RESEARCH-ARTICLE

Uno: A One-Stop Solution for Inter- and Intra-Data Center Congestion Control and Reliable Connectivity

TOMMASO BONATO, Microsoft Corporation, Redmond, WA, United States

SEPEHR ABDOUS, Johns Hopkins University, Baltimore, MD, United States

ABDUL KABBANI, Microsoft Corporation, Redmond, WA, United States

AHMAD GHALAYINI, Microsoft Corporation, Redmond, WA, United States

NADEEN GEBARA, Microsoft Corporation, Redmond, WA, United States

TERRY LAM, Microsoft Corporation, Redmond, WA, United States

View all

Published: 16 November 2025

Citation in BibTeX format

SC '25: The International Conference for High Performance Computing, Networking, Storage and Analysis
November 16 - 21, 2025
MO, St. Louis, USA

Conference Sponsors:
SIGHPC

https://doi.org/10.1145/3712285.3759884
ISBN: 9798400714665

Uno: A One-Stop Solution for Inter- and Intra-Data Center Congestion Control and Reliable Connectivity

Tommaso Bonato*
ETH Zürich
Zurich, Switzerland
Microsoft Corporation
Redmond, USA
tommaso.bonato@inf.ethz.ch

Sepehr Abdous*
Johns Hopkins University
Baltimore, USA
Microsoft Corporation
Redmond, USA
sabdous1@jh.edu

Abdul Kabbani
Microsoft Corporation
Redmond, USA
abdulkabbani@microsoft.com

Ahmad Ghalayini
Microsoft Corporation
Redmond, USA
aghalayini@microsoft.com

Nadeen Gebara
Microsoft Corporation
Redmond, USA
nadeengebara@microsoft.com

Terry Lam
Microsoft Corporation
Redmond, USA
thelam@microsoft.com

Anup Agarwal
Carnegie Mellon University
Pittsburgh, USA
anupa@cmu.edu

Tiancheng Chen
ETH Zürich
Zurich, Switzerland
tiancheng.chen@inf.ethz.ch

Zhuolong Yu
Microsoft Corporation
Redmond, USA
zhuolongyu@microsoft.com

Konstantin Taranov
Microsoft Corporation
Redmond, USA
kotaranov@microsoft.com

Mahmoud Elhaddad
Microsoft Corporation
Redmond, USA
maelhadd@microsoft.com

Daniele De Sensi
Sapienza University
Rome, Italy
daniele.desensi@uniroma1.it

Soudeh Ghorbani
Johns Hopkins University
Baltimore, USA
soudeh@soudeh.net

Torsten Hoefler
ETH Zürich
Zurich, Switzerland
torsten.hoefler@inf.ethz.ch

Abstract

Cloud computing and AI workloads are driving unprecedented demand for efficient communication within and across datacenters. However, the coexistence of intra- and inter-datacenter traffic within datacenters plus the disparity between the RTTs of intra- and inter-datacenter networks complicates congestion management and traffic routing. Particularly, faster congestion responses of intra-datacenter traffic causes rate unfairness when competing with slower inter-datacenter flows. Additionally, inter-datacenter messages suffer from slow loss recovery and, thus, require reliability. Existing solutions overlook these challenges and handle inter- and intra-datacenter congestion with separate control loops or at different granularities. We propose Uno, a unified system for both inter- and intra-DC environments that integrates a transport protocol for rapid congestion reaction and fair rate control with a load balancing scheme that combines erasure coding and adaptive routing. Our findings show that Uno significantly improves the completion times of both inter- and intra-DC flows compared to state-of-the-art methods such as Gemini.

*Both authors contributed equally to this research.

SC '25, St Louis, MO, USA

ACM ISBN 979-8-4007-1466-5/25/11
https://doi.org/10.1145/3712285.3759884

CCS Concepts

• **Networks** → **Network control algorithms**.

Keywords

datacenter congestion control; distributed training; erasure coding

ACM Reference Format:
Tommaso Bonato, Sepehr Abdous, Abdul Kabbani, Ahmad Ghalayini, Nadeen Gebara, Terry Lam, Anup Agarwal, Tiancheng Chen, Zhuolong Yu, Konstantin Taranov, Mahmoud Elhaddad, Daniele De Sensi, Soudeh Ghorbani, and Torsten Hoefler. 2025. Uno: A One-Stop Solution for Inter- and Intra-Data Center Congestion Control and Reliable Connectivity. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3712285.3759884

1 Introduction

With the drastic growth in cloud computing, HPC, and AI workloads, ensuring congestion-free communication and efficient traffic routing both inside and across multiple datacenters (DCs) is becoming more crucial than ever [25, 37, 55]. Specifically, reports from Google highlight a 100× increase in their inter-DC WAN traffic volume during a five-year period [36]. Additionally, with the growth of large-scale AI models, fitting entire training jobs inside a single datacenter is becoming infeasible [48], *e.g.,* Google's Gemini was trained on several Google supercomputers [29], and, more recently, OpenAI used several clusters to train its GPT-4.5 model [54].

Many congestion control protocols have been developed throughout the years to separately ensure efficient communication in intra-DC [3, 9, 11, 41, 43, 47, 51, 61, 62] and inter-DC [20, 35, 37, 40, 59]

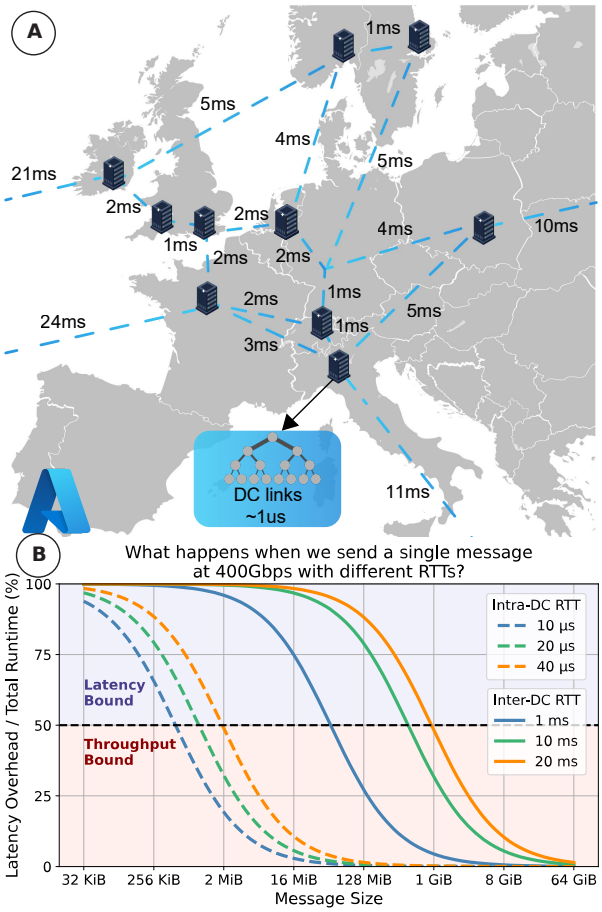

Figure 1: (A) shows inter-DC links for Azure in Europe and their delay assuming point-to-point connections. (B) shows that inter-DC links make even medium-large messages latency-bound.

environments, but very little research has been done on simultaneously handling both [63]. While inter- and intra-datacenter traffic are usually treated as separate entities, they co-exist within datacenters and compete over resources. Therefore, it is crucial to ensure efficient communication for each entity while also considering the other entity [63]. However, doing so is challenging due to the inherent differences between datacenter networks and WANs. In particular, within a single datacenter, cable lengths and propagation delays are small and, mostly, homogeneous. However, inter-datacenter WANs are built using long physical links with large propagation delays [59, 63]. Additionally, inter-DC links traverse different geographical paths thereby introducing heterogeneity in link propagation delays and additional risks of failures.

Unlike datacenter networks, in inter-DC WANs, the completion time of even large messages is bounded by latency rather than throughput due to the large propagation delays. Particularly, in a modern datacenter infrastructure, it might take at most tens of microseconds for a packet to go from any given node to any destination, assuming a lightly-loaded network [23, 51]. On the other hand, when going across datacenters, such delay increases dramatically to multiple milliseconds, *e.g.,* Figure 1 (A) presents link delays between Microsoft Azure's datacenters across Europe. Figure 1 (B) presents the percentage of a message's completion time, *i.e.,* time taken from sending the first packet of the message to receiving the last ACK, that is due to the aggregate propagation delay across distinct message sizes and intra- and inter-DC propagation delays (indicated as RTTs). For typical intra-datacenter RTTs (*i.e.,* 10μs to 40μs [41, 51]), as we increase the message size, the completion time quickly becomes dominated by the sending throughput for sizes greater than 256 KiB. On the other hand, for inter-datacenter RTTs (*i.e.,* 1 ms to 60 ms [59, 63]), the completion time becomes mostly bounded by the propagation delay. For instance, when the inter-DC RTT is 20 ms, the completion time is dominated by propagation delay if messages are smaller than 1 GiB, which is quite large (message sizes recorded from Alibaba's inter-DC traces are all smaller than 300 MB [65]). Moreover, with increasing link bandwidths, this will only become more extreme in the future. While this theoretical study gives us some useful insights, we note that actual bounds would slightly change depending on the network's conditions.

This massive delay gap between intra-DC and inter-DC networks introduces several challenges for simultaneous and efficient congestion management of both intra- and inter-DC traffic:

1) Diverse congestion feedback granularity: *The congestion feedback loop for inter-DC flows is significantly delayed compared to intra-DC traffic.* Consequently, upon receiving a congestion signal for inter-DC traffic, it is hard to know if the path is still congested. On top of that, the delay mismatch makes it hard to maintain fairness between intra- and inter-DC flows while competing over a bottleneck link.

