Revisiting Network Support for RDMA

Radhika Mittal[†], Alex Shpiner[♢], Aurojit Panda[†], Eitan Zahavi[♢],
Arvind Krishnamurthy[♠], Sylvia Ratnasamy[†], Scott Shenker[†‡]
[†]*UC Berkeley*, [‡]*ICSI*, [♢]*Mellanox*, [♠]*Univ. of Washington*

Abstract

The advent of RoCE (RDMA over Converged Ethernet) has led to a significant increase in the use of RDMA in datacenter networks. To achieve good performance, RoCE requires a lossless network which is in turn achieved by enabling Priority Flow Control (PFC) within the network. However, PFC brings with it a host of problems such as head-of-the-line blocking, congestion spreading, and occasional deadlocks. Rather than seek to fix these issues with PFC, we instead ask: *is PFC fundamentally required to support RDMA over Ethernet?*

We show that the need for PFC is an artifact of current RoCE NIC designs rather than a fundamental requirement. We propose an *improved RoCE NIC* (IRN) design that makes a few simple changes to the RoCE NIC for better handling of packet losses. We show that IRN (without PFC) outperforms RoCE (with PFC) by 6-83% for typical network scenarios. Thus not only does IRN eliminate the need for PFC, it *improves* performance in the process! We further show that the changes that IRN introduces can be implemented with modest overheads to NIC resources: 3-10% increase in memory requirements and $<$4% in logic. Based on our results, we argue that research and industry should rethink the current trajectory of network support for RDMA.

1 Introduction

Datacenter networks offer higher bandwidth and lower latency than traditional wide-area networks. However, traditional endhost networking stacks, with their high latencies and substantial CPU overhead, have limited the extent to which applications can make use of these characteristics. As a result, several large datacenters have recently adopted RDMA, which bypasses the traditional networking stacks in favor of direct memory accesses.

RDMA over Converged Ethernet (RoCE) has emerged as the canonical method for deploying RDMA in Ethernet-based datacenters [24, 38]. The centerpiece of RoCE is a NIC that (i) provides mechanisms for accessing host memory without CPU involvement and (ii) supports very basic network transport functionality. Early experience revealed that RoCE NICs only achieve good end-to-end performance when run over a lossless network, so operators turned to Ethernet's Priority Flow Control (PFC) mechanism to achieve minimal packet loss. The combination of RoCE and PFC has enabled a wave of datacenter RDMA deployments.

However, the current solution is not without problems. In particular, PFC adds management complexity and can lead to significant performance problems such as head-of-the-line blocking, congestion spreading, and occasional deadlocks [24, 25, 34, 37, 38]. Rather than continue down the current path and address the various problems with PFC, in this paper we take a step back and ask whether it was needed in the first place. To be clear, current RoCE NICs require a lossless fabric for good performance. However, the question we raise is: *can the RoCE NIC design be altered so that we no longer need a lossless network fabric?*

We answer this question in the affirmative, proposing a new design called IRN (for Improved RoCE NIC) that makes two incremental changes to current RoCE NICs (i) more efficient loss recovery, and (ii) basic end-to-end flow control to bound the number of in-flight packets (§3). We show, via extensive simulations on a RoCE simulator obtained from a commercial NIC vendor, that IRN performs better than current RoCE NICs, and that IRN does not require PFC to achieve high performance; in fact, IRN often performs better without PFC (§4). We detail the extensions to the RDMA protocol that IRN requires (§5) and use comparative analysis and FPGA synthesis to evaluate the overhead that IRN introduces in terms of NIC hardware resources (§6). Our results suggest that adding IRN functionality to current RoCE NICs would add as little as 3-10% overhead in resource consumption, with no deterioration in message rates.

A natural question that arises is how IRN compares to iWARP? iWARP [33] long ago proposed a similar philosophy as IRN: handling packet losses efficiently in the NIC rather than making the network lossless. What we show is that iWARP's failing was in its design choices. As we elaborate on in §2, the differences between iWARP and IRN stem from their starting point: iWARP aimed for full generality which led them to put the full TCP/IP stack on the NIC, requiring multiple layers of translation between RDMA abstractions and traditional TCP bytestream abstractions. As a result, iWARP NICs are typically far more complex than RoCE ones, with higher cost and lower performance (§2). In contrast, IRN starts with the much simpler design of RoCE and asks what minimal features can be added to eliminate the need for PFC.

More generally: while the merits of iWARP vs. RoCE has been a long-running debate in industry, there is no conclusive or rigorous evaluation that compares the two architectures.

Instead, RoCE has emerged as the de-facto winner in the marketplace, and brought with it the implicit (and still lingering) assumption that a lossless fabric is necessary to achieve RoCE's high performance. Our results are the first to rigorously show that, counter to what market adoption might suggest, iWARP in fact had the right architectural philosophy, although a needlessly complex design approach.

Hence, one might view IRN and our results in one of two ways: (i) a new design for RoCE NICs which, at the cost of a few incremental modifications, eliminates the need for PFC and leads to better performance, or, (ii) a new incarnation of the iWARP philosophy which is simpler in implementation and faster in performance.

2 Background

We begin with reviewing some relevant background.

2.1 Infiniband RDMA and RoCE

RDMA has long been used by the HPC community in special-purpose Infiniband clusters that use credit-based flow control to make the network lossless [5]. Because packet drops are rare in such clusters, the RDMA Infiniband transport (as implemented on the NIC) was not designed to efficiently recover from packet losses. When the receiver receives an out-of-order packet, it simply discards it and sends a negative acknowledgement (NACK) to the sender. When the sender sees a NACK, it retransmits all packets that were sent after the last acknowledged packet (i.e., it performs a go-back-N retransmission).

To take advantage of the widespread use of Ethernet in datacenters, RoCE [6, 10] was introduced to enable the use of RDMA over Ethernet.[1] RoCE adopted the same Infiniband transport design (including go-back-N loss recovery), and the network was made lossless using PFC.

2.2 Priority Flow Control

Priority Flow Control (PFC) [7] is Ethernet's flow control mechanism, in which a switch sends a pause (or X-OFF) frame to the upstream entity (a switch or a NIC), when the queue exceeds a certain configured threshold. When the queue drains below this threshold, an X-ON frame is sent to resume transmission. When configured correctly, PFC makes the network lossless (as long as all network elements remain functioning). However, this coarse reaction to congestion is agnostic to *which* flows are causing it and this results in various performance issues that have been documented in numerous papers in recent years [24, 25, 34, 37, 38]. These issues range from mild (*e.g.,* unfairness and head-of-line blocking) to severe, such as "pause spreading" as highlighted in [24] and even network deadlocks [25, 34, 37]. In an attempt to mitigate these issues, congestion control mechanisms have been proposed for RoCE (*e.g.,* DCQCN [38] and Timely [30]) in order to reduce the sending rate on detecting congestion. However, they were not enough to eradicate the need for PFC. Hence, there is now broad agreement that PFC makes networks harder to understand and manage, and can lead to myriad performance problems that need to be dealt with.

[1] We use the term RoCE for both RoCE [6] and its successor RoCEv2 [10] that enables running RDMA, not just over Ethernet, but also over IP-routed networks.

NIC	Throughput	Latency
Chelsio T-580-CR (iWARP)	3.24 Mpps	2.89 μs
Mellanox MCX416A-BCAT (RoCE)	14.7 Mpps	0.94 μs

Table 1: An iWARP and a RoCE NIC's raw performance for 64 bytes RDMA Writes on a single queue-pair.

2.3 iWARP vs RoCE

iWARP [33] was designed to support RDMA over a fully general (i.e., not loss-free) network. iWARP implements the entire TCP stack in hardware along with multiple other layers that it needs to translate TCP's byte stream semantics to RDMA segments. Early in our work, we engaged with multiple NIC vendors and datacenter operators in an attempt to understand why iWARP was not more broadly adopted (since we believed the basic architectural premise underlying iWARP was correct). The consistent response we heard was that iWARP is significantly more complex and expensive than RoCE, with inferior performance [14].

