HyGenICC: Hypervisor-based Generic IP Congestion Control for Virtualized Data Centers

Ahmed M. Abdelmoniem, Brahim Bensaou, Amuda James Abu
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
{amas, brahim, ajabu}@cse.ust.hk

***Abstract*—In today's modern cloud supported applications, the traffic relies on both congestion-responsive transport protocols like TCP and congestion-oblivious transport like UDP including all their variations. As bandwidth is not totally virtualized in today's data centers, the diverse responses of these protocols to congestion lead to inefficiencies and service disruption to some applications. In this paper we present HyGenICC, a simple, distributed and practical congestion control mechanism that puts congestion control back where it belongs, in the network layer, albeit, without modifying the network layer behaviour. To this end IP ECN marking in the switches to indicate congestion, combined with additional packet processing in the hypervisor enable HyGenICC to partition bandwidth among competing virtual machines (VMs) in datacenters effectively. To enable easy deployment in existing data center, HyGenICC design is subjected to several constraints such as: i) freedom of changes to the guest VMs' congestion control mechanism, and ii) reliance only on switch capabilities that are already available in today's commodity switches. We evaluate HyGenICC via extensive simulation in the ns2 network simulator.**

Keywords*—*Congestion Control, Data Center Networks, Kernel Module, Rate Control, Open vSwitch, Virtualization

I. Problem Statement

Resource sharing via virtualization has become a common practice in today's public and private datacenters. Most typically, a tenant is provisioned with virtual machines each having dedicated CPU cores or virtual CPU, dedicated memory and storage space, and virtual network interface over the underlying shared physical network interface. In many practical datacenters, the control plane is very richly provisioned with many novel methods to make the virtualization easy to manage; for example, Amazon Web Services adopted a control-plane concept of "Virtual Private Cloud (VPC)" where a tenant can easily create and manage its own private virtual network as an abstraction layer running on top of the shared network infrastructure of AWS's public cloud. In contrast, the data plane has seen little progress in provisioning the network bandwidth to combat congestion, improve physical network efficiency, achieve better scalability, and provide true isolation between competing VMs, allocating each one its share of bandwidth and throttling careless or subversive ones.

To tackle these issues, cloud providers should deploy a network abstraction layer that represents a dedicated switch with guaranteed capacity, connecting various tenants' VMs [1]. In such environment, different VMs may reside on any machine in the datacenter, but each VM should be able to send traffic at a full line rate specified by the abstraction layer, regardless of traffic patterns or workload nature generated by competing VMs.

The following are a few necessary components that can be integrated together towards this ultimate goal: *i)* a smart and scalable VM placement mechanism within the datacenter network that decouples bandwidth allocation from other resources. To achieve this, topologies with bottlenecks within the network core (such as uplink over-subscription or a low bisection bandwidth) should whenever possible be avoided; *ii)* a methodology to fully utilize the available high bisection bandwidth (e.g., a load balancing mechanism and/or multi-path transport/routing protocols); and *iii)* a rate control mechanism to ensure conformance of VM rates to the provided bandwidth, and to police misbehaving or non-conforming VMs.

A number of promising research works that tackled successfully the first two mechanisms are available today [2], [3], [4]. The works in [2], [3] built scalable network topologies offering a 1:1 over-subscription and a high bisection bandwidth. These topologies are shown to be easily deployable in practice and can simplify the VM placement at any end-host with sufficient bandwidth. The work in [4] targets the second issue, achieving a high utilization of the available capacity via routing and transport protocols designed for datacenters. For the third issue, most proposed solutions [5], [6], [7], [8], [9] focus on TCP congestion control and its variations to share bandwidth fairly among flows or to reduce the overall completion time, nevertheless: *i)* all such protocols tend to be agnostic to the nature of VM aggregate traffic demands and cannot evenly distribute the capacity across competing VMs (for instance a VM could gain more throughput by opening parallel TCP connections); *ii)* the inefficiency is exacerbated by the coexistence of several TCP flavours in the same network due to the variations in guest operating systems deployed in the VMs (e.g., TCP New Reno, compound TCP, Cubic TCP, DCTCP, and so on); and, *iii)* finally, the increasing trend of relying on UDP transport in many emerging cloud applications (e.g., [10]), sounds the knell of any solution to the problem that only relies on TCP.

