RDMA over Commodity Ethernet at Scale

Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni,
Jianxi Ye, Jitendra Padhye, Marina Lipshteyn
Microsoft
{chguo, hwu, zdeng, gasoni, jiye, padhye, malipsht}@microsoft.com

ABSTRACT

Over the past one and half years, we have been using RDMA over commodity Ethernet (RoCEv2) to support some of Microsoft's highly-reliable, latency-sensitive services. This paper describes the challenges we encountered during the process and the solutions we devised to address them. In order to scale RoCEv2 beyond VLAN, we have designed a DSCP-based priority flow-control (PFC) mechanism to ensure large-scale deployment. We have addressed the safety challenges brought by PFC-induced deadlock (yes, it happened!), RDMA transport livelock, and the NIC PFC pause frame storm problem. We have also built the monitoring and management systems to make sure RDMA works as expected. Our experiences show that the safety and scalability issues of running RoCEv2 at scale can all be addressed, and RDMA can replace TCP for intra data center communications and achieve low latency, low CPU overhead, and high throughput.

CCS Concepts

•**Networks** → **Network protocol design; Network experimentation; Data center networks;**

Keywords

RDMA; RoCEv2; PFC; PFC propagation; Deadlock

1. INTRODUCTION

With the rapid growth of online services and cloud computing, large-scale data centers (DCs) are being built around the world. High speed, scalable data center networks (DCNs) [1, 3, 19, 31] are needed to connect the servers in a DC. DCNs are built from commodity Ethernet switches and network interface cards (NICs). A state-of-the-art DCN must support several Gb/s or higher throughput between any two servers in a DC.

TCP/IP is still the dominant transport/network stack in today's data center networks. However, it is increasingly clear that the traditional TCP/IP stack cannot meet the demands of the new generation of DC workloads [4, 9, 16, 40], for two reasons.

First, the CPU overhead of handling packets in the OS kernel remains high, despite enabling numerous hardware and software optimizations such as checksum offloading, large segment offload (LSO), receive side scaling (RSS) and interrupt moderation. Measurements in our data centers show that sending at 40Gb/s using 8 TCP connections chews up 6% aggregate CPU time on a 32 core Intel Xeon E5-2690 Windows 2012R2 server. Receiving at 40Gb/s using 8 connections requires 12% aggregate CPU time. This high CPU overhead is unacceptable in modern data centers.

Second, many modern DC applications like Search are highly latency sensitive [7, 15, 41]. TCP, however, cannot provide the needed low latency even when the average traffic load is moderate, for two reasons. First, the kernel software introduces latency that can be as high as tens of milliseconds [21]. Second, packet drops due to congestion, while rare, are not entirely absent in our data centers. This occurs because data center traffic is inherently bursty. TCP must recover from the losses via timeouts or fast retransmissions, and in both cases, application latency takes a hit.

In this paper we summarize our experience in deploying RoCEv2 (RDMA over Converged Ethernet v2) [5], an RDMA (Remote Direct Memory Access) technology [6], to address the above mentioned issues in Microsoft's data centers. RDMA is a method of accessing memory on a remote system without interrupting the processing of the CPU(s) on that system. RDMA is widely used in high performance computing with Infiniband [6] as the infrastructure. RoCEv2 supports RDMA over Ethernet instead of Infiniband.

Unlike TCP, RDMA needs a lossless network; i.e. there must be no packet loss due to buffer overflow at the switches. RoCEv2 uses PFC (Priority-based Flow Control) [14] for this purpose. PFC prevents buffer

SIGCOMM '16, August 22 - 26, 2016, Florianopolis , Brazil

 ISBN 978-1-4503-4193-6/16/08…$15.00

DOI: http://dx.doi.org/10.1145/2934872.2934908

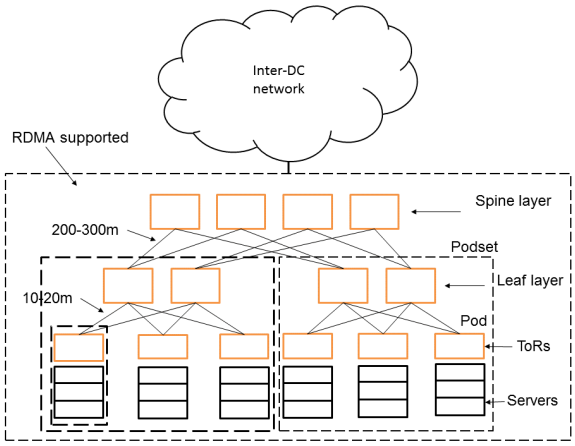


Figure 1: Our goal is to support RDMA for intra data center (intra-DC) communications.

overflow by pausing the upstream sending entity when buffer occupancy exceeds a specified threshold. While some problems with PFC such as head-of-the line blocking and potential for deadlock are well known [22, 33], we see several issues such as the RDMA transport livelock, the NIC PFC pause frame storm and the slow-receiver symptom in our deployment that have not been reported in the literature. Even the root cause of the deadlock problem we have encountered is quite different from the toy examples often discussed in the research literature [22, 33].

We also note that VLAN [32] tags are typically used to identify PFC-enabled traffic in mixed RDMA/TCP deployments. As we shall discuss, this solution does not scale for our environment. Thus, we introduce a notion of DSCP (Differentiated Services Code Point) based PFC to scale RDMA from layer-2 VLAN to layer-3 IP.

Our RDMA deployment has now been running smoothly for over one and half years, and it supports some of Microsoft's highly-reliable and latency-sensitive online services. Our experience shows that, by improving the design of RoCEv2, by addressing the various safety issues, and by building the needed management and monitoring capabilities, we can deploy RDMA safely in large-scale data centers using commodity Ethernet.

2. BACKGROUND

Our data center network is an Ethernet-based multi-layer Clos network [1, 3, 19, 31] as shown in Figure 1. Twenty to forty servers connect to a top-of-rack (ToR) switch. Tens of ToRs connect to a layer of Leaf switches. The Leaf switches in turn connect to a layer of tens to hundreds of Spine switches. Most links are 40Gb/s, and we plan to upgrade to 50GbE and 100GbE in near future [11, 25]. All switches use IP routing.

The servers typically use copper cables of around 2 meters to connect to the ToR switches. The ToR switches and Leaf switches are within the distance of 10

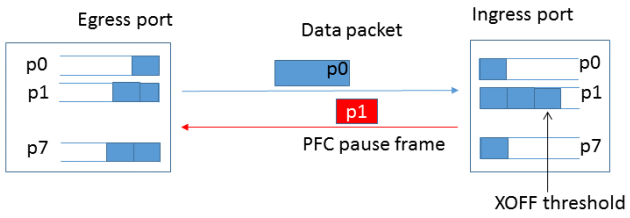


Figure 2: How PFC works.

- 20 meters, and the Leaf and Spine switches are within the distance of 200 - 300 meters. With three layers of switches, tens to hundreds of thousands of servers can be connected in a single data center. In this paper, we focus on supporting RDMA among servers under the same Spine switch layer.

RoCEv2: We deployed RDMA over Converged Ethernet v2 (RoCEv2) [5] for both technical and economical reasons. RoCEv2 encapsulates an RDMA transport [5] packet within an Ethernet/IPv4/UDP packet. This makes RoCEv2 compatible with our existing networking infrastructure. The UDP header is needed for ECMP-based [34] multi-path routing. The destination UDP port is always set to 4791, while the source UDP port is randomly chosen for each queue pair (QP) [5]. The intermediate switches use standard five-tuple hashing. Thus, traffic belonging to the same QP follows the same path, while traffic on different QPs (even between the same pair of communicating end points) can follow different paths.

PFC and buffer reservation: RoCEv2 uses PFC [14] to prevent buffer overflow. The PFC standard specifies 8 priority classes to reduce the head-of-line blocking problem. However, in our network, we are able to use only two of these eight priorities for RDMA. The reason is as follows.

PFC is a hop-by-hop protocol between two Ethernet nodes. As show in Figure 2, the sender's egress port sends data packets to the receiver's ingress port. At the sending egress port, packets are queued in up to eight queues. Each queue maps to a priority. At the receiving ingress port, packets are buffered in corresponding ingress queues. In the shared-buffer switches used in our network, an ingress queue is implemented simply as a counter – all packets share a common buffer pool.