We also looked for empirical datapoints to validate or refute these claims. We ran RDMA Write benchmarks on two machines connected to one another, using Chelsio T-580-CR 40Gbps iWARP NICs on both machines for one set of experiments, and Mellanox MCX416A-BCAT 56Gbps RoCE NICs (with link speed set to 40Gbps) for another. Both NICs had similar specifications, and at the time of purchase, the iWARP NIC cost $760, while the RoCE NIC cost $420. Raw NIC performance values for 64 bytes batched Writes on a single queue-pair are reported in Table 1. We find that iWARP has $3\times$ higher latency and $4\times$ lower throughput than RoCE.

These price and performance differences could be attributed to many factors other than transport design complexity (such as differences in profit margins, supported features and engineering effort) and hence should be viewed as anecdotal evidence as best. Nonetheless, they show that our conjecture (in favor of implementing loss recovery at the endhost NIC) was certainly not obvious based on current iWARP NICs.

Our primary contribution is to show that iWARP, somewhat surprisingly, did in fact have the right philosophy: explicitly handling packet losses in the NIC leads to better performance than having a lossless network. However, efficiently handling packet loss does not require implementing the entire TCP stack in hardware as iWARP did. Instead, we identify the incremental changes to be made to current RoCE NICs, leading

to a design which (i) does not require PFC yet achieves better network-wide performance than both RoCE and iWARP (§4), and (ii) is much closer to RoCE's implementation with respect to both NIC performance and complexity (§6) and is thus significantly less complex than iWARP.

3 IRN Design

We begin with describing the transport logic for IRN. For simplicity, we present it as a general design independent of the specific RDMA operation types. We go into the details of handling specific RDMA operations with IRN later in §5.

IRN makes two key changes to current RoCE NICs, as described in the following subsections: (1) improving the loss recovery mechanism, and (2) basic end-to-end flow control (termed BDP-FC) which bounds the number of in-flight packets by the bandwidth-delay product of the network. These changes are orthogonal to the use of explicit congestion control mechanisms (such as DCQCN [38] and Timely [30]) that, as with current RoCE NICs, can also be enabled with IRN.

3.1 IRN's Loss Recovery Mechanism

As discussed in §2, current RoCE NICs use a go-back-N loss recovery scheme. In the absence of PFC, redundant retransmissions caused by go-back-N loss recovery result in significant performance penalties (as evaluated in §4). Therefore, the first change we make with IRN is a more efficient loss recovery, based on selective retransmission (inspired by TCP's loss recovery), where the receiver does not discard out of order packets and the sender selectively retransmits the lost packets, as detailed below.

Upon every out-of-order packet arrival, an IRN receiver sends a NACK, which carries both the cumulative acknowledgment (indicating its expected sequence number) and the sequence number of the packet that triggered the NACK (as a simplified form of selective acknowledgement or SACK).

An IRN sender enters loss recovery mode when a NACK is received or when a timeout occurs. It also maintains a bitmap to track which packets have been cumulatively and selectively acknowledged. When in the loss recovery mode, the sender selectively retransmits lost packets as indicated by the bitmap, instead of sending new packets. The first packet that is retransmitted on entering loss recovery corresponds to the cumulative acknowledgement value. Any subsequent packet is considered lost only if another packet with a higher sequence number has been selectively acked. When there are no more lost packets to be retransmitted, the sender continues to transmit new packets (if allowed by BDP-FC). It exits loss recovery when a cumulative acknowledgement greater than the *recovery sequence* is received, where the recovery sequence corresponds to the last regular packet that was sent before the retransmission of a lost packet.

SACKs allow efficient loss recovery only when there are multiple packets in flight. For other cases (example, for single packet messages), loss recovery gets triggered via timeouts. A high RTO timer value can increase the tail latency of these short single-packet messages. However, keeping the RTO timer value too small, can result in too many spurious retransmissions, affecting the overall results. An IRN sender, therefore, uses a low timer value of RTO_{low} only when there are a small N number of packets in flight, and a higher value of RTO_{high} otherwise. We discuss how the timeout feature in current RoCE NICs can be easily extended to support this in §6. We also experimented with dynamically computed timeout values (as with TCP), which not only complicated the design, but did not help with this case, since these effects were then be dominated by the initial timeout value.

3.2 IRN's BDP-FC Mechanism

The second change we make with IRN is introducing the notion of a basic end-to-end packet level flow control, called BDP-FC, which bounds the number of outstanding packets in flight for a flow by the bandwidth-delay product (BDP) of the network, as suggested in [17]. This is a static cap that we compute by dividing the BDP (in bytes) with the packet MTU set by the RDMA queue-pair (typically 1KB in RoCE NICs). An IRN sender transmits a new packet only if the number of packets in flight (computed as the difference between current packet's sequence number and last acknowledged sequence number) is less than this BDP cap.

BDP-FC improves the performance by reducing unnecessary queuing in the network. Furthermore, by strictly upper bounding the number of out-of-order packet arrivals, it greatly reduces the amount of state required for tracking packet losses in the NICs (discussed in more details later in §6).

As mentioned before, IRN's loss recovery has been inspired by TCP's loss recovery. However, rather than incorporating the entire TCP stack as is done by iWARP NICs, IRN: (1) decouples loss recovery from congestion control and does not incorporate any notion of TCP congestion window control involving slow start, AIMD or advanced fast recovery, (2) operates directly on RDMA segments instead of using TCP's byte stream abstraction, which not only avoids the complexity introduced by multiple translation layers (as needed in iWARP), but also allows IRN to simplify its selective acknowledgement and loss tracking schemes. We discuss how these changes effect performance towards the end of §4.

4 Evaluating IRN's Transport Logic

We now confront the central question of this paper: *Does RDMA require a lossless network?* If the answer is yes, then we must address the many difficulties of PFC. If the answer is no, then we can greatly simplify network management

by letting go of PFC. To answer this question, we evaluate the network-wide performance of IRN's transport logic via extensive simulations. Our results show that IRN performs better than RoCE, without requiring PFC. We test this across a wide variety of experimental scenarios and across different performance metrics. We end this section with a comparison of IRN and iWARP's network-wide performance.

4.1 Experimental Settings

We begin with describing our experimental settings.

Simulator: Our simulator, obtained from a commercial NIC vendor, extends INET/OMNET++ [1, 2] to model the Mellanox ConnectX4 RoCE NIC [11]. RDMA queue-pairs (QPs) are modelled as UDP applications with either RoCE or IRN transport layer logic, that generate flows (as described later). We define a flow as a unit of data transfer comprising of one or more messages between the same source-destination pair as in [30, 38]. When the sender QP is ready to transmit data packets, it periodically polls the MAC layer until the link is available for transmission. The simulator implements DCQCN as implemented in the Mellanox ConnectX-4 ROCE NIC [35], and we add support for a NIC-based Timely implementation. All switches in our simulation are input-queued with virtual output ports, that are scheduled using round-robin. The switches can be configured to generate PFC frames by setting appropriate buffer thresholds.

Default Case Scenario: For our default case, we simulate a 54-server three-tiered fat-tree topology, connected by a fabric with full bisection-bandwidth constructed from 45 6-port switches organized into 6 pods [16]. We consider 40Gbps links, each with a propagation delay of 2μs, resulting in a bandwidth-delay product (BDP) of 120KB along the longest (6-hop) path. This corresponds to $\sim$110 MTU-sized packets (assuming typical RDMA MTU of 1KB).