Unfair competition between TCP and UDP has already been known to exist for two decades in the Internet. Recent studies [11] have shown that the problem also exist in data-

Manuscript is accepted for publication in proceedings of IEEE ICC16 ©2017 IEEE. This work is supported in part under Grants: HKPFS PF12-16707 and FSGRF14EG24.

center networks with small delays, small buffers and different topologies from those found in the Internet. We also conducted a small scale simulation study (not shown here) to ascertain the existence of such problems and found that TCP NewReno is always at a disadvantage against aggressive UDP and DCTCP. With the proliferation of a plethora of transport protocols in data center it is evident that a new solution to the problems of congestion is needed, and it must appeal to cloud operators and cloud tenants alike, as such the following intuitive design requirements are desirable in such solution: R1) it should be simple enough to be readily deployable in existing production datacenters; R2) it should be agnostic to the transport protocol in use to be able to stand the test of time; R3) it should not require changes to the tenant VM guest OS, nor assume any advanced network hardware capability other than those available in cheap commodity servers and switches; R4) it should scale well with the volume of traffic.

In this paper we propose a hypervisor-based generic IP congestion control (HyGenICC) mechanism that fulfils all four design requirements. In the sequel, we first introduce the idea behind the design of HyGenICC in Section II. We discuss our proposed methodology and present the proposed HyGenICC framework in Section III. We show via ns2 simulations how HyGenICC achieves its requirements and discuss simulation results in Section V. In Section VI, we discuss some related work and finally, conclude the paper in Section VII.

II. Introduction to HyGenICC

To enable responsiveness to congestion regardless of the transport protocol, one needs to return to the fundamentals and put the burden of congestion control in principle where it belongs: in the network layer. As such, in principle, such congestion control mechanism must be transparent to the transport layer protocol. However, to reconcile the principle with the practice, design requirements R1-R4 must be fulfilled and thus HyGenICC outsources its congestion control building blocks to the hypervisor.

To meet requirement R1, HyGenICC can be implemented either as a hypervisor-level shim-layer or as an added feature to any of the current commercial virtual switches' data-path module. The job of the added shim-layer to the hypervisor is to enforce per-VM rate control without VM cooperation nor any knowledge about its traffic patterns, workloads, or used transport protocol (TCP/UDP). To this end, HyGenICC maintains a *rate allocation mechanism* at each server to partition the available uplink bandwidth among VMs locally at the sending and receiving servers. In each such server, HyGenICC only needs to maintain state information per VM which meets design requirement R4. HyGenICC deploys a simple hypervisor-to-hypervisor (IP-to-IP) *congestion control mechanism* that relies on ECN markings (readily available in commodity switches) to infer core network congestion. HyGenICC operates at the IP level and does not interact directly with the VMs, which meets requirements R1, R2 and R3. In addition, when detecting a highly congested path in the core network towards a destination (via ECN), HyGenICC performs admission control by refraining from accepting any further connections to this destination VM until the congestion subsides. Our design is highly scalable, responsive, work conserving and since it is IP based, it enforces the allocated bandwidth even in the presence of highly dynamic and changing traffic patterns and transport protocols. The rate allocator resolves the contention among tens-to-hundreds of co-located VMs at the servers, while the congestion control mechanism addresses the contention in the network core and pushes it back to the sources. HyGenICC also allows administrators to assign per-VM weights which directly affect the bandwidth reservation for the VMs making it appealing from cloud providers' perspective as it enables easier and more tangible bandwidth pricing and accounting.

TABLE I: *Flow attributes and variables tracked in our mechanism*

Entry name (VM-to-VM)	Description
$source$	IP address of source VM
$dest$	IP address of destination VM
out_packet_count	Sent packets count
ipr_packet_count	Received packets with "IPR-bit" mark
ecn_packet_count	Received packets with ECN mark
Variable name (per VM)	**Description**
$rate$	The share rate or speed of NIC
$bucket$	The capacity of the token bucket in bytes
$tokens$	The number of available tokens to be used for transmission

III. Proposed Methodology

First we discuss HyGenICC by imagining the datacenter network as contained within one end-host where the VMs are connected via a single virtual switch. Then, we extend this design to operate in a network of end-hosts where the datacenter fabric is treated as black box that generates congestion signals whenever congestion is experienced. In a single virtual switch connecting all VMs, bandwidth contention happens at the output link to the destination when multiple senders compete to send through the same output port of the virtual switch. The virtual switch need to distribute the available physical port's capacity among VMs and ensure compliance of the VMs with the allocated shares. Hence it needs a mechanism that detects and accounts for active VMs and apply rate limiters on a per-VM basis to share the bandwidth among them.

