Computer Engineering
ISSN: 10003428
Volume 14 | Issue 1 | Year 2025
https://journalofcomputerengineering.com/

Optimizing Network Congestion Control for High-Performance Remote Direct Memory Access Transfers

Alexandros Karampelas, Georgios Tsapatsoulis, Anastasios Kallias, Ioannis Fragkou, Dimitris Chatzigeorgiou, Evangelos Koutsoukos

University of Patras, Department of Electrical and Computer Engineering; University of Crete, Department of Computer Science

Abstract—High-performance interconnects need congestion control to deal with traffic bursts. In this paper, we propose *ACCurate*, a congestion control protocol that assigns exact maxmin fair rates to flows, without relying on costly per-flow state inside the network. ACCurate keeps the backlogs outside of the network, protects innocent flows, and promptly recovers the flows' rates after congestive episodes. Comparisons with TCP and PAUSE-only RDMA under datacenter-resembling workloads further show that ACCurate provides up to 10x faster flow completion times. ACCurate relies on simple hardware that can be readily implemented inside switches. In our implementation, the additional circuitry needed in a 16×16 switch occupies less than 2% of FPGA resources.

I. Introduction

In modern computing clusters, a large number of servers work collectively in order to complete challenging tasks. Current research examines ways to replace the expensive and power-hungry processors of today with thinner, costefficient computing units [1]–[3]. These systems can consist of many tightly coupled end-nodes and fast in-node storage devices increasing the intensity of inter-server traffic. Already happening in datacenters today, VM migration, checkpointing and storage traffic contribute to sudden bursts, which are responsible for transient congestion. To mitigate the bad effects of congestion, many datacenters and several supercomputers still use lossy Ethernet and rely on TCP for congestion control. However, traditional TCP in lossy networks is sluggish and suboptimal, and the industry is looking for alternative solutions [4], [5].

An increasing number of datacenters and HPC installations employ Remote Direct Memory Access (RDMA)-based networks. Multi-channel RDMA engines enable user-level initiated transfers that bypass the kernel network stack and its large latency and CPU utilization [6]–[9]. Without the kernel involvement, handling congestion at the network hardware level is both necessary and a good opportunity to outperform TCP. For similar reasons, modern RDMA NICs already provide reliable packet delivery.

RDMA networks typically use link-level flow control in order to improve performance [10] and simplify the hardware responsible for re-transmissions. In lossless networks, congestion manifests with large in-network backlogs, which can decrease system throughput and increase the flow completion time (FCT) of latency-sensitive flows, even of those not contributing to congestion. Using proactive congestion control, we can prevent congestion from occurring in the first place, by reserving resources before injecting traffic [11], [12]. However, this introduces a connection setup latency, which gets more pronounced with increasing system size due to a larger number of hops. In this paper, we instead favour *latency-optimized protocols* that allow flows to start immediately at full speed.

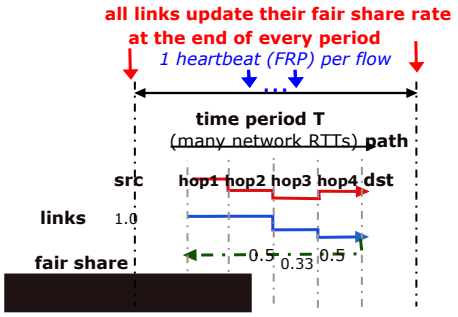
In principle, a congestion management scheme needs to regulate the injection rate of flows in order to avoid overloading network links. Additionally, it must distribute the available capacity fairly among network users and control the in-network backlogs to decrease the FCTs.

To achieve these goals, in this paper we propose *Accurate Congestion Control (ACCurate)*, a new scheme suitable for efficient hardware implementation. At each network interface, we use a *Remote Direct Memory Access (RDMA)* engine with multiple channels. Each such channel executes a data transfer (flow) and can control its transmission rate on a per-packet basis using a rate limiter. Our congestion control relies on a *contention point (CP)* in front of every link. On top of that, we employ a *novel heartbeat mechanism* that allows CPs to keep track of the active flows (RDMA transfers) passing through them, without relying on expensive per-flow state.

We assume that time is slotted in network-scale time periods (from a few to a few tens of microseconds). In every time period, every flow injects *exactly one "heartbeat"* that travels to the destination, effectively communicating its current transmission rate with every link on its path, as shown in Fig. 1. This protocol allows links to compute their fair shares at the end of every period. Our congestion control algorithm uses these estimates to accurately rate-limit each flow to the fair share of its dominant bottleneck.

To further reduce the in-network backlogs, ACCurate reserves enough headroom so that links can breath out sudden backlogs [13].With shallow in-network buffers (~2.5KB per input port in our experiments), this can adequately control the in-network latencies.