2) BDP heterogeneity: *Inter-DC Bandwidth Delay Product (BDP) is significantly larger than intra-DC BDP due to its long RTTs, e.g.,* with 10 ms RTT and 400 Gbps link bandwidth, the inter-DC BDP is ≈ 500 MB. While commodity switch buffers have increased in size during the years [13], they are still quite small, especially intra-DC switch buffers, compared to inter-DC BDP. Meanwhile, many congestion control algorithms [9, 61, 66] assume the capacity of switches to be at least a fraction of BDP, *e.g.,* 17% for DCTCP [9]. While this assumption holds for intra-datacenter flows, whose BDPs are typically less than 1 MiB, it becomes unrealistic for inter-DC flows. The lack of buffering space in switches is amplified as cloud providers are using shallow-buffered commodity switches to be cost-efficient and improve scalability [26, 37].

3) Inefficient loss handling: *Packet loss and its consequent retransmission significantly increase the message completion time in latency-bound WANs.* Even with advanced loss detection mechanisms such as packet trimming [33], the loss notification mechanism takes a long time due to the large propagation delay. Therefore, deploying efficient loss recovery mechanisms traffic is necessary.

While existing proposals, such as Gemini [63] and BBR [20], try to tackle some of these challenges, to the best of our knowledge, no proposals have addressed them all. Specifically, BBR is dedicated only to WAN traffic and requires another transport, such as DCTCP [9], to handle intra-DC traffic. With separate transports, it is challenging to guarantee fairness. Gemini, on the other hand, is a window-based congestion control for both intra- and inter-DC communication that is proven to achieve bandwidth fairness among intra- and inter-DC flows. However, as Gemini's granularity for reacting to intra-DC and inter-DC congestion signals significantly varies, Gemini experiences slow convergence to fairness and is

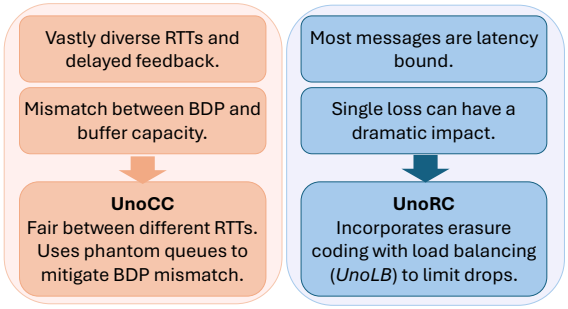

Figure 2: To ensure efficient communication within and across datacenters, Uno integrates congestion control, load balancing, and loss resiliency.

prone to network under-utilization (§2). Lastly, all these techniques suffer from inefficient loss handling for WAN traffic. To resolve these problems, we introduce Uno, a system that tightly integrates congestion control, load balancing, and loss resiliency to create a unified solution for both intra- and inter-DC communication. As shown in Figure 2, Uno employs two key components:

1) Congestion Control component (UnoCC): When messages become throughput-bound and congestion control becomes vital, UnoCC exploits Explicit Congestion Notification (ECN) to efficiently handle congestion both inside and across datacenters and simultaneously provide low latency and flow-level fairness for intra-DC and inter-DC traffic. To ensure effective congestion management across datacenters, Uno uses phantom queues [10], *i.e.,* virtual queues with arbitrary sizes that mimic the behavior of physical queues, to match the high BDPs of the inter-DC connections regardless of the physical queue capacity.

2) Reliable Connectivity component (UnoRC): Since most messages are latency-bound when traversing inter-DC WAN, Uno augments message transmission with *erasure coding* [52] to increase loss recovery without packet re-transmission for cross datacenter communication. Furthermore, Uno integrates the erasure coding logic with a sub-flow level load balancing (UnoLB) scheme that leverages multi-pathing of modern networks.

We evaluate Uno using htsim simulations [33] and compare it against MPRDMA+BBR ([47]+[20]) and Gemini [63]. Our results show that Uno significantly improves latency, fairness, and loss resiliency. For instance, under 60% load and a mixture of both inter- and intra-DC workloads, Uno improves the 99^{th} percentile FCT by 31% and 30% compared to BBR+MPRDMA and Gemini, respectively.

2 Coexistence among inter- and intra-DC traffic: challenges and opportunities

This section outlines the challenges introduced by the inherent heterogeneity between intra-DC and inter-DC propagation delays.

2.1 Diverse congestion feedback granularity

Typically, the workload inside a datacenter is comprised of both intra-DC and inter-DC traffic. Meanwhile, there exists a huge gap between the congestion feedback granularity of inter- and intra-DC flows due to the significant difference between their propagation delays [63]. For instance, considering intra-DC RTT of 10μs [41, 51] and inter-DC RTT of 10ms [59, 63], every inter-DC RTT corresponds to 1000 intra-DC RTTs. This means that intra-DC flows *potentially* receive congestion signals 1000× more frequently than inter-DC flows. Therefore, in cases of network congestion caused by a mixture of inter- and intra-DC traffic, the intra-DC flows adjust their rates more frequently than inter-DC flows which can potentially victimize intra-DC traffic and hurt flow-level fairness.

Furthermore, as shown in Figure 4 Ⓐ in §3, the large gap between the propagation delay of inter- and intra-DC traffic can cause network under-utilization and queue occupancy oscillations in steady state. Specifically, the congestion feedback for inter-DC flows can be piggybacked to the senders long after the actual congestion had been resolved by intra-DC flows adjusting their rates. In such scenario, reducing the send rate of inter-DC flows upon receiving the congestion signals creates long periods of under-utilization before the flows ramp up and fill the excess network capacity.

2.2 Heterogeneous hot spots

Commodity switches, especially those deployed in inter-datacenter WAN, limit the signals used for congestion detection [63]. Therefore, packet loss, delay, and Explicit Congestion Notification (ECN) are commonly used to detect congestion both inside and between datacenters [9, 41, 47, 51, 61, 63, 66]. Solely relying on packet loss and packet re-transmission for detecting congestion and reacting to it imposes significant extra latency as it is only triggered when the switch buffers are extremely congested [9, 41, 61].

With ECN as the congestion signal, it is difficult to properly set the ECN marking threshold when having mixed inter-DC and intra-DC traffic [63]. This is because the buffer capacity of WAN switches can be larger than that of the switches deployed inside datacenters, as is the case in Gemini [63]. Additionally, inter-DC Bandwidth Delay Products (BDPs) are usually much larger than intra-DC BDPs. Therefore, inter-DC traffic requires much larger ECN marking thresholds compared to traffic staying within the datacenter. With delay as the congestion signal, it is challenging to distinguish inter-datacenter hot spots from intra-datacenter. Specifically, it is non-trivial to know whether the increased delay indicates extreme congestion in shallow-buffered intra-DC switches or minor congestion in deep-buffered inter-DC switches. Using shallow buffers everywhere could help the delay signal but it would still be a noisy signal due to the large inter-DC latency. Annulus [59] introduces Quantized Congestion Notification (QCN) [7, 38] to detect early congestion for inter-DC flows. However, it only helps if the congestion happens near source before crossing the datacenter boundary since it relies on sending an early warning on the reverse path from the congestion hot spot to the source.

2.3 BDP heterogeneity

Most reactive congestion control protocols assume a minimum amount of available buffer capacity for proper operation, *e.g.,* DCTCP [9] requires the buffer space to be at least 17% of BDP. If we consider the latest switches from Broadcom such as the Trident4 [2], we are given ~ 4 MB of buffering per port. Assuming 10 ms inter-DC round-trip delay and 400 Gbps link bandwidth, DCTCP requires at least ~ 100 MB of buffering per port, which is much larger than today's switching fabric. Accordingly, intra-DC switches cannot

support inter-DC traffic that co-exists with intra-DC traffic inside datacenters. This gap is likely to increase even further as the network bandwidth keeps increasing at a faster rate than the buffering sizes [59].

Furthermore, having a buffer significantly smaller than the BDP makes it harder to properly assess the extent of the congestion, as small changes in the sending rate can easily result in either over- or under-utilization of the network. Therefore, in §4, we re-purpose phantom queues [10], *i.e.,* virtual queues that were originally designed to provide low-latency within a datacenter, for inter-datacenter communication to easily match the inter-DC BDP.

2.4 Inefficient loss handling

As discussed previously, most of the message sizes that cross WAN links are bounded by the propagation delay. This has a very important implication: a single packet loss could significantly increase message delivery time since detecting an inter-DC packet loss and retransmitting it is proportional to the large inter-datacenter RTT. To shed more light on the importance of efficiently handling loss for inter-DC flows, we measure the failure rates between pairs of cloud VMs located in different regions of North America. More specifically, we set up a simple RDMA API that sends 320 million 2KiB packets between the pairs of datacenters. The first selected pair of datacenters (Setup 1) has an RTT of ≈ 65 ms and an average loss rate of 5.01×10^{-5} while the second pair (Setup 2) observes an RTT of ≈ 33 ms and an average loss rate of 1.22×10^{-5}. We observe that while losses are rare, they can impose significant extra latency. Thus, fast loss resolution is crucial for inter-DC traffic. In addition to measuring the overall loss rate, we grouped the packets into consecutive chunks of 10 packets and determined the probability of losing more than one packet within the chunks. The results (Table 1) uncovered that link-correlated drops within a chunk exist, implying that a multi-link failure resilient scheme is preferred.

Losses Within a Block	Setup 1 (65ms RTT)		Setup 2 (33ms RTT)	
	Drops	Loss Rate	Drops	Loss Rate
1	97 403	3.0×10^{-4}	12 785	4.0×10^{-5}
2	23 984	7.5×10^{-5}	7 262	2.3×10^{-5}
3	5 007	1.6×10^{-5}	1 560	4.9×10^{-6}

Table 1: Packet loss information for two datacenter configurations.

3 Uno Design Goals

Before outlining the details of our design in §4, this section highlights the goals we aim to achieve by proposing Uno.