HyGenICC deploys a flow table (for congestion control purpose) to track state information shown in Table I on a VM-to-VM granularity (i.e., source VM-destination VM pairs). In addition, per-VM token-bucket state is used to enforce the VM's share of bandwidth.

A. VM detection and bandwidth allocation

As soon as a VM's port becomes active (sending or receiving traffic), an associated entry is created in the flow table. Whenever a new VM becomes active on a given NIC, the NIC's nominal capacity is redistributed among the token buckets of active VMs to account for the new one. This is done by readjusting the rate and bucket size of all active VMs' token buckets on that NIC. Any extra traffic sent by the VM in excess of its share is simply dropped and resent later by the transport layer or otherwise a per-VM queue is used for holding the traffic for later transmission whenever the tokens are regenerated[1].

B. Congestion Control Mechanism

In practice, congestion may always happen within the network as shown in Figure 1, if the network is over-subscribed

[1]We have experimented with both approaches and the queuing mechanism achieves slightly better performance which did not motivate its usage due to management and memory overhead.

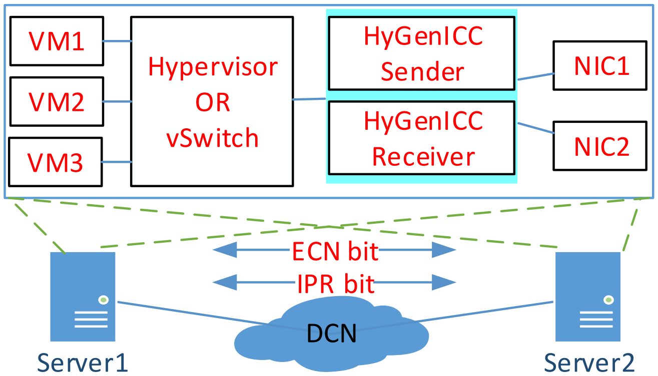


Fig. 1: *HyGenICC high-level system design*

or does not provide full bisection bandwidth. HyGenICC therefore relies on readily available features in switches hardware[2] , to convey congestion signals to the sources. To be more abstract, HyGenICC treats the datacenter network as a black box in which source servers inject traffic and the black box generates ECN marks in response to congestion towards the receivers. ECN marks are a fast proactive mechanism that can help in quickly detecting any congestion from a shared queue when buffers exceed a pre-configured queue occupancy threshold along a packet's path.

HyGenICC uses the flow table to track, for each source-destination pair, the number of IP packets received with congestion notification marks, regardless of the type of transport protocol (TCP, UDP, or otherwise). This information is a valuable indication of the level of congestion along the path between the source VM and the destination VM starting at that particular NIC. Since HyGenICC implements a network-layer congestion control, any ECN or other marking used to track congestion is cleared before delivering the datagrams to the VM[3]. In addition, to force universal ECN marking along the path, all outgoing packets are marked with the ECN-enabled bit. HyGenICC typically creates a network layer congestion control loop between hypervisors and is fully transparent to the overlying VM transport protocol.

At the receiver side, upon receiving ECN marks, HyGenICC needs to reflect the information back to the source to trigger reduction of the sending rate of that particular source VM. To avoid introducing any additional overhead and hinder the operation of any on-path middleboxes by introducing a new protocol, we propose to piggyback the information on any returning data. For this we identify three types of traffic flows: TCP, which is by default bidirectional, other non-TCP bidirectional traffic and finally unidirectional traffic; for the three categories of traffic, we propose to use the unused reserved bit in the IP header "IPR-bit" of any reverse packet to reflect the ECN marking synchronously to the origin. While this might be sufficient for the first two categories of traffic to carry all marking back to the source, for the third category, there might be a dramatic imbalance in the forward traffic and reverse traffic leading to some proportion of forwarded markings not being reflected back. As a solution HyGenICC crafts a special small IP packet with header only (20 bytes of IP and 14 for Ethernet headers) and piggybacks explicitly the number of remaining ECN marks on the identification field of this IP packet. The IP protocol field is destined to an unused number that has meaning only for HyGenICC.

