Flowcut Switching: High-Performance Adaptive Routing with In-Order Delivery Guarantees

Tommaso Bonato, *ETH Zurich*, Daniele De Sensi, *Sapienza University of Rome*, Salvatore Di Girolamo, *ETH Zurich*, Abdulla Bataineh, *HPE*, David Hewson, *HPE*, Duncan Roweth, *HPE*, Torsten Hoefler, *ETH Zurich*

***Abstract*—Network latency severely impacts the performance of applications running on supercomputers. Adaptive routing algorithms route packets over different available paths to reduce latency and improve network utilization. However, if a switch routes packets belonging to the same network flow on different paths, they might arrive at the destination out-of-order due to differences in the latency of these paths. For some transport protocols like TCP, QUIC, and RoCE, out-of-order (OOO) packets might cause large performance drops or significantly increase CPU utilization. In this work, we propose *Flowcut switching*, a new adaptive routing algorithm that provides high-performance in-order packet delivery. Differently from existing solutions like *Flowlet switching*, which are based on the assumption of bursty traffic and that might still reorder packets, *Flowcut switching* guarantees in-order delivery under any network conditions, and is effective also for non-bursty traffic, as it is often the case for RDMA. On top of this, Flowcut can be implemented either at the switch or NIC level providing flexibility and different tradeoffs.**

***Index Terms*—networking, routing, load balancing, data centers, rdma**

I. Introduction and Motivation

THE interconnection network is a performance critical component for applications running on data centers. With the emergence of new paradigms such as disaggregated memory [1] and distributed storage [2], the network must guarantee high throughput and low latency, with close-to-zero queuing delay [3].

Because most data center networks are characterized by multiple paths between any pair of endpoints [4]–[7], *adaptive routing* (a subset of *traffic load balancing*) algorithms can be used to distribute packets on multiple paths, thus reducing queuing delays [8]. In contrast, ECMP [9] is a widely used routing algorithm, which distributes packets over equal-length paths based on the result of a hash function computed on some fields in the packet header. However, although widely adopted for its simplicity [10], vanilla ECMP does not use any congestion information when selecting paths. For this reason, packets often experience congestion, even when there are alternative non-congested paths that could have been used instead [7], [11]–[13]. To avoid congestion, some routing algorithms estimate the congestion of different paths and route packets on the least congested path. Routing decisions can be taken at different degrees of granularity, ranging from per packet [12], [14] to per flow[1] basis [6], [15], [16], with some intermediate solutions [17], [18].

A. Per Packet Adaptive Routing

Adaptively routing on a per packet basis usually provides the lowest packet latency [8], because each packet could be sent on the least congested path [14], [19], [20]. However, due to differences in latencies of different paths, packets of a flow can arrive at their destination in a different order. Because protocols like RoCE, TCP, and QUIC [21], [22] need to deliver the packets to the application in-order, out-of-order (OOO) packets increase the flow completion time (FCT) due to reordering of the packets on the receiver side and/or packets retransmission [21], [23].

For example, it has been shown that, even if only 0.6% of the packets are lost on a 40 Gbps network (and hence generate out-of-order packets), RoCE consumes 80% of an Intel Xeon E5-2630 v3 2.4GHz core time [24] to reorder those packets when using selective ACKs (i.e., almost one entire core is dedicated to packet reordering). Moreover, many RoCE implementations use the *go-back-n* protocol to retransmit OOO packets [25], [26], further exacerbating the problem. This has been illustrated in a recent work [27] where under per-packet multipath, the fraction of out-of-order packets skyrockets once load exceeds 0.3, and RDMA's simple go-back-n retransmit drops goodput to below 10% of ideal. Similarly, a recent study on the performance of several libraries implementing the QUIC protocol shows that, even when less than 1% of the packets arrive out of order, this can cause a performance drop of up to 10 times if OOO packets are treated as packet lost, or a 30% increase in the CPU usage if they are reordered [21].

Moreover, acceleration techniques like *Generic Receive Offload* (GRO) in some cases are not applicable in presence of out-of-order packets, since they rely on in-order delivery [28] (e.g., Linux GRO [29]). Whereas reordering can be offloaded to programmable NICs to avoid wasting CPU cycles, this is usually a complex task and complicates the hardware design of these devices [21].

B. Per Flowlet/Flowcell Routing

To reduce the number of OOO packets, some load routing algorithms take new decisions at a coarser granularity, for

[1]A flow is usually defined as a sequence of packets from a source to a destination. For example, in an IP network a flow can consist of packets with the same 5-tuple, composed of: source and destination addresses, source and destination ports, and transport protocol.

example, on a per *Flowcell* [18] or per *Flowlet* [5], [7], [17], [30], [31] basis. A flowcell is a sequence of consecutive packets, with a cumulative length not larger than some threshold. A flowlet is instead defined as a sequence of consecutive packets with a variable cumulative length, and separated in time from the subsequent flowlet by a fixed time interval. Different flowlets can be routed on different paths and, if the time gap is sufficiently large, packets arrive at the destination in order.

C. Limitations of Existing Algorithms

Flowlet switching works well for bursty traffic [31] (as it is often the case for TCP traffic), as burstiness creates more opportunities for the creation of new flowlets and, thus, for better balancing the traffic on different paths. However, *Remote Direct Memory Access* (RDMA) protocols such as *RDMA over Converged Ethernet* (RoCE) [25], [32] and *Scalable Reliable Datagram* (SRD) [33] are often used for east-west traffic in large data centers (like those of Microsoft [26], [34], Google [35]–[38], and Amazon [33]). Differently from TCP, on RDMA hardware rate limiters are often utilized [37], [39], [40]. As a consequence, RDMA is often characterized by fewer flowlets [26], [41], thus limiting adaptive routing opportunities when using Flowlet switching.

Moreover, although algorithms based on flowcells and flowlets can reduce the number of out of order packets, they cannot completely guarantee in-order delivery. Indeed, in-order delivery can only be guaranteed if the threshold used to identify a burst (i.e., a flowlet), is larger than the latency difference between the paths on which the flowlets are forwarded. This depends on the workload running on the network, the network topology (e.g., when having paths of different lengths), and the underlying network conditions [17], [42].

However, network conditions vary with time in unpredictable ways [43], [44] due to, e.g., changes in traffic patterns and link failures [45]. Thus, having a fixed threshold for all the scenarios cannot be optimal. Figure 1 shows how the best window size for Flowlet switching can heavily change depending on the type of workload being run. Moreover, even for the same workload, the optimal window can be affected by the input data and the actual network configuration. Finally, partial or total failures can dynamically change the network state and affect the optimal threshold choice (Figure 1).

While out of order delivery remains a challenge for many high-performance transports, some recent NICs (e.g. NVIDIA Mellanox ConnectX-5 [46]) now include on-card reordering engines. However, these features are proprietary, tied to specific switch fabrics and transport protocols, and are not broadly available across vendors or workloads. By contrast, Flowcut's in-flight-drain mechanism works entirely in software (or with minimal switch assistance), is transport-agnostic, and can be deployed on any commodity NIC without specialized hardware support.

D. Flowcut Switching

For the aforementioned reasons, in this paper we design a new adaptive routing algorithm for in-order RDMA traffic

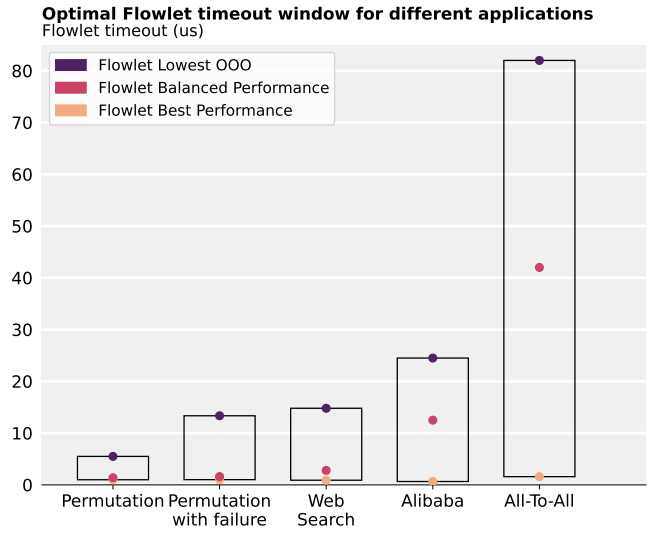

Fig. 1. Optimal flowlet window for different workloads and network conditions. The optimal performance is calculated using the average flow completion time. Web search and Alibaba are traces with their distribution presented in Figure 6

(RoCE), named *Flowcut switching*. Flowcut switching can adaptively route non-bursty traffic regardless of the network conditions, while providing at the same time in-order delivery guarantees. A Flowcut is a sequence of consecutive packets of a flow, all being forwarded on the same path. Differently from Flowlet switching, each switch tracks the number of in-flight packets for each active flow, and creates a new Flowcut (and route it on a different path) only if there are no in-flight packets for the flow between the switch and the destination host. By doing so, all the in-flight packets of a flow are always routed on the same path, thus guaranteeing in-order delivery. As an optimization to reduce memory footprint, Flowcut can also be implemented only at the ingress switch or the NIC level.

In addition, if the NIC or ingress switch (i.e., the switch directly connected to the source of the flow) detects that a flow is experiencing congestion and it cannot create a new Flowcut (because there are still in-flight packets), it can temporarily pause the transmission of packets for that flow. When there are no in-flight packets for the flow, transmission is resumed, and a new Flowcut is created. The new Flowcut is routed on a different (and less congested) path, allowing non-bursty flows to react to congestion.

The Rosetta switches of HPE's Slingshot network [14], designed by some of the co-authors of this paper, implement key ideas that we discuss here, such as the use of flowcuts and an intelligent draining process. In our model, we use timing-based congestion detection to initiate re-routing, which is a key strategy implemented by Slingshot to detect and respond to congestion. We evaluate Flowcut switching through simulations on popular data center network topologies (blocking and non-blocking fat trees [10], [33], and Dragonfly [35]) and workloads, and we compare it with different state of the art adaptive routing algorithms. We show that Flowcut switching improves *flow completion time* (FCT) up to 50% with respect

to per flow routing algorithms like ECMP, and up to 40% compared to Flowlet switching (when setting the timeout value to re-order only few packets, more details in Section III-C2), while guaranteeing in-order delivery of packets. Moreover, we also showcase how Flowcut switching can handle failures effectively and how it beats ECMP by 5x in such scenario.