Once the ingress queue length reaches a certain threshold (XOFF), the switch sends out a PFC pause frame to the corresponding upstream egress queue. After the egress queue receives the pause frame, it stops sending packets. A pause frame carries which priorities need to be paused and the pause duration. Once the ingress queue length falls below another threshold (XON), the switch sends a pause with zero duration to resume transmission. XOFF must be set conservatively to ensure that there is no buffer overflow, while XON needs be set to ensure that there is no buffer underflow.

It takes some time for the pause frame to arrive at the upstream egress port, and for the switch to react to

it. During this time, the upstream port will continue to transmit packets. Thus, the ingress port must reserve buffer space for each priority to absorb packets that arrive during this "gray period". This reserved buffer is called *headroom*. The size of the headroom is decided by the MTU size, the PFC reaction time of the egress port, and most importantly, the propagation delay between the sender and the receiver.

The propagation delay is determined by the distance between the sender and the receiver. In our network, this can be as large as 300 meters. Given that our ToR and Leaf switches have shallow buffers (9MB or 12MB), we can only reserve enough headroom for two lossless traffic classes even though the switches support eight traffic classes. We use one lossless class for real-time traffic and the other for bulk data transfer.

Need for congestion control: PFC works hop by hop. There may be several hops from the source server to the destination server. PFC pause frames propagate from the congestion point back to the source if there is persistent network congestion. This can cause problems like unfairness and victim flow [42].

In order to reduce this collateral damage, flow based congestion control mechanisms including QCN [13], DCQCN [42] and TIMELY [27] have been introduced. We use DCQCN, which uses ECN for congestion notification, in our network. We chose DCQCN because it directly reacts to the queue lengths at the intermediate switches and ECN is well supported by all the switches we use. Small queue lengths reduce the PFC generation and propagation probability.

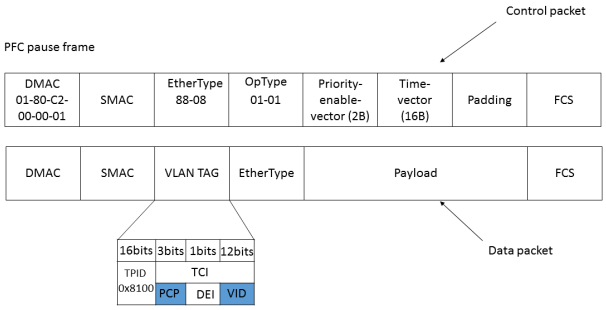
Though DCQCN helps reduce the number of PFC pause frames, it is PFC that protects packets from being dropped as the last defense. PFC poses several safety issues which are the primary focus of this paper and which we will discuss in Section 4. We believe the lessons we have learned in this paper apply to the networks using TIMELY as well.

Coexistence of RDMA and TCP: In this paper, RDMA is designed for intra-DC communications. TCP is still needed for inter-DC communications and legacy applications. We use a different traffic class (which is not lossless), with reserved bandwidth, for TCP. Different traffic classes isolate TCP and RDMA traffic from each other.

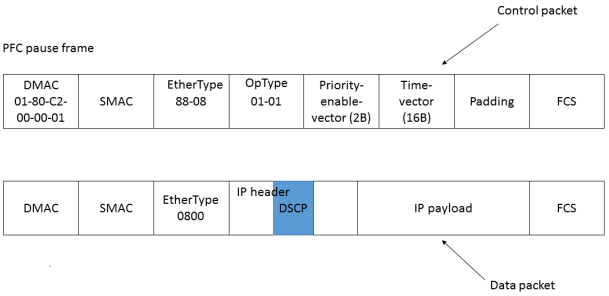
3. DSCP-BASED PFC

In this section we examine the issues faced by the original VLAN-based PFC and present our DSCP-based PFC solution. VLAN-based PFC carries packet priority in the VLAN tag, which also contains VLAN ID. The coupling of packet priority and VLAN ID created two serious problems in our deployment, leading us to develop a DSCP-based PFC solution.

Figure 3(a) shows the packet formats of the PFC pause frame and data packets in the original VLAN-based PFC. The pause frame is a layer-2 frame, and



(a) VLAN-based PFC.



(b) DSCP-based PFC.

Figure 3: The packet formats of VLAN-based PFC and DSCP-based PFC. Note that the PFC pause frame format is the same in both Figure 3(a) and Figure 3(b).

does not have a VLAN tag. The VLAN tag for the data packet has four parts: TPID which is fixed to 0x8100, DEI (Drop Eligible Indicator), PCP (Priority Code Point) which is used to carry packet priority, and VID (VLAN identifier) which carries the VLAN ID of the packet.

For our purpose, although we need only PCP, VID and PCP cannot be separated. Thus, to support PFC, we have to configure VLAN at both the server and the switch side. In order for the switch ports to support VLAN, we need to put the server facing switch ports into trunk mode (which supports VLAN tagged packets) instead of access mode (which sends and receives untagged packets). The basic PFC functionality works with this configuration, but it leads to two problems.

First, the switch trunk mode has an undesirable interaction with our OS provisioning service. OS provisioning is a fundamental service which needs to run when the server OS needs to be installed or upgraded, and when the servers need to be provisioned or repaired. For data centers at our scale, OS provisioning has to be done automatically. We use PXE (Preboot eXecution Environment) boot to install OS from the network. When a server goes through PXE boot, its NIC does not have VLAN configuration and as a result cannot send or receive packets with VLAN tags. But since the server facing switch ports are configured with trunk mode, these ports can only send packets with VLAN tag. Hence the PXE boot communication between the

server and the OS provisioning service is broken. We tried several "hacks" to fix this problem, including letting the switches change the switch port configuration based on the guessed state of the servers, and letting the NICs accept all the packets with or without VLAN tag. However, all these proved to be complex and unreliable, and needless to say, non-standard.

Second, we have moved away from a layer-2 VLAN, and all our switches including the ToR switches are running layer-3 IP forwarding instead of MAC-based layer-2 bridging. A layer-3 network has the benefits of scalability, better management and monitoring, better safety, all public and standard instead of proprietary protocols. However, in a layer-3 network, there is no standard way to preserve the VLAN PCP value when crossing subnet boundaries.

In both problems, the fundamental issue is that VLAN-based PFC unnecessarily couples packet priority and the VLAN ID. We broke this coupling by introducing DSCP-based PFC. Our key observation is that the PFC pause frames do not have a VLAN tag at all. The VLAN tag in data packets is used only for carrying the data packet priority. In the IP world, there is a standard and better way to carry packet priority information, which is the DSCP field in the IP header.

The solution, as shown in Figure 3(b), is to move the packet priority from the VLAN tag into DSCP. As we can see, the change is small and only touches the data packet format. The PFC pause frame format stays the same. With DSCP-based PFC, data packets no longer need to carry the VLAN tag, which solves both of the problems mentioned earlier. The server facing ports no longer need to be in trunk mode, which means that PXE boot works without any issues. Also, the packet priority information, in form of DSCP value, is correctly propagated by IP routing across subnets.

Of course, DSCP-based PFC does not work for the designs that need to stay in layer-2, e.g., Fibre Channel over Ethernet (FCoE). This is not a problem for us since we do not have any layer-2 networks in our data centers.

DSCP-based PFC requires both NICs and switches to classify and queue packets based on the DSCP value instead of the VLAN tag. In addition, the NIC needs to send out data packets with the right DSCP value. Fortunately, the switch and NIC ASICs are flexible enough to implement this. Internally, at each port, the switch or NIC maintains eight Priority Groups (PGs), with each PG can be configured as lossless or lossy. If a PG i ($i \in [0, 7]$) is configured as lossless, once its ingress buffer occupation exceeds the XOFF threshold, pause frame with priority i will be generated. The mapping between DSCP values and PFC priorities can be flexible and can even be many-to-one. In our implementation, we simply map DSCP value i to PFC priority i.

Our DSCP-based PFC specification is publicly available, and is supported by all major vendors (Arista Networks, Broadcom, Cisco, Dell, Intel, Juniper, Mellanox, etc.). We believe DSCP-based PFC provides a simpler and more scalable solution than the original VLAN-based design for IP networks.