Each end host generates new flows with Poisson inter-arrival times [17, 31]. Each flow's destination is picked randomly and size is drawn from a realistic heavy-tailed distribution derived from [19]. Most flows are small (50% of the flows are single packet messages with sizes ranging between 32 bytes-1KB representing small RPCs such as those generated by RDMA based key-value stores [22, 26]), and most of the bytes are in large flows (15% of the flows are between 200KB-3MB, representing background RDMA traffic such as storage). The network load is set at 70% utilization for our default case. We use ECMP for load-balancing [24]. We vary different aspects from our default scenario (including topology size, workload pattern and link utilization) in §4.4.

Parameters: RTO_{high} is set to an estimation of the maximum round trip time with one congested link. We compute this as the sum of the propagation delay on the longest path and the maximum queuing delay a packet would see if the switch

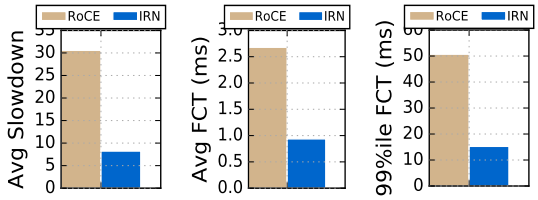


Figure 1: Comparing IRN and RoCE's performance.

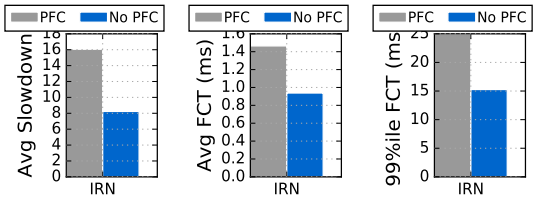


Figure 2: Impact of enabling PFC with IRN.

buffer on a congested link is completely full. This is approximately 320μs for our default case. For IRN, we set RTO_{low} to 100μs (representing the desirable upper-bound on tail latency for short messages) with N set to a small value of 3. When using RoCE without PFC, we use a fixed timeout value of RTO_{high}. We disable timeouts when PFC is enabled to prevent spurious retransmissions. We use buffers sized at twice the BDP of the network (which is 240KB in our default case) for each input port [17, 18]. The PFC threshold at the switches is set to the buffer size minus a headroom equal to the upstream link's bandwidth-delay product (needed to absorb all packets in flight along the link). This is 220KB for our default case. We vary these parameters in §4.4 to show that our results are not very sensitive to these specific choices. When using RoCE or IRN with Timely or DCQCN, we use the same congestion control parameters as specified in [30] and [38] respectively. For fair comparison with PFC-based proposals [37, 38], the flow starts at line-rate for all cases.

Metrics: We primarily look at three metrics: (i) average slowdown, where slowdown for a flow is its completion time divided by the time it would have taken to traverse its path at line rate in an empty network, (ii) average flow completion time (FCT), (iii) 99%ile or tail FCT. While the average and tail FCTs are dominated by the performance of throughput-sensitive flows, the average slowdown is dominated by the performance of latency-sensitive short flows.

4.2 Basic Results

We now present our basic results comparing IRN and RoCE for our default scenario. Unless otherwise specified, IRN is always used without PFC, while RoCE is always used with PFC for the results presented here.

4.2.1 IRN performs better than RoCE. We begin with comparing IRN's performance with current RoCE NIC's. The results are shown in Figure 1. IRN's performance is upto

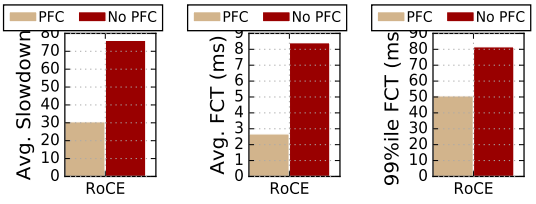


Figure 3: Impact of disabling PFC with RoCE.

2.8-3.7× better than RoCE. This is due to the combination of two factors: (i) IRN's BDP-FC mechanism reduces unnecessary queuing and (ii) unlike RoCE, IRN does not experience any congestion spreading issues, since it does not use PFC. (explained in more details below).

4.2.2 IRN does not require PFC. We next study how IRN's performance is impacted by enabling PFC. If enabling PFC with IRN does not improve performance, we can conclude that IRN's loss recovery is sufficient to eliminate the requirement for PFC. However, if enabling PFC with IRN significantly improves performance, we would have to conclude that PFC continues to be important, even with IRN's loss recovery. Figure 2 shows the results of this comparison. Remarkably, we find that not only is PFC not required, but it significantly degrades IRN's performance (increasing the value of each metric by about 1.5-2×). This is because of the head-of-the-line blocking and congestion spreading issues PFC is notorious for: pauses triggered by congestion at one link, cause queue build up and pauses at other upstream entities, creating a cascading effect. Note that, without PFC, IRN experiences significantly high packet drops (8.5%), which also have a negative impact on performance, since it takes about one round trip time to detect a packet loss and another round trip time to recover from it. However, the negative impact of a packet drop (given efficient loss recovery), is restricted to the flow that faces congestion and does not spread to other flows, as in the case of PFC. While these PFC issues have been observed before [24, 30, 38], we believe our work is the first to show that *a well-design loss-recovery mechanism outweighs a lossless network.*

4.2.3 RoCE requires PFC. Given the above results, the next question one might have is whether RoCE required PFC in the first place? Figure 3 shows the performance of RoCE with and without PFC. We find that the use of PFC helps considerably here. Disabling PFC degrades performance by up to 1.5-3× across the three metrics. This is largely because of the go-back-N loss recovery used by current RoCE NICs, which penalizes performance due to (i) increased congestion caused by redundant retransmissions and (ii) the time and bandwidth wasted by flows in sending these redundant packets.

4.2.4 Effect of Explicit Congestion Control. The previous comparisons did not use any explicit congestion control.

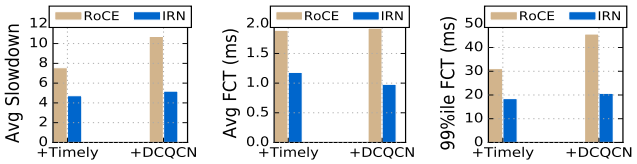


Figure 4: Comparing IRN and RoCE's performance with explicit congestion control (Timely and DCQCN).

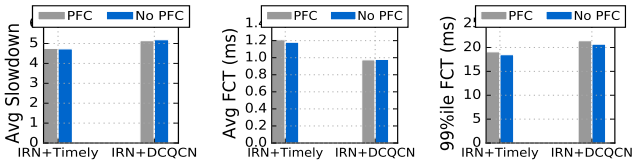


Figure 5: Impact of enabling PFC with IRN, when explicit congestion control (Timely and DCQCN) is used.

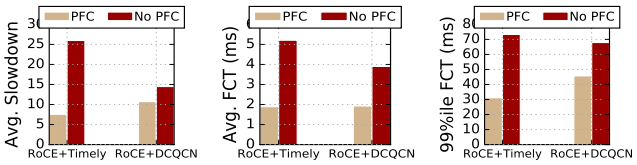


Figure 6: Impact of disabling PFC with RoCE, when explicit congestion control (Timely and DCQCN) is used.

However, as mentioned before, RoCE today is typically deployed in conjunction with some explicit congestion control mechanism such as Timely or DCQCN. We now evaluate whether using such explicit congestion control mechanisms with IRN and RoCE affect the key trends described above.

Figure 4 compares IRN and RoCE's performance when Timely or DCQCN is used. IRN continues to perform better by up to 1.5-2.2× across the three metrics.

Figure 5 evaluates the impact of enabling PFC with IRN, when Timely or DCQCN is used. We find that, IRN's performance is largely unaffected by PFC, since explicit congestion control mitigates both packet drop rate as well as the number of pause frames generated. The largest performance improvement due to enabling PFC was less than 1%, while its largest negative impact was about 3.4%.