II. General Design

By design, Flowcut switching supports three deployment models to accommodate hardware and operational constraints: **Full switch** (state and re-routing at every switch); **Ingress only** (state and re-routing only at the ingress switches); and **NIC only** (state and re-routing at the endpoints). In Section III-C3, we show that while the **Full switch** deployment achieves the best overall performance, the far more memory-efficient **NIC only** variant performs comparably in all tested scenarios.

We discuss in the following how Flowcut can be implemented in the switches, and in Section II-D how it can be implemented in the NICs without any support from the switches. As previously mentioned, when using Flowcut switching switches needs to keep a small state (a few bytes) for each active flow (i.e., for flows that have in-flight packets). The Flowcut state includes the port on which the last packet for the flow has been transmitted, the port from which that packet has been received, and the number of in-flight bytes. Each switch stores information about active flowcuts in a *Flowcut table*, indexed by a unique key (e.g., the 5-tuple for IP flows), and implemented, for example, as a *content-addressable memory* (CAM). We show in Section III-A2 that given current network characteristics, the *Flowcut table* can fit in the memory of most existing switches.

To simplify the exposition, we show in Figure 2 a simple reference topology. We define as *ingress* switch the switch connected to the source host where the flow originates, and as *egress* switch the one connected to the destination host of the flow. Although in this example there is a single hop between ingress and egress switches (traffic goes either through *Switch A* or *B*), Flowcut switching works for any arbitrary number of hops.

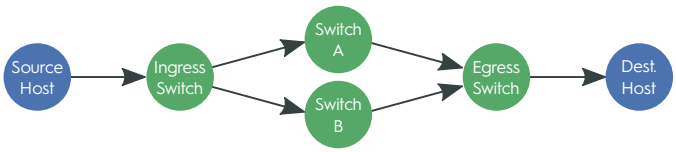

Fig. 2. Network example.

A. Flowcut Creation and Termination

The switch keeps the information on a flow in the Flowcut table as long as there are in-flight packets for that flow. When a switch receives a packet, it uses the flow key to index the Flowcut table. If an entry is found, the packet is forwarded to the output port stored in the Flowcut table entry, and the number of in-flight bytes is increased by the size of the packet. If an entry is not found, the switch creates a new entry for that flow and stores it in the table. Then, it selects a new output port (according to the destination), stores it in the Flowcut table, and forwards the packet to this port, thus creating a new Flowcut.

On fat tree networks, Flowcut switching uses up/down routing and selects, in the *up* direction, the least loaded port. On Dragonfly networks [35], [47], Flowcut switching uses UGAL routing for selecting the least loaded path [47], [48]. Flowcut switching is independent from the specific algorithm used for selecting the output port, and any other adaptive routing algorithm can be used. In particular, for UGAL, the choice whether to route subflows minimally or not is still done by the global adaptive algorithm irrespective of Flowcut, which is orthogonal to it. For each Flowcut, the switch keeps track of the number of in-flight bytes that are still being routed in the network between itself and the egress switch. For this reason, after the egress switch forwards the packet to the destination host of the flow, it sends back an acknowledgment packet (*ACK* for short) to the ingress switch. Each switch forwards the ACK packet on the input port on which the data packets of that Flowcut were received, thus forwarding the ACK through the same switches crossed by the data packet (but in the reverse order).

An ACK packet contains the key of the flow it belongs to, the size (in bytes) of the corresponding data packet and, a timestamp, and a counter (discussed later), for a total of 20 bytes (we provide a more detailed analysis in Section III-A1). Because existing networks have a *Maximum Transmission Unit* (MTU) of a few thousand bytes, ACKs introduce a negligible bandwidth overhead.

When a source switch receives an ACK packet, the in-flight bytes counter for the corresponding Flowcut is decreased by the number of bytes carried by the ACK packet. When the counter drops to zero, the Flowcut entry is removed from the table. By doing so, if a switch receives later a new packet for that flow, it will create a new Flowcut, possibly routing its packets on a different path. By allowing a flow that has no in-flight packets to choose a new route, Flowcut switching guarantees in-order delivery of packets while choosing less congested routes for new arriving packets.

We observe that if we want to preserve the in-order guarantee, the first packets that are sent out before receiving any ACK back must be all sent using the same path (using ECMP for example) in order to avoid any possibility of generating out-of-order packets. If, for example, we initially send out a BDP (Bandwidth-delay product) worth of data, then they need to follow the same path. Only once we start getting ACKs back the normal Flowcut switching mechanism starts to kick in.

To better describe how Flowcut switching works, let us consider the example in Figure 3, showing the packets exchanged between the host and the switches depicted in Figure 2. The source host sends the first packet (data packets are denoted with →), and the ingress switch decides to forward it to *Switch A*. Because the ingress switch receives the three subsequent packets while there are still in-flight packets, it forwards them to *Switch A* again. In the meanwhile, the ingress switch receives four ACK packets (—◆). When the fourth ACK is received, there are no in-flight bytes for that flow, and the Flowcut entry is removed from the table. When the fifth

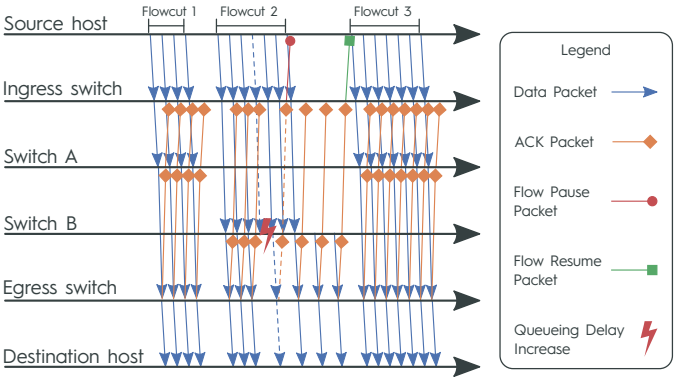

Fig. 3. Flowcut switching example.

packet arrives, the ingress switch does not find any entry for that Flowcut in its table, and a new Flowcut is created (Flowcut 2). This time, the ingress switch decides to send the packet to *Switch B* (e.g., because the output port towards *Switch A* became congested in the meanwhile).

B. Flow Draining

Whereas Flowcut switching guarantees in-order delivery, a flow could always have some in-flight bytes and thus, even if the selected path is congested, switches may not be able to reroute that flow on a different path. This might be an issue for protocols like RoCE that exhibit a more steady behaviour with not too many bursts [26]. To balance the traffic even in presence of steady traffic, if the ingress switch detects that a flow is experiencing congestion, it can decide to force the creation of a new Flowcut, that will potentially be routed on a different path. To do so, the switch asks the source of the flow to stop sending packets for that flow. This causes the flow to be completely drained (i.e., all the in-flight packets are received by the destination and the ingress switch receives all the corresponding ACKs). When the ingress switch receives the last ACK for that flow (i.e., there are no in-flight packets), it deletes the Flowcut from the table and asks the source of the flow to resume the transmission of that flow. When a new packet of that flow is received, the ingress switch does not find a Flowcut for that flow in the table, and can thus create a new one, to be forwarded on a different (and less congested) path.

The draining process is determined by two main aspects: how the ingress switch communicates to the source of the flow to pause/resume the transmission of the packets of that flow (that depends on the specific network) and how it decides that a flow needs to be drained and rerouted. Regarding the first aspect, an effective mechanism, called Fine Grain Flow Control (FGFC) [49], is available on the Slingshot [14] network. With such mechanism the switch returns FGFC frames for flows causing congestion (similarly to pause frames). These identify a flow using either the normal 5-tuple hash or an identifier provided by the NIC. The possible operations are XOFF, XOFF with credit, and XON. An XOFF with credit instructs the NIC that a flow should send a certain amount of data in the pause period. If the NIC ignores FGFC the traffic class will be paused. FGFC is designed to interoperate with an Ethernet NIC managing large numbers of flows, many of which can make progress while others are blocked or being paced. Otherwise, if such a mechanism is not available and cannot be implemented, the ingress switch can rely on *Priority-based Flow Control* (PFC) packets [50], thus pausing all the flows belonging to the same traffic class of the flow to be drained.

To estimate the congestion, the ingress switch adds a timestamp to the packet before forwarding it (e.g., by inserting a custom header between Ethernet and IP headers). Then, the egress switch removes this additional header from the data packet before forwarding it to the destination host, and copies the timestamp in the ACK packet. We note that this process is done completely in hardware by the switch with no cpu utilization or increase in the end-to-end latency [14], [51]. Moreover, to address the extra space in the header due to the timestamp, the advertised MTU size is slightly reduced. This is similar to the approach taken by Swift where 4 bytes are used by timestamp in the packet header [52]. Every time the ingress switch receives an ACK packet, it computes the round trip time (RTT) $\tilde{r}$ by computing the difference between the current timestamp and the one carried in the ACK packet. Because packets can have different size, the switch could observe an increase in $\tilde{r}$ due to larger transmission latency for larger packets, even without any increase in the queueing latency. For this reason, the switch removes from $\tilde{r}$ the transmission latency, by subtracting the quantity $p \cdot h \cdot t$, where p is the packet size, h is the number of traversed hops, and t is the transmission rate of the switch. To compute h each data packet also carries a counter (incremented at each hop) representing the number of traversed hops, and copied back in the ACK packet by the egress switch. We note that, with low diameter topologies where multiple paths are present between the sender and the receiver, it is important that the ACK takes the same reversed path as the corresponding data packet. To do so, we propose two possible solutions: the most straightforward one is to add a single extra field in the switch entry to indicate the input port from which the packets of the flow have been received. That port will be used as the next hop when the ACK is received, thus allowing ACKs to follow the reverse path. The other approach is to move the entire implementation to the NIC (more details in Section II-D) since the intermediate paths then become of no importance for Flowcut.