Our results show that ACCurate: *(i)* improves the FCTs of latency-critical flows by a factor of 10 compared to TCP under typical datacenter workloads carrying a mix of bulky and latency-sensitive flows; *(ii)* throttles the offensive flows



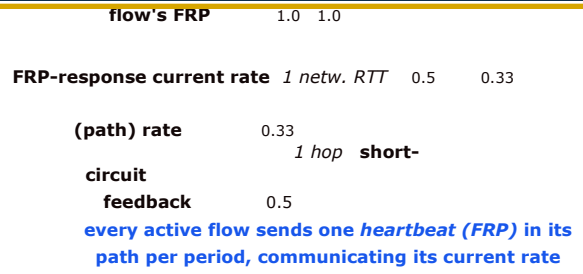


Fig. 1. Heartbeats (one per period per flow) disseminate flows current rates, allowing all links along the flow's path to keep track of the number of active flows and to properly compute their fair shares at the end of each period. The current rate of each heartbeat is updated on every hop such that it does not exceed the fair share of a previously visited link, and is bounced back to the source after reaching the destination, throttling the injection rate of the flow. Short-circuit feedback can shorten the reaction delay when the rate difference is large.

by assigning them a globally feasible fair share, and *(iii)* converges fast (in 20-40μs in our experiments) to a max-min fair rate allocation without relying on per-flow state.

The remainder of this paper is organized as follows. We begin with Section II, describing the new congestion control scheme. In Section III, we use extensive computer simulations to evaluate its performance under various workloads. Then, in Section IV, we present our hardware implementation and its complexity. In Section VI we present related work, and we conclude in Section VII.

II. Congestion control using ACCurate

ACCurate Congestion Control relies on *contention points (CPs)* located in front of network links and *reaction points (RPs)* placed at flow sources, which can control the injection rate of individual flows.

As shown in Fig. 1, time is divided into periods or epochs, which we call *Rate Re-evaluation Periods (RRPs)*. In every RRP, each flow issues one special control packet ("heartbeat"), which we call *Flow Rate Packet (FRP)*. It is important to note that *every active flow issues one FRP per period*. Following the same path as the payload packets of the corresponding flows, FRPs inform the CPs along the flow's path about the flow's current transmission rate. Initially the current rate of an FRP is equal to the maximum transmission rate of the flow as enforced by its source rate limiter. Travelling in its path, the current rate field of FRPs may be updated by contention points, as we describe below.

We assume single-path routing for the packets belonging to the same flow, i.e. all packets of a transfer (or flow) follow the same path inside the network, which is typical for many high-performance networks, such as Infiniband. Note though that there can be many transfers between a source and a destination; these transfers may follow different paths, following ECMP-style routing [14].

The contention points (CPs) rely on three state variables: *(i)* current fair share rate of the link (FSR), *(ii)* number of flows bottlenecked on this link (M), and *(iii)* total capacity of flows bottlenecked elsewhere (B). Receiving an FRP, the CP compares the incoming Current Rate(CR) field against its current FSR:

- If FSR ≤ CR, the flow is considered as bottlenecked on this link; for such locally bottlenecked flows, the CP equates the CR field of the FRP packet to FSR, and increments M by one.
- Otherwise, the flow is considered as bottlenecked elsewhere; in this case, the FRP is left unmodified, while the CP updates $B = B + CR$.

When the destination node receives an FRP packet, it swaps the source-destination header fields of the packet, and injects it back into the network – see Fig. 1. It also sets a special bit in the FRP header to indicate that this is an *FRP-response* packet. The CPs do not process FRP-response packets. Receiving an FRP response, the flow source sets its rate limit to the value indicated by the CR field.

At the end of each RRP, every CP updates its current FSR as:

$$FSR = \frac{C \cdot (1 - \alpha) - B}{M} \quad (1)$$

where C is the capacity of the link, and α is a parameter, typically set between 0.01 and 0.1. Intuitively, the algorithm computes the bandwidth that the (M) locally bottlenecked flows would receive by a fair scheduler, considering also that the present link cannot (and should not) modify the rates of flows bottlenecked elsewhere.

In our algorithm, a portion of the link capacity, $C \cdot \alpha$, is reserved and not allocated to flows, creating headroom to accommodate for control overhead and drain backlogs; increasing parameter α will allow the network queues to drain faster the backlogs formed during congestive episodes.

Fig. 2 presents a CP processing a number of FRP messages. In this example, the RRP period is 20 microseconds, the link capacity C = 10 Gb/s, and the FSR starts at 3 Gb/s. Within the first RRP period, the CP receives several FRPs and updates their CR field. At the end of the FRP, the CP re-calculates the FSR according to Eq. 1: having M = 2 flows bottlenecked here and B = 5 Gb/s from flows bottlenecked elsewhere, the new FSR becomes (10 - 5)/2 = 2.5 Gb/s – for clarity, we assumed that α = 0.

To enable prompt reaction to traffic changes, the RRP must be set low. However, if we set it too small, then FRP overhead will increase. In our experiments, the length of standalone FRP packets is 20 bytes, and the RRP is set at 20 microseconds. Thus the overhead of a single active flow is 20 bytes every 20 microseconds, i.e. 8 Mb/s. This overhead is manageable for up to 100 flows (0.8 Gb/s) passing through a 10 Gb/s link.

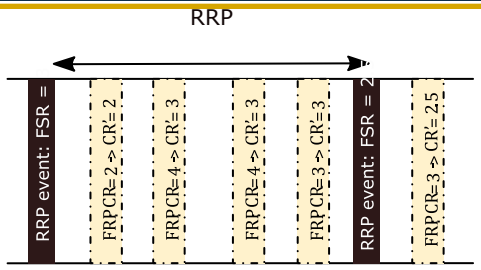


Fig. 2. Timeline of a CP processing a number of FRPs within an 20μsec RRP period; capacity units are measured in Gb/s.

