2010 International Workshop on Storage Network Architecture and Parallel I/Os

XCo: Explicit Coordination for Preventing Congestion in Data Center Ethernet

Vijay Shankar Rajanna, Smit Shah, Anand Jahagirdar, and Kartik Gopalan
Computer Science, State University of New York, Binghamton, NY
Contact: kartik@binghamton.edu

***Abstract*—Cluster-based storage systems increasingly use commodity communication technologies, such as Fibre Channel over Ethernet (FCoE), for accessing stored data over the network. Data is striped over multiple storage nodes, and storage traffic often shares the network with non-storage traffic. In such conditions, storage clients can experience severely degraded performance, such as TCP throughput collapse and network congestion due to competing network traffic. Furthermore, consolidation of multiple virtual machines (VMs) onto fewer physical nodes can worsen the performance of network storage systems. The root cause of this performance problem is that network traffic from multiple sources can cause transient overloads in the switch buffers. In this paper, we make the case that virtualization opens up a new set of opportunities to alleviate and solve such performance problems experienced by network storage in particular, and data center Ethernet in general. We present an architecture, called *XCo*, for *explicit coordination* of network traffic among VMs in a data center Ethernet that is inexpensive, fully transparent, currently feasible, and complementary to any switch-level hardware support. We present experimental evidence via proof-of-concept implementation and evaluation to support this claim and describe the challenges and opportunities in a complete solution.**

I. Introduction

Considerations of cost, sustainability, and consolidation are driving the widespread deployment of network storage systems over commodity networking technologies such as Gigabit Ethernet and 10GigE. For example, iSCSI [1] allows storage clients to send SCSI commands over IP networks to storage devices on remote servers. Similarly, Fibre Channel over Ethernet (FCoE) [2] enables the use of fibre channel protocol over Gigabit and 10GigE networks. Commodity Ethernet hardware is cheap, easy to install and manage, and can be used in a shared mode with non-storage network traffic. However, this commoditization often comes at a price, such as higher latency and smaller, lower-performance packet buffers. Another related trend is CPU virtualization which leads administrators to consolidate more virtual machines (VMs) on fewer physical servers and less networking hardware. These VMs rely on the commodity networking hardware for both their storage and communication needs. Consequently, Ethernet switch buffers can become overwhelmed by high-throughput traffic that can be bursty and synchronized, leading to packet losses.

A typical network congestion scenario could be as follows. Suppose service-oriented VMs are handling peak client loads over the network. At the same time, network storage traffic, such as database access or storage backup operations are fired up over FCoE. Meanwhile, administrators may need to live-migrate multiple VMs across the network fabric [3], [4] for load-balancing. The resulting network contention across the switching fabric could easily bring down the effective network throughput (or "goodput"), increase latency for critical services and possibly cause congestion collapse.

The root cause of congestion (and hence lost packets) in a layer-2 switch is that the number of packets contending for an output port at any instant exceeds the buffering capacity of the switch for that port. Hardware and software mechanisms to control network congestion or support QoS in layer-2 switches are far from well-developed [5]. Current hardware support for QoS is limited [6] and not scalable. Cluster administrators are often reluctant to enable whatever little switch-level support that currently exists for the fear of slowing down the forwarding performance in switches. Current industry practice is to simply throw more hardware at the problem by adding higher capacity network switches, expensive multi-port network cards, and physically separate layer-2 networks for storage, communication, and control traffic. However this merely tends to increase the network cost and complexity without addressing the fundamental problem.

Data center nodes are closely coupled with extensive central monitoring and resource control. This makes certain coordination mechanisms practical in data centers that may not be feasible in a wide-area network. This paper makes the case for *explicit coordination* of network transmission activities among virtual machines (VMs) in the data center Ethernet to proactively prevent network congestion. We argue that virtualization has opened up new opportunities for explicit coordination that are simple, effective, currently feasible, and complementary to switch-level hardware support. We show that explicit coordination can be implemented transparently without modifying any applications, standard protocols, network switches, or VMs. Our solution, called *XCo*, co-ordinates the network transmissions from multiple VMs so that they do not overflow the intermediate switch buffers, while simultaneously increasing network utilization. We present experimental evidence via a proof-of-concept implementation and outline challenges and opportunities in a complete solution.

978-0-7695-4025-2/10 $26.00 © 2010 IEEE
DOI 10.1109/SNAPI.2010.17

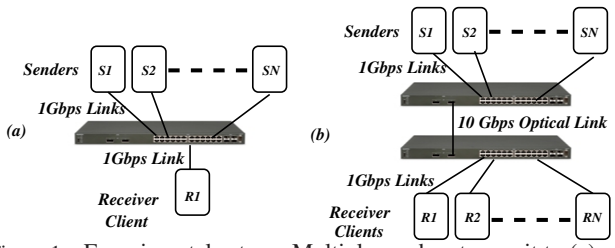

Figure 1. Experimental setups: Multiple senders transmit to (a) one receiver via 1Gbps link, (b) different receivers via 10Gbps uplink.

II. The Throughput Collapse Problem

Consider two experimental setups shown in Figure 1. End nodes running Xen [7] VMs are connected via a layer-2 switched network consisting of Nortel 4526-GTX switches, each having 24 1000Base-T ports (for end host connectivity) and two XFP slots with 10Gbps optical transceiver modules (for uplink between switches). Hosts run Xen 3.3.1/Linux 2.6.29.2. Although both setups in Figure 1 are somewhat simple, they serve to illustrate the basic traffic contention problem in larger switched Ethernet hosting networked storage.