3.1 Unified congestion control logic

As discussed in the previous section, in a congested network, there is a massive gap in the granularity at which transports such as Gemini [63] react to congestion signals for inter- and intra-DC flows, which can potentially victimize intra-DC flows. At the same time, the scale of AI training tasks is exceeding the resources available in one datacenter, meaning that messages of the same importance can flow both within and across datacenters. Therefore, ensuring bandwidth

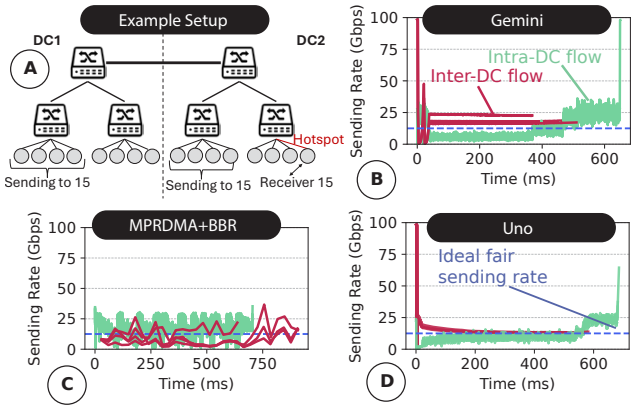

Figure 3: Gemini and MPRDMA+BBR fall short in efficiently achieving bandwidth fairness while Uno provides fast convergence to fairness during a mixed incast scenario.

fairness among inter- and intra-DC flows and fast convergence to the bandwidth fair share is critical [63][1].

Alas, existing solutions [20, 59, 63] either do not achieve bandwidth fairness as they separate congestion control for inter- and intra-DC flows, *e.g.,* applying BBR [20] for inter-DC and DCTCP [9] for intra-DC traffic, or experience slow convergence time because they react to inter- and intra-DC congestion at different granularities. To illustrate this, we simulate two 8-ary fat-tree datacenters [5] connected by eight 100 Gbps links via two border switches [63], with inter-DC RTT set 128× larger than intra-DC RTT [63]. We create incast by generating four intra-DC and four inter-DC 1 GiB flows toward the same destination and record sending rates for fairness. Figure 3 (A) shows a simplified model of this setup.

We start by measuring rates as we use Gemini [63] as congestion control, *i.e.,* Figure 3 (B). While Gemini guarantees convergence to fairness [63], we observe that the convergence occurs so slowly that it outlives the flows' completion times. We repeat the same experiment with BBR's [20] and MPRDMA's [47] control loop for inter- and intra-DC flows, respectively. Figure 3 (C) highlights the unfairness among send rates of distinct flows as they are controlled by separate congestion control mechanisms. To address this, in Uno's design, we deploy a unified control loop for both inter- and intra-DC traffic that guarantees fairness while reacting to congestion signals at the same granularity for inter- and intra-DC workload. Our results, (D), show that Uno converges to fairness considerably faster than Gemini. To also ensure fast reaction to congestion, Uno deploys *Quick Adapt, i.e.,* under extreme network congestion, indicated by a sharp drop in the number of ACKed bytes, Uno dramatically reduces the send rates to quickly resolve over-utilization. In §5, we show that Uno improves the overall latency against different baselines and under distinct scenarios.

[1]One approach to flow-level fairness is using multiple priority queues, but inter-DC switches may not support them [59]. It also fails to address the lack of inter-DC buffer space [59] and requires constant tracking of competing flows to apply weighted round-robin scheduling between inter- and intra-DC traffic.

3.2 Near-zero queuing

To ensure low latency, especially for small messages, without significantly under-utilizing the network, keeping switch buffers lightly-occupied is essential [9]. However, with empty queues, there is a serious chance of network under-utilization. To avoid this, ECN-based protocols typically keep some packets in the queue with small queue occupancy fluctuations around the ECN marking threshold. However, doing so is challenging with inter-DC traffic in the picture as inter-DC flows can easily overwhelm small commodity switches due to their large BDPs. To address this, we use phantom queues [10, 22], *i.e.,* virtual queues with arbitrary sizes and drain rates that mimic physical queues. They increase occupancy on ingress and drain at a constant rate, typically slightly below the line rate. Intuitively, phantom queues offer two advantages: early congestion signaling (due to lower drain rates) and burst smoothing.

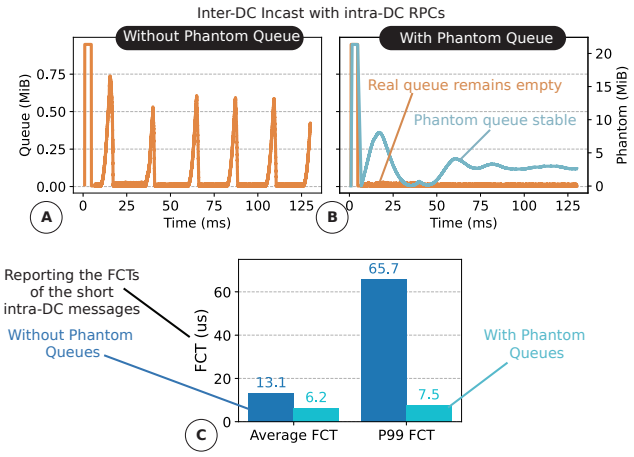

Figure 4: Showcasing the effect of phantom queues for intra-DC traffic. (A) shows the queue at the incast receiver over time without phantom queues and with phantom queue ((B)). (C) shows the FCTs for the intra-DC flows.

To highlight the potential advantages of phantom queues, we simulate a simple scenario where we initiate long-lived flows from 8 different senders in a local datacenter toward a single receiver in a remote datacenter, creating an incast scenario. In the receiver's datacenter, we also simulate sending several small messages from the "Google RPC" CDF distribution [53]. Figure 4 illustrates the queue occupancies over time with and without phantom queue at the receiver bottleneck and the flow completion times of the small RPC messages. As expected, phantom queues facilitate near-zero queuing that results in 2× and 8× improvement in the average and 99^{th} percentile FCT of the RPC messages, respectively.

3.3 Reliability and Load Balancing

To mitigate the delay penalties induced by packet loss and retransmissions, explained in §2.4, we adopt Maximum Distance Separable (MDS) [4] erasure coding as a proactive countermeasure. In our scheme, data is organized into distinct blocks, each composed of both the original data packets and additional parity packets computed via MDS coding. This "block" represents the minimal unit of encoded data, ensuring that the original information can be fully recovered as long as a sufficient number of packets are received, even if some fail during transit. Given that our experiments reveal that packet losses are not purely random but tend to occur in correlated clusters, the redundancy introduced by MDS coding is crucial. It allows the system to tolerate minor burst losses without waiting for slow retransmission timeouts, thereby maintaining low latency across WAN links. This approach reduces recovery delays and optimizes resource utilization by minimizing unnecessary retransmissions.

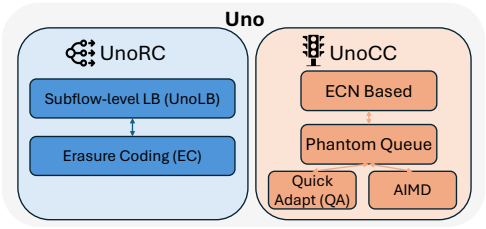

Figure 5: Uno's overall architecture.

While erasure coding solves the problem with certain failure modes, it cannot completely help in all cases. For instance, if we use ECMP routing, *e.g.,* with Gemini [63] and Annulus [59], if a link goes down, temporarily or permanently, all packets in a block would be lost until the routing table gets updated making it harder to reconstruct the message. To resolve this, we develop our custom load balancing scheme to mitigate the classical shortcomings of ECMP, *e.g.,* hash collisions [30], while also improving the resilience of erasure coding. We describe the details in §4.2.

4 Uno: a unified system for both intra- and inter-DC communication

Given the goals outlined in the previous section, we design Uno, a unified system that facilitates low-latency and fair communication in both intra-DC and inter-DC environments. As shown in Figure 5, Uno has two components: 1) Congestion Control component (UnoCC) and 2) Reliable Connectivity component (UnoRC).

UnoCC is a window-based congestion control scheme for both intra- and inter-DC traffic that employs Additive Increase Multiplicative Decrease (AIMD) window adjustment to ensure fair bandwidth sharing. To quickly converge to bandwidth fairness, UnoCC reacts to congestion signals at the same granularity for intra- and inter-DC flows. It further employs *Quick Adapt*[2], which, under extreme congestion (*i.e.,* sharp drop in ACKed bytes), promptly reduces the congestion window to quickly alleviate congestion and avoid persistent over-utilization. To efficiently handle ECN marking in both inter- and intra-DC switch buffers, UnoCC is augmented with phantom queues [10], *i.e.,* virtual queues with arbitrary sizes and drain rates that mimic physical ones. Delay is used to distinguish physical from phantom queue congestion.

UnoRC combines subflow-level load balancing (UnoLB) with *erasure coding* [4, 52] for inter-DC flows to improve routing performance between datacenters and ensure loss resiliency. The key idea is to use erasure coding to maximize the chances of latency-bound messages getting delivered correctly to the receiver. To do so, we send a certain number of parity packets for every *block*. However, to

[2]Quick Adapt has been previously proposed for communication inside datacenters [15]. However, we tailor it for both intra- and inter-DC communication in this paper.

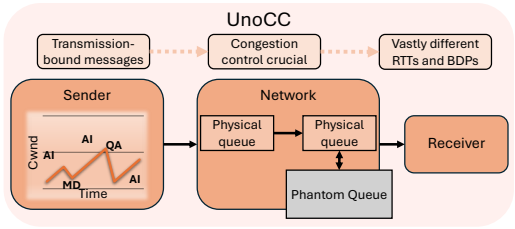

Figure 6: UnoCC's design.

further improve loss resiliency for a block, we also spread packets of a single block across different paths to maximize the probabilities of successful deliveries even in case of link failures. Finally, we adaptively remove paths from our routing options when we identify them as failed or congested paths which is detected either via a sender-based timeout or a NACK from the receiver.

4.1 UnoCC

As presented in Figure 6, UnoCC assumes three congestion states for the network: 1) Uncongested, 2) Congested, and 3) Extremely congested. To cope with states 1 and 2, UnoCC employs an *AIMD* rate control mechanism that uses ECN as the congestion signal. UnoCC also uses relative delay, *i.e.,* $RTT - RTT_{base}$, but only to differentiate between phantom and physical queue congestion events (RTT and RTT_{base} are a packet's measured RTT and the minimum RTT in an uncongested network). Additionally, UnoCC deploys Quick Adapt to facilitate fast reaction to extreme network congestion (State 3). Algorithm 1 outlines distinct mechanisms in UnoCC's design.