At the sender, to match the current sending rate to the congestion level in the network, upon receiving "IPR-bit" marks or the special packet, the source decreases the VM's current allocated rate in proportion to the amount of marks and gradually increases the rate when no congestion bits are received in a period.

IV. Implementation

As explained above, HyGenICC needs two mechanisms: rate limiters at the source server and congestion controller that run from source to destination server. These mechanisms can either be implemented in software, or hardware or a combination of both as necessary. We simplified the design and concepts of HyGenICC so that the built system is able to maintain line rate performance at 1-10Gb/s while reacting quickly to deal with congestion within a datacenter's short RTT time scale.

A. HyGenICC sender

HyGenICC sender processing is described in Algorithm 1. At the senders HyGenICC tracks the *rate*, the number of *tokens*, the depth of the *bucket* and the fill-rate variables per-VM per-NIC where the per-VM rate limiters are implemented as counting token buckets that have a rate $R(i,j)$ each, a bucket capacity $B(i,j)$ each and number of tokens $T(i,j)$ each. In addition, the sender will also handle the received congestion signals from different destinations on a per-source basis.

1) Rate Allocation: Initially, the installed on-system NICs are probed and the values of their nominal data rate $R(i)$, bucket capacity $B(i)$ and tokens $T(i)$ are calculated correspondingly. Thereafter, when packets start flowing from each source VM, NIC capacities are redistributed and a new capacity share "$Capacity\_Share$" is calculated and used to update the entries for each active VM in the rate, tokens and bucket matrices and the VM is marked as currently active on all outgoing physical NICs.

After a certain time of inactivity[4], the bucket entries for a VM are reset and its allocation is reclaimed and redistributed among currently active VMs. As shown in Table I, flow-table entries are established immediately after arrival of the first packet using source-destination IP address[5]. First, on arrival or departure of each packet P, its outgoing port j and incoming port i are detected. The current value of available tokens $T(i,j)$ is retrieved and replenished based on the elapsed time since the last transmission. Then, using the new $T(i,j)$, the packet is allowed for transmission if $T(i,j) \geq size(pkt)$, in

[2]Most current commodity switches used in datacenters are equipped with QoS mechanisms like Strict Priority (SP), Weighted Fair Queuing (WFQ) and Weighted Random Early Detection (WRED) in addition to the ability of ECN marking of IP packets.

[3]Supposedly, If the tenants are willing to deploy ECN in their overlay networks then our mechanism should not clear ECN mark and let the transport layer handle it as well. We believe the two would not conflict rather complement each other as shown in the simulations.

[4]Inactivity timeout is set to 1 sec in simulations.

[5]We track the state of the communicating VM pairs not individual flows plus we find that the operations involved do not add burden on the hypervisor/vswitch.

Algorithm 1 HyGenICC Sender Algorithm

procedure PACKET_DEPARTURE(P, i, j)
    look up flow entry f in flow table
    $T(i,j) = T(i,j) + R(i,j) \times (now() - f.senttime)$
    $T(i,j) = MIN(B(i,j), T(i,j))$
    if $T(i,j) \geq Size(P)$ **then**
        $T(i,j) = T(i,j) - Size(P)$
        $f.senttime = now()$
        Enable ECN Capable bits (ECT) in IP header
    else
        Drop the packet
procedure PACKET_ARRIVAL(P, i, j)
    look up flow entry f in flow table
    if Packet is congestion feedback message **then**
        $f.feedback = f.feedback + int(P.data)$
        $f.rbdetected = true$
        $f.feedbacktime = now()$
        Drop the packet
    else if Packet is "IPR-bit" marked **then**
        $f.feedback = f.feedback + 1$
        $f.rbdetected = true$
        $f.feedbacktime = now()$
        Clear the mark and forward to the VM
procedure TIMER_TIMEOUT
    for each flow f in $FlowTable$ **do**:
        if $now() - f.senttime \geq 1sec$ **then**
            $f.active = false$
            Reset f entry in Flow Table
            redistribute NIC capacity among active flows
    for each Active flow f in $FlowTable$ **do**:
        if now() - f.feedbacktime$\geq$ Congestion_Timeout **then**
            $f.rbdetected = false$
        if $f.rbdetected == false$ **then**
            $R(i,j) = R(i,j) + scale(NIC\_Cap)$
        else if $f.feedback \geq 0$ **then**
            $R(i,j) = R(i,j) - (f.feedback \times scale(NIC\_Cap))$
        else
            $R(i,j) = R(i,j) + scale(NIC\_Cap)$
            $f.feedback = 0$
        $R(i,j) = MAX(0, MIN(Capacity\_Share, R(i,j)))$