[TCP Throughput Collapse] We first illustrate the problem of TCP throughput collapse, also known as TCP Incast Collapse, that is observed with synchronized request workloads and was documented earlier in [8], [9], [10]. This problem is observed in high-bandwidth low-delay networks where multiple servers send barrier-synchronized traffic to a single client receiver node. For example, assume that nodes in Figure 1(a) form an iSCSI storage network in which data blocks are striped across N storage servers $S1...SN$. We wrote a client application which requests striped blocks of data from server applications. If block size is B bytes then each server is requested B/N bytes of data where N is the number of servers. The client node requests such a block of data from N servers, sending the request for next block only after all servers have responded. Each server responds to the client request with the fragment of the data block that it holds. Figure 2, shows the steep collapse in observed throughput at the client as the number of servers increase from 1 to 10 for a fixed block size of 1MB.

Such barrier-synchronized requests are often observed in cluster-based applications such as parallel I/O in cluster filesystems, multiple back-end responses to search queries, or in parallel query processing in parallel distributed databases. The near-simultaneous response from multiple servers overload the buffer at the receiver's port. Since commodity switches have limited buffer space, excess packets arriving at the client port are dropped by the switch. When a packet is dropped, TCP waits for atleast RTO_{min} time before retransmitting the packet. This in turn delays the client's request for next block of data. The ratio of TCP's RTO_{min} (typically 200ms) to the time taken to transfer the data blocks is high. Consequently, synchronized packet

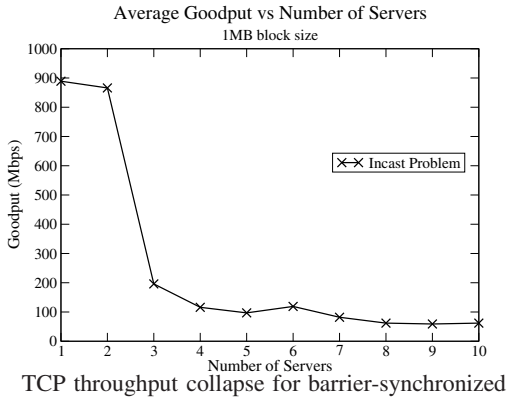

Figure 2. TCP throughput collapse for barrier-synchronized reads.

losses and timeouts result in large network idle times (akin to idle-time bubbles) leading to TCP throughput collapse.

While a number of application-specific and TCP-specific solutions have been proposed to address TCP throughput collapse problem (see Section VI), we show in Section III that our *XCo* solution can alleviate TCP throughput collapse without modifying any application-specific or TCP-specific parameters. In addition, it can also alleviate a more general congestion collapse problem due to the presence of non-TCP-friendly traffic in the network.

[Throughput collapse due to non-TCP-friendly Traffic] We now show that even in the absence of synchronized traffic, the presence of traffic that is not TCP-friendly can lead to throughput collapse in Gigabit networks. Figure 1(a) shows a setup where five Xen VMs executing on five different hosts send a mix of TCP and UDP network traffic to a common receiver VM on another host. Each sender uses the Netperf [11] benchmark to generate either TCP or UDP traffic to a Netperf server on the receiver. The plot in Figure 3 shows the received throughput as the number of UDP senders is increased and, correspondingly, number of TCP senders is decreased. All five senders transmit without any coordination. When all five senders send TCP traffic, the received throughput is more than 900Mbps since competing TCP streams reduce their congestion window (and hence sending rate) in response to packet losses. In contrast, UDP traffic does not back off in the face of packet loss. Hence, as we increase the number of UDP senders in the mix, we see an immediate drop in the received throughput due to output port contention.

The problem illustrated above is the well-known problem of *congestion collapse*, and is precisely the reason why TCP was designed. However, not all network traffic is TCP (or TCP-friendly) and not all protocols perform self-regulation the way TCP does. In fact, a growing proportion of network traffic is not TCP-friendly [12] such as streaming media, voice/video over IP, and peer-to-peer traffic. Although TCP is self-regulating, even a small amount of non-TCP-friendly traffic can disrupt fair-sharing of switched network resources.

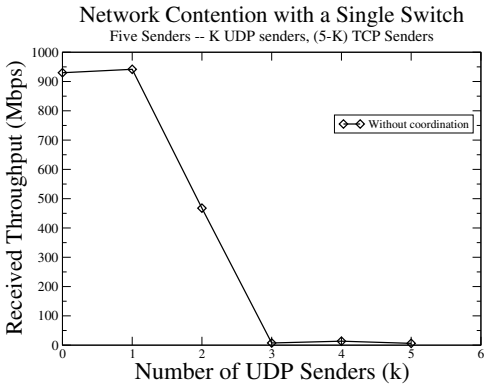

Figure 3. Throughput collapse at receiver's port in Figure 1(a).