A. Short-circuit notifications for prompt reaction

If a flow starts transmitting with a very high rate, before it receives an FRP reply, it may cause congestion on all links that it passes through.

To slow down such a sudden flow, we use *short-circuit notifications*: when a CP observes a large difference between the current rate of a flow and the FSR variable of the link, it immediately generates an FRP reply and forwards it towards the source. As shown in Fig.1, the source can reduce the flow's rate without having to wait for the destination response, which will arrive after a network round-trip time (RTT). Note that the CP always propagates the FRP to the next switch, in order to allow the flow discover other bottlenecks along its path. Thus, multiple switches may generate a short-circuit notification for the same FRP.

B. Flow init & flow stop messages

When a new flow commences, the CPs inside the network are still unaware of its presence. What will happen in this case is that within the next RRP the flow will issue an FRP. Subsequently, the affected CPs will update their FSR at the end of this RRP, and will start throttling flows in the next RRP, taking into account the presence of the new flow. This reaction latency can induce large backlogs inside the network. To accelerate this procedure, flows inject a *flow-init* FRP together or in advance of the first network packet. Flow-init FRPs trigger *a prompt update of the link's fair share*, and may also cause short-circuit notifications.

If a flow wants to stop transmitting, it sends a Flow Stop FRP message. With this message, it lets the CPs know that its bandwidth can be distributed right on the next RRP, so that flow rates can reach an optimal state, and utilization can reach maximum.

C. Ultra-fast rate recovery

So far we have described the ACCurate mechanisms used to throttle congestive flows. In order to allow throttled flows to recover their rate once congestion has settled, ACCurate uses a positive feedback mechanism: FRP messages also carry a *Desired Rate (DR)* field, which is set by the FRP source and lets the CPs know at what rate the flow wants to transmit, regardless of its current rate (CR). The CPs look at the DR field of FRP packets, and if it is smaller than the FSR, they pass the DR as is; otherwise, the DR field of the packet is set equal to the current FSR. Therefore, if the source gets an FRP with DR greater than the CR, it receives a positive feedback that it can increase its current sending rate to the value indicated by the DR value. This allows a flow's rate to recover as soon as the congestion episode elapses.

III. Performance evaluation

In this section, we use computer simulations in order to test the ACCurate scheme. We use event-driven simulations based on Omnet++ [15] on 10 Gb/s networks. With ACCurate, the flows are initiated by firing up transfers in the multi-channel RDMA engine. The RDMA engine generates network packets with maximum size of 256B, parameter $\alpha = 0.05$, and RRP = 20μsec. The FRP messages travel with high priority in order to bypass backlogs formed inside the network. We also use the implementation of TCP that comes together with the INET library in Omnet. The TCP is configured with maximal receiver buffer capacity of 21KB, no delayed ACKs, maximum segment size of 1500 Bytes and TCPReno congestion control. We test lossless TCP, i.e. TCP in a network with flow control that prevents packet drops, and two types of lossy TCP, one performing tail-drop at switch inputs and one at switch outputs.

A. Solving the parking lot problem

In this experiment, we examine the fairness properties of ACCurate under the parking lot benchmark [16]. We consider a fat-tree network as shown in Fig. 3(a). Two hosts (H1 and H2) connected to the same leaf of the tree send packets (flows f1 and f2) to H4 which resides in a remote leaf. These two transfers cross the same link due to routing conflict (e.g. caused by ECMP or d-mod-k routing). H4 also receives packets from flow f3, coming from a local host (H3). All flows initially send packets at full link speed. Ideally all flows should get an equal share of the link connecting to H4.

First we examine how lossless TCP performs in this scenario. Our results are presented in Fig. 3(b). As can be seen, this configuration fails to deliver desired rates: the remote flows get almost 2x lower bandwidth. Unfortunately, popular hardware congestion control solutions, like DCQCN [8] and QCN [5] also experience similar problems, because the remote flows, facing two bottlenecks, can receive two times more congestion notifications.

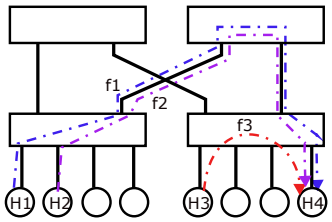
With ACCurate (Fig. 3(c)), the rate allocation is globally fair. All flows receive their max-min fair share 10/3 Gb/s – they get slightly less than that because $\alpha = 0.05$.

B. Victim flow protection, max-min fairness, and rate recovery

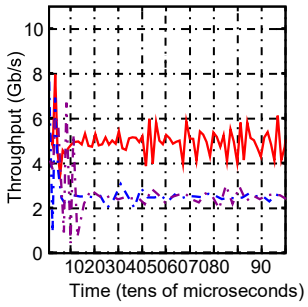
In this experiment, we configured three RDMA flows in the network topology depicted in Fig. 4(a) –for clarity, only a

subset of the network links are shown. Here we test three key properties of ACCurate:

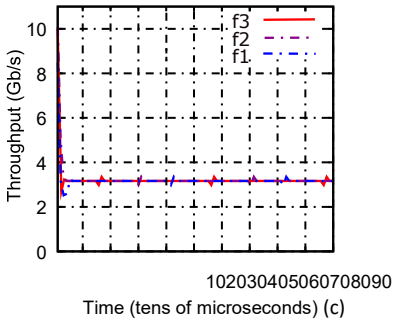
- *protect victim flows that do not contribute to congestion*,
- *redistribute the unused bandwidth in a max-min fair way*,
- *recover the rate of previously congested flows fast*.