Algorithm 1 UnoCC's control loop

```
1:  procedure OnAck
2:      if ECN not marked then                          ▷ Uncongested network (AI)
3:          cwnd = cwnd + α × bytes_acked / cwnd
4:      end if
5:  end procedure
6:
7:  procedure OnEpoch
8:      if ecn_fraction > 0 then                        ▷ Congested network (MD)
9:          if delay == 0 then                          ▷ Congestion in phantom queues
10:             MD_scale = MD_scale × 0.3               ▷ Gentle Reduction
11:         else                                        ▷ Congestion in physical queues
12:             MD_scale = 1
13:         end if
14:         cwnd = cwnd × (1 − MD_ECN × MD_scale)
15:     end if
16: end procedure
17:
18: procedure OnQA
19:     if bytes_acked_in_qa < cwnd × β then            ▷ Very congested network (QA)
20:         cwnd = bytes_acked_in_qa
21:     end if
22: end procedure
```

4.1.1 Additive Increase – Multiplicative Decrease (AIMD). When an ACK packet arrives and it is not ECN marked, UnoCC increases the congestion window ($cwnd$) by $\alpha \times \frac{bytes\_acked}{cwnd}$. α is the AI factor and is set as a fraction of BDP, *e.g.,* $0.001 \times BDP$ for our simulations. Thus, after one RTT in an uncongested network, $cwnd$ increases by α. Note that α should be scaled depending on the queue size and the degree of incast that a network can support without losses in the steady state. Unlike AI, that is applied per ACK, MD is applied at most once per *epoch*. Specifically, upon receiving the first ACK of the flow, UnoCC stores an epoch activation time ($\mathcal{T}_{epoch}$) for that flow which is initialized to the time of the ACK arrival. Additionally, when a packet is being sent/re-sent, UnoCC stores its send/re-send time ($\mathcal{T}_{pkt}$). An epoch terminates when we receive an ACK for a data packet whose $\mathcal{T}_{pkt}$ is $\geq \mathcal{T}_{epoch}$. Upon epoch termination, UnoCC increases $\mathcal{T}_{epoch}$ by $epoch\_period$, *i.e.,* a time span proportional to the packet's RTT, and re-activates the epoch. Using this approach, we ensure that enough packets are received before deciding to apply MD which gives us better grasp of the network's condition. UnoCC considers an epoch period as congested if any packets has been ECN-marked during the epoch. When applying MD, UnoCC computes the MD factor (*i.e.,* MD_{ECN}) as $\mathcal{E} \times (\frac{4 \times K}{K + BDP})$. $\mathcal{E}$ represents the exponential weighted moving average (EWMA) of the fraction of ECN-marked packets across epochs, and K is a user-set constant that indicates the extent of UnoCC's reaction to congested epochs.

We select UnoCC's AI and MD factors (*i.e.,* α and MD_{ECN}, respectively) similar to Gemini [63] to achieve guaranteed convergence to fairness. However, Gemini experiences slow convergence time due to reacting to congestion signals at different granularities for inter- and intra-DC traffic (§2). Through extensive empirical experimentation, we found that by reacting to congestion signals at the same granularity for both inter- and intra-DC flows, we can better capture congestion events inside and across datacenters and, thus, considerably improve the speed of convergence to the fair bandwidth share. To this end, we use the same $epoch\_period$, set based on intra-DC RTT, for both inter- and intra-DC flows. In §5, we quantitatively show that, despite having similar AI and MD factors, UnoCC achieves much faster convergence to fairness than Gemini [63].

Finally, since phantom queues have slower drain rates than physical queues, they can signal the sender to slow down more than needed and for too long. To avoid this, if UnoCC detects that the physical queues are empty and phantom queues are congested, *i.e.,* packets are ECN marked but packet delays indicate no congestion, it employs a gentler reduction (MD_{scale}) by scaling down MD_{ECN}.

4.1.2 Quick Adapt (QA). Events such as the arrival of new flows or incast can potentially create extreme network congestion. Solely relying on MD for resolving such congestion events is slow and can significantly hurt latency. To avoid this, UnoCC deploys the *QA* mechanism. Specifically, once every RTT, UnoCC evaluates if the network is extremely congested by checking if the number of ACKed bytes is considerably low, *i.e.,* less than $cwnd \times \beta$ (β is the user-set QA ratio). If the network is deemed as extremely congested, the $cwnd$ is sharply decreased to the number of bytes ACKed during the QA period to quickly match the network's instantaneous capacity. To avoid over-reacting to congestion, after triggering QA, UnoCC skips one RTT without triggering any QAs or MDs.

4.1.3 Phantom Queues. As discussed in §2, efficiently setting ECN marking thresholds for a mixture of inter- and intra-DC traffic is challenging as intra- and inter-DC switch buffer capacities differ, and using only shallow buffers everywhere can also lead to oscillations for modern CCs due to the large inter-DC BDPs. To address

this, we use phantom queues [10] in conjunction with UnoCC. The phantom queue occupancy increases every time a new packet is enqueued in the physical queue and decreases at a constant rate (*i.e.,* draining rate), which is a fraction of the link bandwidth. Note that a phantom queue is easily implementable using a counter that keeps track of its occupancy.

Using phantom queues enables us to correctly mark packets with ECN signals regardless of the physical queue's capacity. As illustrated in §2, by properly setting the draining rate of the phantom queues, we practically experience zero queuing at the physical queues as long as the network is at its steady state. Specifically, in steady state, as the phantom queues are drained at a lower speed than the physical queues, the physical queues become empty before the phantom queue occupancy reaches zero. This is important as it gives extra bandwidth headroom, especially for small and latency-sensitive intra-DC flows. Using phantom queues can potentially penalize large throughput-intensive flows. However, our experiment results show that by setting the phantom queue's draining rate slightly lower than physical queues, *e.g.,* 10% lower, we avoid victimizing large flows while keeping the benefiting small flows[3].

4.2 UnoRC

As shown in Figure 7, UnoRC has two components: an erasure coding component to enhance reliability, especially under failure, and a simple but effective sub-flow level load balancer (UnoLB).

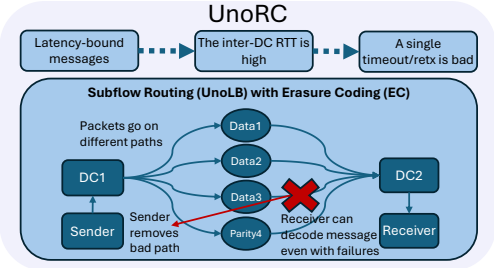

Figure 7: UnoRC's design.

Erasure Coding (EC): For reliability, each inter-DC message is divided into *blocks* of n packets, with x data and y parity packets. A block can be reconstructed if at most y out of n packets are lost. Upon receiving the first packet of a block, the receiver starts a timer set to the estimated maximum queuing and transmission delay. If the timer expires before enough packets arrive, a NACK is sent to the sender requesting retransmission of the missing block. UnoRC applies erasure coding only to inter-DC traffic due to long recovery delays. While EC adds fixed overhead (e.g., 20%), it reduces packet loss and improves completion times under failure and congestion events, which is crucial in latency-bound scenarios (§5).

Load Balancing (UnoLB): Our load balancing scheme uses n subflows and each subflow gets assigned its own path (either via source-based assignment or by changing the source port value for ECMP hashing). By itself, this simple action (somewhat similar to MPTCP [27]) vastly improves the performance due to a decrease in hash collisions. To integrate UnoLB with the reliability aspect (*i.e.,* erasure coding), we spread the packets of a block across n subflows. Doing so increases the resilience to link failures. Moreover, UnoRC switches from *bad paths* when it detects extreme congestion on them. In particular, upon receiving a NACK (indicating an unrecoverable block) or when a sender timeout occurs (possibly due to lost NACKs caused by failures or corruption), UnoRC re-routes the affected flows by randomly selecting a subflow that has recently received ACKs, thereby reducing the likelihood of switching to another congested or failed path. Algorithm 2 presents UnoRC's logic.

[3]This result aligns with prior work on phantom queues [10].

Algorithm 2 Pseudocode for UnoLB

```
1: procedure onSend(packet)
2:     packet[header.source_port] ← subflow[index]
3:     index ← (index + 1) mod total_subflows
4: end procedure
5: procedure onNackOrTimeout(packet)
6:     if (now() − last_reroute) > base_rtt then
7:         update_subflow(packet)
8:         last_reroute ← now()
9:     end if
10: end procedure
```

5 Performance Evaluation

We use *htsim*, a packet-level network simulator [33], to evaluate Uno across distinct workloads and traffic patterns. Our key findings are summarized below:

- Uno improves the average and tail latency compared to the state-of-the-art solutions. Specifically, under 40% load and a mixture of inter-DC and intra-DC flows, Uno improves the 99^{th} percentile FCT by 1.4× compared to both MPRDMA+BBR and Gemini.
- UnoCC provides fast convergence to fairness. Particularly, with UnoCC under incast events created from various combinations of intra- and inter-DC flows, all flows quickly converge to their fair bandwidth share.
- UnoRC (*i.e.,* UnoLB + EC) further improves the performance under several failure scenarios by up to 3× compared to Uno without erasure coding and 2× and 6× compared to RPS and PLB, respectively.

5.1 Simulation Setup

Topology. We simulate two 8-ary fat-tree datacenters [5], each consisting of 16 core switches and 8 pods with 4 aggregate and 4 edge switches. Each edge switch is connected to 4 servers. The datacenters are connected through two border switches that are interconnected through eight links. Also, every core switch is connected to a border switch through eight links. Unless stated otherwise, we exploit 100 Gbps links for all our interconnects and set the switch buffer capacities to 1 MiB per port.

Microbenchmarks. We first run microbenchmark experiments similar to those performed in prior works [63] to demonstrate the fairness advantages of Uno. Two types of traffic are evaluated: 1) incast traffic originating from different sources and 2) permutation traffic with randomly selected source and destination nodes.

Realistic Workload. The characteristics of intra-DC traffic differ from traffic spanning multiple datacenter networks [58, 65].