Finally, Figure 6 compares RoCE's performance with and without PFC, when Timely or DCQCN is used. We find that, unlike IRN, RoCE (with its inefficient go-back-N loss recovery) requires PFC, even when explicit congestion control is used. Enabling PFC improves RoCE's performance by 1.35× to 3.5× across the three metrics.

Key Takeaways: The following are, therefore, the three takeaways from these results: (1) IRN (without PFC) performs better than RoCE (with PFC), (2) IRN does not require PFC, and (3) RoCE requires PFC.

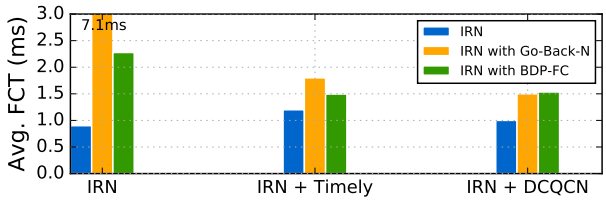


Figure 7: The figure shows the effect of doing go-back-N loss recovery and disabling BDP-FC with IRN. The y-axis is capped at 0.003s to better highlight the trends.

4.3 Factor Analysis of IRN

We now perform a factor analaysis of IRN, to individually study the significance of the two key changes IRN makes to RoCE, namely (1) efficient loss recovery and (2) BDP-FC. For this we compare IRN's performance (as evaluated in §4.2) with two different variations that highlight the significance of each change: (1) enabling go-back-N loss recovery instead of using SACKs, and (2) disabling BDP-FC. Figure 7 shows the resulting average FCTs (we saw similar trends with other metrics). We discuss these results in greater details below.

Need for Efficient Loss Recovery: The first two bars in Figure 7 compare the average FCT of default SACK-based IRN and IRN with go-back-N respectively. We find that the latter results in significantly worse performance. This is because of the bandwidth wasted by go-back-N due to redundant retransmissions, as described before.

Before converging to the current SACK-based mechanism for IRN, we also experimented with other simpler designs. In particular we explored the following two questions:

(1) Can go-back-N be made more efficient? Go-back-N does have the advantage of simplicity over selective retransmission, since it allows the receiver to simply discard out-of-order packets. We, therefore, tried to explore whether we can mitigate the negative effects of go-back-N. We found that explicitly backing off on losses improved go-back-N performance for Timely (though, not for DCQCN). Nonetheless, SACK-based loss recovery continued to perform significantly better across different scenarios (with the difference in average FCT for Timely ranging from 20%-50%).

(2) Do we need SACKs? We also tried a selective retransmit scheme without SACKs (where the sender does not maintain a bitmap to track selective acknowledgements). This performed better than go-back-N. However, it fared poorly when there were multiple losses in a window, requiring multiple round-trips to recover from them. The corresponding degradation in average FCT ranged from $<$1% up to 75% across different scenarios when compared to SACK-based IRN.

Significance of BDP-FC: The first and the third bars in Figure 7 compare the average FCT of IRN with and without BDP-FC respectively. We find that BDP-FC significantly improves performance by reducing unnecessary queuing. Furthermore, it prevents a flow that is recovering from a packet loss from sending additional new packets and increasing congestion, until the loss has been recovered.

Efficient Loss Recovery vs BDP-FC: Comparing the second and third bars in Figure 7 shows that the performance of IRN with go-back-N loss recovery is generally worse than the performance of IRN without BDP-FC. This indicates that of the two changes IRN makes, efficient loss recovery helps performance more than BDP-FC.

4.4 Robustness of Basic Results

We now evaluate the robustness of the basic results from §4.2, across different experimental scenarios, other metrics, and more explicit congestion control schemes.

4.4.1 Varying Experimental Scenario. We evaluated the robustness of our results, as the experimental scenario is varied from our default case. We experimented with (1) link utilization levels varied between 30%-90%, (2) link bandwidths varied from the default of 40Gbps to 10Gbps and 100Gbps, (3) larger fat-tree topologies with 128 and 250 servers, (4) a different workload with flow sizes uniformly distributed between 500KB to 5MB, representing background and storage traffic for RDMA, (5) the per-port buffer size varied between 60KB-480KB, (6) varying other IRN parameters (increasing RTO_{high} value by upto 4 times the default of 320μs, increasing the N value for using RTO_{low} to 10 and 15).

Overall Results: Our results across these different scenarios have been summarized below (details excluded for brevity).

(a) IRN (without PFC) always performed better than RoCE (with PFC), with the performance improvement ranging from 6% to 83% across different cases.

(b) When used without any congestion control, enabling PFC with IRN always degraded performance, with the maximum degradation across different scenarios being as high as 2.4×.

(c) Even when used with Timely and DCQCN, enabling PFC with IRN often degraded performance (with the maximum degradation being 39% for Timely and 20% for DCQCN). Any improvement in performance due enabling PFC with IRN stayed within 1.6% for Timely and 5% for DCQCN.

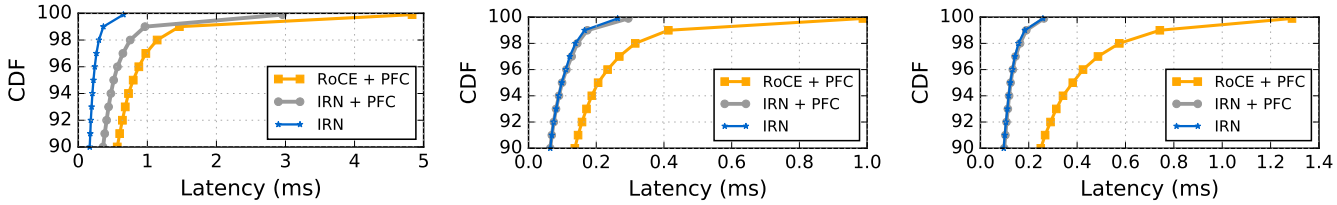
Some observed trends: We observed that the drawbacks of enabling PFC with IRN:

(a) generally increase with increasing link utilization, as the negative impact of congestion spreading with PFC increases.

(b) decrease with increasing bandwidths, as the relative cost of a round trip required to react to packet drops without PFC also increases.

(c) increase with decreasing buffer sizes due to more pauses and greater impact of congestion spreading.

4.4.2 Tail latency for small messages. We now look at the tail latency (or tail FCT) of the single-packet messages from



(a) No CC (b) Timely (c) DCQCN

Figure 8: The figures compare the tail latency for single-packet messages for IRN, IRN with PFC, and RoCE (with PFC), across different congestion control algorithms.

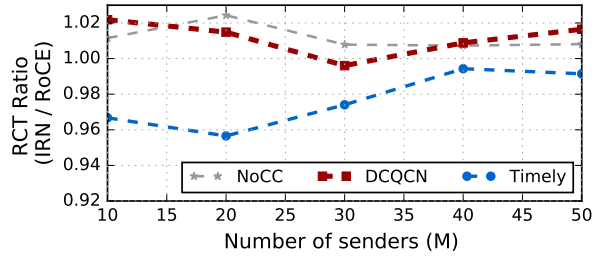


Figure 9: The figure shows the ratio of request completion time of incast with IRN (without PFC) over RoCE (with PFC) for varying degree of fan-ins across congestion control algorithms.

our default scenario, which is another relevant metric in datacenters [30]. Figure 8 shows the tail CDF of this latency (from 90%ile to 99.9%ile), across different congestion control algorithms. Our key trends from §4.2 hold even for this metric. This is because IRN (without PFC) is able to recover from single-packet message losses quickly due to the low RTO_{low} timeout value. With PFC, these messages end up waiting in the queues for similar (or greater) duration due to pauses and congestion spreading. For all cases, IRN performs significantly better than RoCE.

4.4.3 Incast. We now evaluate incast scenarios, both with and without cross-traffic. The incast workload without any cross traffic can be identified as the best case for PFC, since only valid congestion-causing flows are paused without unnecessary head-of-the-line blocking.