To assess if a packet experienced a too high queuing latency, the switch needs to compare the measured RTT to some reference RTT. For this reason, the switch keeps track of the minimum observed RTT $r_{min}(h)$ for each hop count h. It is worth remarking that this information is globally stored and does not need to be stored for each flow. Because existing data center networks have a diameter lower than ten [14], [53], this process requires storing only a few extra bytes in the switch memory.

Then, the switch can compute a normalized RTT (always ≥ 1) as $\frac{\tilde{r}}{r_{min}(h)+p \cdot h \cdot t}$. To avoid reacting to temporary increases in the RTT due to transient congestion, the switch stores, for each flow, an exponential moving average r of the normalized RTTs. If the average exceeds a threshold, the flow is drained

Param.	Description
l	Average packet latency (ingress to egress switch).
b	Average rate of data packets.
a	Average rate of ACK packets.
r	Average RTT.
$\tilde{r}$	Measured RTT for a packet.
$r_{min}(h)$	Min. observed RTT for packets crossing h hops.
h	Number of hops traversed by a packet.

TABLE I
DESCRIPTION OF USED PARAMETERS.

and rerouted. Limiting the RTT also limits the number of in-flight packets, thus in turn limiting the maximum time required to drain a flow. To proactively react to congestion, the switch also keeps a moving average of the difference between the current and the previously measured normalized RTT. If the average exceeds a threshold, the congestion on the path is growing too quickly and the switch drains and reroutes the flow currently being forwarded. We summarize the parameters used in our discussion in Table I.

It is worth remarking that although the measured RTT also includes the queuing latency experienced on the reverse path by the ACK packets, this is usually negligible because ACK packets can be forwarded with higher priority with respect to data traffic, as it happens in the Slingshot [14] network. Moreover it has been shown that, in the presence of congestion on the reverse path, RTT measurements with ACK prioritization are indistinguishable from measurements taken in a scenario with no congestion on the reverse path [54].

Some existing congestion control algorithms also use the RTT as a congestion signal [52], [54]. However, while in those cases this is used to reduce the transmission rate of the sender so that it matches the bandwidth of the path, in our case we use it to trigger the selection of a better path, hopefully with a higher bandwidth. We elaborate more on the interaction between the adaptive routing and congestion control algorithms in Section IV-B.

To better describe the draining process, we report an example in Figure 3. When the fourth packet of *Flowcut 2* (denoted with a dashed arrow) arrives at *Switch B*, it experiences some delay (), e.g., because packets of other flows are already queued for transmission to the egress switch. As a consequence, the transmission of the corresponding ACK is delayed as well. When the ingress switch receives the ACK, it realizes that the packet experienced congestion (because its normalized RTT exceeded the threshold).

Thus, it asks the source of the flow to not send any other packet for that flow (). Eventually, when the ingress switch receives all the ACKs for the in-flight packets, it removes the Flowcut from the table and resumes the packets transmission (). The source starts transmitting the packets again and, because there are no entries for that flow in the table of the ingress switch, the switch creates a new Flowcut, routing it through a less congested switch (*Switch A*).

Finally, we observe that the draining logic generalizes from *ingress only* to a *full switch* implementation that accounts for per-hop latency. While it increases memory requirements and the number of timestamping operations, it achieves the highest precision and effectiveness.

Variant	Memory impact	Adaptivity	ACK routing	Deployment effort
Full switch	Per-flow entry in every switch (highest memory requirements)	Re-routing capabilities at all switches for best performance	Must follow reverse path storing the input port at switches	Upgrade all network switches
Ingress only	State only in ingress switches (medium memory requirements)	Reroute only at ingress, slower and less precise	Free to take any path	Upgrade ToR; core untouched
NIC only	No switch state; per-flow info lives in endpoints (lowest memory requirements)	Adaptivity limited to ECMP hash change at source	Free to take any path	Flowcut-aware NIC firmware/-driver; fabric unchanged

TABLE II
QUALITATIVE COMPARISON OF FLOWCUT DEPLOYMENT VARIANTS

C. Switch Driven Variants

While so far we described a full switch implementation of Flowcut, the design also supports an ingress-switch only implementation that is attractive for incremental rollouts and for fabrics with tight memory budgets.

Full switch Every switch along the path maintains per flow state and may reroute locally. This yields the fastest reaction and highest precision at the cost of memory across the fabric.
Ingress only Only the ingress switch keeps per flow state and decides when to reroute. Core and aggregation switches forward packets without Flowcut state. Because intermediate switches do not update counters for packets in flight, ACK packets do not need to follow the exact reverse path; they can return along any path from the egress to the ingress by carrying and the ingress progresses ACKs before triggering rerouting. In practice, each top of rack switch holds state only for flows sourced by its directly attached hosts.

D. NIC-only Implementation

In the previous section, we assumed Flowcut switching to be implemented by the switch, without any support from the NIC (except for PFC-like mechanisms). Due to its simplicity, it could be implemented on most existing programmable switches [55], [56]. However, Flowcut switching could also be entirely implemented in hosts's NICs (e.g., by using programmable NICs [57]–[62]) without any switch support, similarly to *Flowbender* [63] or *REPS* [64]. The switch would not need to store any information about the active flowcuts, and the NICs would only need to store information about the flowcuts it generates, thus also reducing resource occupancy. Because NICs usually have larger memory than switches (and

could also use the host DRAM [57]), this would allow the use of flowcuts on networks with larger RTTs.

For example, the source NIC would add to the packet the timestamp, that would be copied back by the destination NIC in an ACK packet. The source NIC then would compute the RTT and, if the current path is congested, it would stop the transmission of packets for that flow until all the ACKs are received. If the switches use ECMP, the source NIC could force the selection of a different path by changing one of the fields on which the ECMP hash is computed (e.g., the source port), so that the packet (and the subsequent ones), will be forwarded on a different path (similar to what is done in SRD [33]). The original source port can be inserted in an additional header, and then restored by the destination NIC. However, in this case, the selection of the path is random and congestion oblivious. Moreover, because there is no support from the switch, ACKs must also have Ethernet and IP headers, thus increasing their network bandwidth occupancy (even if still negligible for MTU-sized data packets).

This flexibility in moving functionalities from NICs to switches and vice versa allows incremental deployment of Flowcut switching, either by implementing it on programmable NICs without changes in the switches, or by implementing it in top-of-the-rack (ToR) switches without any change at the endpoints.

In Table II we summarize the different variants of Flowcut with their different advantages and disadvantages.

III. Evaluation

In this section we first analyze the resource requirements of Flowcut switching in terms of network bandwidth and switch memory (Section III-A). We then describe our simulation environment (Section III-B) and evaluate Flowcut switching on different networks and workloads, by comparing it with other state of the art adaptive routing algorithms (Section III-C). Finally, we share some results based on a real cluster running Slingshot on 2,048 endpoints (Section III-C5).

For Flowcut, we use the *NIC only* version (Section II-D) for all experiments unless specified differently. This is to showcase the performance of the easiest to implement version of Flowcut. Finally, in Section III-C3 we showcase how implementing Flowcut on switches can further improve the performance.

A. Network Resources Occupancy

Flowcut switching increases the consumption of network resources, both in terms of network bandwidth due to the transmission of additional information in data and ACK packets, and of switch memory.

1) Network Bandwidth: Flowcut switching requires explicit ACK packets to be sent from the egress to the ingress switch. To reduce the Ethernet header overhead, ACKs can be sent using a custom packet format, using the first byte of the Ethernet preamble to distinguish them from data packets. ACKs do not have the IP header, because they are routed on the backward path using information stored in the switches (input and output port for the flow). Thus, ACKs are only composed of a preamble byte and a payload containing the key of the flow (13 bytes if we consider the 5-tuple composed of the source and destination IP addresses, source and destination ports, and transport protocol identifier). To further reduce the number of extra transferred bytes, the flow key could be mapped to a smaller space. For example, by transmitting only the least significant bits of the addresses (depending on the subnet mask) or by using a unique flow key generated by the source NIC and encoded on fewer bytes. To be as general as possible, however, we consider the case where all the 13 bytes of the 5-tuple key are transmitted in the ACKs.

Flowcut switching also needs to add the timestamp and the hop count to both data and ACK packets. Because end-to-end latencies in existing data center networks are in the order of a few microseconds [14], [35], [52], [65], if timestamps have nanosecond resolutions, RTTs can be encoded using 2 bytes (for RTTs up to 64 microseconds). Data center networks have a diameter lower than ten [14], [53], and storing the number of hops only introduces a 4 bits overhead in both the data and ACK packets. To align the packet size to 1 byte, additional 4 bits are unused and reserved for future extensions. Alternatively, on IP networks switches could use the *Time to Live* (TTL) field of the IP header instead of adding an explicit counter to the data packet. We summarize the extra network bandwidth consumption in Table III.

Algorithm	Congestion Signal	Per-Packet Extra Bytes (Wire)	Per-Flow Extra Bytes (Memory)
Flowcell [18]	None	0	2
Flowlet [7], [17]	Idle Time	0	5
Flowcut Full switch	RTT	20	11
Flowcut Ingress/NIC only	RTT	20	10

TABLE III
Resource occupancy of different adaptive routing algorithms. We report the extra bytes sent for each packet, and the extra bytes stored for each flow in the switch memory. We note that although the per-packet or per-flow cost is almost the same for all Flowcut variants, in practice *NIC/Ingress only* require significantly less resources on the NIC/Switch due to less active flows.

For 1KiB packets, the per-packet Flowcut switching network bandwidth overhead is smaller than 2%. We note that modern switches, such as the Rosetta switches used in Slingshot [14], are capable of supporting per-packet ACKs even at modern network bandwidths ($\geq 200Gb/s$).