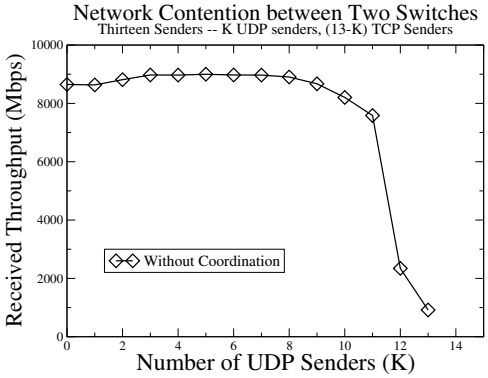

Figure 4. Throughput collapse at the 10Gbps uplink in Figure 1(b).

To illustrate further, Figure 1(b) shows another experimental setup where 13 sender VMs on different hosts send a mix of TCP and UDP Netperf traffic to 13 receiver VMs, again on separate hosts. Each sender VM is connected via a 1Gbps link to one Nortel 4526-GTX switch. The receiver VMs are similarly connected to another Nortel switch. The two switches are connected to each other by a 10GigE optical link via their XFP transceiver modules. The plot in Figure 4 shows the link utilization at the 10Gbps link as the number of UDP senders is increased. With increasing number of UDP senders (and correspondingly fewer TCP senders), there is a steep drop in the utilization of the 10Gbps link due to port contention at the first switch. *The second experiment above shows that simply replacing 1Gbps hardware (switches, links, and network cards) with their 10Gbps counterparts is not going to avoid the problem of congestion collapse.* If anything, it could worsen the problem, because one multi-core server can easily run 8 to 16 VMs in parallel with today's technology. This means that one end host alone can very quickly fill up a 10Gbps pipe originating from its NIC [13].

[Impact of Congestion on Live VM Migration Time] We now use another simple example of live VM migration [3] to understand the actual impact of throughput collapse on real operations in data center clusters. Again consider the same setup as in Figure 1(a). We initiate the live migration of a 256MB idle VM from one of the host S1 to

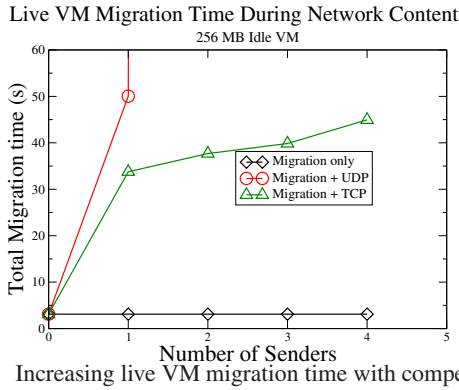

Figure 5. Increasing live VM migration time with competing traffic.

R1. Simultaneously, we initiate Netperf traffic from different number of senders S2...S5 to R1. We measure the total time taken to perform live migration as the number of Netperf senders is increased. Figure 5 plots the total migration time when all Netperf senders transmit either TCP or UDP traffic. We see that in the presence of even one or more competing Netperf UDP sender, the total migration time increases exponentially since UDP traffic tends to gobble up any available uplink bandwidth. With competing Netperf TCP senders, there is an almost ten fold increase in total migration time, though the increase is not as dramatic as with UDP. Note that even though all TCP sessions back off when network congestion is detected, it does not completely eliminate output port contention.

III. XCo: Design and Implementation

This section presents the design and implementation of the *XCo* framework for preventing congestion collapse. Although presented in the context of Xen [7] virtualization platform, the fundamental principles presented here are applicable across different virtuallization technologies.

A. Xen Network Subsystem Background

Xen exports virtualized views of network devices to each VM, as opposed to real physical network cards. The actual network drivers that interact with the real network card execute within Domain 0 – a privileged domain that can directly access all hardware in the system. The privileged Domain 0 and the unprivileged VMs communicate by means of a split network-driver architecture shown in Figure 6. Domain 0 hosts the backend of the split network driver, called *netback*, and the VM hosts the frontend, called *netfront*. All netbacks attach to a software bridge (when in bridged mode) in Domain 0, which multiplexes and demultiplexes packets from/to the VMs. Under normal operations, all outgoing packets are forwarded by the software bridge to the native driver for the network interface card (NIC). *XCo* interposes a *local coordinator* module after the software bridge to control transmissions.

B. Overview of the XCo Framework

XCo works on the principle that if one can prevent too many end hosts from sending too much data at the

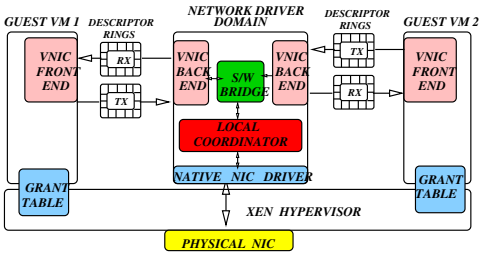

Figure 6. Xen networking subsystem in end hosts.

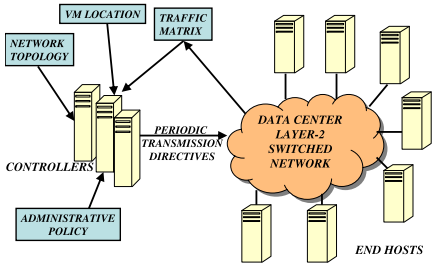

Figure 7. Explicit coordination framework for data centers.

same time then we can avoid the throughput collapse seen earlier. The cluster administrator may not have complete control over the type of traffic being generated by each VM, such as in a subscriber based cloud computing model where different external users may own individual VMs. The cluster administrator also may not have the freedom to modify the application or operating system image within each VM. *However, the cluster administrator does have control over a privileged VM, such as Domain 0, which can intercept, monitor, and control the traffic being transmitted from all other VMs in a host machine.*

Figure 6 shows a *local coordinator* in Domain 0. It is a kernel module that intercepts outgoing network traffic and enforces any form of transmission control. The specific form of transmission control is determined by one or more *central controllers* shown in Figure 7, which reside in the same switched network as the other nodes. A central controller takes as input (i) switch interconnection topology, (ii) location of VMs on physical nodes, (iii) current traffic matrix of the network (from local coordinator feedback) and (iv) policies such as VM priority, bandwidth, and response times, and generates periodic *transmission directives* to the local coordinators.