Incast without cross-traffic: We simulate the incast workload on our default topology by striping 150MB of data across M randomly chosen sender nodes that send it to a fixed destination node [17]. We vary M from 10 to 50. We consider the request completion time (RCT) as the metric for incast performance, which is when the last flow completes. For each M, we repeat the experiment hundred times and report the average RCT from these runs. Figure 9 shows the results, comparing IRN with RoCE. We find that the two had comparable performance: any increase in the RCT due to disabling PFC with IRN remained within 2.5%. The results comparing IRN's performance with and without PFC looked very similar. We also varied our default incast setup by changing the bandwidths to 10Gbps and 100Gbps, and increasing the number of connections per machine. Any degradation in performance due to disabling PFC with IRN stayed within 9%.

Incast with cross traffic: In practice we expect incast to occur with other cross traffic in the network [24, 30]. We started an incast as described above with $M = 30$, along with our default case workload running at 50% link utilization level. The incast RCT for IRN (without PFC) was always lower than RoCE (with PFC) by 4%-30% across the three congestion control schemes. For the background workload, the performance of IRN was better than RoCE by 32%-87% across the three congestion control schemes and the three metrics (*i.e.,* the average slowdown, the average FCT and the tail FCT). Enabling PFC with IRN generally degraded performance for both the incast and the cross-traffic by 1-75% across the three schemes and metrics, and improved performance only for one case (incast workload with DCQCN by 1.13%).

4.4.4 Window-based congestion control. We also implemented conventional window-based congestion control schemes such as TCP's AIMD and DCTCP with IRN and observed similar trends as discussed in §4.2. In fact, when IRN is used with TCP's AIMD logic, the benefits of disabling PFC were even stronger. This is because TCP exploits packet drops as a congestion signal (which is lost when PFC is enabled). Detailed results have been excluded for brevity.

Summary: Our key results *i.e.,* (1) IRN (without PFC) performs better than RoCE (with PFC), and (2) IRN does not require PFC, hold across varying realistic scenarios, congestion control schemes and performance metrics.

4.5 Comparison with iWARP.

We finally explore whether IRN's simplicity over the TCP stack implemented in iWARP impacts performance. We compare IRN's performance (without any explicit congestion control) with full-blown TCP stack's, using INET simulator's in-built TCP implementation for the latter. Figure 10 shows the results for our default scenario. We find that absence of slow-start (with use of BDP-FC instead) results in

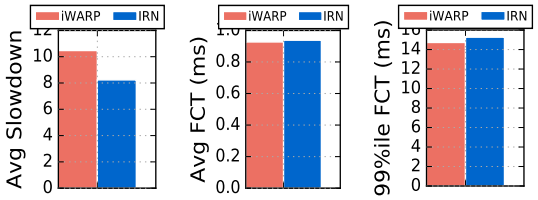


Figure 10: The figures compares iWARP's transport (TCP stack) with IRN.

21% smaller slowdowns with IRN and comparable average and tail FCTs. These results show that in spite of a simpler design, IRN's performance is better than full-blown TCP stack's, even without any explicit congestion control. Augmenting IRN with TCP's AIMD logic further improves its performance, resulting in 44% smaller average slowdown and 11% smaller average FCT as compared to iWARP.

5 Implementation Considerations

We now discuss how one can incrementally update RoCE NICs to support IRN's transport logic, while maintaining the correctness of RDMA semantics as defined by the Infiniband RDMA specification [5]. Our implementation relies on extensions to RDMA's packet format, *e.g.,* introducing new fields and packet types. These extensions are encapsulated within IP and UDP headers (as in RoCEv2) so they only effect the end-host behavior and not the network behavior (i.e. no changes are required at the switches). We begin with providing some relevant context about different RDMA operations before describing how IRN supports them.

5.1 Relevant Context

The two remote endpoints associated with an RDMA message transfer are called a *requester* and a *responder*. RDMA has this notion of *Work Queue Elements* or WQEs (pronounced as *wookies*), which act as an interface between the user application and the RDMA NIC. The application posts a WQE for each RDMA message transfer, which contains the application-specified metadata for the transfer. It gets stored in the NIC while the message is being processed, and is expired upon message completion. The WQEs posted at the requester and at the responder NIC are called Request WQEs and Receive WQEs respectively. Expiration of a WQE upon message completion is followed by creation of a *Completion Queue Element* or a CQE (pronounced as *cookie*), which signals the message completion to the user application. There are four types of message transfers supported by RDMA NICs:

Write: The requester *writes* data to responder's memory. The data length, source and sink locations are specified in the Request WQE, and typically, no Receive WQE is required. However, Write-with-immediate operation requires the user application to post a Receive WQE that expires upon completion to generate a CQE (thus signaling Write completion at the responder as well).

Read: The requester *reads* data from responder's memory. The data length, source and sink locations are specified in the Request WQE, and no Receive WQE is required.

Send: The requester *sends* data to the responder. The data length and source location is specified in the Request WQE, while the sink location is specified in the Receive WQE.

Atomic: The requester reads and atomically updates the data at a location in the responder's memory, which is specified in the Request WQE. No Receive WQE is required. Atomic operations are restricted to single-packet messages.

5.2 Supporting RDMA Reads and Atomics

IRN relies on per-packet ACKs for BDP-FC and loss recovery. RoCE NICs already support per-packet ACKs for Writes and Sends. However, when doing Reads, the requester (which is the data sink) does not explicitly acknowledge the Read response packets. IRN, therefore, introduces packets for *read (N)ACKs* that are sent by a requester for each Read response packet. RoCE currently has eight unused opcode values available for the reliable connected QPs, and we use one of these for *read (N)ACKs*. IRN also requires the Read responder (which is the data source) to implement timeouts. New timer-driven actions (such as, for implementing DCQCN) have been added to the NIC hardware implementation in the past [35]. Hence, this is not an issue. RDMA Atomic operations are treated similar to a single-packet RDMA Read message.

Our simulations from §4 did not use ACKs for the RoCE (with PFC) baseline, modelling the extreme case of all Reads. Therefore, our results take into account the overhead of per-packet ACKs in IRN.

5.3 Supporting Out-of-order Packet Delivery

One of the key challenge for implementing IRN is supporting out-of-order (OOO) packet delivery at the receiver – current RoCE NICs simply discard OOO packets. A naive approach for handling OOO packet would be to store all of them in the NIC memory. The total number of OOO packets with IRN is bounded by the BDP cap (which is about 110 MTU-sized packets for our default scenario as described in §4.1) [2]. Therefore to support a thousand flows, a NIC would need to buffer 110MB of packets, which exceeds the memory capacity on most commodity RDMA NICs.

We therefore explore an alternate implementation strategy, where the NIC DMAs OOO packets directly to the final address in the application memory and keeps track of them

[2]For QPs that only send single packet messages less than one MTU in size, the number of outstanding packets is limited to the maximum number of outstanding requests, which is typically much smaller than the BDP cap [26, 27].

using bitmaps (which are sized at BDP cap). This reduces NIC memory requirements from 1KB per OOO packet to only a couple of bits. However, it introduces some additional challenges that need to be addressed. Note that partial support for OOO packet delivery was introduced in the Mellanox ConnectX-5 NICs to enable adaptive routing [12]. However, it is restricted to Write and Read operations. We improve and extend this design to support all RDMA operations with IRN.

We classify the issues due to out-of-order packet delivery into four categories.

5.3.1 First packet issues. For some RDMA operations, critical information is carried in the first packet of a message, which is required to process other packets in the message. Enabling OOO delivery, therefore, requires that some of the information in the first packet be carried by all packets.

In particular, the RETH header (containing the remote memory location) is carried only by the first packet of a Write message. IRN requires adding it to every packet.