4. THE SAFETY CHALLENGES

Use of PFC and RDMA transport lead to several safety challenges. We now describe these challenges and the solutions we devised to address them.

4.1 RDMA transport livelock

RDMA transport protocol is designed around the assumption that packets are not dropped due to network congestion. This is achieved by PFC in RoCEv2. However, packet losses can still happen for various other reasons, including FCS errors, and bugs in switch hardware and software [21]. Ideally, we want RDMA performance to degrade gracefully in presence of such errors.

Unfortunately, we found that the performance of RDMA degraded drastically even with a very low packet loss rate. We illustrate this with the following simple experiment. We connected two servers A and B, via a single switch (W), and carried out three experiments for RDMA SEND, WRITE, and READ. In the first experiment, A used RDMA SENDs to send messages of size 4MB each to B as fast as possible. The second experiment was similar, except A used RDMA WRITE. In the third experiment B used RDMA READ to read 4MB chunks from A as fast as possible. The switch was configured to drop any packet with the least significant byte of IP ID equals to 0xff. Since our NIC hardware generates IP IDs sequentially, the packet drop rate was 1/256 (0.4%).

We found that even with this low packet drop rate, the application level goodput was zero. In other words, the system was in a state of livelock – the link was fully utilized with line rate, yet the application was not making any progress.

The root cause of this problem was the go-back-0 algorithm used for loss recovery by the RDMA transport. Suppose A is sending a message to B. The message is segmented into packets $0, 1, \cdots, i, \cdots, m$. Suppose packet i is dropped. B then sends an $NAK(i)$ to A. After A receives the NAK, it will restart sending the message from packet 0. This go-back-0 approach caused live-lock. A 4MB message is segmented into 4000 packets. Since the packet drop rate is a deterministic 1/256, one packet of the first 256 packets will be dropped. Then the sender will restart from the first packet, again and again, without making any progress.

Note that TCP and RDMA transport make different assumptions on the network. TCP assumes a best-effort network, in which packets can be dropped. Thus, TCP stacks incorporate sophisticated retransmission schemes such as SACK [24] to deal with packet drops. On the other hand, RDMA assumes a lossless network, hence our vendor chose to implement a simple go-back-0 approach. In go-back-0, the sender does not need to keep any state for retransmission.

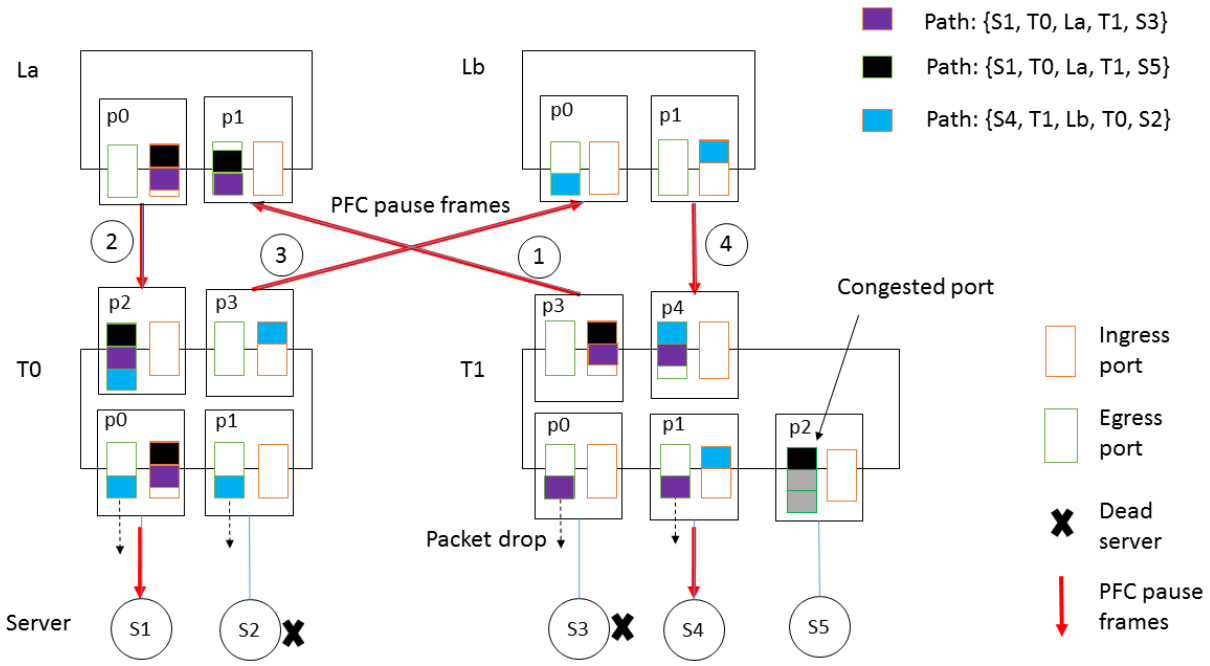


Figure 4: An example to show that the interaction between Ethernet packet flooding and PFC pause frame propagation can cause deadlock.

This experiment clearly shows that for large network like ours, where packet losses can still occur despite enabling PFC, a more sophisticated retransmission scheme is needed. Recall however, that the RDMA transport is implemented in the NIC. The resource limitation of the NIC we use meant that we could not implement a complex retransmission scheme like SACK. SACK would also be overkill, as packet drops due to network congestion have been eliminated by PFC.

Our solution is to replace the go-back-0 with a go-back-N scheme. In go-back-N, retransmission starts from the first dropped packet and the previous received packets are not retransmitted. Go-back-N is not ideal as up to $RTT \times C$ bytes , where C is the link capacity, can be wasted for a single packet drop. But go-back-N is almost as simple as go-back-0, and it avoids livelock. We worked with our NIC provider to implement the go-back-N scheme, and since doing that, we have not observed livelock in our network. We recommend that the RDMA transport should implement go-back-N and should not implement go-back-0.

4.2 PFC Deadlock

We once believed that our network is deadlock-free because of its Clos network topology and up-down routing [1, 3, 19]. In such a topology, when a source server sends a packet to a destination server, the packets first climb up to one of the common ancestors of the source and the destination, then go down the path to the destination. Hence there is no cyclic buffer dependency. But to our surprise, we did run into PFC deadlock when we ran a stress test in one of our test clusters.

As we will see later, this occurred because the unexpected interaction between PFC and Ethernet packet flooding broke the up-down routing.

Before diving into the details of this example, let's briefly review how a ToR switch forwards an IP packet to a server. Typically servers connected to the same ToR are in the same IP subnet. This subnet is then advertised to the rest of the network, so the rest of the network can forward packets to the ToR switch. Once the ToR receives an IP packet which belongs to one of its servers, it needs to query two tables. One is the ARP table from which the ToR switch can figure out the MAC address of the destination server. The second is the MAC address table from which the ToR switch can figure out with which physical port the MAC address is associated. The ARP table is for layer-3 IP whereas the MAC address table is for layer-2. The ARP table is maintained by the ARP protocol. The switch watches which packet comes from which port to establish the MAC address table.

Both tables use timeout to retire outdated entries. The typical timeout values for the ARP and MAC tables are very different: 4 hours and 5 minutes, respectively. The reason for using such disparate timeout values is that the overhead of refreshing the entries in the two tables is very different. The MAC table is automati-

cally refreshed by hardware as new packets are received, while the ARP table is updated by ARP packets, which are handled by the switch CPU. Hence the ARP table maintenance is more costly and thus the ARP table has a much longer timeout value. Such disparate timeout values can lead to an "incomplete" ARP entry – i.e. a MAC address is present in the ARP table, but there is no entry in the MAC address table for that MAC address. When a packet destined to such a MAC address arrives, the switch cannot figure out to which port to forward the packet. The standard behavior in this case is for the switch to *flood* the packet to all its ports.

Below let's use a simplified example as shown in Figure 4 to illustrate how the deadlock happens. We assume all the packets in the example are lossless packets protected by PFC.