Transmission directives are explicit instructions to local coordinators at each physical node on how to regulate the high throughput transmission activity of each VM. For example, a transmission directive could take the form of *explicit time scheduling*, such as "which VM gets to transmit packets when and for how long". Or a transmission directive could be *explicit rate scheduling*, such as "at what rate should a VM's traffic be transmitted for the next N milliseconds". Or the directives could be a combination of both or could take any other form suited to the performance objectives.

In theory, if central controllers in the above framework can carefully coordinate the transmission activities of all VMs across the data center, then one could precisely control the extent of network load at each link in the switched network. Of course, there are a number of practical challenges to implementing such a network-wide transmission control.

In this section, we provide an overview of one form of transmission directive issued by the central controller, namely *global timeslice scheduling*, illustrating how to solve the congestion collapse problems demonstrated earlier in Section II for simple network topologies. Section III-C describes the complete timeslice scheduling algorithm for general network topologies. Section V discusses larger challenges in implementing a complete explicit coordination framework.

[Timeslice scheduling]: A central controller temporally divides network transmission into equal-sized timeslices and explicitly decides which VM gets to transmit in which timeslice. *The goal is to prevent excessive concurrent VM transmissions that can potentially cause congestion collapse in some part of the switched network, while permitting sufficiently high network activity to achieve high utilization.* In each timeslice, the central controller transmits a transmission directive to all local coordinators indicating which VMs are eligible to transmit during that timeslice. The granularity of the timeslice is small, in the order of 1ms to 10ms. The local coordinators act as transmission gatekeepers for VMs at their nodes.

For example, consider the experimental setup shown in Figure 1(a) where the output port leading to the common receiver VM is susceptible to congestion collapse. In this case, the central controller can permit only one of the sender VMs to transmit at a time. Hence it sends a periodic bitmask of the form shown in Figure 8(a) every 10ms. In this case, only one VM gets the green light to transmit in any one 10ms period.

Similarly, consider the experimental setup shown in Figure 1(b) where the output port leading to the 10Gbps uplink is susceptible to congestion collapse. In this case, it would be inefficient to allow only one VM to transmit in each timeslice since each VM is only capable of sending at a maximum 1Gbps. Hence the central controller permits up to 10 VMs to transmit simultaneously in each timeslice by sending the bitmask sequence shown in Figure 8(b).

[Distributed Work Conservation] It may so happen that some of the VMs may not have any traffic to send when the central controller allows them to send whereas other VMs without permission may be backlogged. To prevent network under-utilization in such cases, we designed timeslice scheduling to be *work-conserving* [14]. When a local coordinator finds that a VM with permission to transmit has no pending packets, it gives up the VM's share of the

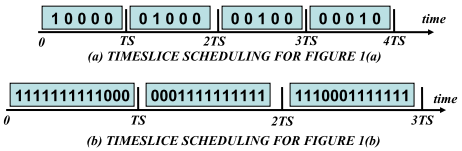

Figure 8. Global timeslice scheduling for setups in Figure 1.

unused timeslice back to the central controller. The central controller then transfers the remaining timeslice to another backlogged VM. To avoid giving up a VM's timeslice too quickly, the local coordinator introduces a a small hysteresis delay ($200\mu s$ in our implementation) before returning the unused timeslice.

C. Timeslice Scheduling in Arbitrary Network Topologies

[Notations and Assumptions] Consider a network with a set N of (physical) end-hosts. Each end-host can host one or more VMs. Assume that the central controller knows the interconnection topology of all the layer-2 switches in the network, which could be arbitrary. There are L links in the network interconnection topology. A transmission link from node i to node j is represented by an ordered pair (i, j), where a node could be either a switch or an end-host. Conversely, the transmission link in the reverse direction from node j to node i is represented by (j, i), even though physically a single bidirectional cable might connect the two nodes. Assume that the central controller has knowledge of the subset of links ST which constitute the spanning tree among switches at any point in time. This information can be queried from modern managed switches via their management interfaces. Maximum bandwidth capacity of link (i, j) is represented by C_{ij}. For ease of exposition, the maximum bandwidth capacity of the outgoing link C_{nj} from an end-host n to switch j is represented by C_n (assuming there is only one outgoing link per end-host). Given the spanning tree ST, we pre-compute the path P_{xy} from every end-host x to every other end-host y.

$$P_{xy} = \{\ (i,j) \mid (i,j) \text{ lies in the path from } x \text{ to } y \text{ in the spanning tree } ST\ \}$$

[Backlog and Active Contention Groups] At any instant in time, we define the *backlog group* B_n of an end-host n as the set of destination end-hosts for whom n has backlogged traffic. Further, we define the *active contention group* AG_{ij} of a link (i, j) as the set of end-hosts which have backlogged traffic that would pass through link (i, j) and are allowed to transmit by the central controller during a given quantum.