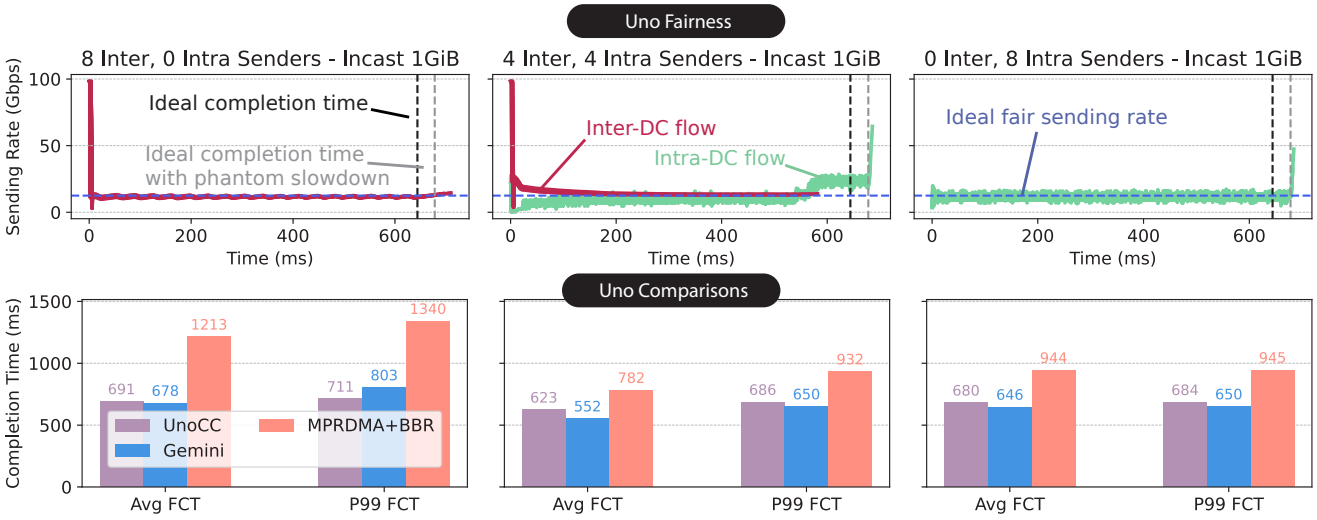

Figure 8: The top plot showcases the fairness of Uno while the bottom plot shows the performance against other algorithms. The title on top of each plot indicates how many inter- and intra-DC flows take part in the incast.

Therefore, we simulate different workloads within and across datacenters. Similar to [63], we use the flow size distributions of Google's web search [9] for generating intra-DC traffic. To generate inter-DC traffic, we exploit the flow size distribution recorded between two datacenters in Alibaba's regional WAN [65]. Unless stated otherwise, the flows' arrival rates are generated based on an exponential distribution, and the rates are scaled to achieve a desired network load. Flow source and destination servers are selected using a uniform random distribution, similar to previous works [9, 63, 65]. The ratio of datacenter to WAN traffic is set to 4:1.

We also use an AI training workload for the inter-DC traffic in one of our experiments. Particularly, we assume a data parallel training strategy [42, 57] across the two datacenters, where each datacenter has at least one replica of the model being trained. After computing the gradients during the backward pass of each iteration, an Allreduce (or separate Reducescatter and Allgather) collective operation is initiated to synchronize the gradients across the datacenters. Our experiments simulate inter-datacenter training of the Llama 70B model and the parallelization strategy in its technical report [32], which generates periodic traffic bursts of approximately 70-500 MiB per iteration [50]. The total number of send operations depends on the number of Allreduce groups in the collective.

Evaluation metrics. We measure the mean and tail (99^{th} percentile) Flow Completion Time (FCT) as our main evaluation metric. We also report metrics such as queue occupancy and sending rate.

Baseline state-of-the-art (SOTA) approaches. We compare Uno against two major baselines: 1) MPRDMA+BBR ([47]+[20]): Widely deployed for enterprise WAN, BBR targets accurate RTT and bandwidth estimation to maximize throughput and minimize latency. Since BBR is only used for inter-DC communication, we combine it with MPRDMA, an ECN-based congestion control scheme, for managing intra-DC communication. 2) Gemini [63]: Similar to UnoCC, Gemini is a congestion control protocol designed for both intra-DC and inter-DC communication that exploits ECN and RTT to detect intra-DC and inter-DC congestion, respectively.[4] To evaluate UnoRC, we evaluate it against different routing schemes such as Random Packet Spraying (RPS) [24] and PLB [56].

Parameter	Default Value
α (UnoCC's AI factor)	$0.001 \times BDP$
β (UnoCC's QA factor)	0.5
K (UnoCC's MD constant)	$\frac{1}{7}\times$intra-DC BDP
Intra-DC RTT	$14\mu s$
Inter-DC RTT	$2ms$
Phantom queue drain rate	0.9×physical queue drain rate

Table 2: Parameter table.

Parameter settings. To mark ECN, we use Random Early Detection [45]. Specifically, the packets are never ECN marked as long as the destination queue occupancy is less than the minimum ECN threshold (*i.e., MinECNThresh*) and always marked when the occupancy is more than the maximum ECN threshold (*i.e., MaxECNThresh*). Otherwise, the probability of marking packets increases linearly. *MinECNThresh* and *MaxECNThresh* are set to 25% and 75% of queue capacity. For Uno, we set α, β, and K to 0.1% of the BDP, 0.5, and $\frac{1}{7}\times$intra-DC BDP, respectively. The phantom queues drain at 90% of the line rate. Unless stated otherwise, MTU, intra-DC RTT, and inter-DC RTT are set to 4096 B, $14\mu s$, and $2ms$, respectively [63]. Table 2 summarizes the main parameters of our experiments.

5.2 Simulation results

5.2.1 Micro-benchmarks. As the first experiment, we simulate Uno under intra-DC incast, inter-DC incast, and a mixture of the two. In all scenarios, we generate a total of eight 1 GiB flows. Also, we use packet spraying for all schemes as load balancing has a negligible impact under receiver-side incast. In Figure 8, we present Uno's

[4] Annulus [59], which works on top of other schemes such as BBR, could also be used to enhance the performance of Uno under oversubscribed topologies. However, we leave this add-on for future work.

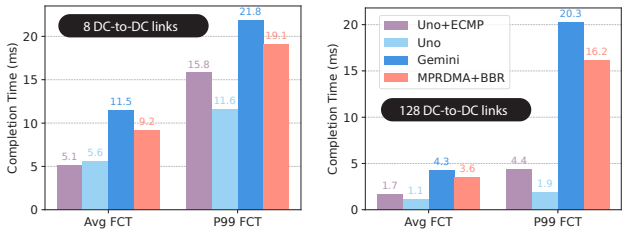

Figure 9: Showcasing results with a permutation workload.

fairness by plotting the sending rate of the flows involved in the incast and evaluate its performance against other baselines. Our findings show that Uno outperforms or matches the performance of the other algorithms in all three scenarios and achieves near-ideal latency. Moreover, using Uno, in all scenarios, the send rates of inter- and intra-DC flows quickly converge to their fair bandwidth share, highlighting Uno's fast convergence to fairness.

We then consider a permutation scenario where each sender is sending to another randomly selected node (within the same DC or cross DCs). Since this could result in a lot of inter-DC communication (easily overwhelming the eight inter-DC links), we showcase two scenarios: one where the topology is as-is with 800Gbps of inter-DC bandwidth and another scenario where the inter-DC links are fully provisioned. Here we showcase Uno with ECMP and Uno with our custom load balancing solution (UnoLB) as part of UnoRC. In Figure 9, we show the two results. Under the same ECMP load balancing assumption, Uno is still significantly better than the alternatives in both scenarios. The gap between Uno and the other approaches further increases with UnoLB. As expected, average FCTs are higher when fewer links connect the DCs.

5.2.2 Realistic workloads. To test Uno against realistic workloads, we generate intra- and inter-DC traffic using Google's web search [9] and Alibaba's WAN [65] traffic distributions, respectively, and compare Uno+ECMP (*i.e.,* UnoCC deployed beside ECMP for load balancing) and Uno (*i.e.,* UnoCC+UnoRC) against other baselines. Figure 10 presents the results. We observe that, compared to Gemini and MPRDMA+BBR, UnoCC reduces both average and tail latency of inter-DC flows but slightly increases the latency of intra-DC web search flows. This is because phantom queues' drain rates are lower than physical queues which can occasionally hurt the latency of intra-DC flows. However, despite marginally increasing the latency of intra-DC flows, UnoCC manages to improve the overall latency by providing better congestion management and faster convergence to fair bandwidth share, *e.g.,* under 40% load, UnoCC improves mean latency by 30% and 37% compared to MPRDMA+BBR and Gemini, respectively. Additionally, by achieving balanced load and loss resiliency on top of UnoCC, Uno manages to reduce the latency of both intra- and inter-DC flows compared to other baselines. For example, compared to MPRDMA+BBR and Gemini under 40% load, Uno decreases the tail FCT by 4.4× and 5.3× for intra-DC flows and by 1.7× and 2.1× for inter-DC flows, respectively.

We also evaluate Uno against different inter-DC propagation delays. To this end, we repeat the previous experiments with 40% network load and gradually increase the propagation delay of inter-DC links. Figure 11 reports the FCT slowdown ratios of distinct schemes as we increase the ratio of inter-DC minimum RTT to intra-DC minimum RTT from 8 to 512 by increasing the inter-DC propagation delay (intra-DC minimum RTT = 14μs). We observe that, when the gap between intra- and inter-DC RTT is small, Uno is slightly outperformed by MPRDMA+BBR due to the phantom queue slowdown. However, as the RTT ratio increases, and gets closer to ratios observed in today's networks (*e.g.,* Figure 1), Uno significantly outperforms MPRDMA+BBR and Gemini. In particular, when the RTT ratio is 512, Uno's tail FCT slowdown is 5× lower than MPRDMA+BBR and Gemini. This shows that the existing solutions fall short in efficiently handling the big gap between intra- and inter-DC delays, while Uno is well-designed for it.