(a) Topology & traffic scenario

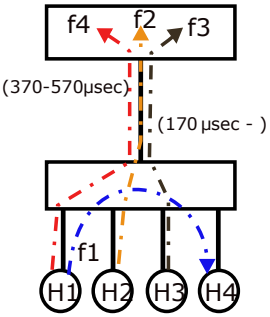


(b) Lossless TCP results

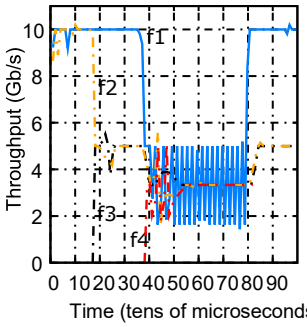


(c) ACCurate results

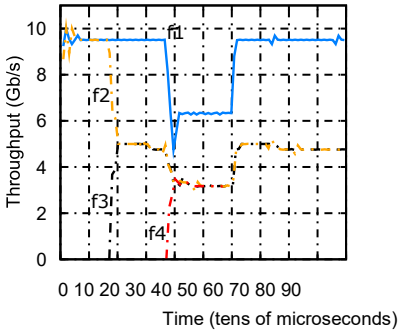
Fig. 3. Performance under the parking lot scenario.



(a) Topology and traffic scenario



(b) Results for Pause-only



(c) Results for ACCurate

Fig. 4. Traffic scenario and time-series depicting the evolution of flows rates.

Flows f1 and f4 are sourced from host H1, but diverge inside the leaf switch. Flows f2 and f3 source from hosts H3 and H4, respectively, in the leaf switch; they travel through the same link with f4 to the spine, where they all diverge to different destinations. The active periods of the flows are described below.

- f1 and f2 are active throughout the experiment, • f3 is activated at time 170 μsec, and stays on till the end,
- f4 is active from 370 to 570 μsec.

Before 170 μsec, only flows f1 and f2 are active. Because these flows have disjoint paths, they can reach full link rate. Problems emerge with flow f3 (170 μsec), which together with f2 congest the leaf to spine link. As shown in Fig. 4(b), in a network with link-level flow control (Pause-only), once f3 becomes active, f2 and f3 get 5 Gb/s because they share a link. Subsequently, a backlog is formed in front of this link, which triggers link-level flow control that moves the backlog upstream at the network interfaces of H2 and H3. However, the backlog does not reach H1, which sources flow f1, because the path of f1 is disjoint from that of f2 and f3. Effectively, the rate of f1 is unaffected.

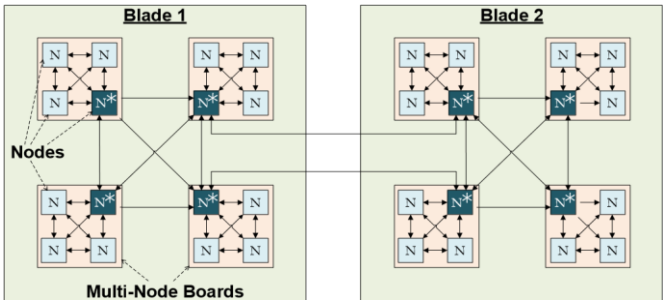
Flow f1 is impacted once flow f4 is activated at time 370 μsec. Since f2, f3 and f4 share a link, they all get a rate of 3.33 Gb/s. Now, due to link-level flow control, the backlog of f4 reaches H1, wherein it strikes the *victim flow f1*, limiting its rate to 3.33 Gb/s. This happens due to head-of-line blocking inside the queue that f1 shares with f4: this queue drains f4 packets at a rate of 3.33 Gb/s, effectively limiting the departure rate of f1 packets as well.

Next, we examine whether ACCurate can correct this behavior. Our results are shown in Fig. 4(c). When the congestion begins at 370 μsec, flow f1 is initially throttled to 5 Gb/s. This is expected because it shares a link with f4. Note that ACCurate removes the head-of-line blocking that could get f1 down to 3.33 Gb/s; hence, ACCurate protects the victim flow.

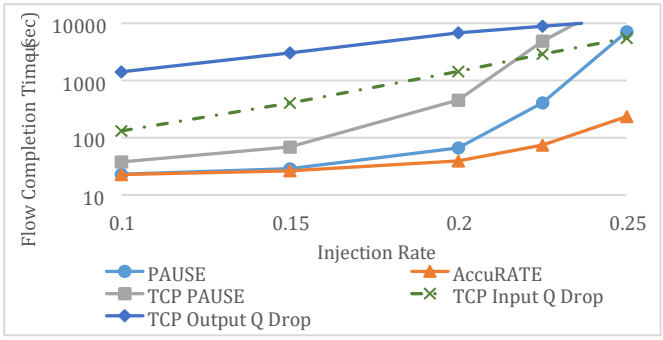
In addition, ACCurate allows flow f1 to take advantage of f4's bottleneck, and use the excess bandwidth: one RRP time after its appearance, flow f4 informs the CP in front of the link connecting H1 with the leaf switch that its current rate is 3.33 Gb/s. Effectively, the CP can in the next RRP period update the desired rate (DR) field of f1's FRPs to 6.66 Gb/s. In this

example, *ACCurate converges to max-min fair rates in two RRP periods*, i.e. just 40 μsec.[1]