$$AG_{ij} = \{\ n \mid \exists m \in B_n \text{ s.t. } (i,j) \in P_{nm} \text{ and } n \text{ is allowed to send via link } (i,j)\ \}$$

[Feasibility Condition] The *feasibility condition* $F(AG_{ij}, C_{ij})$ is the condition under which the link (i, j) will not experience congestion collapse. Specifically, the sum of all traffic allowed through link (i, j) should not exceed the link capacity C_{ij}. More formally, $F(AG_{ij}, C_{ij})$ is defined by the following inequality:

$$F(AG_{ij}, C_{ij}) : \sum_{\forall n \in AG_{ij}} C_n \leq C_{ij} \quad (1)$$

[Scheduling Algorithm] We now outline the algorithm used by the central controller to coordinate the transmission activities of end-hosts across the network. The central controller takes as input (1) the spanning tree ST, (2) Pre-computed paths $P_{xy} \forall$ end-host pairs x, y, and (3) Current backlog group $B_n\ \forall$ end-hosts n.

As output, the central controller generates a set of destination nodes D_n for each end-host n, such that n is allowed to transmit only to the nodes in D_n during the next scheduling quantum without violating any feasibility condition. If $D_n = \emptyset$, then n is not allowed to transmit during the scheduling quantum. Formally:

$$D_n = \{\ m \mid m \in B_n \text{ and } n \in AG_{ij}\ \ \forall (i,j) \in P_{nm}\ \}$$

In other words, D_n is allowed to transmit to node m in a scheduling quantum if and only if n had backlog traffic for m and n is a member of the active contention group AG_{ij} for every link (i, j) along the path from n to m. Algorithm 1 describes the algorithm to compute D_n. Algorithm 1 has a worst-case complexity of $O(N^2D)$, where N is the number of end-hosts in the network and D is the network diameter.

Algorithm 1 Algorithm to compute the destination set $D_n\ \forall n$.

```
1:  Input: (a) Current spanning tree ST
2:        (b) Pre-computed paths P_xy ∀ end-host pairs x, y
3:        (c) Current backlog group B_n ∀ end-hosts n
4:  Output: Destination set D_n ∀n
5:
6:  for each link (i, j) do
7:     AG_ij := ∅
8:  end for
9:
10: for each end-host n do
11:    D_n = ∅
12:    for each end-host m ∈ B_n do
13:       for each link (i, j) ∈ P_nm do
14:          if n ∉ AG_ij and F({n} ∪ AG_ij, C_ij) = false
             then
15:             Skip to next m on line 12.
16:          end if
17:       end for
18:       for each link (i, j) ∈ P_nm do
19:          AG_ij := AG_ij ∪ {n}
20:       end for
21:       D_n = D_n ∪ {m}
22:    end for
23: end for
```

[Generating Schedules for Successive Quantums] Algorithm 1 will generate one transmission schedule for a given

scheduling quantum. For successive scheduling quantums, the central controller has to ensure that distinct transmission schedules are generated and that no backlogged traffic is indefinitely starved. Algorithm 2 lists the algorithm to generate distinct transmission schedules for successive time quantums while avoiding starvation. This algorithm has the worst-case time complexity of $O(NL + N^2D)$ for each time quantum, where N is the number of end-hosts, L is the number of links in the spanning tree ST, and D is the network diameter.

Algorithm 2 Algorithm to compute transmission schedules for successive time quantums.

```
1:  Input: (a) Current spanning tree ST
2:        (b) Maximum link capacity C_ij ∀(i.j) ∈ ST
3:  Output: Sequence of transmission schedules for each
    quantum.
4:
5:  for each end-host n do
6:     Set B_n to current backlog at n
7:  end for
8:  for each time quantum do
9:     Compute schedule D_n  ∀n using Algorithm 1
10:    Output D_n  ∀n
11:
12:    /*Ensure distinct schedule for next quantum*/
13:    for each end-host n do
14:       for each end-host m do
15:          if m ∈ D_n then
16:             B_n := B_n − {m}
17:          end if
18:          if B_n = ∅ then
19:             Skip to line 22
20:          end if
21:       end for
22:       Append the new backlog at end-host n to B_n
23:    end for
24: end for
```

D. Global Rate Scheduling

The end-system coordination framework in Figure 7 is flexible enough to allow for transmission control strategies other than timeslice scheduling described above. For example, the periodic directives from the central controller could alternatively specify the maximum rate at which each VM should transmit until the next directive. Thus the central controller could proactively throttle VMs that are contributing to an imminent congestion buildup. We have also developed corresponding algorithms for this strategy and implemented them in the *XCo* framework. We omit the details of the rate scheduling algorithm here for space constraints, but include preliminary evaluation in Section IV.

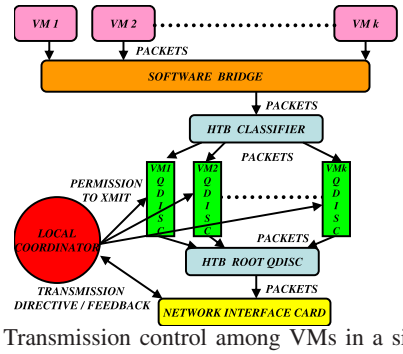

Figure 9. Transmission control among VMs in a single node.