1. Server S1 is sending packets to S3 and S5 via path {T0, La, T1}. The purple packets are to S3 and the black packets to S5. S3 is dead, so the purple packets received at port T1.p3 are flooded to the rest ports of T1 including p4. The egress queue of T1.p4 will drop the purple packets once they are at the head of the queue since the destination MAC does not match. But before that, these purple packets are queued there. Also T1.p2 is congested due to incast traffic from S1 and other sources. Hence the black packets are queued in T1. As a result, the ingress port of T1.p3 begins to pause the egress port of La.p1.

2. Consequently, as the black and purple packets build up queues in La, the ingress port of La.p0 begins to pause the egress port of T0.p2. For the same reason, T0.p0's ingress port begins to pause S1.

3. Server S4 begins to send blue packets to S2 via path {T1, Lb, T0}. S2, unfortunately, is also dead. Port T0.p3 then floods the blue packets to the rest ports including T0.p2. Since all packets, including the blue packets, at the egress port of T0.p2 cannot be drained, the ingress port of T0.p3 begins to pause Lb.p0.

4. As a result, the ingress port of Lb.p1 begins to pause T1.p4, and T1.p1 begins to pause S4.

Note that T1.p3 will continue to pause La.p1 even if the black packets leave T1 to S5 after the congestion at T1.p2 is gone. This is because the purple packets cannot be drained as T1.p4 is paused by Lb. A PFC pause frame loop among the four switches is then formed. A deadlock therefore occurs. Once the deadlock occurs, it does not go away even if we restart all the servers.

This deadlock is a concrete example of the well-known cyclic buffer dependency (see [12, 18, 22, 36] and references therein). The cause of the cyclic dependency, however, is 'new'. It is caused by the flooding behavior of the switch. In an Ethernet switch, once the destination MAC address of a packet is unknown, the packet is flooded to all the ports except the receiving port. This 'legitimate' behavior causes the dependency circle to be formed as we have shown in the above example.

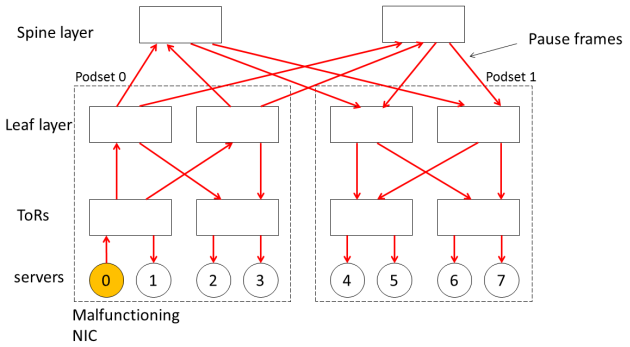


Figure 5: The PFC pause frame storm caused by the malfunctioning NIC of one single server (server 0).

We need to stop flooding for lossless packets to prevent deadlock from happening. There are several options for us to choose when an ARP entry becomes incomplete (i.e., the IP address to MAC address mapping is there, but the MAC address to port number mapping is not). (1) We forward the packets to the switch CPU and let the switch CPU figure out what to do. (2) We set up the timeout value of the MAC table to be longer than that of the ARP table, so that an ARP entry cannot be incomplete. (3) We drop the lossless packets if their corresponding ARP entry is incomplete.

We have chosen option (3). Option (1) may increase the switch CPU overhead. Option (2) needs to either reduce the ARP table timeout value or increase the MAC address table timeout value. If we reduce the ARP table timeout value, we increase the switch CPU overhead for ARP handling. If we increase the MAC address table timeout value, we need longer time to tell when a server becomes disconnected from the switch. Option (3) is a better way to prevent deadlock as it directly prevents the cyclic buffer dependency.

The lesson we have learned from the PFC deadlock is that broadcast and multicast are dangerous for a lossless network. To prevent deadlock from happening, we recommend that broadcast and multicast packets should not be put into lossless classes.

4.3 NIC PFC pause frame storm

PFC pause frames prevent packets from been dropped by pausing the upstream devices. But PFC can cause collateral damage to innocent flows due to the head-of-line blocking. We illustrate the worst-case scenario in Figure 5 [1]:

1. The malfunctioning NIC of server 0 continually sends pause frames to its ToR switch;

2. The ToR switch in turn pauses all the rest ports including all the upstream ports to the Leaf switches;

[1]The scenario involves a malfunctioning NIC.

3. The Leaf switches pause the Spine switches;
4. The Spine switches pause the rest of the Leaf switches;
5. The rest of the Leaf switches pause their ToR switches;
6. The ToR switches pause the servers that connect to them.

In this case, a single malfunctioning NIC may block the entire network from transmitting. We call this NIC PFC pause frame storm, or PFC storm for abbreviation. To our surprise, we observed PFC storms in our networks multiple times and PFC storms caused several incidents which we will describe in Section 6.

The root-cause of the PFC storm problem is a bug in the NIC's receiving pipeline. The bug stopped the NIC from handling the packets it received. As a result, the NIC's receiving buffer filled, and the NIC began to send out pause frames all the time.

We have worked with the NIC provider to fix this NIC bug. Furthermore, to prevent PFC storms from hurting the network, we have implemented two watchdogs at both the NIC and the ToR switches as follows.

On the NIC side, we worked with the NIC provider to build a PFC storm prevention watchdog. This is possible because the NIC has a separate micro-controller which can be used to monitor the NIC receiving side pipeline. Once the NIC micro-controller detects the receiving pipeline has been stopped for a period of time (default to 100ms) and the NIC is generating the pause frames, the micro-controller will disable the NIC from generating pause frames. Our experience with PFC storm is that once the NIC enters the storm mode, the server is disconnected from the network since the NIC is not functioning well anymore. The NIC watchdog is not able to rescue the server. Instead, its goal is to prevent the pause frame storms from hurting the rest of the network.

On the ToR switch side, we worked with the switch providers to build a switch watchdog to monitor the server facing ports. Once a server facing egress port is queuing packets which cannot be drained, and at the same time, the port is receiving continuous pause frames from the NIC, the switch will disable the lossless mode for the port and discard the lossless packets to and from the NIC. Similar to the NIC side watchdog, it is to prevent pause frames from the malfunctioning NIC from propagating into the network. Once the switch detects that the pause frames from the NIC disappear for a period of time (default to 200ms), it will re-enable the lossless mode for the port.

These two watchdogs are complementary to each other. One of them should be sufficient to stop the NIC PFC storm. We have implemented both for double insurance.

Note that there is a small difference in the two watchdog implementations. The switch watchdog will re-enable the lossless mode once pause frames are gone, whereas the NIC watchdog does not re-enable the lossless mode. This is because we have observed once the NIC enters the PFC storm mode, it never comes back. Hence re-enabling the lossless mode is not needed for the NIC.

We also have observed that the NIC PFC storm problem typically can be fixed by a server reboot. Hence once the NIC is not functioning, our server management system will try to repair (reboot, reimage etc.) the server. Repairing takes tens of minutes. The NIC watchdog is to limit the damage of the problematic NIC to hundreds of milliseconds before server repair kicks in. Once the server is repaired successfully and pause frames from the servers are gone, the switch can re-enable the lossless mode for the corresponding switch port automatically.

Knowledgable readers may wonder about the interactions between the two watchdogs. Once the NIC watchdog disables the NIC pause frames, the switch watchdog will re-enable the lossless mode for the corresponding switch port. The packets to the NIC will either dropped by the switch (if the MAC address of the NIC times out) or dropped by the NIC (since the NIC receiving pipeline is not functioning). In either case, the NIC PFC storm cannot cause damage to the whole network.

We recommend both switches and NICs should implement the watchdogs for NIC PFC storm prevention.

4.4 The Slow-receiver symptom

In our data centers, a server NIC is connected to a ToR switch using a point-to-point cable. The NIC is connected to the CPU and memory systems via PCIe. For a 40 GbE NIC, it uses PCIe Gen3x8 which provides 64Gb/s raw bidirectional bandwidth which is more than the 40Gb/s throughput of the NIC. Hence there seems to be no bottleneck between the switch and the server CPU and memory. We thought that the server NIC should not be able to generate PFC pause frames to the switch, because there is no congestion point at the server side.