In principle, for protocols like TCP we could avoid sending explicit ACKs and piggyback the additional information on top of the TCP ACKs. This has, however, several disadvantages. For example, relying on TCP ACKs would delay the reception of the congestion feedback on the switch, due to the extra latency required to cross the receiver host network stack. This latency would increase even more if the TCP ACK also carries data (due to the additional transmission latency at each hop) or if ACK coalescing is performed by the receiver host. Having said that, Flowcut would still be able to work, although slightly delayed, even in cases where per-packet acking is not feasible and cumulative ACKs are used.

2) Switch/NIC Memory: Flowcut switching, when implemented in all switches, require switches to store, for each

flow crossing them, the input and output ports for the flow (1 byte each for 256-port switches) and the number of in-flight bytes (3 bytes for up to 2 MiB of in-flight data per flow), for a total of 5 bytes per flow. The switch also needs to store the normalized RTT moving average, that requires 2 bytes, as discussed in Section III-A1, the normalized RTT measured for the last received ACK, and an exponential average of the RTT difference, requiring an additional 4 bytes per flow. For the Ingress only and NIC-based implementations, Flowcut does not need to store the input port saving one byte. This is because in most topologies the NICs are linked only to one switch and the ACK packet will necessarily travel to that switch regardless of what path it takes to get there. We summarize the memory requirements in Table III. We also report the per flow state for Flowlet switching, that requires 5 bytes per flow [7], [17], and for flowcell switching, that requires 2 bytes per flow [18].

The memory occupancy depends on the number of active flows (i.e., flows that have at least one in-flight packet). Indeed, the NIC or a switch deletes the information about a Flowcut from the table as soon as there are no in-flight packets for that flow. To simplify the analysis, we assume that all the transmitted packets have size M (equal to the MTU). If each host has f flows, and the network bandwidth B is equally divided among these flows, each of the flow has a bandwidth $b = \frac{B}{f}$. The number of in-flight packets for a flow can be computed as the bandwidth-delay product. Thus, if the packet latency is l, each flow has $\frac{b \cdot l}{M}$ in-flight packets.

If each flow has at least one in-flight packet, all the f flows are active. Otherwise all the flows have, on average, less than one in-flight packet. Each host can have at most $\frac{B \cdot l}{M}$ in-flight packets, and in this case each in-flight packet belongs to a different flow. Thus, each host would have $\frac{B \cdot l}{M}$ active flows. To summarize, if there are H hosts in the system, there will be a total number of active flows F in the network equal at most to:

$$F = \begin{cases} H \cdot f, & \text{if } \frac{B \cdot l}{f \cdot M} \geq 1 \\ \frac{H \cdot B \cdot l}{M}, & \text{if } \frac{B \cdot l}{f \cdot M} < 1 \end{cases} \quad (1)$$

It is worth remarking that when each flow has, on average, less than one in-flight packet, increasing the number of active flows per host f does not increase the number of active flows in the network. Thus, the maximum number of active flows that can be in the network at any time is only determined by the number of hosts H, the network bandwidth B, the latency l, and the MTU M (assuming that each packet is as large as the MTU).

This can also be observed in Figure 4(a), where we report the modelled maximum memory occupancy of Flowcut switching for different values of RTTs, network bandwidth, and number of hosts. The memory occupancy is computed as the total number of active flows (derived from Equation 1) multiplied by the number of bytes that a switch needs to store for each flow (Table III). For the *Full switch* variant of Flowcut, we consider a worst case scenario where a switch stores information for all the flows in the network (that is usually not the case since each flow crosses only a portion of the switches). On the other hand, the Ingress only variant has a much smaller memory requirement as only flows coming from its input ports need to be saved. Similarly, the *NIC only* version of Flowcut has a much smaller memory requirement since it only needs to store flows active on a given NIC.

In Figure 4(a), we fix the number of hosts to 1024, and the network bandwidth to 200 Gb/s, and we analyze how the memory occupancy changes for different RTTs. We can see that memory occupancy grows linearly with the RTT but, for a fixed RTT, when increasing the number of flows per host after a certain point the switch memory occupancy remains constant. Because *North-South* traffic in data centers (i.e., traffic directed to hosts outside the data center) can be characterized by RTT in the order of milliseconds, Flowcut switching is only used for *East-West* traffic (i.e., to traffic between hosts inside the data center). We also observe that even for a latency of 50 microseconds, much higher than that observed for intra-datacenter traffic [14], [35], [52], [65], the memory occupancy on the switches does not grow beyond 7 MiB for the *full switch* implementation.

In Figure 4(b), we instead fix the number of hosts to 1024, and the RTT to 5 microseconds, and we analyze the memory occupancy for different values of the network bandwidth, ranging from 200 Gb/s to 1.6 Tb/s. Even in this case, we can see that Flowcut switching memory occupancy is limited even for future 1.6 Tb/s networks. Similarly, we analyze in Figure 4(c) the memory occupancy for a 800 Gb/s network with 5 microseconds RTT, for different number of hosts, ranging from 1024 to 65536. In this case we can see that with more than 16384 hosts, the memory occupancy exceeds 50 MiB for the Full switch implementation. However, it is worth remarking that on such a high node count, the *ingress switch only* and the *NIC only* variants of Flowcut switching could be a better solution as they only require slightly more than 100KiB and 10KiB respectively.

We also report in Figure 5 the memory occupancy of Flowlet and Flowcell switching. If we consider that existing switches have memories of tens of MiB (excluding the packets buffer) [66], [67], and that our analysis considers the worst case (with the highest number of active flows on each switch), we believe that Flowcut switching can effectively be implemented with the current technology. Finally, we note that the flexibility of the different Flowcut variants allows users to choose what best fit their needs with the *NIC only* version requiring significantly less space than Flowcell and Flowlet.

B. Test Environment

1) Simulated networks: To analyze Flowcut performance we simulate different types of traffic using the SST simulator [68], [69]. Hosts in the simulated systems communicate through an RDMA-like protocol, and the network uses credit-based flow control to guarantee lossless behavior (protocols implementing RDMA are usually deployed on lossless networks [34], [37]). In our analysis we simulated the following systems, all based on 1024 nodes, 200 Gb/s networks and 1us links latency:

- **Fat tree (1:1)** This system connects the servers through a 3-level non-blocking fat tree. There are 16 pods with a

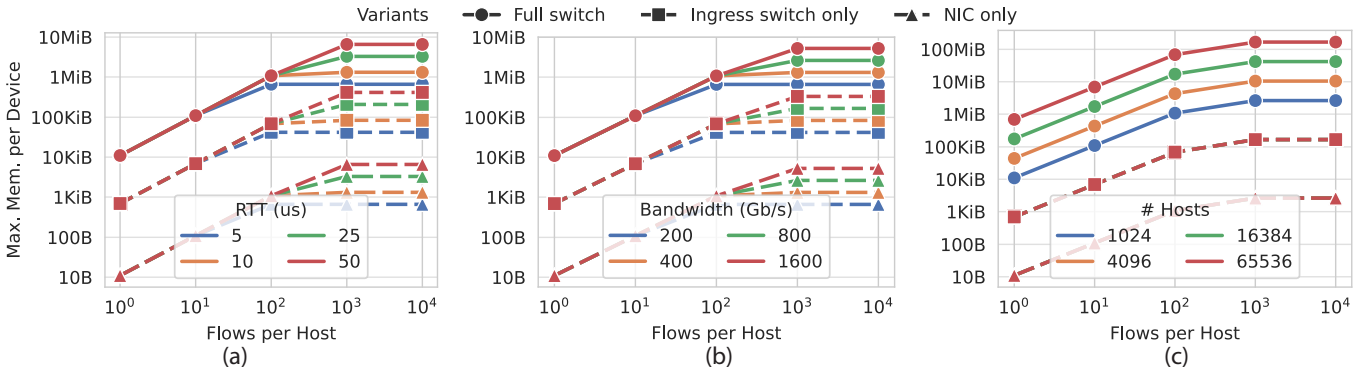

Fig. 4. Maximum switch memory occupancy (MiB) of Flowcut switching on different configurations on a network with 2KiB MTU and 64 input and output ports per switch. (a) 1024 hosts on a 200Gb/s network, for different RTTs. (b) 1024 hosts, 5 microseconds maximum RTT, for different network bandwidths. (c) 800 Gb/s network, 5 microseconds maximum RTT, for different hosts count.

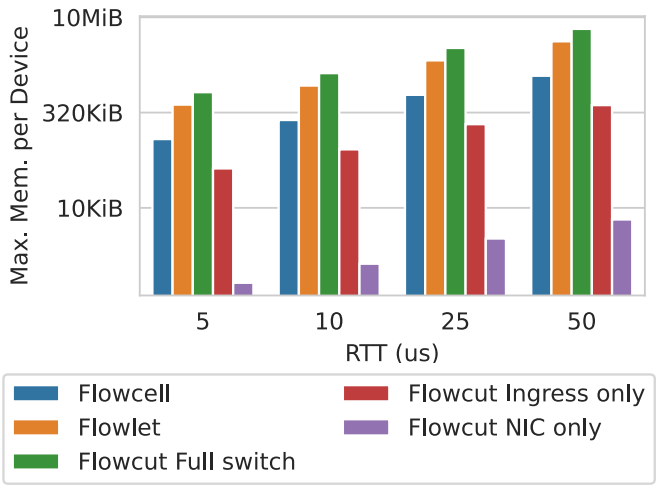

Fig. 5. Maximum memory occupancy (MiB) of different algorithms (10^4 flows per host), on a 200Gb/s network with 2KiB MTU and connecting 1024 hosts. The memory requirement is for switches for all cases but the NIC only one where it is per NIC.

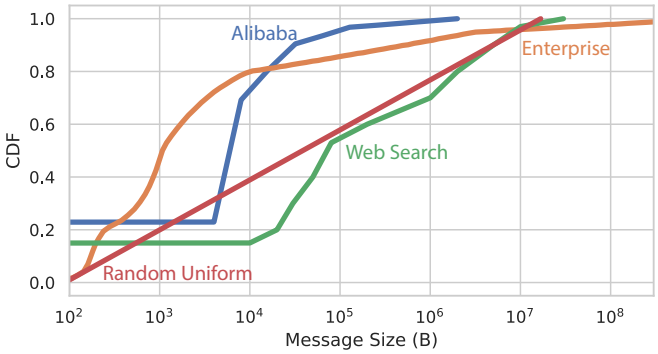

Fig. 6. Flow size distribution for some of the different workloads used during benchmarking.

total of 128 ToR switches and 128 aggregation switches. The aggregation switches are connected to a total of 64 core switches.