E. Implementation Issues

[Transmission control within each node] Figure 9 shows the internal architecture of transmission control among multiple VMs within a single node. We first created a custom Linux Traffic Control [15] setup in Domain 0. We use Hierarchical Token Bucket (HTB) queuing discipline (qdisc) as the root queue communicating with the network card. This root queue is configured to contain one Priority (PRIO) qdisc for each VM in the host. The root HTB qdisc has a packet classifier which queues each VM's outgoing packets in their corresponding PRIO qdiscs. The root HTB qdisc dequeues packets from these PRIO qdiscs for transmission over the NIC. We modified the implementation of PRIO qdisc in Domain 0 to add the local coordinator. The local coordinator and the central controller communicate directly using a new layer-3 protocol type for control messages and thus bypass the standard TCP/IP stack in Domain 0. Each VM (and its PRIO qdisc in Domain 0) is assigned a globally unique ID by the central controller. For timeslice scheduling, only the PRIO qdiscs which are marked in the directive from central controller can transmit. For rate scheduling, each PRIO qdisc transmits at or below the rate specified in the directive.

[Synchronization] of network transmission activity among different physical nodes is particularly important for timeslice scheduling. Hence each local coordinator synchronizes its "network transmission clock" with the central controller at the boundary of every timeslice whenever it receives a periodic transmission directive. Within a timeslice the local coordinator uses the local high resolution clock to track events.

[Dynamic Join/Leave] When a physical node joins the network, its local coordinator registers itself with the central controller. Whenever a new VM is started (or migrated) in a physical node, the local coordinator obtains a globally unique ID for the VM and sets up its PRIO qdisc. Conversely, when a VM terminates or the node shuts down, the local coordinator informs the central controller.

IV. Performance Evaluation

[Solving Incast Problem] Consider the same experimental setup described in Section II. As we saw earlier in Figure 2 the plot labeled "Incast Problem" shows the effect

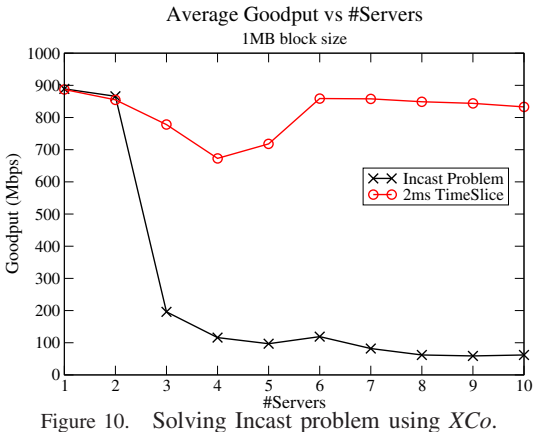

Figure 10. Solving Incast problem using *XCo*.

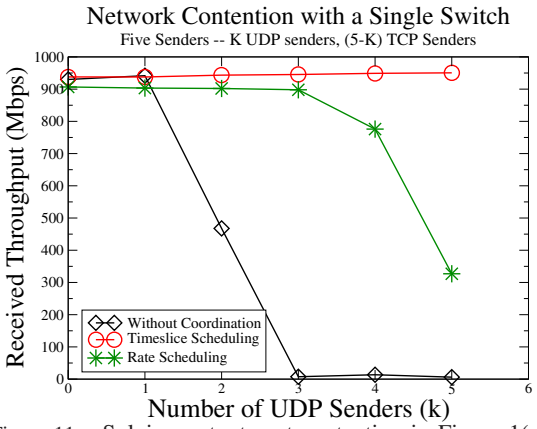

Figure 11. Solving output port contention in Figure 1(a)

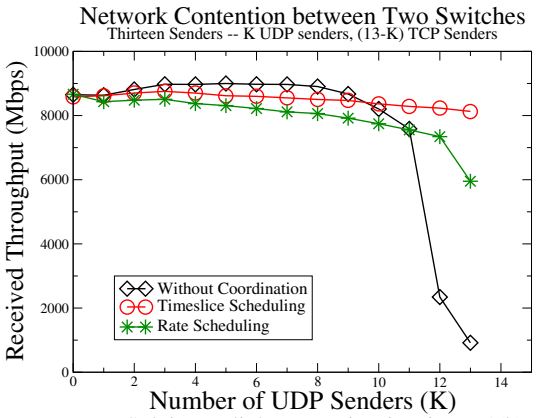

Figure 12. Solving uplink contention in Figure 1(b)

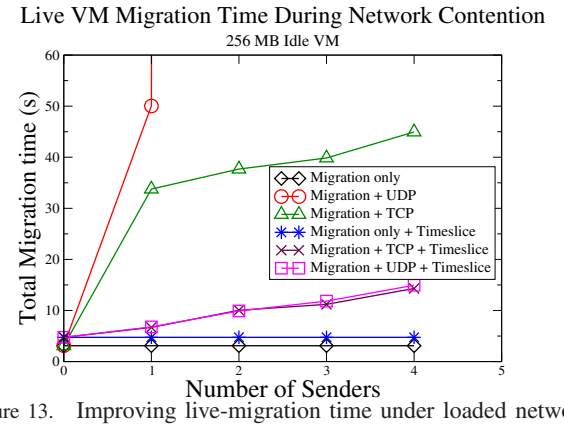

Figure 13. Improving live-migration time under loaded network.

of TCP throughput collapse in networked storage setup. To solve this problem in *XCo*, the central controller decides which storage server sends data to the client in a given timeslice. The central controller broadcasts a control packet every 2ms and allows one of the storage servers to transmit. Figure 10 shows that the plot labeled "2ms TimeSlice" maintains a high receive throughput between 700Mbps to 900Mbps, thus overcoming the incast problem. A slight reduction in throughput between 3 to 5 servers is likely due to inefficiencies in our implementation of distributed work conservation mechanism. This is being further investigated.