But this was not what we have observed. We found that many servers may generate up to thousands of PFC pause frames per second. Since RDMA packets do not need the server CPU for processing, the bottleneck must be in the NIC. It turned out that this is indeed the case. The NIC has limited memory resources, hence it puts most of the data structures including QPC (Queue Pair Context) and WQE (Work Queue Element) in the main memory of the server. The NIC only caches a small number of entries in its own memory. The NIC has a Memory Translation Table (MTT) which translates the virtual memory to the physical memory. The MTT has only 2K entries. For 4KB page size, 2K MTT entries can only handle 8MB memory.

If the virtual address in a WQE is not mapped in the MTT, it results in a cache miss, and the NIC has to replace some old entries for the new virtual address. The NIC has to access the main memory of the server to get the entry for the new virtual address. All those operations take time and the receiving pipeline has to

wait. The MTT cache miss will therefore slow down the packet processing pipeline. Once the receiving pipeline is slowed down and the receiving buffer occupation exceeds the PFC threshold, the NIC has to generate PFC pause frames to the switch.

We call this phenomenon the slow-receiver symptom. Though its damage is not as severe as the NIC PFC storm, it may still cause the pause frames to propagate into the network and cause collateral damage.

The slow-receiver symptom is a 'soft' bug caused by the NIC design. We took two actions to mitigate it. On the NIC side, we used a large page size of 2MB instead of 4KB. With a large page size, the MTT entry miss becomes less frequent. On the switch side, we enabled dynamic buffer sharing among different switch ports. Compared with static buffer reservation, dynamic buffer sharing statistically gives RDMA traffic more buffers. Hence even if the NICs are pausing the switch ports from time to time, the switches can absorb additional queue buildup locally without propagating the pause frames back into the network. Compared with static buffer allocation, our experience showed that dynamic buffer sharing helps reduce PFC pause frame propagation and improve bandwidth utilization.

5. RDMA IN PRODUCTION

We added new management and monitoring capabilities to debug the various RDMA and PFC safety issues described in Section 4, and to detect RDMA related bugs and incidents. We now discuss these new capabilities which include the RDMA/PFC configuration monitoring, the PFC pause frame and lossless traffic monitoring, and the active RDMA latency monitoring. We also present the latency and throughput measurements.

5.1 Configuration management and monitoring

To enable RDMA, we need to configure PFC at the switch side, and RDMA and PFC at the server side. At the switch side, the PFC configuration is part of the QoS configuration. The PFC configuration has a global part which reserves buffer size, classifies packets into different traffic classes based on the DSCP value, maps different traffic classes into different queues, and assigns different bandwidth reservations for different queues. The PFC configuration also has a per port part which enables PFC for every individual physical port.

At the server side, there are configurations to enable/disable RoCEv2, PFC configuration, DCQCN configuration, and traffic configuration. In traffic configuration, users specify which type of traffic they would like to put into PFC protection. The specification is based on the destination transport port which is similar to the TCP destination port.

We have a configuration monitoring service to check if the running configurations of the switches and the servers are the same as their desired configurations. Our RDMA management and monitoring service handles the complexities introduced by the combinations of multiple switch types, multiple switch and NIC firmware versions, and different configuration requirements for different customers.

5.2 PFC pause frame and traffic monitoring

Besides configuration monitoring, we have also built monitoring for the PFC pause frames and the two RDMA traffic classes. For pause frame, we monitor the number of pause frames been sent and received by the switches and servers. We further monitor the pause intervals at the server side. Compared with the number of pause frames, pause intervals can reveal the severity of the congestion in the network more accurately. Pause intervals, unfortunately, are not available for the switches we currently use. We have raised the PFC pause interval monitoring requirement to the switching ASIC providers for their future ASICs.

For RDMA traffic monitoring, we collect packets and bytes been sent and received per port per priority, packet drops at the ingress ports, and packet drops at the egress queues. The traffic counters can help us understand the RDMA traffic pattern and trend. The drop counters help us detect if there is anything wrong for the RDMA traffic: normally no RDMA packets should be dropped.

5.3 RDMA Pingmesh

We have developed an active latency measurement service for RDMA similar to the TCP Pingmesh service [21]. We let the servers ping each other using RDMA and call the measurement system RDMA Pingmesh. RDMA Pingmesh launches RDMA probes, with payload size 512 bytes, to the servers at different locations (ToR, Podset, Data center) and logs the measured RTT (if probes succeed) or error code (if probes fail).

From the measured RTT of RDMA Pingmesh, we can infer if RDMA is working well or not.

Our RDMA management and monitoring took a pragmatic approach by focusing on configurations, counters, and end-to-end latency. We expect this approach works well for the future 100G or higher speed networks. RDMA poses challenges for packet-level monitoring due to the high network speed and NIC offloading, which we plan to address in our next step.

5.4 RDMA Performance

In what follows, we present the RDMA performance results in both testbed and production networks.

Latency reduction: Figure 6 shows the end-to-end latency comparison of TCP and RDMA for a highly-reliable, latency-sensitive online service. This service has multiple instances in Microsoft global data centers and it has 20K servers in each data center. The measurements are from one of the data centers. At the time of measurement, half of the traffic was TCP and

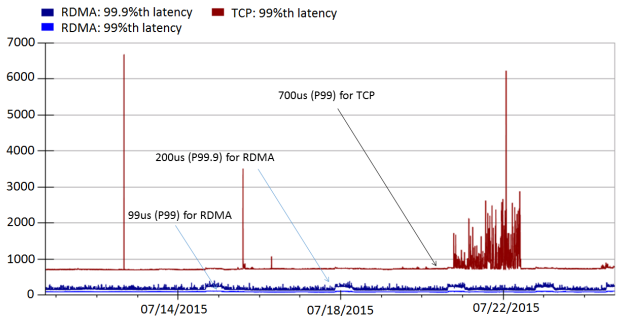
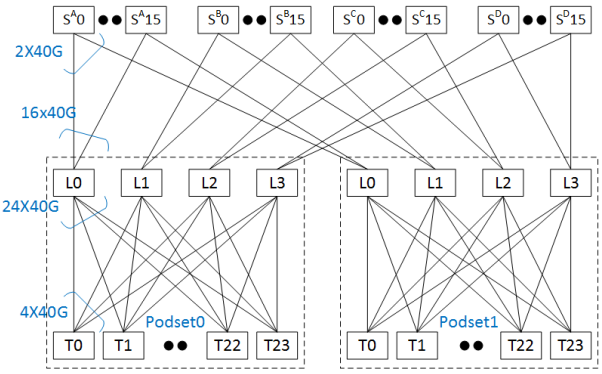
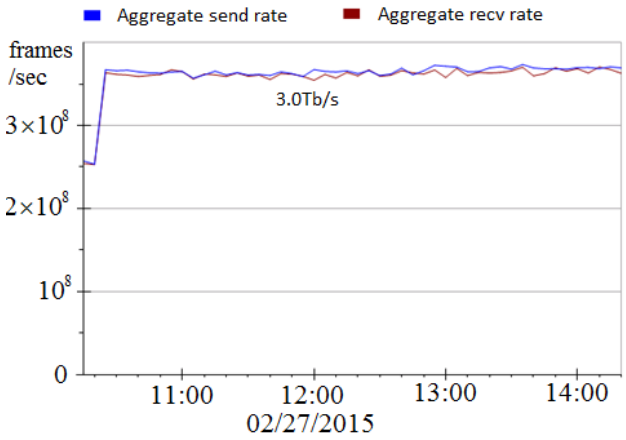


Figure 6: The comparison of the measured TCP and RDMA latencies for a latency-sensitive service.



(a) The network topology.



(b) The aggregate RDMA throughput.

Figure 7: The aggregate RDMA throughput in a three-layer Clos network. The y-axis shows the number of frames/second. A frame size is 1086 bytes.

half of the traffic was RDMA. The RDMA and TCP latencies were all measured by Pingmesh. The latency measurements for both TCP and RDMA were for intra-DC communications. Since the online service is latency sensitive, the peak traffic volume per sever was around 350Mb/s, and the aggregate server CPU load of the service was around 20% - 30% during the measurement. The network capacity between any two servers in this data center is several Gb/s. The network was not the bottleneck, but the traffic was bursty with the typical many-to-one incast traffic pattern.