5.3.2 WQE matching issues. Some operations require every packet that arrives to be matched with its corresponding WQE at the responder. This is done implicitly for in-order packet arrivals. However, this implicit matching breaks with OOO packet arrivals. A work-around for this is assigning explicit WQE sequence numbers, that gets carried in the packet headers and can be used identify the corresponding WQE for each packet. IRN uses this workaround for the following RDMA operations:

Send and Write-with-immediate: It is required that Receive WQEs be consumed by Send and Write-with-immediate requests in same the order in which they are posted. Therefore, with IRN every Receive WQE, and every Request WQE for these operations, maintains a *recv_WQE_SN*, that indicates the order in which they are posted. This value is carried in all Send packets and in the last Write-with-Immediate packet, [3] and is used to identify the appropriate Receive WQE. IRN also requires the Send packets to carry the relative offset in the packet sequence number, which is used to identify the precise address when placing data.

Read/Atomic: The responder cannot begin processing a Read-/Atomic request *R*, until all packets expected to arrive before *R* have been received. Therefore, an OOO Read/Atomic Request packet needs to be placed in a Read WQE buffer at the responder (which is already maintained by current RoCE NICs). With IRN, every Read/Atomic Request WQE maintains a *read_WQE_SN*, that is carried by all Read/Atomic request packets and allows identification of the correct index in this Read WQE buffer.

[3] A Receive WQE is consumed only by the last packet of a Write-with-immediate message, and is required to process all packets for a Send message.

5.3.3 Last packet issues. For many RDMA operations, critical information is carried in last packet, which is required to complete message processing. Enabling OOO delivery, therefore, requires keeping track of such last packet arrivals and storing this information at the endpoint (either on NIC or main memory), until all other packets of that message have arrived. We explain this in more details below.

A RoCE responder maintains a *message sequence number (MSN)* which gets incremented when the last packet of a Write/Send message is received or when a Read/Atomic request is received. This MSN value is sent back to the requester in the ACK packets and is used to expire the corresponding Request WQEs. The responder also expires its Receive WQE when the last packet of a Send or a Write-With-Immediate message is received and generates a CQE. The CQE is populated with certain meta-data about the transfer, which is carried by the last packet. IRN, therefore, needs to ensure that the completion signalling mechanism works correctly even when the last packet of a message arrives before others. For this, an IRN responder maintains a 2-bitmap, which in addition to tracking whether or not a packet p has arrived, also tracks whether it is the last packet of a message that will trigger (1) an MSN update and (2) in certain cases, a Receive WQE expiration that is followed by a CQE generation. These actions are triggered only after all packets up to p have been received. For the second case, the *recv_WQE_SN* carried by p (as discussed in §5.3.2) can identify the Receive WQE with which the meta-data in p needs to be associated, thus enabling a *premature CQE* creation. The premature CQE can be stored in the main memory, until it gets delivered to the application after all packets up to p have arrived.

5.3.4 Application-level issues. Certain applications (for example FaRM [22]) rely on polling the last packet of a Write message to detect completion, which is incompatible with OOO data placement. This polling based approach violates the RDMA specification (Sec o9-20 [5]) and is more expensive than officially supported methods (FaRM [22] mentions moving on to using the officially supported Write-with-Immediate method in the future for better scalability). IRN's design provides all of the Write completion guarantees as per the RDMA specification.

OOO data placement can also result in a situation where data written to a particular memory location is overwritten by a restransmitted packet from an older message. Typically, applications using distributed memory frameworks assume relaxed memory ordering and use application layer *fences* whenever strong memory consistency is required [15, 36]. Therefore, both iWARP and Mellanox ConnectX-5, in supporting OOO data placement, expect the application to deal with the potential memory over-writing issue and do not handle it in the NIC or the driver. IRN can adopt the same strategy.

Another alternative is to deal with this issue in the driver, by enabling the fence indicator for a newly posted request that could potentially overwrite an older one.

5.4 Other Considerations

Currently, the packets that are sent and received by a requester use the same packet sequence number (*PSN*) space. This interferes with loss tracking and BDP-FC. IRN, therefore, splits the *PSN* space into two different ones (1) *sPSN* to track the request packets sent by the requester, and (2) *rPSN* to track the response packets received by the requester. This decoupling remains transparent to the application and is compatible with the current RoCE packet format. IRN can also support shared receive queues and send with invalidate operations and is compatible with use of end-to-end credit. Details about these have been omitted for brevity.

6 Evaluating Implementation Overheads

We now evaluate IRN's implementation overheads over current RoCE NICs along the following three dimensions: in §6.1, we do a comparative analysis of IRN's memory requirements; in §6.2, we evaluate the overhead for implementing IRN's packet processing logic by synthesizing it on an FPGA; and in §6.3, we evaluate, via simulations, how IRN's implementation choices impact end-to-end performance.

6.1 NIC State overhead due to IRN

Mellanox RoCE NICs support several MBs of cache to store various metadata including per-QP and per-WQE contexts [3]. The additional state that IRN introduces consumes a total of only 3-10% of the current NIC cache for a couple of thousands of QPs and tens of thousands of WQEs, even when considering large 100Gbps links. We present a breakdown this additional state below.

Additional Per-QP Context:

State variables: IRN needs 52 bits of additional state for its transport logic: 24 bits each to track the packet sequence to be retransmitted and the recovery sequence, and 4 bits for various flags. Other per-flow state variables needed for IRN's transport logic (e.g., expected sequence number) are already maintained by current RoCE NICs. Hence, the per-QP overhead is 104 bits (52 bits each at the requester and the responder). Maintaining a timer at the responder for Read timeouts and a variable to track in-progress Read requests in the Read WQE buffer adds another 56 bits to the responder leading to a total of 160 bits of additional per-QP state with IRN. For context, RoCE NICs currently maintain a few thousands of bits per QP for various state variables [3].

Bitmaps: IRN requires five BDP-sized bitmaps: two at the responder for the 2-bitmap to track received packets, one at the requester to track the Read responses, one each at the requester and responder for tracking selective acks. Assuming each bitmap to be 128 bits (*i.e.,* sized to fit the BDP cap for a network with bandwidth 40Gbps and a two-way propagation delay of up to 24μs, typical in today's datacenter topologies [30]), IRN would require a total of 640 bits per QP for bitmaps. This is much less than the total size of bitmaps maintained by a QP for the OOO support in Mellanox ConnectX-5 NICs [3].

Others: Other per-QP meta-data that is needed by an IRN driver when a WQE is posted (e.g counters for assigning WQE sequence numbers) or expired (e.g. premature CQEs) can be stored directly in the main memory and do not add to the NIC memory overhead.

Additional Per-WQE Context: As described in §5, IRN maintains sequence numbers for certain types of WQEs. This adds 3 bytes to the per-WQE context which is currently sized at 64 bytes [3].

Additional Shared State: IRN also maintains some additional variables (or parameters) that are shared across QPs. This includes the BDP cap value, the RTO_{low} value, and N for RTO_{low}, which adds up to a total of only 10 bytes.

6.2 IRN's packet processing overhead

We evaluate the implementation overhead due to IRN's per-packet processing logic, which requires various bitmap manipulations. The logic for other changes that IRN makes – e.g., adding header extensions to packets, premature CQE generation, etc. – are already implemented in RoCE NICs and can be easily extended for IRN.

We use Xilinx Vivado Design Suite 2017.2 [4] to do a high-level synthesis of the four key packet processing modules (as described below), targeting the Kintex Ultrascale XCKU060 FPGA which is supported as a bump-on-the-wire on the Mellanox Innova Flex 4 10/40Gbps NICs [13].

6.2.1 Synthesis Process. To focus on the *additional* packet processing complexity due to IRN, our implementation for the four modules is *stripped-down*. More specifically, each module receives the relevant packet metadata and the QP context as streamed inputs, relying on a RoCE NIC's existing implementation to parse the packet headers and retrieve the QP context from the NIC cache (or the system memory, in case of a cache miss). The updated QP context is passed as streamed output from the module, along with other relevant module-specific outputs as described below.