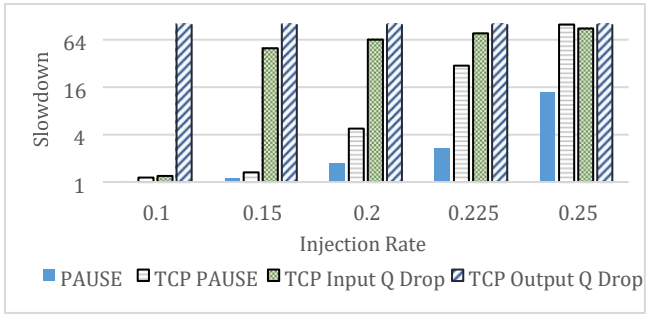
When f4 becomes inactive again at 570 μsec, we want the rates of the other flows to increase to the levels they were before 370μsec. With ACCurate, the rate recovery using the DR positive feedback is very fast – it takes only two RRP periods. With Pause-only, the flows can recover their previous rates only after the network backlogs have drained,



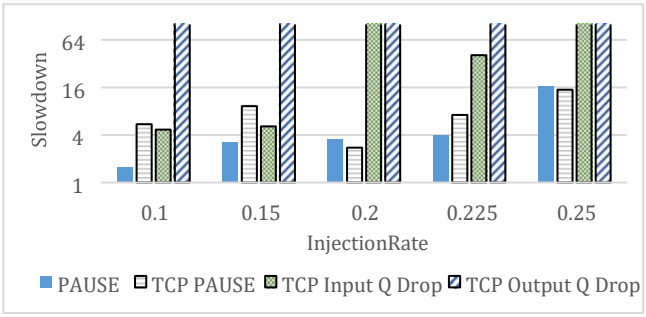
(a) Topology: 2 Blades hosting 4 Multi-Node Groups, each of which hosts 4 nodes. N* nodes are also responsible for network communication.



(b) Average Flow Completion Time (FCT) for critical flows.



(c) Slowdown relative to ACCurate for the 99.9th percentile of FCTs: All Critical flows.



(d) Slowdown relative to ACCurate for the 99.9th percentile of FCTs: flows.

Fig. 5. Simulation results under datacenter-resembling workload in multinode topology.

which takes approximately 150 μsec. With ACCurate, these in-network backlogs are avoided, and the positive feedback from the network allows them to recover the rate limiters.

C. Flow completion times under datacenter workloads

In this experiment, we use an adversarial traffic pattern, inspired by modern datacenter workloads. As described in [4], datacenter workloads exhibit a heavy-tail flow size distribution. In our experiments, the bursty flows (20%) have a size of 1 MB, whereas the small, latency-critical ones (80%) have a size of 20 KB. We use uniformly-destined traffic, and we vary the injection load.We also run experiments with richer packet size distributions (additional sizes) and the main tradeoffs did not change.

We use the hierarchical direct topology shown in Fig. 5(a). Each computing node generates traffic either from an RDMA interface or an Ethernet interface (TCP tests) at 10 Gb/s. Four (4) computing nodes are grouped together in a *MultiNode Group* (MNG). The nodes inside each MNG are fullyconnected, using 10 Gb/s High Speed Serial (HSS) links. One node in each MNG additionally implements inter-MNG connectivity (shown as N*). At the inter-MNG level, sets of four (4) MNGs form a *Blade group*. Each blade/group has its MNGs fully-connected through HSS links. In this experiment, we configured two blades, connected using two HSS links.

In Fig. 5(b) we depict the latency of critical (small) flows. For uniformly destined traffic, the links connecting the two blades are congested once we reach the 0.25 injection load per node – half of the traffic is local within each blade, therefore the 16 local nodes generate $16 \times \frac{10}{2}$ Gb/s traffic that crosses the two links, generating a 4x overload on each one. As can be seen, ACCurate outperforms all other variants. The TCP variants perform quite poorly even at very low loads. As we approach the saturation point, ACCurate delivers 5x to 10x lower latency than all other schemes.

Next, we measure the tail latency. In these results, ACCurate's performance shines thanks to precise rate limiting that keeps backlogs out of the fabric. In Fig. 5(c), we plot the slowdown in 99.9-th percentile latency for all schemes compared to ACCurate. As can be seen, lossy TCP performs poorly even at low loads. Pause-based schemes (with or without TCP) perform better at low loads, but their

[1] The exact rates are a bit lower due to α begin equal to =0.05.

performance suffers under high load. By avoiding backlogs and congestion spreading, *ACCurate reduces the tail latency of all flows*. In Fig. 5(d), we consider only small latency-critical flows. As can be seen, the slow down is now considerable starting from low loads. Only PAUSE keeps up, but again the slowdown is almost 4x even at low loads. Overall, ACCurate reduces the latency variations that contribute to large tail latency.