As we can see, the 99^{th} percentile latencies for RDMA and TCP were 90us and 700us, respectively. The 99^{th} percentile latency for TCP had spikes as high as several milliseconds. In fact, even the 99.9^{th} latency of RDMA was only around 200us, and much smaller than TCP's 99^{th} percentile latency. Although the network was not the bottleneck, TCP's latency was high at the 99^{th} percentile. This is caused by the kernel stack overhead and occasional incast packet drops in the network. Although RDMA did not change the incast traffic pattern, it eliminated packet drops and kernel stack overhead. Hence it achieved much smaller and smoother high percentile latency than TCP.

Throughput: The following experiment shows the RDMA performance with hundreds of servers in a three-tier Clos network. We ran this experiment using two podsets after a data center was online but before it was handed to the customer – i.e. there is no customer traffic during the experiment.

The network topology is shown in Figure 7(a). All the ports in the network are 40GbE. A podset is composed of 4 Leaf switches, 24 ToR switches, and 576 servers. Each ToR switch connects 24 servers. The 4 Leaf switches connect to a total of 64 Spine switches. The oversubscription ratios at the ToR and the Leaf are 6:1 and 3:2, respectively. The aggregate bandwidth between a podset and the Spine switch layer is 64x40Gb/s = 2.56Tb/s.

We used a ToR-to-ToR traffic pattern. We paired the ToRs in the two podsets one by one. ToR i in podset 0 was paired with ToR i in podset 1. In each ToR, we selected 8 servers, and let each server establish 8 RDMA connections to the corresponding server in the other ToR. All these RDMA connections needed to traverse the Leaf-Spine links. All the RDMA connections sent data as fast as possible. In total we had 3074 connections distributed among the 128 Leaf-Spine links, which were the bottlenecks in this experiment.

Figure 7(b) shows the aggregate throughput measured from the servers. The unit of the y-axis is frames per second. The RDMA frame size is 1086 bytes with 1024 bytes as payload. The aggregate throughput is 3.0Tb/s. This is 60% network utilization of the total 5.12Tb/s network capacity. During the whole experiment, not a single packet was dropped. Every server was sending and receiving at 8Gb/s with the CPU utilization close to 0%.

Since we use ECMP for multi-path routing in our network, 60% utilization is what we can achieve for this experiment. This 60% limitation is caused by ECMP hash collision, not PFC or HOL blocking. Both our simulation and the results in [2], in which no PFC was used, showed similar utilization numbers for ECMP routing in three-tier Clos networks.

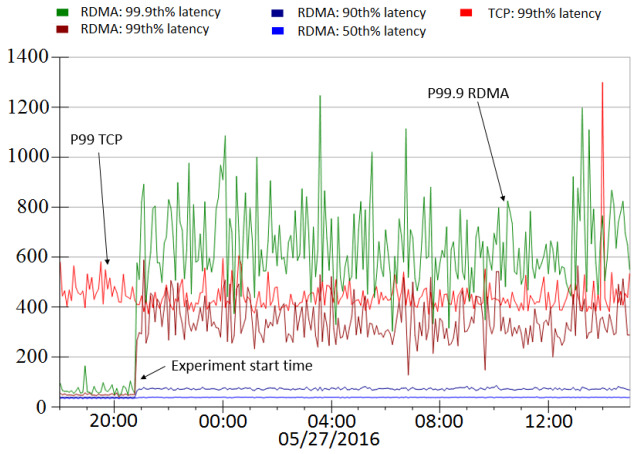


Figure 8: The end-to-end RDMA latency jumped up as the experiment started and network throughput increased.

We unfortunately did not record the end-to-end RDMA latency in the above throughput experiment. To further investigate the relationship between network latency and throughput, we conducted the following experiment in our testbed with a two-tier network. We had two ToR switches in this testbed. Each ToR switch had 24 servers, and each ToR used 4 uplinks to connect to four Leaf switches. All the links were 40GbE. The oversubscription ratio was 6:1. We mimicked The traffic pattern in Figure 7. We chose 20 servers in every ToR and paired every server in one ToR with one server in another ToR and let every server-pair establish 8 RDMA connections. Every server achieved 7Gb/s sending/receiving throughput. We show the RDMA latency measured in Pingmesh in Figure 8. Once the experiment started, the end-to-end RDMA latencies increased from 50us at the 99^{th} percentile and 80us at the 99.9^{th} percentile to 400us and 800us, respectively.

Note that the 99^{th} percentile latency of TCP did not change during the experiment in Figure 8. This is because we put RDMA and TCP packets into two different queues in the switches. Hence RDMA and TCP did not interfere with each other. We note that the 99^{th} percentile latency of TCP was 500us in Figure 8, whereas it was 700us in Figure 6. The difference was caused by the fact that the servers in Figure 6 were servicing real-world workload whereas the servers in Figure 8 were almost idle (except running the RDMA traffic generator). Figure 8 also demonstrated that the RDMA latency increase was due to the network congestion created by the RDMA traffic.

The above measurement results show that, compared to TCP, RDMA achieves low latency and high throughput by bypassing the OS kernel and by eliminating packet drops. But RDMA is not a panacea for achieving *both* low latency and high throughput. The RDMA latency can still increase as the network becomes congested and queues build up.

6. EXPERIENCES

6.1 RDMA Deployment

RoCEv2 was a new technology to us when we began this work three years ago. We were unaware of any large-scale RoCEv2 deployment at that time. Though the benefits (zero packet drops, low latency, and low CPU overhead) were attractive, we were concerned about the maturity of RoCEv2. We devised a step-by-step procedure to onboard RDMA.

For the first step, we built a small lab network with tens of servers. This step helped us eliminate most of the bugs at early stage. In the second step, we used test clusters to improve the maturity of RoCEv2. The test clusters' setup and management were the same as their production counterparts. In the third step, we enabled RDMA in production networks at ToR level only. In the fourth step, we enabled PFC at the Podset level, i.e., we enabled PFC in the ToR and Leaf switches within the Podsets. In the last step, we enabled PFC up to the Spine switches. In every step when we carried out deployment in production, we followed our safe deployment procedure to enable RDMA through several phases in our global data centers.

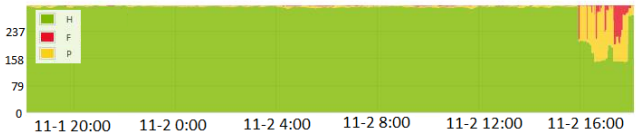
This step-by-step procedure turned out to be effective in improving the maturity of RoCEv2. The RDMA transport livelock and most of the bugs were detected in lab tests. The PFC deadlock and slow-receiver symptom were detected in the test clusters. Only the NIC PFC pause frame storm and a few other bugs hit our production networks.

Using the same management and monitoring for both the test clusters and the production networks turned out to be invaluable. It made our life easier as the test clusters were always well managed. At the same time, it let us thoroughly test RoCEv2 as well as the management and monitoring systems before RoCEv2 went into production.

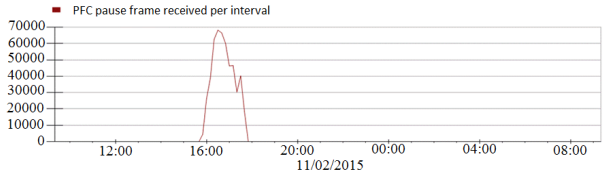
6.2 Incidents

NIC PFC storm. The following is one of the few NIC PFC storm incidents we have encountered. In this incident, one of our customers experienced a service availability issue. Many of their servers became unavailable as shown in Figure 9(a). At the same time, we observed that many of the servers were continuously receiving large number of PFC pause frames as shown by our monitoring system in Figure 9(b). The y-axis shows the number of PFC pause frames sent/received in every five minutes.

We were able to trace down the origin of the PFC pause frames to a single server. That server was unresponsive and was in Failing (F) state as detected by our data center management system. But from the connected ToR switch, we could observe the number of pause frames from the server was always increasing, at more than two thousands pause frames per second.



(a) Server availability reduction. H (healthy), F (failing), and P (probation) are server states.



(b) The PFC pause frames received by the servers.

Figure 9: An incident caused by the NIC PFC storm problem of a single server.

We also observed that the server was not sending or receiving any data packets. After we power-cycled that server, the server came back up and the pause frames were gone.