- **Fat tree (2:1)** This system also connects the servers through a 3-level fat tree, but with a 2:1 tapering. That means that the ToR switches have less (half) up-links going to the aggregation switches.
- **Dragonfly** This system has the same configuration of a real system deployed at the *Swiss National Supercomputing Center* [70], using the Slingshot interconnect [14]. Nodes are interconnected through a Dragonfly network [47]. The network is composed of 64-port switches. Servers are divided into four Dragonfly groups (256 servers per group). Switches within a group are fully connected, and each switch is connected to 16 servers, as it happens in the Slingshot interconnect [14]. Each group is connected to each other group through 16 links.

2) Workloads: In our experiments we consider the following workloads:

- **Web Search/Enterprise/Alibaba/Random distribution** In these workloads, each host iteratively selects a random partner and sends a message to that partner. The size of the messages is extracted from distributions (Figure 6) collected from real data centers, and used to evaluate several adaptive routing algorithms [7], [17]. On fat tree networks, we adopt the same approach used by CONGA [7], forcing all the traffic to cross aggregation switches.
- **Permutation** This microbenchmark consists in all nodes of the network sending a fixed amount of data to another random node in the network. Each node interacts with at most 1 node, meaning at any given time there will be one send and one receive active. This pattern is extremely important in many applications and even AI workloads. For example, the butterfly all-reduce operation uses several permutation operations during its execution. The same can be true for some all-to-all implementations, for instance when using a windowed algorithm [71].
- **All-to-all** An All-to-all benchmark to stress the network. Note that all-to-all collectives are used in a large number of applications including as a training step for the latest deep learning applications [72], [73]. Moreover, all-to-all collectives are also frequently used in high-performance computing (HPC) [74].

3) Failures: One important aspect of adaptive routing is correctly handling failures. Practically this means avoiding the utilization of failed links as much as possible. We simulate

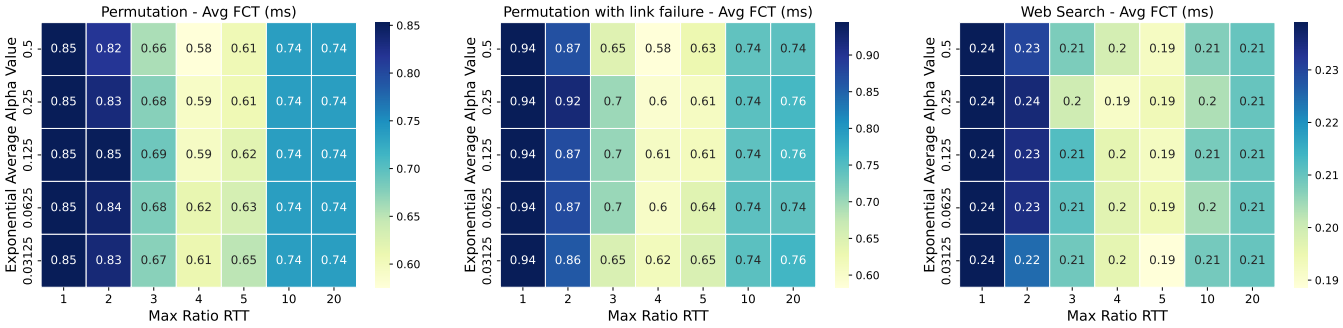

Fig. 7. Heatmap observing how changing the RTT ratio together with the α value from the exponential average affects runtime.

network failures by downgrading the capacity of a number of links (we show results for 1% of link failures) to a bandwidth of 1/10th of the initial capacity (20Gbps). We showcase results for some of the experiments and configurations previously mentioned.

C. Experimental Results

1) Tuning Flowcut parameters: As a first evaluation step we run simulations to observe the behaviour of Flowcut as we change some of its most important parameters previously introduced in Section II-B. We show in Figure 7 the results using a different set of workload and network conditions. We do this to demonstrate that, regardless of the workload or the network's configuration and conditions, Flowcut switching can work effectively without needing extensive parameter tuning.

For each heatmap, on the x-dimension we show different values for the RTT threshold triggering the draining. Because Flowcut computes the exponential moving average of the last observed RTT values as $r = \alpha\tilde{r}+(1-\alpha)r$, on the y-dimension, we show different values of α to analyze Flowcut sensitivity to this parameter. Each heatmap reports the average flow completion time in the cells (darker colors represent higher runtimes).

First, we observe that, for all workloads and conditions, α has a minimal impact compared to the RTT threshold. Still, having large values for α introduces some benefits since Flowcut can react to congestion faster. Regarding the RTT threshold, we briefly recall that Flowcut is triggered when the ratio between the average RTT and the minimum observed RTT exceeds such threshold. We observe that a small value always leads to the worst performance. This is expected, because the algorithm would drain flows too often, hence introducing a high overhead from the draining operation. Specifically, a value of 1 means that the algorithm drains a flow every time the measured RTT is higher than the minimum RTT. A similar conclusion can be made for a value of 2 where the draining time would still introduce large overhead to the overall runtime and flow completion time. On the other hand, when setting a large RTT threshold Flowcut switching would never or rarely react to congestion, thus mimicking ECMP behaviour.

In general, differently from Flowlet Switching, any value in a reasonable range (e.g., between 3 and 5 for the RTT threshold) always leads to good performance, as we show in the following section when comparing these results to other algorithms. It may be counter-intuitive that choosing a large value for the RTT ratio threshold (like 4 or 5) still produces good results. However, such a choice has the advantage of having a minimal draining overhead, while also dealing with the most severe congestion scenarios in the network. Last, we obtained comparable results also when running the same analysis on different network topologies (with different over-subscription ratios).

2) Analysis on fat tree networks: We now compare Flowcut with other state of the art algorithms on fat tree networks. We define a packet as arriving out-of-order if the expected PSN does not match the received one. Since Flowlet switching performance significantly changes depending on the choice of its parameters, we consider in the analysis the following alternatives. In *"Flowlet Best Performance"*, the parameters have been tuned so that Flowlet switching can provide the best performance while keeping the number of out-of-order packets below 20%. Likewise, *"Flowlet Balanced Performance"* keeps the number of out-of-order packets below 5%, and *"Flowlet Lowest OOO"* below 2%. For Flowcut switching, we select an RTT ratio of 4, meaning a flow will be re-routed when the measured RTT is 4 times larger than the base one. *Spraying* is the packet spraying algorithm [75], distributing each packet randomly across the available paths. This is close to ideals in fat tree networks when there are no failures and each path is equivalent to the alternatives in length. MP-RDMA [76] is a multi-path RDMA transport that marries ECN-based congestion control with per-packet ACK clocking to continuously steer traffic onto lightly loaded paths and prune underperforming ones, thus bounding out-of-order packet arrivals. A single tunable threshold Δ lets you cap the maximum tolerated reordering. We note that we do not consider the impact of re-ordering in this experiments, thus presenting a worst case situation for Flowcut switching as all other algorithms (expect ECMP) would need extra-overhead to deal with the re-ordering of packets. In the first experiment (Fig 8), we showcase the results of a 8 MiB permutation using a non-blocking fat tree. We report the 99th percentile average flow completion time and number of out-of-order packets. We focus on the 99th percentile since tightly coupled operations complete only when the last flow has finished. Although packet

spraying minimizes the FCT, more than 50% of packets arrive out-of-order. On the opposite extreme, ECMP has the largest FCT, but no out-of-order packets. Flowcut switching, with its in-order delivery guarantee, is characterized by a FCT lower than *Flowlet Lowest OOO* and on-par with *Flowlet Balanced Performance*. MPRDMA performs slightly better than Flowcut but can't guarantee that all packets are delivered in-order. We note that, in all the experiments, the results for Flowcut also includes the time spent draining (more details in Section IV-C), while we assume a best case scenario (worst for Flowcut) of zero re-ordering cost for all the other algorithms.

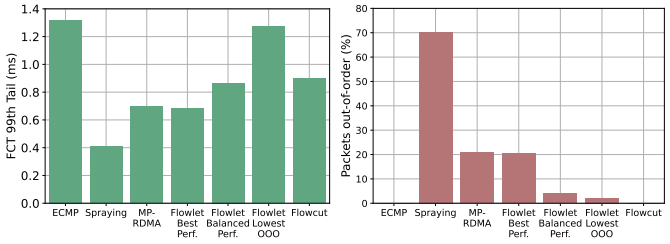

Fig. 8. 99th percentile FCT and amount of out-of-order packets when running a permutation on fattree (untapered).

We now run the same experiment but by disconnecting 1% of the links, as previously described. We show in Figure 9 the results about the average FCT and the 99th percentile FCT. We observe how, in this case, the average flow completion

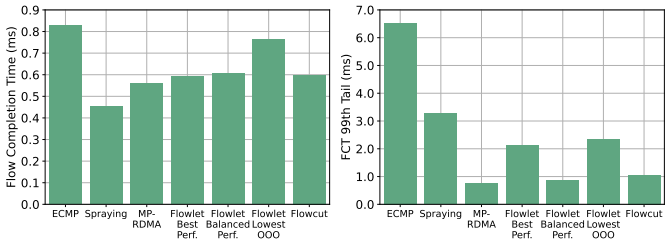

Fig. 9. Average FCT and 99th percentile FCT when running a permutation on fattree while also simulating few links failing.

time confirms our previous findings with Flowcut getting very similar results to *Flowlet Best Performance*, *Flowlet Balanced Performance*, and MPRDMA. This is expected as the average flow is not affected by the failure for the most part and hence, the results are not much different compared to Figure 8. However, when looking at the 99th FCT, the situation becomes the opposite with Flowcut and Flowlet competing for the best performance, while Spraying and ECMP lag behind. Indeed, they are both unable to react to link failures. On the other hand, both Flowlet and Flowcut can somewhat successfully circumvent failed links. Flowcut does this by observing the RTT of the packets going through the failed links increasing, while Flowcut achieves this by re-routing packets that arrive with enough delay between them. Interestingly, this also makes the choice of the Flowlet timeout even more challenging, as in this case all values perform very similarly for the tail latency but not for the average latency where they follow the same pattern we will see throughout these results.