Lastly, we evaluate Uno's performance under realistic workloads and distinct inter- and intra-DC queue sizes. We re-run experiments under 40% load using shallow- and deep-buffered switches inside and across datacenters, respectively. Particularly, we set intra- and inter-DC queue sizes to ≈ 175 KiB (*i.e.,* intra-DC BDP) and ≈ 2.2 MiB (*i.e.,* 0.1× inter-DC BDP) per port. Figure 12 shows average and tail FCTs. Consistent with prior results (*i.e.,* Figure 10), Uno+ECMP lowers overall FCT by reducing inter-DC completion times with only slight intra-DC latency increase, while Uno improves both: compared to Gemini, tail FCT drops by 3.1× (intra) and 1.7× (inter); versus MPRDMA+BBR, it drops by 3.6× (intra) and 1.8× (inter).

5.2.3 Failure Scenarios. We now move to the evaluation of the reliability (RC) aspect of Uno. To do so, we evaluate Uno's performance under different failure scenarios. Since we have already evaluated the effectiveness of UnoCC on its own against other state-of-the-art algorithms, we now focus only on different variants of Uno when it comes to load balancing and reliability and use UnoCC as congestion control for all experiments. In particular, we compare three different state-of-the-art load balancing schemes: packet spraying [24], PLB [56] and UnoLB. We exclude ECMP since we already evaluated the superiority of multipathing schemes in the previous section and because ECMP is oblivious to network failures.

For erasure coding (EC), we note that the choice of the correct block size and number of parity packets depends on the initial assumptions of the network. Higher failure rates require more redundancy but also introduce more overhead. Although there is no single *one-size-fits-all* answer, we ran several experiments and decided to use a *(8, 2)* scheme, *i.e.,* 10 packets in a group with 8 data packets and 2 parity packets.

The first experiment that we run consists of failing one of the eight border links while simulating latency-sensitive 5MiB flows between two datacenters that can theoretically saturate all the inter-DC bandwidth. Since a single run can be heavily influenced by the initial path selection, we re-run every experiment 100 times to reduce biases and use violin plots to show detailed statistics across runs. Figure 13 (A) presents the results. We observe that Uno outperforms both spraying and PLB with and without EC. This is due to Uno's ability to adaptively avoid problematic links and intelligently distribute packets within the same block.

Then, we move to a different failure scenario and simulate random loss events based on our real measurements from Table 1. We implement and simulate the full failure logic comprehensive of the correlation between different failure events. In Figure 13 (B), we show the results for this scenario considering a single flow across

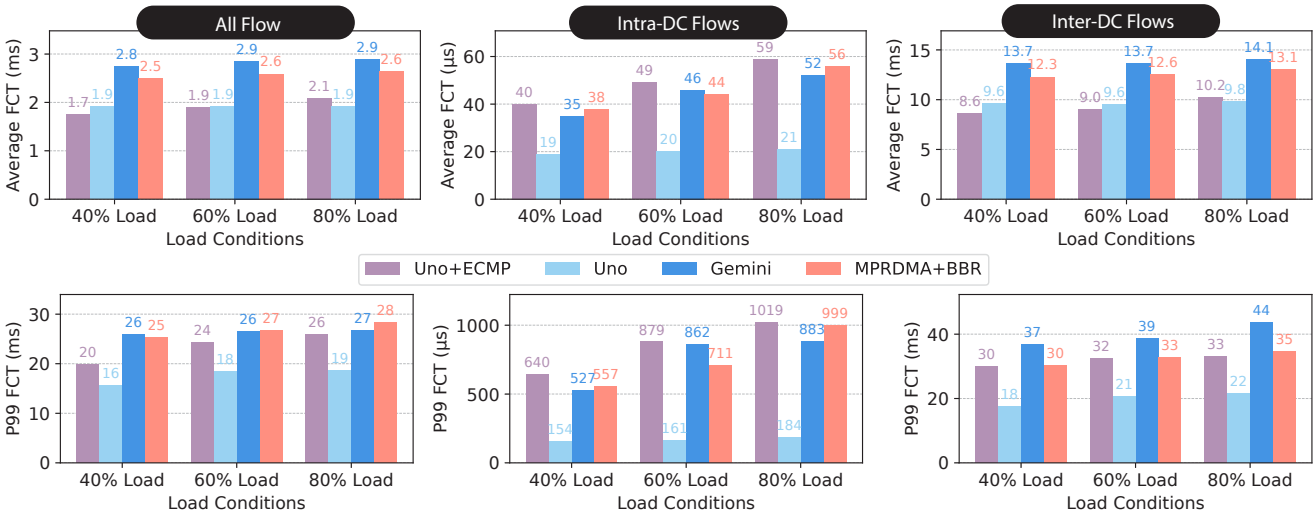

Figure 10: Uno is superior to other baselines under distinct network loads.

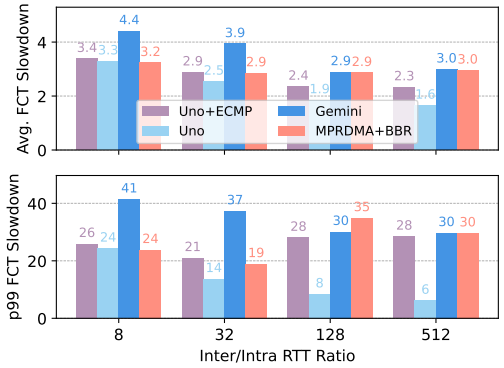

Figure 11: Uno improves latency when there exists a significant gap between intra- and inter-DC RTTs.

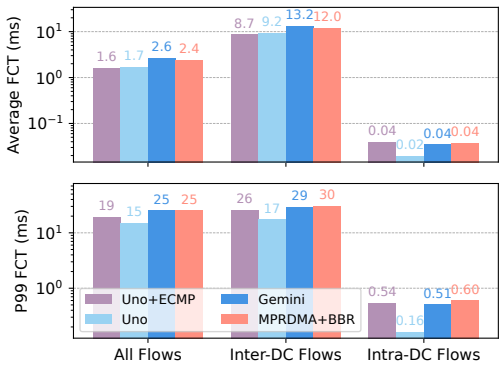

Figure 12: Uno is superior to MPRDMA+BBR and Gemini even when the queue capacities inside and across datacenters differ.

datacenters. In this case, blocks are only dropped in the unlikely event that three or more packets are dropped within a block. What we observe is that Uno is mostly matching packet spraying, as expected, and outperforming PLB both with and without EC. This is because PLB sticks to one path for a given flow, and when a link becomes temporarily flaky, it negatively affects the entire block, resulting in a higher runtime for the worst-case scenario.

Finally, in Figure 13 Ⓒ, we simulate the training strategy described in §5.1. In this experiment, we simulate both link failures and random drops. We generate 100 training iterations and focus on the inter-DC communication. We report the ratio of the measured Allreduce collective runtime per training iteration to the ideal runtime. We observe that Uno consistently outperforms other baselines both with and without EC. Specifically, with EC, Uno performs over 2× better than the second best algorithm and only 30% slower than the ideal completion time that assumes no ECMP collisions or random drops.

6 Discussion

Leveraging modern datacenter congestion control for inter-DC traffic. Using MPRDMA+BBR, we show the drawbacks of separating intra- and inter-DC control loops. While alternatives like HPCC [43] and PowerTCP [3] exist, they too suffer from fairness issues due to this separation. Unfortunately, most SOTA intra-DC congestion control protocols, *e.g.,* [9, 11, 33, 41, 43, 47], rely on fast RTT feedback and specialized switch support (*e.g.,* INT and packet trimming), making them impractical across inter-DC environments. Additionally, protocols like PowerTCP and HPCC use single-pathing, limiting their ability to balance load. In contrast, Uno applies congestion control uniformly across intra- and inter-DC traffic, relies on widely supported ECN [9, 44], and employs multi-pathing to achieve low latency, high reliability, and fairness.

Hardware implementation. While we showcase Uno's potential in simulations, we design it with hardware feasibility in mind:

UnoCC's key operations (*i.e.,* AIMD and QA) are easily deployable in the Linux kernel, similar to other widely deployed protocols like TCP/DCTCP. For the congestion signal, UnoCC only uses ECN, which is commonly supported by today's switches [9, 44]. Phantom queues, already supported by several vendors [19, 31] (such as in

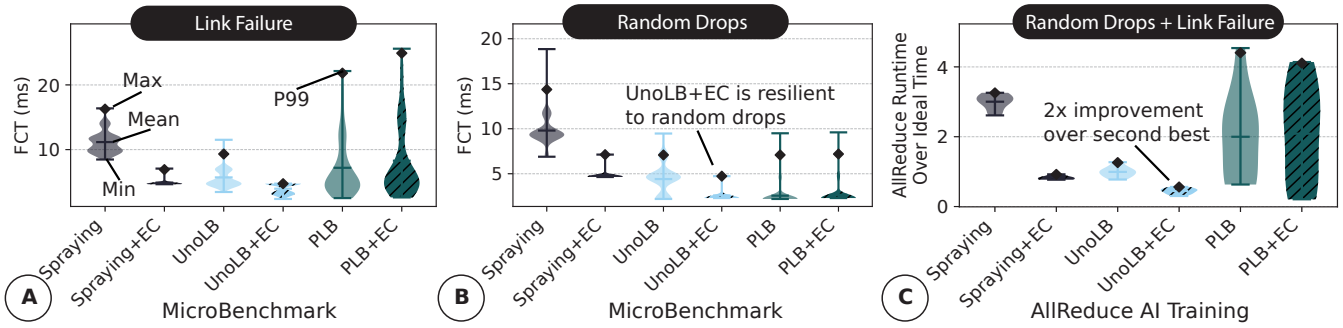

Figure 13: Uno's performance under distinct failure scenarios and different workloads.

Cisco Nexus 5548P [1]) and implementable with simple packet-increment/decrement counters, are also easily deployable on most switches. Additionally, Uno uses hardware pacing for congestion control, which is commonly available at the sender NIC [10, 41, 59].