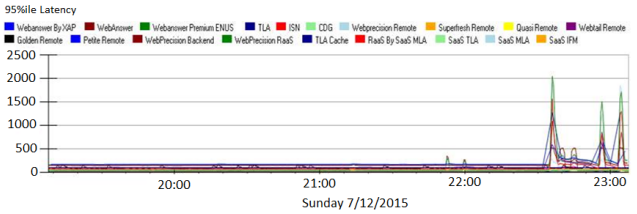
NIC PFC storms happened very infrequently. With hundreds of thousands of servers in production, the number of the NIC PFC storm events we have experienced is still single digit. Nonetheless, once NIC PFC storm happens, the damage is huge due to the PFC pause frame propagation. As we can see from this incident, half of our customers servers were affected and put into non healthy state.

After we put the NIC PFC storm prevention watchdogs at both the servers and the ToR switches, we did not experience NIC PFC storms anymore.

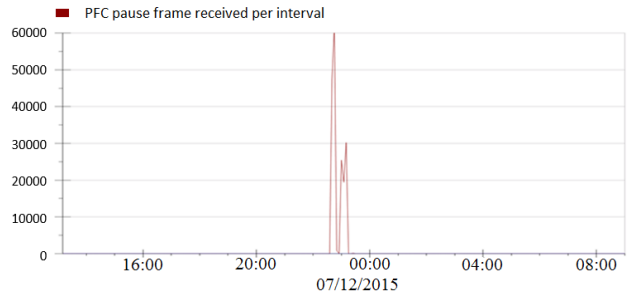
Switch buffer misconfiguration. The ToR and Leaf switches we use have a small and limited buffer size of 9MB or 12MB. To better utilize the scarce buffer space, we need to enable dynamic buffer sharing. In dynamic buffer sharing, the ports allocate memory from a shared buffer pool. The shared buffer allocation per port per traffic class is controlled by a parameter called α. As long as $\alpha \times UB > B_{p,i}$, where UB is the unallocated shared buffer size and $B_{p,i}$ is the allocated buffer size for traffic class i of ingress port p, we can allocate memory from the shared buffer for traffic class i from ingress port p. Hence a large α can help reduce the chance of PFC pause frames from been generated. But a large α may cause imbalanced and unfair buffer allocation.

We have found that the default α value ($\alpha = \frac{1}{16}$) for a type of ToR switch worked well in our production network. When we onboarded a new type of switch from the same switch provider, we took it for granted that it would use the same default settings as before.

Then in the midnight of 07/12/2015, we ran into an incident. As shown in Figure 10(a), the latencies of many latency-sensitive services increased dramatically. Also we have observed that many servers were receiving



(a) Services latency increase caused by the PFC pause frame propagation. Every color here represents an impacted service.



(b) The PFC pause frames received by the servers.

Figure 10: An incident caused by the buffer misconfiguration of a newly introduced switch type.

a large number of PFC pause frames, up to 60000 pause frames in 5 minutes (Figure 10(b)).

Further analysis revealed the origins of the pause frames. The pause frames were generated by two ToR switches, then propagated into the rest of the network, and affected thousands of servers.

Why there were so many pause frames been generated? There were two reasons. The first was the incast traffic pattern. These two ToR switches hosted many chatty servers, which sent queries to more than one thousand servers simultaneously. Once the responses came back to the chatty servers, incast happened, which created network congestion condition for PFC pause frame generation.

The second reason was that we found the α value of the new type of ToR switch was changed to $\frac{1}{64}$, though these two types of switches were from the same provider. A much smaller α made the dynamic buffer allocated to the congested ingress ports much smaller. Hence the PFC pause frames could be triggered much more easily.

We could not change the traffic pattern, so we tuned the α value back to $\frac{1}{16}$ for these switches.

The lesson we learned from this incident is that PFC pause frames did propagate and cause collateral damage in our production network. To reduce the damage, we need to reduce PFC pause frames from being generated. Our work on the NIC PFC storm and the slow-receiver symptom prevent servers from been generating pauses. Moreover, parameter tuning of the dynamic buffer sharing and the per-flow based DCQCN [42] congestion control reduce the pauses generated by the switches.

6.3 Lessons learned and discussion

During the three years period of designing, building, and deploying RoCEv2, we have learned several lessons which we share as follows.

Deadlock, livelock, and PFC pause frames propagation did happen. The PFC deadlock we met was a surprise to us, as we once believed that our Clos-based network topology was free of cyclic buffer dependency hence free of deadlock. We did not expect the slow-server symptom, though we were fully aware that PFC backpressure can cause PFC pause frame propagation in the network. We did not foresee the RDMA transport livelock either. The lesson we learned is that a design works in theory is not enough, as there may be many hidden details which invalidate the design. We have to use well designed experiments, test clusters, and staged production deployments, to verify the designs and to unveil the unexpected facets methodologically.

NICs are the key to make RDMA/RoCEv2 work. Most of the RDMA/RoCEv2 bugs we ran into were caused by the NICs instead of the switches. We spent much more time on the NICs than on the switches. In hindsight, this happened for two reasons. The first reason is because the NIC implements the most complicated parts of the RDMA functionalities, including the RDMA verbs and the RDMA transport protocol. As a comparison, the switch side functionalities are relatively simple (e.g., PFC implementation) or well tested (e.g., ECN implementation). The second reason is that the NICs we use are resource constrained. The NIC leverages the server's DRAM to store its data structures and uses its own local memory as the cache. Cache management then becomes a big part of the NIC and introduces bugs as well as performance bottlenecks, e.g., the slow-receiver symptom.

Be prepared for the unexpected. Our experiences of running one of the largest data center networks in the world taught us that network incidents happen. From day one when we began to work on RoCEv2, we put RDMA/RoCEv2 management and monitoring as an indispensable part of the project. We upgraded our management and monitoring system for RDMA status monitoring and incidents handling at the same time when we worked on the DSCP-based PFC design and the safety and performance bugs. As a result, when our customers began to use RDMA, the RDMA management and monitoring capabilities were already in production. This RDMA management and monitoring system is essential for RDMA health tracking and incident troubleshooting. It helped us detect, localize, and root-cause the RoCEv2 bugs and incidents as we have shown in Sections 6.2 and 4.

Is lossless needed? RoCEv2 depends on a lossless network to function well. In this work, we have demonstrated that we indeed can build a lossless network using PFC, and all the real-world scalability and safety challenges can be addressed. Looking forward into the future, the question we would like to ask is: do we really need a lossless network to get the benefits of RoCEv2? Given the progress on programmable hardware, e.g., FPGA and FPGA integrated CPU [8], it may become feasible and economical to build much faster and more advanced transport protocols and forward error correction algorithms directly in commodity hardware, hence relieving RoCEv2 from been depending on lossless network.

7. RELATED WORK

This paper focuses on how to safely deploy RoCEv2 on a large-scale. Besides RoCEv2, there are two other RDMA technologies: Infiniband [6] and iWarp [30].

Infiniband is a complete networking protocol suite, which has its own layer-1 to layer-7 protocols. An Infiniband network is composed of multiple Infiniband subnets which are connected via Infiniband routers. Within a subnet, servers are connected via Infiniband switches. However, to the best of our knowledge, there are still no Infiniband routers in production. Hence Infiniband does not meet our scalability requirement. Furthermore, Infiniband is not compatible with Ethernet, which is the dominant networking technology for data centers.

iWarp runs RDMA over TCP/IP. The TCP/IP protocol is offloaded to the NIC. Since TCP guarantees reliable delivery even if some of the packets are dropped, iWarp does not necessarily need a lossless network. iWarp has one advantage over RoCE in that it can be use for inter-DC communications. But since iWarp uses TCP for packet transmission, it faces the same issue of TCP: long latency caused by packet drops and retransmission timeout. As we have discussed in 6.3, we expect new transport protocols different from the Infiniband transport and TCP to be introduced in the future driven by new hardware innovations.

Deadlock is well studied in the literature and it is well known cyclic buffer dependency is necessary for deadlock [12, 18, 22, 33, 36]. Due to the specific Clos network topology, we once thought our network should be free from deadlock since it should be free of cyclic buffer dependency. But the 'conspiracy' of PFC and Ethernet packet flooding has shown that deadlock can happen in Clos networks.