(1) receiveData: Triggered on a packet arrival, it outputs the relevant information required to generate an ACK/NACK packet and the number of Receive WQEs to be expired, along with the updated QP context (e.g. bitmaps, expected sequence number, MSN).

(2) txFree: Triggered when the link's Tx is free for the QP to transmit, it outputs the sequence number of the packet to be (re-)transmitted and the updated QP context (e.g. next

Module Name	Resource Usage		Max	Min
	FF	LUT	Latency	Throughput
receiveData	0.62%	1.93%	16.5 ns	45.45 Mpps
txFree	0.32%	0.95%	15.9 ns	47.17 Mpps
receiveAck	0.4%	1.05%	15.96 ns	46.99 Mpps
timeout	0.01%	0.08%	<6.3 ns	318.47 Mpps
Total Resource Usage: 1.35% FF and 4.01% LUTs				
Min Bottleneck Tpt: 45.45Mpps				

Table 2: Performance and resource usage for different packet processing modules on Xilinx Kintex Ultrascale KU060 FPGA.

sequence to transmit). During loss-recovery, it also performs a look ahead by searching the SACK bitmap for the next packet sequence to be retransmitted.

(3) receiveAck: Triggered when an ACK/NACK packet arrives, it outputs the updated QP context (e.g. SACK bitmap, last acknowledged sequence).

(4) timeout: If triggered when the timer expires using RTO_{low} value (indicated by a flag in the QP context), it checks if the condition for using RTO_{low} holds. If not, it does not take any action and sets an output flag to extend the timeout to RTO_{high}. In other cases, it executes the timeout action and returns the updated QP context. Our implementation relies on existing RoCE NIC's support for setting timers, with the RTO_{low} value being used by default, unless explicitly extended.

The bitmap manipulations in the first three modules account for most of the complexity in our synthesis. Each bitmap was implemented as a ring buffer, using an arbitrary precision variable of 128 bits, with the *head* corresponding to the expected sequence number at the receiver (or the cumulative acknowledgement number at the sender). The key bitmap manipulations required by IRN can be reduced to the following three categories of known operations: (i) *finding first zero*, to find the next expected sequence number in *receiveData* and the next packet to retransmit in *txFree* (ii) *popcount* to compute the increment in MSN and the number of Receive WQEs to be expired in *receiveData*, (iii) *bit shifts* to advance the bitmap *heads* in *receiveData* and *receiveAck*. We optimized the first two operations by dividing the bitmap variables into chunks of 32 bits and operating on these chunks in parallel.

We validated the correctness of our implementation by generating input event traces for each synthesized module from the simulations described in §4 and passing them as input in the test bench used for RTL verification by the Vivado Design Suite. The output traces, thus, generated were then matched with the corresponding output traces obtained from the simulator. We also used the Vivado HLS tool to export our RTL design to create IP blocks for our modules.

6.2.2 Synthesis Results. Our FPGA synthesis report has been summarized in Table 2 and discussed below.

Resource Usage: The second and third columns in Table 2 report the percentage of flip-flops (FF) and look-up tables (LUT) used for the four modules (no BRAM or DSP48E units were consumed). We find that each of IRN's packet processing modules consume less than 1% FFs and 2% LUTs (with a total of 1.35% FFs and 4% LUTs consumed). Increasing the bitmap size to support 100Gbps links consumed a total of 2.66% of FFs and 9.5% of LUTs on the same device (though we expect the relative resource usage to be smaller on a higher-scale device designed for 100Gbps links).

Performance: The third and fourth column in Table 2 report the worst-case latency and throughput respectively for each module. [4] The latency added by each module is at most only 16.5ns. The *receiveData* module (requiring more complex bitmap operations) had the lowest throughput of 45.45Mpps. This is high enough to sustain a rate of 372Gbps for MTU-sized packets. It is also higher than the maximum rate of 39.5Mpps that we observed on Mellanox MCX416A-BCAT RoCE NIC across different message sizes (2 bytes - 1KB), after applying various optimizations such as batching and using multiple queue-pairs. A similar message rate was observed in prior work [26]. Note that we did not use pipelining within our modules, which can further improve throughput.

While we expect IRN to be implemented on the ASIC integrated with the existing RoCE implementation, we believe that the modest resources used on an FPGA board supported as an *add-on* in recent RDMA-enabled NICs, provides some intuition about the feasibility of the changes required by IRN. Also, note that the results reported here are far from the optimal results that can be achieved on an ASIC implementation due to two sources of sub-optimality: (i) using HLS for FPGA synthesis has been found to be up to 2× less optimal than directly using Verilog [28] and (ii) FPGAs, in general, are known to be less optimal than ASICs.

6.3 Impact on end-to-end performance

We now evaluate how IRN's implementation overheads impact the end-to-end performance. We identify the following two implementation aspects that could potentially impact end-to-end performance and model these in our simulations.

Delay in Fetching Retransmissions: While the regular packets sent by a RoCE NIC are typically pre-fetched, we assume that the DMA read request for retransmissions is sent only after the packet is identified as lost (i.e. when loss recovery is triggered or when a look-ahead is performed). The time taken to fetch a packet over PCIe is typically between a few hundred nanoseconds to $<2\mu$s [9, 32]. We set a worst-case retransmission delay of 2μs for every retransmitted packet i.e. the sender QP is allowed to retransmit a packet only after 2μs have elapsed since the packet was detected as lost.

[4] The worst-case throughput was computed by dividing the clock frequency with the maximum initiation interval, as reported by the Vivado HLS synthesis tool [8].

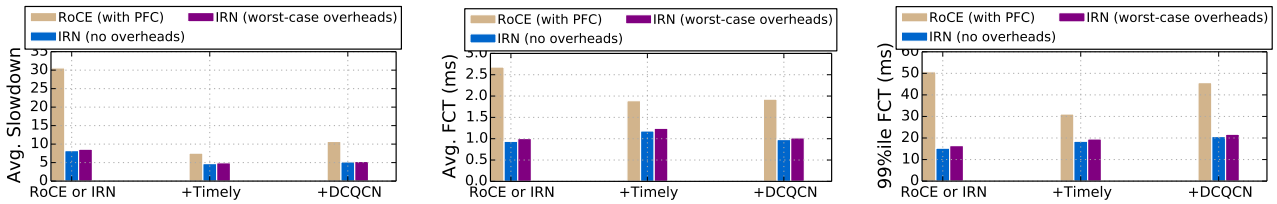


Figure 11: The figures show the performance of IRN with worse case overheads, comparing it with IRN without any overheads and with RoCE for our default case scenario.

Additional Headers: As discussed in §5, some additional headers are needed in order to DMA the packets directly to the application memory, of which, the most extreme case is the 16 bytes of RETH header added to every Write packet. Send data packets have an extra header of 6 bytes, while Read responses do not require additional headers. We simulate the worst-case scenario of all Writes with every packet carrying 16 bytes additional header.

Results: Figure 11 shows the results for our default scenario after modeling these two sources of worst-case overheads. We find that they make little difference to the end-to-end performance (degrading the performance by 4-7% when compared to IRN without overheads). The performance remains 35%-63% better than our baseline of RoCE (with PFC). We also verified that the retransmission delay of 2μs had a much smaller impact on end-to-end performance (2μs is very small compared to the network round-trip time taken to detect a packet loss and to recover from it, which could be of the order of a few hundred microseconds). The slight degradation in performance observed here can almost entirely by attributed to the additional 16 bytes header in every packet. Therefore, we would expect the performance impact to be even smaller when there is a mix of Write and Read workloads.