this case the packet length is deducted from $T(i,j)$, otherwise the packet is dropped.

2) Congestion Reaction: The sender module reacts on regular intervals to incoming "IPR-bit" and cuts the sending rate in proportion to the amount of marking received. Hence, sources causing congestion in the network will receive "IPR-bit" signals and will react by decreasing their sending rates proportionally until the congestion subsides and congestion signals start disappearing at which time sources start to gradually increase their rates. The process will increase the rate conservatively, and if no feedback arrives within $Congestion\_Timeout$ seconds, the rate is increased fast until it reaches its "Capacity_Share" or an "IPR-bit" is detected again. Function "scale(NIC_Cap)" is used to scale the amount of rate increase and decrease to account for a single packets transmission over a single RTT (i.e., 1000 bytes over 1 Gb/s in average RTT range of 100μs-10ms would give us $\approx$ 80-8 Mb/s of increments).

B. HyGenICC receiver

At the receiver, HyGenICC needs to track incoming congestion ECN marks from the network on a per-source-destination basis and feed this information back by piggybacking it on outgoing packets heading back to corresponding sources. Hence, the operations of the receiver is quite simple and does not incur much processing overhead onto incoming traffic. The receiver processing is described in Algorithm 2.

Algorithm 2 HyGenICC Reciever Algorithm

procedure PACKET_ARRIVAL(P, i, j)
    look up flow entry f in flow table
    if Packet is ECN marked **then**
        $f.ecnmarks = f.ecnmarks + 1$
        Clear the mark and forward to the VM
    if now() - f.feedbacksenttime $\geq$ feedback_timeout **then**
        Create IP feedback message and send to $f.source$
        $f.feedbacksenttime = now()$
        $f.ecnmarks = 0$
procedure PACKET_DEPARTURE(P, i, j)
    look up flow entry f in flow table
    if $f.ecnmarks \geq 1$ **then**
        Set "IPR-bit" flag in IP header
        $f.feedbacksenttime = now()$
        $f.ecnmarks = f.ecnmarks - 1$

Each incoming packet is checked for ECN mark and the number of packets with and without the mark are traced in the flow table, Table I, and immediately the ECN mark is cleared before re-injecting the packet in the normal packet processing path. For each ECN marked packet, an IPR-bit mark is reflected in the first available outgoing packet to that destination (it could be a TCP ACK if the flow is TCP or a UDP reply data packet) until all the ECN marks are cleared. However, when ingress and egress traffic are out of balance on a given flow, non-reflected ECN marks may start to accumulate at the receiver, to address this issue, we periodically use an explicit ICMP-like feedback packet to convey the remaining amount of ECN marks to the source. On a regular intervals close to an RTT, we scan through the flow table asynchronously for any flow with remaining ECN marks and that has not sent any feedback for a period of $Feedback\_Timeout$. If any is found, then an IP packet is created with unused protocol ID value and the current value of ECN marks added as a 2-bytes payload of this packet addressed to the source of the flow. This event is infrequent and unlikely to exist but if so, will not incur much network overhead as the packet size would be 36 bytes (14-bytes Ethernet header + 20-bytes IP header + 2-bytes payload data). To compress further the explicit feedback, the 2 bytes payload can be piggybacked instead in the IP header identification field.