[Solving throughput collapse due to non-TCP-friendly traffic] Now we reconsider the problem of throughput collapse due to non-TCP-friendly traffic demonstrated earlier in Figures 3 and 4. Recall Figure 1(a) where the output port leading to the common receiver VM is susceptible to congestion collapse. In Figure 11, the plot labeled "Timeslice Scheduling" shows the corresponding improvement in received throughput when using *XCo*. Note that the received throughput stays close to the maximum 1Gbps even as the number of UDP senders increase. The plot labeled "Rate Scheduling" in Figure 11 shows a better received throughput than having no coordination at all, but slightly worse than timeslice scheduling. This is because merely reducing the transmission rate does not completely eliminate port contention, which happens due to transient overload of output port capacity.

Similarly, recall that in Figure 1(b), the 10Gbps uplink is susceptible to congestion collapse. Hence the central controller permits up to 10 VMs to transmit simultaneously in each timeslice by sending the bitmask sequence shown in Figure 8(b). In Figure 12, the plot labeled "Timeslice Scheduling" shows that the link utilization at the common 10Gbps uplink stays close to 9Gbps even as the number of UDP senders increases. As before, rate scheduling performs slightly worse than timeslice scheduling but better than no coordination.

[Improving Live VM Migration Time] Recall from Figure 5 that, without any transmission control, live VM migration time experiences an exponential increase with competing UDP Netperf senders and an almost ten fold increase with competing TCP senders. When using *XCo*, the plots labeled "... + Timeslice" in the Figure 13 show that increase in live VM migration time is far less than the corresponding plots without timeslice scheduling for both UDP and TCP senders. Although there is still an increase in migration time, the increase is linear and there is almost no difference between the cases of competing TCP and UDP sessions.

V. *XCo* Challenges and Opportunities

[Minimizing Coordination Overhead] Central coordination is unnecessary when the network is uncongested. To minimize coordination overhead, central controller could permit uncoordinated transmissions under normal traffic conditions, but keep monitoring the feedback from local coordinators, including traffic load, packet delays or drops, and any available switch-level congestion notifications. Explicit coordination can kick in only when imminent congestion is detected in some part of the network. Even then, the central controller need not impose transmission control on all the VMs. The local coordinators could identify and report those VMs generating large network flows. These VMs could then be regulated by the central controller without limiting transmissions from other well-behaved VMs. The rule of thumb is that central controller should impose minimal constraints on the VMs in order to preempt congestion only where necessary.

[Scalability] In large data center clusters, a single central controller could become a potential bottleneck in transmission control activities. We are investigating the use of multiple central controllers that could self-organize themselves into a hierarchy of controllers. Each controller in the hierarchy could manage transmission permissions for different parts of the switched subnet. Peer controllers at the same level coordinate with each other and with the controller next level up.

[Fault Tolerance] There are three major types of failures: controller failure, local node failure, and loss of transmission directives. Failure of a central controller(s) does not need to imply that network activity comes to a grinding halt. If the local controllers do not receive a transmission directive from a central controller for a certain time, they could switch to uncontrolled transmission to keep their network activities alive. Non-arrival of multiple successive directives indicates to local coordinator that either central controller has failed, or that the network is partitioned. Another alternative is for a backup controller node to monitor the presence of a primary central controller via its transmission directives and take over the operations if the primary controller ever goes silent. Central controller can also remove the VMs in a physical node from its schedule if that node's local controller does not send feedback.

[Handling Direct NIC Access by VMs] Recent development of hardware IOMMU support [16] enables VMs to directly access the network card and bypass the network datapath via Domain 0 (or the hypervisor). Such hardware support is still in early stages of adoption. Fortunately, another parallel trend is the development of intelligent programmable network cards [17] with virtualization support. Since local coordinator is a lightweight entity, it can be easily implemented in such programmable NICs to support *XCo*-like capability.

VI. Related Work

[Congestion in Data Center Ethernet] The Data Center Bridging Task Group [6] is developing specifications for hardware QoS support in future data center Ethernet fabric through congestion notification, priority based flow control, and enhanced transmission selection. Work in [18] shows that it is possible to design a fat-tree network topology for *non-virtualized* data center networks that allows any pair of hosts to communicate at full bisection bandwidth. The architecture ties the IP address assignment to the location of nodes in the topology. This renders the solution unsuitable for clusters that host VMs because VMs may often migrate from one physical host to another, carrying their original IP address to the new location. [18] also proposes performing centralized scheduling of large flows so that they can be rerouted via disjoint paths. This requires reprogramming the forwarding logic in the switches and the ability of switches to identify and report large flows to the controller. Currently, Virtual LANs (VLAN) [19] are extensively used to set up *logical* layer-2 networks between groups of related VMs to form virtual clusters. VLANs are intended for logical isolation of Ethernet traffic and by themselves do not provide any congestion control or QoS. ECMP [20] is a multi-path routing technique supported in many enterprise core switches that helps reduce traffic bottlenecks by using multiple paths via static load splitting. However, one could still end up with congested network links since ECMP does not consider per-flow bandwidth while splitting. Ethernet flow control in the 802.3x standard [21] allows an overloaded downstream port to request a temporary pause of all traffic from the upstream port. While useful in low-end edge switches, this feature is counter-productive when enabled in backbone switches due to head-of-the-line blocking. The *XCo* framework proposed in this paper is complementary to any switch-level and transport-level support for congestion control and QoS in the switched network. Moreover, *XCo* can also work with today's commodity Ethernet switches that provide no such switch-level support.