UnoRC is deployable as software shim layers: the sender inserts parity packets and the receiver decodes once enough packets arrive, using a coarse software timer suited for long-latency links. Finally, Uno's re-routing can be implemented by either changing the IPv6 flow-label [56] or updating the UDP source port as described in Ultra Ethernet [34].

7 Related Work

Congestion within and across datacenters. Most of the existing proposals on congestion control either focus on intra-DC networks or inter-DC WANs and very little research has been done on simultaneously addressing intra- and inter-DC congestion [63]. To address intra-DC congestion, existing body of work typically relies on ECN [9, 10, 61, 66], delay [41, 51], or receiver-driven transmission [21, 28, 33, 53]. WAN congestion management proposals [12, 18, 20, 60], on the other hand, typically use delay to detect WAN congestion due to the limited congestion detection support from WAN switches. Gemini [63] is among the first proposals that tries to bring inter- and intra-DC congestion control together as a system by using ECN and delay to detect intra- and inter-DC congestion, respectively. However, as presented in §5, Gemini suffers slow convergence to bandwidth fairness. Annulus [59] tries to address congestion near source but does not effectively handle congestion that occurs far from the source and closer to the destination. Moreover, Annulus works on top of other protocols, thus not a standalone solution on its own. Uno, however, acts as an effective system for both intra- and inter-DC environments by ensuring fast reaction to congestion, bandwidth fairness, and loss resiliency. HULL [10] and LGC-ShQ [22] have tried using phantom queues but limited to the datacenter networks.

Addressing reliability and load balancing. Throughout the years, many protocols have been proposed for load balancing inside datacenters, spanning across different granularity: per flow, per subflow, or per packet. Protocols such as ECMP, Hedera [6], MicroTE [14], Flowcut [17] work on per-flow level but they either suffer from hash collision or require complex global controllers. Solutions working on sub-flow level load balancing try to mitigate the above issues. FlowBender [39], PLB [56], CONGA [8] significantly improve routing performance over ECMP. However, they can still be prone to collisions since they use one given path at a time or require specialized hardware as is the case of CONGA. Finally, packet-level load balancers, such as RPS [24], Drill [30], and REPS [16], provide the best absolute performance but require out-of-order support at the receiver and complicate loss detection. UnoLB tries to get the best of both worlds by using multiple sub-flow at any given time (similar to MPTCP [27]), but also by adaptively re-routing flows away from congested paths. This ensures good load balancing performance while limiting the amount of out-of-order packets. Besides load balancing, erasure coding also improves loss resiliency. Specifically, erasure coding enhances data reliability by splitting the data into fragments and adding strategic redundancy, allowing recovery even with some failures [46, 49, 52]. In Cloudburst [64], the idea of multi-pathing with erasure coding was explored but focusing on intra-DC communication.

8 Conclusion

Simultaneously handling congestion both inside and across datacenters is challenging due to the huge gap between intra- and inter-DC propagation delays. To address this, we proposed Uno, a unified system for efficient DC communication with two components: (1) UnoCC, a congestion control scheme ensuring fairness and fast reaction, and (2) UnoRC, a reliable routing mechanism combining subflow-level load balancing and erasure coding. Through extensive simulations, we showed that Uno improves latency and fairness compared to the state-of-the-art solutions. Moreover, we illustrated the effectiveness of Uno in ensuring loss resiliency when encountering failure and against different load balancers.

Acknowledgments

This project was funded by the NSF CNS NeTS grant (No. 2313164), Sapienza University Grants ADAGIO and D2QNeT (Bando per la ricerca di Ateneo 2023 and 2024), and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement PSAP, No. 101002047). We also thank the Swiss National Supercomputing Center (CSCS) for providing the computational resources used in this work. The authors used ChatGPT to only assist with editing and quality control throughout this manuscript (ideas and content remain original).

References

[1] 2010. *Cisco Nexus 5548P Switch Architecture. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-5548p-switch/white_paper_c11-622479.html.*

[2] 2024. *Broadcom Trident4. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56880-series.*

[3] Vamsi Addanki, Oliver Michel, and Stefan Schmid. 2022. PowerTCP: Pushing the Performance Limits of Datacenter Networks. In *NSDI*.

[4] Marcos K Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. 2005. On the erasure recoverability of MDS codes under concurrent updates. In *IEEE ISIT*.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* (2008).

[6] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, et al. 2010. Hedera: dynamic flow scheduling for data center networks.. In *NSDI*.

[7] Mohammad Alizadeh, Berk Atikoglu, Abdul Kabbani, Ashvin Lakshmikantha, Rong Pan, Balaji Prabhakar, and Mick Seaman. 2008. Data center transport mechanisms: Congestion control theory and IEEE standardization. In *IEEE Allerton Conference on Communication, Control, and Computing*.

[8] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*.

[9] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *SIGCOMM*.

[10] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is more: Trading a little bandwidth for {Ultra-Low} latency in the data center. In *NSDI*.

[11] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt:Sub-RTT congestion control for Ultra-Low latency. In *NSDI*.

[12] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical {Delay-Based} congestion control for the internet. In *NSDI*.

[13] Wei Bai, Kai Chen, Shuihai Hu, Kun Tan, and Yongqiang Xiong. 2017. Congestion control for high-speed extremely shallow-buffered datacenter networks. In *APNET*.

[14] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. 2011. MicroTE: Fine grained traffic engineering for data centers. In *CoNEXT*.

[15] Tommaso Bonato, Abdul Kabbani, Daniele De Sensi, Rong Pan, Yanfang Le, Costin Raiciu, Mark Handley, Timo Schneider, Nils Blach, Ahmad Ghalayini, et al. 2024. SMaRTT-REPS: Sender-based Marked Rapidly-adapting Trimmed & Timed Transport with Recycled Entropies. *arXiv e-prints* (2024).

[16] Tommaso Bonato, Abdul Kabbani, Ahmad Ghalayini, Michael Papamichael, Mohammad Dohadwala, Lukas Gianinazzi, Mikhail Khalilov, Elias Achermann, Daniele De Sensi, and Torsten Hoefler. 2025. REPS: Recycled Entropy Packet Spraying for Adaptive Load Balancing and Failure Mitigation. *arXiv e-prints* (2025).

[17] Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Abdulla Bataineh, David Hewson, Duncan Roweth, and Torsten Hoefler. 2025. Flowcut Switching: High-Performance Adaptive Routing with In-Order Delivery Guarantees. arXiv:2506.21406 [cs.NI] https://arxiv.org/abs/2506.21406

[18] Lawrence S Brakmo, Sean W O'malley, and Larry L Peterson. 1994. TCP Vegas: New techniques for congestion detection and avoidance. In *SIGCOMM*.

[19] Broadcom. 2021. BCM88480 Traffic Management Architecture. (2021). https://docs.broadcom.com/doc/88480-DG1-PUB

[20] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-based congestion control. *Commun. ACM* (2017).

[21] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-scheduled delay-bounded congestion control for datacenters. In *SIGCOMM*.

[22] Kristjon Ciko, Peyman Teymoori, and Michael Welzl. 2022. LGC-ShQ: Datacenter Congestion Control with Queueless Load-Based ECN Marking. *SIGCOMM Comput. Commun. Rev.* 52, 4 (Dec. 2022), 2–11. doi:10.1145/3577929.3577931

[23] Daniele De Sensi, Tiziano De Matteis, Konstantin Taranov, Salvatore Di Girolamo, Tobias Rahn, and Torsten Hoefler. 2022. Noise in the clouds: Influence of network performance variability on application scalability. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2022).

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

[25] Xiaodong Dong, Wenxin Li, Xiaobo Zhou, Keqiu Li, and Heng Qi. 2020. TINA: A fair inter-datacenter transmission mechanism with deadline guarantee. In *IEEE INFOCOM*.

[26] Nathan Farrington, Erik Rubow, and Amin Vahdat. 2009. Data center switch architecture in the age of merchant silicon. In *IEEE symposium on high performance interconnects*.

[27] Alan Ford, Costin Raiciu, Mark Handley, and Olivier Bonaventure. 2013. *TCP extensions for multipath operation with multiple addresses.* Technical Report.

[28] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *CoNEXT*.

[29] Gemini Team. 2023. Gemini: A Family of Highly Capable Multimodal Models. *arXiv e-prints*, Article arXiv:2312.11805 (Dec. 2023), arXiv:2312.11805 pages. doi:10.48550/arXiv.2312.11805 arXiv:2312.11805 [cs.CL]

[30] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. Drill: Micro load balancing for low-latency data center networks. In *SIGCOMM*.

[31] Richard J. Gibbens and Frank P. Kelly. 1999. Distributed connection acceptance control for a connectionless network. https://api.semanticscholar.org/CorpusID:192826

[32] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint* (2024).

[33] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 29–42. doi:10.1145/3098822.3098825

[34] Torsten Hoefler, Karen Schramm, Eric Spada, Keith Underwood, Cedell Alexander, Bob Alverson, Paul Bottorff, Adrian Caulfield, Mark Handley, Cathy Huang, Costin Raiciu, Abdul Kabbani, Eugene Opsasnick, Rong Pan, Adee Ran, and Rip Sohan. 2025. Ultra Ethernet's Design Principles and Architectural Innovations. arXiv:2508.08906 [cs.NI] https://arxiv.org/abs/2508.08906

[35] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *SIGCOMM*.

[36] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. 2018. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN. In *SIGCOMM*.

[37] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *SIGCOMM* (2013).

[38] Abdul Kabbani, Mohammad Alizadeh, Masato Yasuda, Rong Pan, and Balaji Prabhakar. 2010. AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *IEEE symposium on high performance interconnects*.

[39] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. 2014. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *CoNEXT*.

[40] Umesh Krishnaswamy, Rachee Singh, Paul Mattes, Paul-Andre C Bissonnette, Nikolaj Bjørner, Zahira Nasrin, Sonal Kothari, Prabhakar Reddy, John Abeln, Srikanth Kandula, et al. 2023. OneWAN is better than two: Unifying a split WAN architecture. In *NSDI*.

[41] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Mike Ryan, David J. Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *SIGCOMM*.