6.4 Summary

Our analysis shows that IRN is well within the limits of feasibility, with small chip area and NIC memory requirements and minor bandwidth overhead. We also validated our analysis through extensive discussions with two commercial NIC vendors (including Mellanox); both vendors confirmed that the IRN design can be easily implemented on their hardware NICs. Inspired by the results presented in this paper, Mellanox is implementing a version of IRN in their next release.

7 Discussion and Related Work

Backwards Compatibility: We briefly sketch one possible path to incrementally deploying IRN. We envision that NIC vendors will manufacture NICs that support dual RoCE/IRN modes. The use of IRN can be negotiated between two endpoints via the RDMA connection manager, with the NIC falling back to RoCE mode if the remote endpoint does not support IRN. (This is similar to what was used in moving from RoCEv1 to RoCEv2.) Since IRN performs well both with or without PFC, network operators can continue to run PFC until all their endpoints have been upgraded to support IRN at which point PFC can be permanently disabled. During the interim period, hosts can communicate using either RoCE or IRN, with no loss in performance.

Loss Recovery vs Loss Avoidance: IRN adopts an approach based on better loss recovery to eliminate the need for PFC. A different approach is to *avoid* packet losses (and PFC frames) by careful tuning of advanced congestion control schemes. In particular, [35] explores the use of DCQCN to avoid packet losses in specific scenarios. However, as we showed in §4, DCQCN is not always successful in avoiding packet losses across different realistic scenarios and hence PFC (with its accompanying problems) remains necessary. Another recent proposal [20] uses careful switch configuration to avoid losses but this can result in under-utilization of network capacity. Running these schemes in combination with IRN (instead of PFC) would improve their robustness and allow them to better utilize the available link capacity.

Reordering due to load-balancing: Datacenters today use ECMP for load balancing [24], that maintains ordering within a flow. IRN's OOO packet delivery support also allows for other load balancing schemes that may cause packet reordering within a flow [21, 23]. IRN's loss recovery mechanism can be made more robust to reordering by triggering loss recovery only after a certain threshold of NACKs are received.

Other hardware-based loss recovery: MELO [29], a recent scheme developed in parallel to IRN, proposes an alternative design for hardware-based selective retransmission, where out-of-order packets are buffered in an off-chip memory. Unlike IRN, MELO only targets PFC-enabled environments with the aim of greater robustness to random losses caused by failures. As such, MELO is orthogonal to our main focus which is showing that PFC is unnecessary. Nonetheless, the existence of alternate designs such as MELO's further corroborates the feasibility of implementing better loss recovery on NICs.

HPC workloads: The HPC community has long been a strong supporter of losslessness. This is primarily because HPC clusters are smaller with more controlled traffic patterns, and hence the negative effects of providing losslessness (such as congestion spreading and deadlocks) are rarer. PFC's issues are exacerbated on larger scale clusters [24, 25, 30, 34, 38].

Credit-based Flow Control: Since the focus of our work was RDMA deployment over Ethernet, our experiments used PFC. Another approach to losslessness, used by Infiniband, is credit-based flow control, where the downlink sends credits to the uplink when it has sufficient buffer capacity. Credit-based flow control suffers from the same performance issues as PFC: head-of-the-line blocking, congestion spreading, the potential for deadlocks, *etc*. We, therefore, believe that our observations from §4 can be applied to credit-based flow control as well.

References

[1] http://omnetpp.org/.

[2] https://inet.omnetpp.org.

[3] Personal communication with Mellanox.

[4] Xilinx Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html.

[5] InfiniBand architecture volume 1, general specifications, release 1.2.1. www.infinibandta.org/specs, 2008.

[6] Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A16: RDMA over Converged Ethernet (RoCE). www.infinibandta.org/specs, 2010.

[7] IEEE. 802.11Qbb. Priority based flow control, 2011.

[8] Vivado Design Suite User Guide. https://goo.gl/akRdXC, 2013.

[9] http://www.xilinx.com/support/documentation/white_papers/wp350.pdf, 2014.

[10] Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE),. www.infinibandta.org/specs, 2014.

[11] Mellanox ConnectX-4 Product Brief. https://goo.gl/HBw9f9, 2016.

[12] Mellanox ConnectX-5 Product Brief. https://goo.gl/ODlqMl, 2016.

[13] Mellanox Innova Flex 4 Product Brief. http://goo.gl/Lh7VN4, 2016.

[14] RoCE vs. iWARP Competitive Analysis. http://www.mellanox.com/related-docs/whitepapers/WP_RoCE_vs_iWARP.pdf, 2017.

[15] Sarita V Adve and Hans-J Boehm. Memory models: a case for rethinking parallel languages and hardware. *Communications of the ACM*, 53(8):90–101, 2010.

[16] Mohammad Alizadeh, Shuang Yang, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Deconstructing Datacenter Packet Transport. In *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2012.

[17] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pFabric: Minimal Near-optimal Datacenter Transport. In *Proc. ACM SIGCOMM*, 2013.

[18] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proc. ACM SIGCOMM*, 2004.

[19] Theophilus Benson, Aditya Akella, and David Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. ACM Internet Measurement Conference (IMC)*, 2012.

[20] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 239–252, New York, NY, USA, 2017. ACM.

[21] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2130–2138. IEEE, 2013.

[22] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 401–414, Seattle, WA, 2014. USENIX Association.

[23] Soudeh Ghorbani, Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Micro load balancing in data centers with drill. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, pages 17:1–17:7, New York, NY, USA, 2015. ACM.

[24] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 202–215. ACM, 2016.

[25] Shuihai Hu, Yibo Zhu, Peng Cheng, Chuanxiong Guo, Kun Tan, Jitendra Padhye, and Kai Chen. Deadlocks in datacenter networks: Why do they form, and how to avoid them. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 92–98, New York, NY, USA, 2016. ACM.

[26] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using rdma efficiently for key-value services. *SIGCOMM Comput. Commun. Rev.*, 44(4):295–306, August 2014.

[27] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance rdma systems. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '16, pages 437–450, Berkeley, CA, USA, 2016. USENIX Association.

[28] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 1–14, New York, NY, USA, 2016. ACM.

[29] Yuanwei Lu, Guo Chen, Zhenyuan Ruan, Wencong Xiao, Bojie Li, Jiansong Zhang, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Memory efficient loss recovery for hardware-based transport in datacenter. In *Proceedings of the First Asia-Pacific Workshop on Networking*, APNet'17, pages 22–28, New York, NY, USA, 2017. ACM.

[30] R Mittal, V 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 *Proc. ACM SIGCOMM*, 2015.

[31] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. Recursively Cautious Congestion Control. In *Proc. USENIX NSDI*, 2014.

[32] Sivasankar Radhakrishnan, Yilong Geng, Vimalkumar Jeyakumar, Abdul Kabbani, George Porter, and Amin Vahdat. Senic: Scalable nic for end-host rate limiting. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 475–488, Berkeley, CA, USA, 2014. USENIX Association.

[33] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A Remote Direct Memory Access Protocol Specification. RFC 5040, 2007.

[34] Alex Shpiner, Eitan Zahavi, Vladimir Zdornov, Tal Anker, and Matty Kadosh. Unlocking credit loop deadlocks. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 85–91. ACM, 2016.

[35] Alexander Shpiner, Eitan Zahavi, Omar Dahley, Aviv Barnea, Rotem Damsker, Gennady Yekelis, Michael Zus, Eitan Kuta, and Dean Baram. Roce rocks without pfc: Detailed evaluation. In *Proceedings of the Workshop on Kernel-Bypass Networks*, KBNets '17, pages 25–30, New York, NY, USA, 2017. ACM.

[36] Daniel J. Sorin, Mark D. Hill, and David A. Wood. *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers, 1st edition, 2011.

[37] Brent Stephens, Alan L Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. Practical dcb for improved data center networks. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1824–1832. IEEE, 2014.

[38] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. Congestion Control for Large-Scale RDMA Deployments. In *ACM SIGCOMM*, 2015.