[TCP Throughput Collapse Problem] Also known as *Incast* problem, TCP throughput collapse was first described in [8] in the context of parallel filesystems. The problem was addressed at the application level by limiting the number of servers communicating concurrently with the client and by reducing the advertised TCP receive buffer size. Work in [10] examined a number of TCP improvements to address the Incast problem, concluding that while throughput sometimes improved, none of them substantially prevented TCP throughput collapse. Subsequent work [9], [22], [23] showed that reducing TCP's minimum RTO can maintain high throughput. However too small minimum RTO can lead to spurious timeouts for wide-area network traffic. In addition, none of the TCP-specific solutions to the Incast problem address the case where non-TCP-friendly traffic

might share the data center Ethernet fabric, causing loss of fairness properties. Our *XCo* framework leverages recent advances in virtual machine technology to guarantee that throughput collapse is avoided under all circumstances, irrespective of the mix of TCP and non-TCP-friendly traffic sharing the network fabric.

VII. Conclusion

This paper makes the case that virtualization offers new opportunities to alleviate congestion-driven performance problems experienced by networked storage in particular, and data center Ethernet in general. We present the design, implementation, and evaluation of a prototype, called *XCo*, that explicitly and transparently coordinates network transmissions among virtual machines in a data center Ethernet fabric. We offer evidence through preliminary evaluation that such explicit coordination can go a long way towards preventing congestion and throughput collapse in commodity Gigabit and 10GigE switched Ethernets. Our techniques require no changes to the VMs, applications, protocols, or the network switches. Besides being complementary to future switch-level support for congestion management, we have shown that the *XCo* framework can also squeeze greater performance out of today's unmodified switched Ethernet infrastructure.

Acknowledgement

This work is supported in part by the National Science Foundation through awards CNS-0845832 and CNS-0855204.

References

[1] Internet Small Computer Systems Interface (iSCSI), *http://tools.ietf.org/rfc/rfc3720.txt*.

[2] INCITS Technical Committee T11, *Fibre Channel over Ethernet, http://www.t11.org/fcoe*.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation, Boston, MA*, 2005, pp. 273–286.

[4] J. G. Hansen and E. Jul, "Self-migration of operating systems," in *Proceedings of the 11th ACM SIGOPS European workshop, Leuven, Belgium*, 2004.

[5] K. Kant, "Towards a virtualized data center transport protocol," in *Proc. of INFOCOM workshop on High Speed Networks*, 2008.

[6] IEEE 802.1 Data Center Bridging Task Group, *http://www.ieee802.org/1/pages/dcbridges.html*.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton, NY, USA*, 2003, pp. 164–177.

[8] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale storage cluster: Delivering scalable high bandwidth storage," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Pittsburgh, PA*, 2004.

[9] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *Proceedings of the ACM SIGCOMM, Barcelona, Spain*, 2009, pp. 303–314.

[10] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of tcp throughput collapse in cluster-based storage systems," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies, San Jose, California*, 2008, pp. 1–14.

[11] Netperf, *http://www.netperf.org/netperf/*.

[12] C. Diot and J.-Y. L. Boudec, "Control of best effort traffic," *IEEE Network*, vol. 15(3), pp. 14–15, May 2001.

[13] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner, "Achieving 10gbps using safe and transparent network interface virtualization," in *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 2009)*, March 2009.

[14] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," in *Proc. of IEEE*, vol. 83(10), 1995, pp. 1374–1396,.

[15] Linux Advanced Routing and Traffic Control, *http://lartc.org/howto/*.

[16] Input/Output Memory Management Unit, *http://en.wikipedia.org/wiki/IOMMU*.

[17] G. Gumanow, "Solving the hypervisor network I/O bottleneck solarflare virtualization acceleration," White Paper, SF-101233-TM, Solarflare Communications, 2007.

[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of SIGCOMM 2008*, Aug. 2008.

[19] IEEE 802.1, *802.1Q - Virtual LANs, http://www.ieee802.org/1/pages/802.1Q.html*.

[20] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992, Internet Engineering Task Force, 2000.

[21] O. Feuser and A. Wenzel, "On the effects of the ieee 802.3x flow control in full-duplex ethernet lans," in *Proceedings of the 24th Annual IEEE Conference on Local Computer Networks, Lowell, MA*, 1999.

[22] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding tcp incast throughput collapse in datacenter networks," in *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking, Barcelona, Spain*, 2009, pp. 73–82.

[23] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of ACM*, vol. 51, no. 1, pp. 107–113, 2008.