V. SIMULATION ANALYSIS

We study the performance of HyGenICC via ns2 simulation in network scenarios with a high bandwidth-low delay (as is the case in data centers). We compare the performance achieved by a tagged VM using TCP when competing against other VMs using TCP, DCTCP, and UDP in 1) our system, 2) a system that does not use such traffic management and relies on end-to-end congestion control and 3) a system that uses a central control node to perform static bandwidth allocation. We have compared two TCP flavours with ECN and without ECN to show that TCP's reactive nature to ECN is not sufficient to achieve the desired allocation especially when competing with non-responsive flows running UDP and to show that HyGenICC rather complements the use of ECN by the transport layers.

For HyGenICC, there is a single parameter settings of timeout interval for updating flow rates which should be larger than a single RTT, in the simulation we set it to 5 RTTs. In all simulation experiments, we adjust RED parameters to achieve marking based on instantaneous queue length at the threshold of 20% of the buffer size rather than using the weighted average queue length.

A. Simulation Setup

We use ns2 version 2.35 [12], which we have extended with a HyGenICC module inserted at the link elements in topology setup[6]. In addition, we patched ns2 using the publicly available DCTCP patch. We compare TCP newReno with SACK-enabled when competing against TCP, DCTCP and UDP under the three systems. We considered two cases, one where TCP is ECN-bit responsive and one when TCP is not. We use in our simulation experiments speed links of 1 Gb/s for sending stations, a bottleneck link of 1 Gb/s, low RTT of 100 μs and the default RTO_{min} of 200 ms.

We use a rooted tree topology with single bottleneck at the destination and run the experiments for a period of 15 sec. The buffer size of the bottleneck link is set to be more than the bandwidth-delay product in all cases (100 Packets), the IP data packet size is 1500 bytes.

B. Simulation Results and Discussion

We simulated several scenarios that lead all to the same results. So for ease of exposition and clarity we consider a toy scenario with 2 elephant flows, a tagged flow and a competitor. In the experiments, the tagged flow always uses TCP newReno and competes with other flows (in the toy scenario only one other flow) all using the same protocol either TCP newReno, DCTCP or UDP. The competitor flows start at the beginning and finish at the 10th second whereas the tagged flow starts at the 5th second and runs to the end of the simulation. So typically from 0 to 5s only the competitors occupy the bandwidth, from 5s to 10s bandwidth is shared by the two groups, and from 10s to 15s only the tagged flow uses the bandwidth. This experiment is designed to demonstrate the work conservation-ability, the efficiency, and the convergence speed of HyGenICC compared to other alternatives.

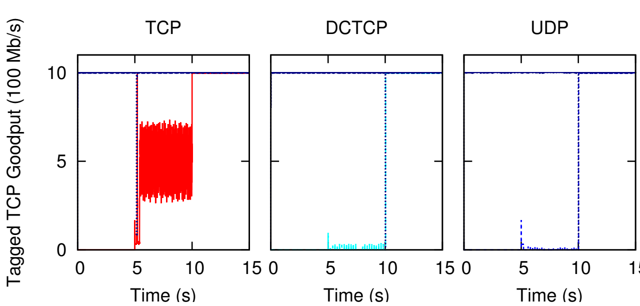


Fig. 2: *No ratelimit without ECN*

[6]Simulation code is available upon request from the authors.

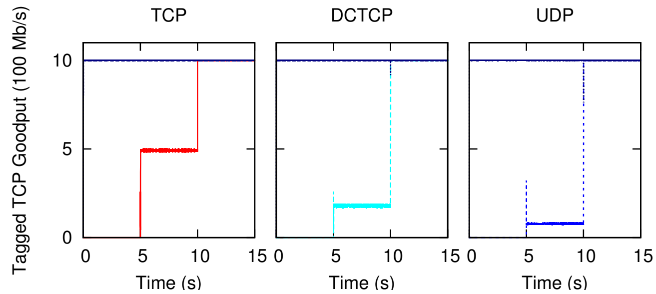
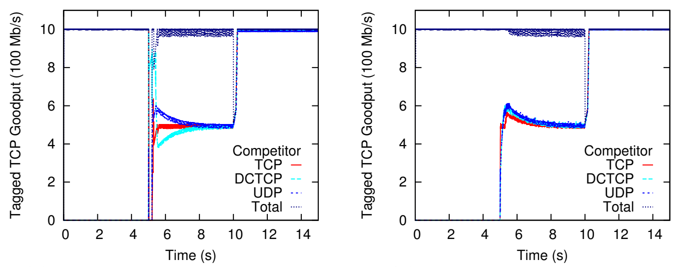