We then simulate a 1 MiB all-to-all using the non-oversubscribed topology and without any failure. We show the results in Figure 10 showing, once more, the 99th percentile flow completion time and the amount of out-of-order packets. Once more, we confirm the previous findings even during

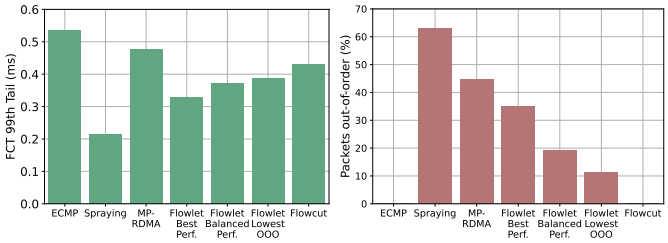

Fig. 10. 99th percentile FCT and amount of out-of-order packets when running an all-to-all on fattree (untapered).

more intensive collectives with Flowcut switching being able to have a tail latency comparable to the one of *Flowlet Balanced Performance*. Interestingly, we note that MPRDMA performs slightly worse in this case, likely due to the different congestion control handling rather than pure load balancing.

Last, in Figure 11 we show the results for the oversubscribed fat tree running the data mining distribution, reporting the average flow completion time and the amount of out-of-order packets.

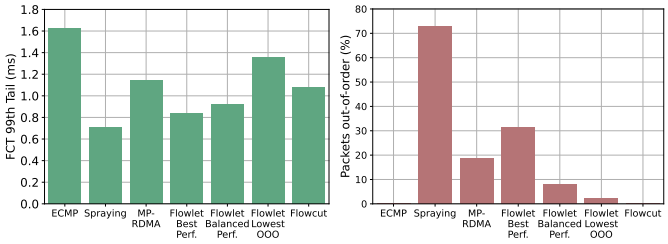

Fig. 11. 99th percentile FCT and amount of out-of-order packets when using a random uniform distribution on a 2:1 oversubscribed fattree.

In this case, there is a smaller difference between the routing scheme in terms of FCT. This is because most of the flows, as shown in Figure 6, are smaller and hence, they complete sending their data before the routing algorithm can make any routing adjustment. However, even in this case, we can still see how Flowcut switching outperforms ECMP and all versions of Flowlet switching. Flowcut switching is only slightly outperformed by packet spraying which, however, is characterized by almost 50% of out-of-order packets. Moreover, in this case MPRDMA performs slightly worse than Flowcut.

3) Analysis of Flowcut variants: We now focus on analyzing the different variants of Flowcut for the results previously presented for the fat tree topology. In particular in Figure 12 we showcase the three variants in 4 different scenarios that we previously presented. We also report again ECMP numbers as a reference. As expected, the Full switch variant is always the best but, interestingly, the NIC version and the Ingress only versions are performing just slightly worse and within 10% of the full and costly implementation.

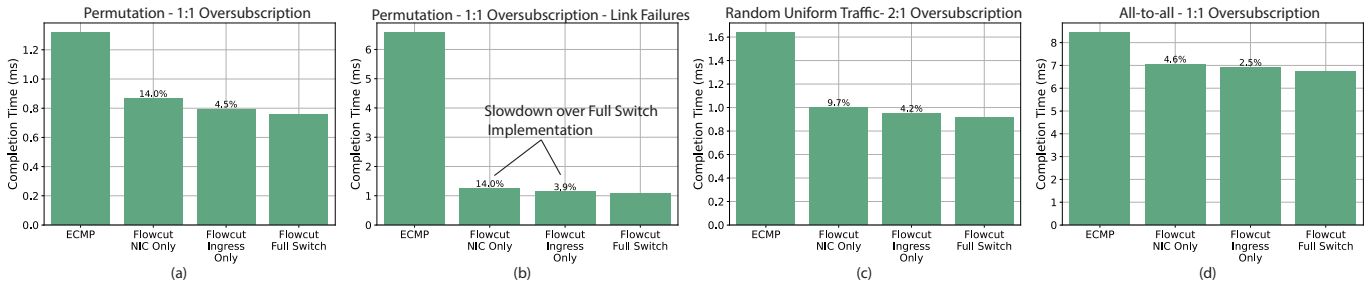

Fig. 12. Showcasing the different performance of the three variants of Flowcut. Annotated is the slowdown over the best variant of Flowcut which is always the one implemented on all switches (Full switch).

4) Analysis on Dragonfly: We now analyze Flowcut performance on the Dragonfly topology, by comparing it with Flowlet switching, ECMP, UGAL [47], and Valiant routing [77] (i.e., data is always forwarded through an intermediate random group). When using the Dragonfly topology, we can observe similar results, to those shown for fat trees. In particular, in Figure 13 we run the *random uniform* distribution on the Dragonfly topology. The results show that Flowcut switching achieves performance close to Flowlet switching, and not too far behind UGAL, while guaranteeing in-order packet delivery. We also note how Valiant performs badly since it forces each packet to go through extra expensive hops, thus always increasing the latency and the overall completion time.

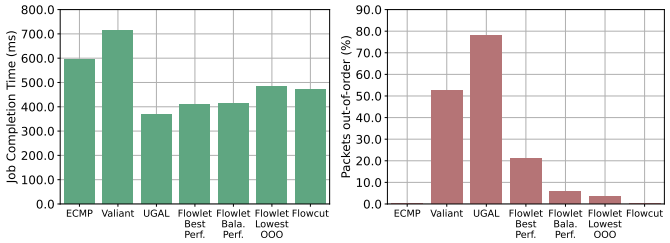

Fig. 13. Runtime and amount of out-of-order packets when using a random uniform distribution on Dragonfly.

A similar result can be observed in Figure 14 where we run the *enterprise* distribution. In this case, there is no performance gap between Flowcut and UGAL, as well as between Flowcut and the balanced performing version of Flowlet. Like for the fat tree case with the data mining distribution, this is because the workload is characterized by many small flows, and hence routing has a smaller impact on FCT. Moreover, it seems changing path slightly less than on a per-packet basis is beneficial as it brings more stability and overall better performance.

5) Empirical Results from a Slingshot System: To further evaluate the performance of Flowcut Switching, we conducted experiments on a Slingshot-based system that implements a variant of Flowcut Switching.

The experiments were performed on a Dragonfly topology consisting of 2,048 nodes, divided into 8 groups. Each group contains 16 switches, with 16 ports per switch dedicated to connecting to local NICs. The remaining ports are allocated for intra-group connections to other switches via local links

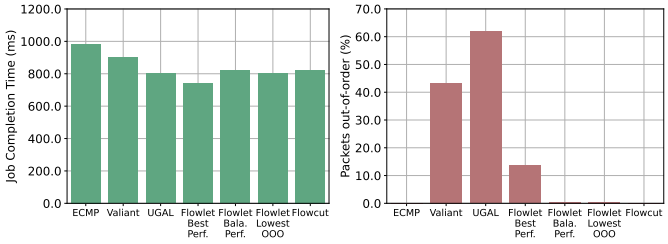

Fig. 14. Runtime and amount of out-of-order packets when using the enterprise distribution on Dragonfly.

and inter-group connections via global links. All links operate at a bandwidth of 200 Gbps.

We run an all-to-all traffic pattern and tested two routing modes of Slingshot. In the first, *ordered* mode, packets are constrained to arrive in order at the receiver using Flowcut Switching. In the second, *unordered* mode, this constraint is relaxed, resembling UGAL behavior as discussed in earlier examples. The *ordered* mode is important for several applications and protocols, for example message envelopes and some amount of eager data, where the receiver needs to process the packets in a strict order to avoid encountering a large performance penalty [21], [23]. On the other hand, several applications and protocols care less about such requirements and can have more relaxed constraints regarding the ordering of packets that they can exploit with Slingshot *unordered* mode [76], [78], [79].

The throughput results are presented in Figure 15. Notably, the ordered mode achieves throughput remarkably close to the unordered mode, with only a slight delay in reaching full utilization and a modest increase in completion time. Furthermore, both adaptive routing modes, ordered and unordered, outperform traditional non-adaptive systems such as Aries [80] by a significant margin based on internal data.

IV. Discussion

A. Packet Loss/Corruption

Flowcut switching is designed to work with RoCE, that is usually deployed on lossless networks. If this is not the case, or if a packet gets discarded by the switch for any other reason (e.g., due to a CRC check failure), Flowcut switching might not be able to resume paused flows. For example, this might happen if an ACK is lost after that the ingress switch paused

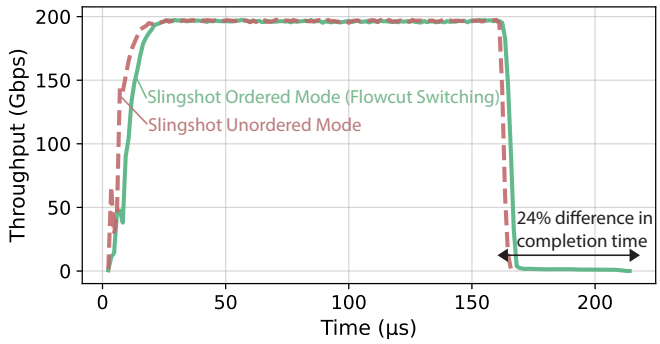

Fig. 15. Throughput of two Slingshot modes when running an all-to-all on a real Dragonfly system with 2,048 endpoints.

a flow. In that case, the number of in-flight bytes will never become zero, and the flow will never be resumed. To avoid that, the source host keeps a timeout for each flow. If the flow is not resumed before the timeout expires, the source resumes the transmission. In that case, all subsequent packets will be routed by the switches on the old path.