IV. Hardware Implementation

One concern with congestion control mechanisms is that they often require complex hardware blocks inside the network or interconnect [17]. ACCurate also implements some functions inside network nodes, but does not to use expensive per-flow state. To demonstrate the simplicity of hardware block needed, we have described the CP functionality in Verilog, and placed and routed the design in a Zynq Ultrascale+ FPGA.

The contention points sit on every switch output. Their purpose is to throttle the monitor FRP packets, and to update their CR and DR fields. The packets that pass through a CP are divided into four groups:

- Normal, payload packets that pass through the switch are not processed.
- FRPs that are processed as explained in Section II and further detailed in this section.
- FRP-responses, i.e. FRPs that reached their destination and are now traveling back to their source, are not processed.
- Flow-init and flow-stop messages that inform the network about a new or a finished transmission and need special processing.

The hardware circuit that implements the CP is coarsely depicted in Fig. 6. Our implementation is generic and can be made to work for any custom interconnect or popular Layer-2 network, such as Ethernet or Infiniband, which may encode the FRP packet fields differently. We deliberately ignore needed signals, such as start-of-packet, end-of-packet, and reset.

Inputs to our circuit are the fields of an FRP packet and the RRP clock. The CP also maintains variables M, B and FSR, as explained in Section II. The circuit estimates the FSR of the link, and updates the CR and DR fields of FRP packets.

Upon receiving an FRP packet, a CP first determines if it is a flow-init or a flow-stop message. New flows must not wait for an RRP clock event in order to get a rate assigned to them; nstead, the CPs may give them a feasible rate immediately. Unfortunately, in every RRP clock event, the M and B variables of the CP are reset to zero (0) and therefore the CP cannot assign a legitimate rate to these new flows without waiting for all flows to send their FRPs, which will only happen at the end of the RRP. To counter this, the CP keeps the *old* (previous RRP) FSR, M and B values in registers. It then assigns a new sending rate to the new flow, considering what its rate should be if it was active in the previous RRP: the CP compares the CR field with the old FSR, and correspondingly updates the old M and B values. It then calculates a new FSR using these updated values.

If the message is a forward FRP, the CP compares the current FSR with the CR. If FSR ≤ CR, the CP increases M and updates the CR field of the packet with FSR. Otherwise, it increases B by the CR value of the packet.

At the end of the RRP, the CP calculates the new FSR value and keeps the old values M, B and FSR variables in its "old" registers; these are not displayed in the figure for clarity.

We have run the place and route process for the CP circuit for a Zynq Ultrascale+ FPGA [18] using Vivado. Our design occupies less than 0.1% of the FPGA (286 LUTs and 90 flipflops), and can achieve the frequency of 156.25 MHz that we targeted in our implementation; thus, our design can handle a new FRP message every 6.4 nanoseconds. In a crossbar switch consisting of 16 ports, the CPs will occupy less than 2% of the FPGA resources.

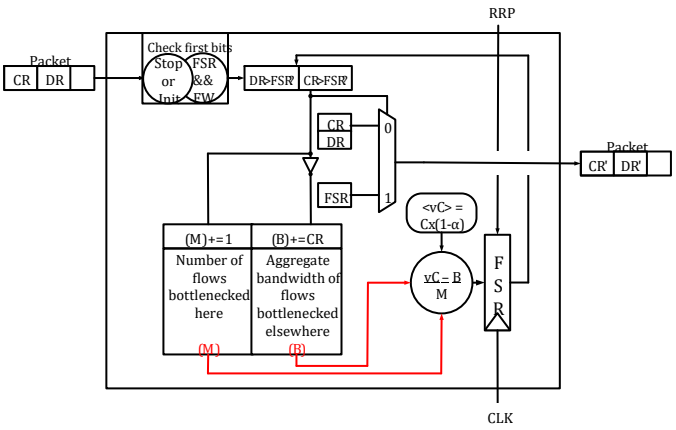


Fig. 6. Outline of RTL implementation of contention points.

A. Handling corner cases

Our circuit also handles a number of corner cases which we describe next. The aggregate capacity (B) of flows bottlenecked elsewhere may exceed the total capacity(C) when many flows start transmitting simultaneously, thus not allowing the system to realize where the real bottleneck is. In this case, Eq. 1 yields a negative FSR. To handle this case, we temporarily set the FSR equal to C divided by the total number of flows passing through.

Another corner case is when $M = 0$. In this case, Eq. 1 will attempt division by zero. We handle this as follows: if $B = 0$, then no flow is active; if instead $B \neq 0$, we use the maximum of the bottlenecked elsewhere rates, b_{max} (the circuit keeps

this in a register, and resets it in every RRP), set $M = 1$, and decrement B by b_{max}.

B. Reaction points

The reaction points in ACCurate can be implemented on the multi-channel RDMA sources residing on network interfaces. Below we outline their operation, leaving a more detailed description for future work.

We consider that every flow (i.e. transfer) is allocated a separate RDMA channel. The reaction point injects an FRP message for each active channel per RRP, and limits its injection rate to not exceed the value specified by FRP responses. In principle, it is preferable to limit flow rates on a per packet-basis (packet packing), allowing only small bursts (1-2 packets) to jump into the network. Rate limiting can also be combined with flow scheduling. The idea is to maintain a *next-service-time (NST)* per flow, and block a flow with NST > T, where T is the current time. After serving a packet from a flow, $NST = T + \frac{S}{R}$, where S is the packet size and R is the current rate limit. A packet scheduler selects the flow (i.e. DMA channel) with the minimum NST; if this minimum NST exceeds the current time, then no rate-limited flow is eligible for service.