Fig. 3: *No ratelimit with ECN*

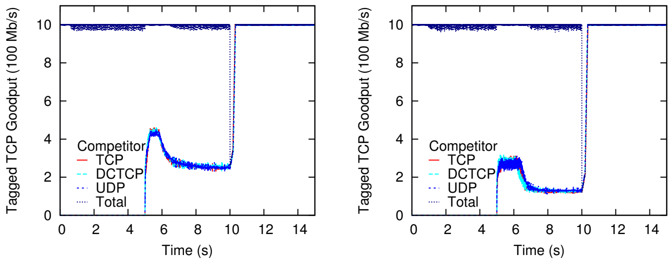


(a) *HyGenICC without ECN* **(b)** *HyGenICC with ECN*

Fig. 4: *Goodput of the tagged TCP flow and aggregate (total) goodput of all senders as measured by the destination.*

Figures 2, 3 and 4 show the goodput of the tagged TCP flow with respect to each competitor and the aggregate goodput observed at the destination in the 2 flows scenario. As shown in Figure 2, without any rate limits, TCP struggles to grab any bandwidth when competing with DCTCP and UDP and its throughput is not stable when competing with TCP without ECN. Figure 3 suggests that ECN can partially solve the problem by allowing TCP flow to be responsive to congestion events at the network, however the achieved throughput reaches the allocated share only when the competitor uses the same TCP protocol. This is attributed to the fact that DCTCP does not react as conservatively as TCP to ECN marks as it does not cut its window by half like TCP does. Simulations with a static rate limit of 500 Mb/s (fair-share) for each of the 2 flows, show that a centralized node assigning per VM static rates can achieve perfect rate allocation but is not efficient as it does not achieve work-conservation (the link utilization is only 500 Mb/s when single flow is active). Figures 4a and 4b show that HyGenICC's dynamic rate limiters that respond to congestion signals achieve both target rate allocation and work conservation regardless of the competing transport protocol. Hence, HyGenICC is able to converge to the current network-wide target-share for the TCP flow in all cases and keeps the network links fully utilized all the time.

Figure 5 shows how HyGenICC reacts to the increasing number of senders by repeating same scenario but with 4 and 8 senders. For the new arriving TCP flow starting at the slow start, it can grab its current share quickly causing congestion in the network. The "IPR-bit" markings coming to the sources will help them adjust their rates up and down until they reach the equilibrium point where each sources is getting their share of 1Gb/4 = 250Mb and 1Gb/8 = 125Mb respectively. HyGenICC's convergence time of ($\leq$ 1 sec) may be a concern but we believe it will not greatly affect the performance of the long-lived elephants and will benefit short-lived mice flows by reducing drops at the end-host rate limiters. To summarize



(a) *4 senders scenario* **(b)** *8 senders scenario*

Fig. 5: *Goodput of the tagged TCP flow and aggregate (total) goodput of all senders or 4 and 8 senders scenarios.*

this simulation study, HyGenICC seems to be able to smooth oscillations and reach a high link utilization and efficient rate allocation among competing flows.