TCP performance issues such as TCP incast [35, 38, 39] and long latency tail [41] have been studied extensively. These solutions are still within the existing TCP framework. They either tune the retransmission timer (as in [35]), or control the TCP receiving window ([39]), or tune the ECN parameter ([38]). RDMA provides a different approach.Compared to [41] which still uses TCP, RDMA bypasses the OS kernel, so that the latency introduced by the kernel is eliminated. Our work shows that RDMA can be safely deployed at scale for intra-DC communications. As we have shown in Figure 5.4, RDMA greatly reduces the high percentile latency compared with TCP.

RDMA has been used to build systems including storage, key-value stores, and distributed transaction systems [17, 26, 28, 37]. Most of these systems use Infiniband or RoCE with tens of servers. In this paper, we have shown that we can scale RDMA to much larger networks using RoCEv2. Hence much larger in-memory systems can be built in the future.

8. CONCLUSION

In this paper, we have presented our practices and experiences in deploying RoCEv2 safely at large-scale in Microsoft data centers. Our practices include the introducing of DSCP-based PFC which scales RoCEv2 from layer-2 VLAN to layer-3 IP and the step-by-step onboarding and deployment procedure. Our experiences include the discoveries and resolutions of the RDMA transport livelock, the RDMA deadlock, the NIC PFC storm and the slow-receiver symptom. With the RDMA management and monitoring in place, some of our highly-reliable, latency-sensitive services have been running RDMA for over one and half years.

8.1 Future Work

There are several directions for our next steps. The hop-by-hop distance for PFC is limited to 300 meters. Hence RoCEv2 works only for servers under the same Spine switch layer. To this end, RoCEv2 is not as generic as TCP. We need new ideas on how to extend RDMA for inter-DC communications.

Our measurement showed ECMP achieves only 60% network utilization. For TCP in best-effort networks, there are MPTCP [29] and per-packet routing [10] for better network utilization. How to make these designs work for RDMA in the lossless network context will be an interesting challenge.

The deadlock we have discovered in this paper reminds us that deadlock in data centers may be worthy of more systematic study. Even though the up-down routing in Clos network can prevent deadlock, designs like F10 [23] may break the assumption by introducing local rerouting. Many other network topologies [20] do not even have the up-down routing property. How can deadlocks be avoided in those designs?

Last but not least, we have shown that RDMA provides low latency and high throughput by eliminating OS kernel packet processing overhead and by relying on a lossless network. A lossless network, however, does not guarantee low latency. When network congestions occur, queues build up and PFC pause frames may still be generated. Both queues and PFC pause frames increase network latency. How to achieve low network latency and high network throughput at the same time for RDMA is still an open problem.

9. ACKNOWLEDGEMENTS

Yan Cai contributed to the design and experimentations of DSCP-based PFC. Gang Cheng and Daniel Firestone worked on the RDMA project at its early stage. Yibo Zhu helped us on the RDMA transport livelock experiments. We were supported by many members of Azure Networking. We have worked closely with our partners Charlie Gu, Sorabh Hamirwasia, Madhav Pandya, Passaree Pasarj, Veselin Petrov, Xin Qian, Junhua Wang, Chenyu Yan to onboard RDMA for several key online services in Microsoft. We received technical support from the engineers of Arista Networks, Broadcom, Cisco, Dell, and Mellanox. Our shepherd Nandita Dukkipati and the anonymous SIGCOMM reviewers gave us many constructive feedback and comments which improved the content and presentation of this paper. We thank them all.

10. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. SIGCOMM*, 2008.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI*, 2010.

[3] Alexey Andreyev. Introducing Data Center Fabric, The Next-generation Facebook Data Center Network. https://code.facebook.com/posts/360346274145943/, Nov 2014.

[4] Hadoop. http://hadoop.apache.org/.

[5] Infiniband Trade Association. RoCEv2. https://cw.infinibandta.org/document/dl/7781, September 2014.

[6] Infiniband Trade Association. InfiniBandTM Architecture Specification Volume 1 Release 1.3, March 2015.

[7] Luiz Barroso, Jeffrey Dean, and Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, March-April 2003.

[8] Diane Bryant. Disrupting the Data Center to Create the Digital Services Economy. https://communities.intel.com/community/itpeernetwork/datastack/blog/2014/06/18/disrupting-the-data-center-to-create-the-digital-services-economy.

[9] Brad Calder et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *SOSP*, 2011.

[10] Jiaxin Cao et al. Per-packet Load-balanced, Low-Latency Routing for Clos-based Data Center Networks. In *ACM CoNEXT*, 2013.

[11] Cisco. Cisco Nexus 3232C Switch Data Sheet. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-3232c-switch/datasheet-c78-734883.html.

[12] William Dally and Charles Seitz. Deadlock-Free Message Routing in Multiprocessor

Interconnection Networks. *IEEE trans. Computers*, C-36(5), 1987.

[13] IEEE DCB. 802.1Qaz - Quantized Congestion Notification. http://www.ieee802.org/1/pages/802.1au.html.

[14] IEEE DCB. 802.1Qbb - Priority-based Flow Control. http://www.ieee802.org/1/pages/802.1bb.html.

[15] Jeffrey Dean and Luiz André Barroso. The Tail at Scale. *CACM*, Februry 2013.

[16] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[17] Aleksandar Dragojević et al. No compromises: distributed transactions with consistency, availability, and performance. In *SOSP*, 2015.

[18] José Duato. A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks. *IEEE trans Parallel and Distributed Systems*, 7(8), 1996.

[19] Albert Greenberg et al. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, August 2009.

[20] Chuanxiong Guo et al. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[21] Chuanxiong Guo et al. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *ACM SIGCOMM*, 2015.

[22] Mark Karol, S. Jamaloddin Golestani, and David Lee. Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks. *IEEE/ACM trans Networking*, 11(6), Dec 2003.

[23] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A Fault-Tolerant Engineered Network. In *NSDI*, 2013.

[24] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options, 1996. IETF RFC 2018.

[25] Mellanox. ConnectX-4 EN Adapter Card Single/Dual-Port 100 Gigabit Ethernet Adapter. http://www.mellanox.com/page/products_dyn?product_family=204&mtag=connectx_4_en_card.

[26] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *USENIX ATC*, 2013.

[27] Radhika Mittal et al. TIMELY: RTT-based Congestion Control for the Datacenter. In *ACM SIGCOMM*, 2015.

[28] Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John Ousterhout, and Mendel Rosenblum. Fast Crash Recovery in RAMCloud. In *SOSP*, 2013.

[29] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving Datacenter Performance and Robustness with Multipath Tcp. In *SIGCOMM*, 2011.

[30] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A Remote Direct Memory Access Protocol Specification. IETF RFC 5040, October 2007.

[31] Arjun Singh et al. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM*, 2015.

[32] IEEE Computer Society. 802.1Q - 2014: Virtual Bridged Local Area Networks, 2014.

[33] Brent Stephens, Alan L. Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. Practical DCB for Improved Data Center Networks. In *Infocom*, 2014.

[34] D. Thaler and C. Hopps. Multipath Issues in Unicast and Multicast Next-Hop Selection, 2000. IETF RFC 2991.

[35] Vijay Vasudevan et al. Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication. In *SIGCOMM*, 2009.

[36] Freek Verbeek and Julien Schmaltz. A Fast and Verified Algorithm for Proving Store-and-Forward Networks Deadlock-Free. In *Proceedings of The 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'11)*, 2011.

[37] Xingda Wei, Jiaxin Shi, Yanzhe Chen, Rong Chen, and Haibo Chen. Fast In-memory Transaction Processing using RDMA and HTM. In *SOSP*, 2015.

[38] Haitao Wu et al. Tuning ECN for Data Center Networks. In *ACM CoNEXT*, 2012.

[39] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. ICTCP: Incast Congestion Control for TCP in Data Center Networks. In *ACM CoNEXT*, 2010.

[40] Matei Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*, 2012.

[41] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *SIGCOMM*, 2012.

[42] Yibo Zhu et al. Congestion Control for Large-Scale RDMA deployments. In *ACM SIGCOMM*, 2015.