V. Discussion

In this section, we discuss a number of design decisions taken by ACCurate.

Reaction control delay: With ACCurate, flows adjust their rates on RRP. With a relatively short RRP (a few tens of microseconds, depending on the number of flows expected – 20μsec in our experiments), we maintain a fairly fast reaction to network changes. The added benefit is that allowing rate adjustments only on RRP intervals, we maintain accurate rate assignments, which take all flow rates into account. Multi-path routing: ACCurate assumes that all packets of a rate-controlled transfer follow the same path. To enable multipath routing, different transfers between a pair of endnodes may use different paths. In order to avoid sudden path overshoots caused by large transfers, we can break the transfer in smaller ones that follow a single path each.

Congestion control for Network-on-chip (NOC): In this paper, we described and evaluated ACCurate for off-chip interconnects. However, ACCurate is also applicable for onchip networks, implementing wormhole or virtual cut-through routing. NOCs traditionally adopt RDMA mechanisms for data transfers. Typically, they have smaller buffers, but this does not mean that congestion control is not needed [19], [20]. Unlike QCN and DCQCN, ACCurate does not rely on backlog size to quantify congestion, and therefore is applicable for ultra-shallow buffer networks, like NOCs. Furthermore, it is straightforward to implement a global clock needed for the RRP in NOCs.

Inactive or small/slow flows: Flows that are too slow or small may not send FRP messages, as they do not interfere with other flows. For instance, one may allow flows smaller than 4KB to proceed without sending FRPs. The same can be configured for flows with small injection bandwidth. In this way, the CPs will not be informed about the current rate of these flows. As described in [4], small flows contribute little in overall traffic, as most of network traffic comes from large bulky flows, which we control with ACCurate.

Max-min fair versus shortest-job-first scheduling: Our protocol allocates equal shares of the link capacity to competing flows using processor sharing (PS). A competing paradigm is shortest-job-first (SJB) scheduling, which can reduce the flow completion times [11]. We consider that when two large storage / data flows conflict on a link, it is preferable to share the bandwidth half-half at all times (PS), rather than having one waiting for the other to complete (SJB). Eliminating the in-network backlogs using rate regulation already reduces the flow completion times. In addition, we do not rate-limit small messages. In future work, we will consider how to further optimize the FCT of latency critical flows.

Reducing the FRP overhead: As said previously, the overhead of FRP is one FRP per RRP per link. For an FRP size of 20 Bytes and RRP of 20 μsecs, the traffic per rate-limited flow is just 8 Mb/s. As described in [4], a large portion of the flows found in a datacenter are relatively small, and we do not have to limit their rate. To further reduce the overhead, one may increase the length of the FRP period, taking into account that doing so will increase the reaction delay of the scheme. Also, one may try to send the FRP information using available bits in the header or footer of a packet. Encoding the current and the desired rates using 8 bits for each, and adding 4 more bits to encode the type of packet (FRP, FRP-response, start/end flow), we need 20 bits. The downside is that an FRP may delay due to backlogs inside the network.

Broadcasting a common RRP clock event: Our protocol assumes that a central clock generates synchronous events to all modules with a frequency of 50kHz for an RRP of 20 μsecs. In general LAN networks, the Precision Time Protocol (PTP) is already approaching this level of accuracy [21]. For rack-scale interconnects, a custom protocol can be used to achieve this distribution of a common clock. For instance, the BXI interconnect from Atos broadcasts a common clock with an accuracy of 2 μsecs [22]. As said above, broadcasting the RRP clock event is straightforward in NOCs.

Implementing CP circuits using Open-Flow: The CP logic required inside switches by ACCurate is very simple. If custom hardware is not an alternative, it may still be possible to implement it using advanced, programmable OpenFlowcapable switches [23].

VI. Related work

Quantized Congestion Notification control (QCN) has been proposed for lossless Converged Enhanced Ethernet (CEE) [5]. QCN and its derivatives detect congestion based on buffer queue occupancy and occupancy variation inside routers, and send notification messages to the sources in order to slow down in the event of congestion. One big advantage of our scheme is that it is more proactive: we do not wait for queues to built up in order to detect congestion and throttle flows. In addition, in our scheme we use absolute rate assignments, whereas QCN uses a stochastic process in order to decide which flow to throttle, and relies on additive-increase multiplicativedecrease (AIMD) feedback control in order to adjust the flows' transmission rates. The combination of the two bring uncertainty which can have detrimental effects on performance for certain workloads.

Like ACCurate, QCN generates congestion notifications; however, QCN decides to throttle a random subset of the incoming flows, which can lead to grossly unfair schedules, whereas the present scheme selectively activates the prompt feedback paths, exactly on the cases when this is mostly needed, producing fair allocations.

Today, the datacenters are dominated by TCP congestion control. Recently, lossless TCP has been tested as a way to avoid re-transmission latency and effectively to optimize the latency of critical flows [10]. Lossleness is also a key requirement of RDMA networks. Our results in this paper showed that ACCurate provides up to 10x faster completion times than lossless TCP. Other TCP derivatives have also been proposed that reduce the backlogs created inside the networks, thereby reducing latency [4]. The software stack of TCP however places a large overhead on communication, which we override completely using RDMA engines.