B. Interaction with Congestion Control

Adaptive routing algorithms are orthogonal to congestion and flow control algorithms used by the hosts. In principle, the adaptive routing algorithm avoids most of the intermediate congestion [14], [63], [81], preventing the congestion control algorithm to kick-in, by keeping the network bandwidth as high as possible. However, if congestion cannot be avoided through adaptive routing (as in the case of endpoint congestion generated by incast-like traffic [65], [81]), the transmission rate of the flow at the source host will eventually decrease and match the bandwidth of the network path. For these reasons, we focused on non-incast scenarios in our evaluation since it is mostly a congestion control issue. Having said that, some of our evaluated scenarios (Figure 14) do have some small incasts happening during their executions.

C. Draining Time

As discussed in Section II-B, draining a flow introduces some overhead, because transmission of packets for that flow is stopped until the flow has no more in-flight packets. In general, this overhead is compensated because subsequent packets are sent on a less congested path. However, this depends on how many packets remain to be transmitted for that flow. To mitigate this issue, the source NIC could add an indication of the number of packets left for the flow, and the switches could drain the flow only if there are enough packets left, so that the cost of the draining will be compensated by the forwarding of the subsequent packets on a less congested path. Finally, we note that with certain protocols, even a single OOO packet might trigger re-transmissions which would always be more expensive (pending failure and edge cases) than draining the flow.

Alternatively, the switch could resume the transmission of the flow while there are still a few in-flight packets, to overlap the delivery of those packets to the destination host with the delivery of the resume packet to the source host. Although this would break the in-order delivery guarantee, it still guarantees a maximum *Out-of-Order Degree* (OOD). The OOD is defined as the maximal difference between the sequence number of a received out-of-order packet, and the expected packet sequence number [26], and some protocols can tolerate an OOD greater than zero. For example, in QUIC [22] a retransmission is not triggered if the OOD is ≤ 3, while other protocols tolerate an OOD ≤ 64 [26]. By allowing resuming the flow while there are still some in-flight packets we could reduce the draining time while providing guarantees on the maximum OOD.

Regardless, even with our standard Flowcut Switching approach, we note that in our simulations the impact of draining is relatively small and greatly justifies the added performance of making the flow use multiple paths. We note this in Table IV, where we show the impact of draining for the fat tree experiments presented in Section III-C.

Experiment	Draining Impact (avg. % of the runtime spent draining)
Permutation	11.3%
Permutation with failures	10.5%
Web Search	5.2%
All-To-All	6.3%

TABLE IV
SHOWING THE IMPACT OF DRAINING ON THE TOTAL RUNTIME. THE NUMBER IS THE AVERAGE CONSIDERING ALL FLOWS IN THE NETWORK.

V. Related Work

Several works address issues related to adaptive routing and in-order packet delivery. Some of these works balance traffic on a per flow granularity [6], [9], [15], [16], [82]–[85]. Although they guarantee in-order packet delivery, it has been shown that applications can still experience congestion due to the coarse granularity of the adaptive routing decisions [7], [11], [12]. On the other hand, per packet adaptive routing algorithms react better to congestion [11], [12], [86]–[88] but are characterized by higher number of OOO packets, posing significant challenges on the hosts for reordering packets. MPRDMA [76] tries to balance this by pruning slow paths in order to reduce the amount of out-of-order packets. However, it cannot guarantee in-order delivery for all packets and requires ACK clocking to work.

Several adaptive routing algorithms rely on Flowlet switching to reduce the number of OOO packets [7], [17], [31], [89]. However, they rely on the assumption that the traffic is bursty, that is usually not the case for RDMA-like protocols [26]. Moreover, they can only guarantee in-order delivery if the threshold used to detect flowlets is large enough. This depends on the network conditions, and having a too high threshold limits the adaptive routing opportunities and impacts the performance. For this reason, some algorithms dynamically tune this threshold according to the network traffic to balance performance and OOO packets [31], [42], [90], but they cannot still provide in-order delivery guarantees.

ConWeave [91] is a recently introduced solution that guarantees in-order delivery by temporarily buffering packets at

the last switch before the receiver. Although it achieves impressive throughput and fairness, it has a few practical drawbacks compared to Flowcut: it requires substantial switch-side changes (while Flowcut can exist entirely on the NIC) and per-flow buffer queues (or traffic classes) at each ToR to hold rerouted packets; it depends on several microsecond-scale tunable timers (RTT-probe timeout, path-busy blacklist, tail-flush recovery); and its design and evaluation are tied to fat-tree topologies.

Other works implement the monitoring of congestion and adaptive routing decisions in the source host software stack [63], [92]. For example, Flowbender [63] monitors the congestion experienced by each flow by using *Explicit Congestion Notification* (ECN) and, if a flow is too congested, it is rerouted on a different path. However, differently from Flowcut switching, paths are selected randomly, and there is no guarantee that the newly selected path is less congested than the old one. Also, these approaches only take decisions at the endpoint, whereas Flowcut switching can reroute packets at any point of the network, thus reacting more promptly to congestion.

Last, LEFT [27] uses an endpoint based reorder buffer in a RNIC to absorb any out of order arrivals and deliver a perfectly in order stream to the application. This requires custom NIC support and allocates per flow buffers on the host.

VI. Conclusions

In this paper we presented Flowcut switching, a simple and easy-to-implement mechanism to achieve good routing performance while guaranteeing in-order delivery of all packets. We show that Flowcut switching, unlike other state-of-the-art solutions like Flowlet switching, requires minimal tuning and is more tolerant to parameter selection, independently of the workload, network configuration, congestion, and failures. Most importantly, Flowcut switching can guarantee in-order delivery of packets. This is important for several protocols like RoCE, where packets are not handled efficiently if out-of-order. We remark that even a few out-of-order packets can cause problems to such protocols, if there is a large distance between their sequence numbers. We propose three possible implementations for Flowcut switching, two requiring switch support (on all switches or only ingress switches) while another requiring only NIC changes. In our testing using fat tree and dragonfly topologies, Flowcut switching consistently outperforms ECMP, by achieving 1.5x lower FCTs under normal conditions and by 5x when simulating link failures. Moreover, Flowcut switching also consistently outperforms a conservative version of Flowlet switching in all scenarios, as well as packet spraying in link failures scenarios.

VII. Acknowledgments

This work is supported by the European Union Commission's Horizon Europe, under grant agreements 101175702 (NET4EXA), by the Sapienza University Grants ADAGIO and D2QNeT (Bando per la ricerca di Ateneo 2023 and 2024), and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement PSAP, No. 101002047). We also thank the Swiss National Supercomputing Center (CSCS) for providing the computational resources used in this work. The authors used ChatGPT and Claude solely for language editing and quality control; all ideas and content are original.

References

[1] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 79–92. [Online]. Available: https://doi.org/10.1145/3445814.3446713

[2] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy, "Gimbal: Enabling multi-tenant storage disaggregation on smartnic jbofs," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 106–122. [Online]. Available: https://doi.org/10.1145/3452296.3472940

[3] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 249–264. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gao

[4] M. Besta, J. Domke, M. Schneider, M. Konieczny, S. D. Girolamo, T. Schneider, A. Singla, and T. Hoefler, "High-performance routing with multipathing and path diversity in ethernet and hpc networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 943–959, 2021.

[5] M. Besta, M. Schneider, M. Konieczny, K. Cynk, E. Henriksson, S. Di Girolamo, A. Singla, and T. Hoefler, "Fatpaths: Routing in supercomputers and data centers when shortest paths fall short," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[6] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 51–62. [Online]. Available: https://doi.org/10.1145/1592568.1592576

[7] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 503–514. [Online]. Available: https://doi.org/10.1145/2619239.2626316

[8] J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu, "Load balancing in data center networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2324–2352, 2018.

[9] D. Thaler and C. Hopps, "Rfc2991: Multipath issues in unicast and multicast next-hop selection," USA, 2000.

[10] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 183–197, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787508

[11] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 49–60. [Online]. Available: https://doi.org/10.1145/2535372.2535375

[12] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 225–238. [Online]. Available: https://doi.org/10.1145/3098822.3098839

[13] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang, S. Zhang, M. J. Fernandez, S. Gandham, and H. Zeng, "Rdma over ethernet for distributed training at meta scale," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 57–70. [Online]. Available: https://doi.org/10.1145/3651890.3672233

[14] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[15] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. USA: USENIX Association, 2010, p. 19.

[16] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: Scalability and commoditization," in *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 57–62. [Online]. Available: https://doi.org/10.1145/1397718.1397732

[17] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 407–420. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini

[18] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 465–478. [Online]. Available: https://doi.org/10.1145/2785956.2787507

[19] F. P. Tso and D. P. Pezaros, "Improving data center network utilization using near-optimal traffic engineering," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1139–1148, 2013.

[20] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 2130–2138.

[21] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making quic quicker with nic offload," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, ser. EPIQ '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–27. [Online]. Available: https://doi.org/10.1145/3405796.3405827

[22] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at quic: An approach for rigorous evaluation of rapidly evolving transport protocols," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 290–303. [Online]. Available: https://doi.org/10.1145/3131365.3131368

[23] H. Ghasemirahni, T. Barbette, G. P. Katsikas, A. Farshin, A. Roozbeh, M. Girondi, M. Chiesa, G. Q. Maguire, and D. Kostić, "Packet order matters! improving application performance by deliberately delaying packets," in *Symposium on Networked Systems Design and Implementation*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:268076420

[24] Y. Lu, G. Chen, Z. Ruan, W. Xiao, B. Li, J. Zhang, Y. Xiong, P. Cheng, and E. Chen, "Memory efficient loss recovery for hardware-based transport in datacenter," in *Proceedings of the First Asia-Pacific Workshop on Networking*, ser. APNet'17. New York, NY, USA: Association for Computing Machinery, 2017, p. 22–28. [Online]. Available: https://doi.org/10.1145/3106989.3106993

[25] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 313–326. [Online]. Available: https://doi.org/10.1145/3230543.3230557

[26] G. Chen, Y. Lu, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, and T. Moscibroda, "Mp-rdma: Enabling rdma with multi-path transport in datacenters," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2308–2323, 2019.