VI. Related Work

HyGenICC can be comparable or complementary to a number of works on cloud network resource allocation that have been proposed recently. Seawall [13] is a system proposed for sharing network bandwidth, it provides per-VM max-min weighted fair share using explicit feedback end-to-end congestion notification based on losses for rate adaptation. Seawall requires modifications to network stack which incurs a large overhead and may interfere with middleboxes operations. SecondNet [14] is designed to divide network among tenants and enforce rate limits, but is limited to providing static bandwidth reservation between pairs of VMs. Oktopus [1] argues for predictability by enforcing a static hose model using rate limiters. It computes rates using a pseudo-centralized mechanism, where VMs communicate their pairwise bandwidth consumption to a tenant-specific centralized coordinator. This control plane overhead limits reaction times to more than 2 seconds which is inadequate for the fast changing and dynamic traffic nature in datacenters. FairCloud [15] designs better policies for sharing bandwidth and explored fundamental trade-offs between network utilization, minimum guarantees and payment proportionality, for a number of sharing policies. EyeQ [16] provides per-VM max-min weighted fair shares in the context of a full bisection bandwidth datacenter topology where congestion is limited to the first and the last hops. By simplifying rate limiters and coupling congestion control to make them dynamic entities rather than static, HyGenICC can achieve similar objectives as these proposals in an easy to deploy manner with minimal CPU and network overhead. HyGenICC is designed to operate with commodity infrastructure and traditional protocols used by current production datacenter/cloud, to be a readily deployable solution. Finally, HyGenICC can leverage the popularity of Open vSwitch (OvS) usage by cloud management frameworks like openstack to implement its mechanism with minor modifications to OvS that do not require any new protocols, software and hardware.

VII. Conclusion and Future Work

In this paper, we set to design and build HyGenICC a system that can fit easily within current production cloud setups to achieve better bandwidth isolation and improved application performance. HyGenICC is a hypervisor (vswitch) level framework to enforce efficient and guaranteed network bandwidth allocation among competing VMs. Our intuitive analysis and ns2 simulation experiments show that simple mechanisms like rate limiting token buckets allied to ingress-egress congestion control protocol can lead to a simple, scalable and readily deployable system design for cloud network resource isolation. HyGenICC is built with three main objectives in mind, low overhead, commodity hardware, and no changes to network hardware or VMs protocol stack. This constitutes a great incentive for deployment in today's production datacenter networks. HyGenICC requires minimal human intervention and can flexibly and efficiently divide network bandwidth across active VMs by giving each VM endpoint a predictable minimum bandwidth and hence bounded latency. Regardless of the transport protocol used by the applications residing in the VMs and even with the existence of misbehaving or bandwidth-hungry traffic, HyGenICC can achieve its design goals. Further testing HyGenICC in a larger scale data center testbed is currently part of our ongoing work.

References

[1] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of the ACM SIGCOMM 2008*, 2008.

[3] B. A. Greenberg, J. R. Hamilton, S. Kandula, C. Kim, P. Lahiri, A. Maltz, P. Patel, S. Sengupta, A. Greenberg, N. Jain, and D. A. Maltz, "VL2: a scalable and flexible data center network," in *Proceedings of ACM SIGCOMM*, 2009.

[4] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proceedings of the ACM SIGCOMM*, 2011.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Computer Communication Review*, vol. 40, 2010.

[6] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Transactions on Networking*, vol. 21, 2013.

[7] A. M. Abdelmoniem and B. Bensaou, "Reconciling mice and elephants in data center networks," in *2015 IEEE International Conference on Cloud Networking (CloudNet)*, Oct. 2015.

[8] A. M. Abdelmoniem and B. Bensaou, "Incast-Aware Switch-Assisted TCP congestion control for data centers," in *2015 IEEE Global Communications Conference (GlobeCom)*, Dec. 2015.

[9] A. M. Abdelmoniem and B. Bensaou, "Efficient switch-assisted congestion control for data centers: an implementation and evaluation," in *2015 IEEE International Performance Computing and Communications Conference (IPCCC)*, Dec. 2015.

[10] R. Nishtala, H. Fugal, and S. Grimm, "Scaling memcache at facebook," *10th USENIX NSDI Conference*, 2013.

[11] S. M. Irteza, A. Ahmed, S. Farrukh, B. N. Memon, and I. A. Qazi, "On the coexistence of transport protocols in data centers," in *IEEE International Conference on Communications (ICC)*, 2014.

[12] NS2, "The network simulator ns-2 project." http://www.isi.edu/nsnam/ns.

[13] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proceedings of the 8th USENIX NSDI Conference*, 2011.

[14] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *6th CoNext Conference*, 2010.

[15] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the Network in Cloud Computing," *ACM SIGCOMM Computer Communication Review*, vol. 42, 2012.

[16] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "Eyeq: Practical network performance isolation at the edge," in *10th USENIX NSDI Conference*, 2013.