Recently, DCQCN was proposed for hardware-level congestion control in L3 datacenter networks [8]. DCQCN combines features from QCN and DCTCP, providing hardware rate limiters, and can be used for RDMA transfers that bypass the linux network stack. One drawback of the scheme is that it has a large number of control variables, which may be difficult to tune in different environments. In addition, DCQCN cannot provide max-min fair allocation, thereby needlessly penalizing some flows in multi-bottleneck (parking lot) scenarios, which ACCurate handles smoothly. Similar to QCN, DCQCN does not use positive feedback, and relies on additive increase, which typically takes longer to recover rates. In this paper, we showed that with ACCurate, flows can recover to new fair share rates within one or two RRP times.

Another alternative is the Infiniband Congestion Control. This congestion control uses a table at sources to store congestion signals from the network, and uses this feedback to throttle flows. However, Infiniband has a lot of parameters that may be difficult to tune correctly with the risk of being unfair [24].

A proactive congestion control scheme is presented in [12]. This scheme relies on per-flow state inside the network, and uses a complicated formula to compute link fair shares, without demonstrating its implementation. In addition, in [12] packets carry a rate demand per link, which creates additional overhead. In contrast, ACCurate relies on minimal hardware resources, making it applicable in demanding environments.

VII. Conclusion

Modern high-speed networks shift away from traditional TCP-based communication, adopting RDMA transfers in order to achieve lower latency. These networks need hardware-based congestion management in order to deal with saturation trees. In this work we described ACCurate, an efficient and fair congestion control scheme that can be readily implemented in hardware. Our simulation results demonstrate that ACCurate reacts promptly to congestion, throttles the offensive flows by allocating bandwidth in a max-min fair manner, and reduces the flow completion time by more than one order of magnitude under demanding realistic workloads. In our implementation for a Xilinx Ultrascale+ FPGA, the CPs needed at the outputs of a 16-port switch occupy less than 2% of the FPGA resources.

References

[1] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," ser. ACM ASPLOS XVII, 2012.

[2] N. Rajovic *et al.*, "Tibidabo1: Making the case for an arm-based hpc system," *Future Generation Computer Systems*, vol. 36, 2014.

[3] H. Fu *et al.*, "The sunway taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, 2016.

[4] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *ACM SIGCOMM*, ser. SIGCOMM '10, 2010.

[5] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data center transport mechanisms: Congestion control theory and ieee standardization," in *Allerton Conference on Communication, Control, and Computing*, Sept 2008.

[6] G. F. Pfister, "An introduction to the infiniband architecture," *High Performance Mass Storage and Parallel I/O*, vol. 42, pp. 617–632, 2001.

[7] M. S. Birrittella *et al.*, "Intel omni-path architecture: Enabling scalable, high performance fabrics," in *IEEE High Performance Interconnects*, 2015.

[8] Y. Zhu *et al.*, "Congestion control for large-scale rdma deployments," in *ACM SIGCOMM CCR, 2015*.

[9] B. Arimilli *et al.*, "The PERCS High-Performance Interconnect," in *Proc. IEEE Hot Interconnects*, 2010.

[10] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," in *ACM SIGCOMM 2012*.

[11] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *ACM SIGCOMM CCR*, 2014.

[12] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High speed networks need proactive congestion control," in *ACM Workshop on Hot Topics in Networks, 2015*.

[13] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, “Less is more: trading a little bandwidth for ultra-low latency in the data center,” in *USENIX NSDI*, 2012.

[14] N. Chrysos *et al.*, “Large switches or blocking multi-stage networks? an evaluation of routing strategies for datacenter fabrics,” *Computer Networks*, vol. 91, pp. 316–328, 2015.

[15] A. Varga and R. Hornig, “An overview of the omnet++ simulation environment,” in *Proc. Simulation tools and techniques for communications, networks and systems & workshops*, 2008, p. 60.

[16] D. A. Hayes, D. R. Sanchez, L. Andrew, and S. Floyd, “Common tcp evaluation suite,” Ph.D. dissertation, 2013.

[17] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. Garc´ıa, and T. N. Frinos,´ “A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks,” in *IEEE HPCA*, 2005.

[18] V. Boppana, S. Ahmad, I. Ganusov, V. Kathail, V. Rajagopalan, and R. Wittig, “Ultrascale+ mpsoc and fpga families,” in *IEEE Hot Chips, 2015*.

[19] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 3–21, 2009.

[20] N. Alfaraj, J. Zhang, Y. Xu, and H. J. Chao, “Hope: Hotspot congestion control for clos network on chip,” in *IEEE/ACM International Symposium on Networks on Chip (NoCS)*, 2011.

[21] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kero,¨ “Towards high accuracy in ieee 802.11 based clock synchronization using ptp,” in *IEEE Precision Clock Synchronization for Measurement Control and Communication (ISPCS)*, 2011.

[22] P. Vigneras and J.-N. Quintin, “The bxi routing architecture for exascale´ supercomputer,” *The Journal of Supercomputing*, vol. 72, no. 12, 2016. [23] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[24] E. G. Gran, E. Zahavi, S. A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, “On the relation between congestion control, switch arbitration and fairness,” in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2011.