[27] P. Huang, X. Zhang, Z. Chen, C. Liu, and G. Chen, "Left: Lightweight and fast packet reordering for rdma," in *Proceedings of the 8th Asia-Pacific Workshop on Networking*, ser. APNet '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 67–73. [Online]. Available: https://doi.org/10.1145/3663408.3663418

[28] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh, "Juggler: a practical reordering resilient network stack for datacenters," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2901318.2901334

[29] Data Plane Development Kit (DPDK), "Generic Receive Offload Library," https://doc.dpdk.org/guides/prog_guide/generic_receive_offload_lib.html, 2022.

[30] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[31] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, p. 51–62, mar 2007. [Online]. Available: https://doi.org/10.1145/1232919.1232925

[32] Infiniband Trade Association, "RoCEv2 Specification," https://cw.infinibandta.org/document/dl/7781, 2021.

[33] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable hpc," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, 2020.

[34] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 202–215. [Online]. Available: https://doi.org/10.1145/2934872.2934908

[35] D. Gibson, H. Hariharan, E. Lance, M. McLaren, B. Montazeri, A. Singh, S. Wang, H. M. G. Wassel, Z. Wu, S. Yoo, R. Balasubramanian, P. Chandra, M. Cutforth, P. Cuy, D. Decotigny, R. Gautam, A. Iriza, M. M. K. Martin, R. Roy, Z. Shen, M. Tan, Y. Tang, M. Wong-Chan, J. Zbiciak, and A. Vahdat, "Aquila: A unified, low-latency fabric for datacenter networks," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1249–1266. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/gibson

[36] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 537–550. [Online]. Available: https://doi.org/10.1145/2785956.2787510

[37] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 523–536. [Online]. Available: https://doi.org/10.1145/2785956.2787484

[38] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. K. Martin, M. McLaren, P. Chandra, R. Cauble, H. M. G. Wassel, B. Montazeri, S. L. Sabato, J. Scherpelz, and A. Vahdat, "1rma: Re-envisioning remote memory access for multi-tenant datacenters," 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 708–721. [Online]. Available: https://doi.org/10.1145/3387514.3405897

[39] Y. Le, B. Stephens, A. Singhvi, A. Akella, and M. M. Swift, "Rogue: Rdma over generic unconverged ethernet," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 225–236. [Online]. Available: https://doi.org/10.1145/3267809.3267826

[40] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "Freeflow: Software-based virtual rdma networking for containerized clouds," in *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'19. USA: USENIX Association, 2019, p. 113–125.

[41] Y. Lu, G. Chen, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, E. Chen, and T. Moscibroda, "Multi-Path transport for RDMA in datacenters," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX

Association, Apr. 2018, pp. 357–371. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/lu

[42] C. H. Benet and A. J. Kassler, "Flowdyn: Towards a dynamic flowlet gap detection using programmable data planes," 2019.

[43] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[44] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 202–208. [Online]. Available: https://doi.org/10.1145/1644893.1644918

[45] S. M. Irteza, H. M. Bashir, T. Anwar, I. A. Qazi, and F. R. Dogar, "Load balancing over symmetric virtual topologies," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[46] NVIDIA Corporation, *NVIDIA ConnectX-5 Ethernet Adapter Cards User Manual*, 2020, accessed: 2025-05-07. [Online]. Available: https://docs.nvidia.com/nvidia-connectx-5-ethernet-adapter-cards-user-manual.pdf

[47] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*, 2008, pp. 77–88.

[48] A. Singh and S. U. D. of Electrical Engineering, *Load-balanced Routing in Interconnection Networks*. Stanford University, 2005. [Online]. Available: https://books.google.it/books?id=RG_BPAAACAAJ

[49] J. P. Beecroft, T. L. Court, A. M. Bataineh, and D. C. Hewson, "System and method for facilitating data-driven intelligent network with per-flow credit-based flow control," Patent US20 220 217 079A1, Jul 7, 2022, filed: 2020-03-23. [Online]. Available: https://patents.google.com/patent/US20220217079A1/en

[50] IEEE 802.1 Working Group. (2021) 802.1Qbb – Priority-based Flow Control. https://1.ieee802.org/dcb/802-1qbb/. [Accessed 24-Dec.-2021].

[51] Y. Gellis, "Calculating and synchronizing time with the precision timing protocol on the nvidia spectrum switch," https://developer.nvidia.com/blog/...-spectrum-switch/, Sep. 2022, accessed: 2025-05-07.

[52] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat, "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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 514–528. [Online]. Available: https://doi.org/10.1145/3387514.3406591

[53] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *2013 Optical Interconnects Conference*, 2013, pp. 49–50.

[54] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 537–550, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787510

[55] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," 2021.

[56] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, p. 87094–87155, 2021. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2021.3086704

[57] S. Di Girolamo, A. Kurth, A. Calotoiu, T. Benz, T. Schneider, J. Beránek, L. Benini, and T. Hoefler, "A risc-v in-network accelerator for flexible high-performance low-power packet processing," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 958–971.

[58] NVIDIA, "NVIDIA Bluefield-3 DPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf, 2021.

[59] Pensando, "Pensando DSC-100 Distributed Services Card," https://pensando.io/wp-content/uploads/2020/03/Pensando-DSC-100-Product-Brief.pdf, 2021.

[60] Marvell, "Marvell LiquidIO III," https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf, 2021.

[61] Broadcom, "Stingray PS250 – 2×50-Gb High-Performance Data Center SmartNIC," https://docs.broadcom.com/doc/PS250-PB, 2021.

[62] Fungible, "Fungible F1 Data Processing Unit," https://pages.fungible.com/rs/038-PGB-059/images/PB0028.02.12020914-Fungible-F1-Data-Processing-Unit.pdf, 2021.

[63] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 149–160. [Online]. Available: https://doi.org/10.1145/2674005.2674985

[64] T. Bonato, A. Kabbani, A. Ghalayini, M. Papamichael, M. Dohadwala, L. Gianinazzi, M. Khalilov, E. Achermann, D. D. Sensi, and T. Hoefler, "Reps: Recycled entropy packet spraying for adaptive load balancing and failure mitigation," 2025. [Online]. Available: https://arxiv.org/abs/2407.21625

[65] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," 2019. [Online]. Available: https://arxiv.org/abs/1909.09923

[66] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," 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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 90–106. [Online]. Available: https://doi.org/10.1145/3387514.3405855

[67] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–28. [Online]. Available: https://doi.org/10.1145/3098822.3098824

[68] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, p. 37–42, Mar. 2011. [Online]. Available: https://doi.org/10.1145/1964218.1964225

[69] S. N. Laboratories, "The Structural Simulation Toolkit," http://sst-simulator.org/.

[70] Swiss National Supercomputing Center, "Alps System," https://www.cscs.ch/computers/alps/.

[71] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731508001767

[72] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 676–691. [Online]. Available: https://doi.org/10.1145/3452296.3472904

[73] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "{GS}hard: Scaling giant models with conditional computation and automatic sharding," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=qrwe7XHTmYb

[74] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, ser. ICPP '08. USA: IEEE Computer Society, 2008, p. 320–329. [Online]. Available: https://doi.org/10.1109/ICPP.2008.83

[75] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 2130–2138.

[76] Y. Lu, G. Chen, B. Li, K. Tan, Y. Xiong, P. Cheng, J. Zhang, E. Chen, and T. Moscibroda, "Multi-path transport for rdma in datacenters," in *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'18. USA: USENIX Association, 2018, p. 357–371.

[77] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, vol. 11, no. 2, pp. 350–361, 1982. [Online]. Available: https://doi.org/10.1137/0211027

[78] NVIDIA, "Nvidia spectrum-x network platform architecture," 2024, https://resources.nvidia.com/en-us-accelerated-networking-resource-library/nvidia-spectrum-x.

[79] U. E. Consortium, "Ultra Ethernet Specification Update - Ultra Ethernet Consortium — ultraethernet.org," https://ultraethernet.org/ultra-ethernet-specification-update/, [Accessed 16-09-2024].

[80] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc ® series network," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:45810707

[81] S. Chunduri, T. Groves, P. Mendygral, B. Austin, J. Balma, K. Kandalla, K. Kumaran, G. Lockwood, S. Parker, S. Warren, N. Wichmann, and N. Wright, "Gpcnet: Designing a benchmark suite for inducing and measuring contention in hpc networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356215

[82] C. Hopps, "Rfc2992: Analysis of an equal-cost multi-path algorithm," USA, 2000.

[83] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "Wcmp: Weighted cost multipathing for improved fairness in data centers," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2592798.2592803

[84] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 151–162. [Online]. Available: https://doi.org/10.1145/2535372.2535397

[85] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2079296.2079304

[86] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 307–318. [Online]. Available: https://doi.org/10.1145/2619239.2626309

[87] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: Reducing the flow completion time tail in datacenter networks," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 139–150. [Online]. Available: https://doi.org/10.1145/2342356.2342390

[88] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, p. 266–277, aug 2011. [Online]. Available: https://doi.org/10.1145/2043164.2018467

[89] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2890955.2890968

[90] X. Diao, H. Gu, X. Yu, L. Qin, and C. Luo, "Flex: A flowlet-level load balancing based on load-adaptive timeout in dcn," *Future Generation Computer Systems*, vol. 130, pp. 219–230, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X21005021

[91] C. H. Song, X. Z. Khooi, R. Joshi, I. Choi, J. Li, and M. C. Chan, "Network load balancing with in-network reordering support for rdma," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ser. ACM SIGCOMM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 816–831. [Online]. Available: https://doi.org/10.1145/3603269.3604849

[92] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "SPAIN: COTS Data-Center ethernet for multipathing over arbitrary topologies," in *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 10)*. San Jose, CA: USENIX Association, Apr. 2010. [Online]. Available: https://www.usenix.org/conference/nsdi10-0/spain-cots-data-center-ethernet-multipathing-over-arbitrary-topologies