[42] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. arXiv:2006.15704 [cs.DC] https://arxiv.org/abs/2006.15704

[43] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *SIGCOMM*.

[44] Hyoyoung Lim, Seonwoo Kim, Jackson Sippe, Junseon Kim, Greg White, Chul-Ho Lee, Eric Wustrow, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2022. A Fresh Look at ECN Traversal in the Wild. *arXiv preprint* (2022).

[45] Dong Lin and Robert Morris. 1997. Dynamics of random early detection. In *SIGCOMM*.

[46] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. 2014. Traffic engineering with forward fault correction. In *SIGCOMM*.

[47] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path transport for RDMA in datacenters. In *NSDI*.

[48] Shouxi Luo, Renyi Wang, and Huanlai Xing. 2024. Efficient inter-datacenter ALLReduce with multiple trees. *IEEE Transactions on Network Science and Engineering* (2024).

[49] Ratul Mahajan, Jitendra Padhye, Sharad Agarwal, and Brian Zill. 2012. High performance vehicular connectivity with opportunistic erasure coding. In *USENIX ATC*.

[50] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. arXiv:1710.03740 [cs.AI] https://arxiv.org/abs/1710.03740

[51] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based congestion control for the datacenter. *SIGCOMM* (2015).
[52] Michael Mitzenmacher. 2004. Digital fountains: A survey and look forward. In *IEEE Information theory workshop*.
[53] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*.
[54] OpenAI. 2025. Pre-Training GPT-4.5. https://www.youtube.com/watch?v=6nJZopACRuQ. YouTube video, accessed on April 14, 2025.
[55] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. 2015. Low latency geo-distributed data analytics. *ACM SIGCOMM Computer Communication Review* (2015).
[56] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. 2022. PLB: congestion signals are simple and effective for network load balancing. In *SIGCOMM*.
[57] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC*.
[58] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *SIGCOMM*.
[59] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. 2020. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *SIGCOMM*.
[60] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *IEEE INFOCOM*.
[61] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. 2012. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review* (2012).
[62] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. 2023. Poseidon: An Efficient Congestion Control using Deployable INT for Data Center Networks. https://www.usenix.org/system/files/nsdi23-wang-weitao.pdf
[63] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, Yibo Zhu, and Lei Cui. 2019. Congestion Control for Cross-Datacenter Networks. In *ICNP*.
[64] Gaoxiong Zeng, Li Chen, Bairen Yi, and Kai Chen. 2022. Cutting tail latency in commodity datacenters with cloudburst. In *IEEE INFOCOM*.
[65] Gaoxiong Zeng, Jianxin Qiu, Yifei Yuan, Hongqiang Liu, and Kai Chen. 2021. FlashPass: Proactive Congestion Control for Shallow-buffered WAN. In *ICNP*.
[66] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *SIGCOMM* (2015).

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

A Overview of Contributions and Artifacts

A.1 Paper's Main Contributions

C_1 We develop Uno, a framework that unifies inter- and intra-datacenter communication from the perspective of congestion control, load balancing, and reliability.

C_2 We improve and extend the htsim simulator to support multiple datacenters and implement our scheme in htsim and several other state-of-the-art algorithms (Gemini, MPRDMA, BBR).

C_3 We evaluate Uno against the state-of-the-art algorithms under distinct workloads and traffic patterns, ranging from microbenchmarks to realistic workloads. We also evaluate Uno under a simple inter-DC AI workload. On top of that, we implement several failure scenarios in htsim to investigate Uno's performance under different failure cases. Our findings show that Uno is superior to existing proposals as it achieves lower latency, better loss resiliency, and faster convergence to flow-level fairness.

A.2 Computational Artifacts

We provide a GitHub repository (https://github.com/spcl/Uno_SC25.git) for the code. We also provide a DOI link (https://doi.org/10.5281/zenodo.15916080) where users can find both the source code of the repository and a Docker container with everything already set up.

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1-C_3	Figures 1,3,4,8-12

B Artifact Identification

B.1 Computational Artifact A_1

Relation To Contributions

The artifact contains the code necessary to run the experiments and extract the results. However, we manually enhanced the quality of certain plots, Figure 3 for example, for stylistic purposes in post-production. Key results are still readable without those annotations though. The file structure of the project is organized as the following:

- `sim/`: contains the code for distinct key modules required for the simulations, such as network topology, congestion control schemes, load balancing paradigms, etc.
- `plotting/`: contains the Python scripts used for processing the results and extracting information such as Flow Completion Times (FCTs).
- `sc25_X.sh`: These files are the shell scripts for installing the required packages, running the experiments, and extracting the results. Examples of such files are `sc25_pkt_install.sh` and `sc_fig10.sh`.
- `artifact_scripts/`: These folder contains most of the Python scripts needed to reproduce the results. We note that these need to be run from the root directory of the project.
- `artifact_results/`: When all simulation have run, the `artifact_results` folder will contain a series of subfolders for each figure of the paper. For example, after running the `sc25_fig1.sh` script, the corresponding figures will be available in `artifact_results/fig1/`.

Expected Results

The results should demonstrate that Uno achieves fairness more quickly than existing proposals. Furthermore, under a realistic workload consisting of Google's web search traffic within datacenters and Alibaba's inter-datacenter traffic, Uno is expected to reduce latency for both intra- and inter-datacenter traffic, regardless of the network load. Finally, Uno is anticipated to offer strong reliability characteristics, *i.e.,* resilience to packet loss and fast loss recovery, especially in inter-datacenter WAN environments prone to failures.

Expected Reproduction Time (in Minutes)

The artifacts can be installed and run on a local machine. The overall time required to install htsim is quite short (≈5 minutes). However, the run time for the simulations really depends on the number of generated flows and the degree of load. Specifically, with realistic workload, it can take quite some time as the number of generated flows and the network load are high. Below, we estimate the complexity of each stage of the computational artifact:

- Artifact setup: 20 minutes to download the artifact, install dependencies, and compile it using the provided files and instructions.
- Artifact execution: ≈2 days (48 hours) to run all the evaluation scripts and generate all the results used in the paper. The times reported are for worst-case scenarios, assuming a relatively slow machine and without running multiple scripts in parallel to speed up the evaluation. We note that most experiments can be run relatively quickly, such as the ones in Figure 8 or Figure 12 (in less than an hour), and only some experiments take significantly longer (like Figure 10).
- Artifact analysis: Most plots are generated automatically upon running various Python scripts. Some scripts require to manually copy the results from one script to the other for plotting purposes. We will also provide raw data, allowing plots to be generated without re-running most of the experiments.

Artifact Setup (incl. Inputs)

Hardware. This work does not rely on any specialized hardware. All experiments were conducted using a standard x86 CPU, and any similar processor capable of running the simulator used in our study should be sufficient to reproduce the results.

Software. The project requires `C++17` and `Python3.8` as the minimum dependencies for building and running both the `htsim` simulator and our accompanying Python scripts. For the experiments and computational artifacts presented in this paper, all code was executed locally on Ubuntu 22.04 LTS via WSL2.

To process and visualize the raw data, we rely on a set of standard Python packages, *i.e.,* `seaborn`, `scipy`, `numpy`, and `pandas`.

Datasets/Inputs. We did not use any particular dataset or public input data for our experiments except for the flow size distribution traces of Google's web search workload and Alibaba's inter-datacenter traffic, which allow us to test Uno under realistic workloads. Such traces have already been used in a number of previous papers and are properly cited in the main paper. Moreover, we include the files having the CDF flow size distribution in the actual repository.

The data used for Table 1 was collected using internal resources of a large-scale cloud provider and is not easily reproducible.

Installation and Deployment. Installing `htsim` is straightforward and can be done by executing the provided `Makefile` located in the `sim/` directory of the repository. This will compile the simulator and set up the necessary binaries using the default system compiler with `C++17` support.

To assist users in getting started, we provide detailed bash commands for running the `Makefile` and executing the associated Python scripts in the `README` file included with the repository. These instructions are intended to facilitate reproducibility and ease of setup across typical Linux environments such as the one mentioned above.

Artifact Execution

Once the `htsim` software has been successfully built (T_1), all experiments and visualizations can be executed using the corresponding Bash and Python scripts T_2. These scripts are located in the root directory of the main repository. For most scripts, the corresponding Figure is automatically generated and store inside the corresponding folder (*i.e.,* `fig1_results`).

The workflow is quite simple and only has the following dependencies: $T_1 \rightarrow T_2$.

Artifact Analysis (incl. Outputs)

For each experiment that is run, the software automatically generates output data in a corresponding results folder. This data can be analyzed manually or processed using the same Bash and Python scripts to produce the plots included in the paper. Most plots are automatically generated after running the programs.

Artifact Evaluation (AE)

C.1 Computational Artifact A_1

Artifact Setup (incl. Inputs)

We provide two alternative methods to install and use the artifact: 1) via a Docker container where all packages are already installed and ready to be used, 2) via source code where the user can manually download, build, and run all the required packages and scripts.

In ≈ 5 minutes, the source code can be downloaded from either GitHub or Zenodo with its unique DOI. The Docker container is only available on Zenodo due to its large size.

- `GitHub: https://github.com/spcl/Uno_SC25.git`
- `Zenodo: https://doi.org/10.5281/zenodo.15916080`

Using Docker: Docker version 26.1.3 was used to generate and test the Docker images.

The Docker container is called `uno_container.tar.gz` and is downloadable from Zenodo.

Load and run the Docker image using the following command. Note, this could take some minutes to run and might need root privileges.

- `sudo docker load -i uno_container.tar.gz`
- `sudo docker run -it uno:1.0`

If viewing images such as plots inside the container is problematic, it is possible to transfer that file to the host machine using:

- `docker cp <containerId>:/file/path/within/container /host/path/target`

It is possible to obtain the `<containerId>` by running

- `sudo docker ps`

Using Source Files: After having downloaded the source code either via `git clone` or downloading and decompressing the archive from Zenodo, the user must install all the required packages using the following commands:

- `./sc25_pkg_install.sh`

If you encounter any error and to avoid conflicts with local packages, we recommend running these commands in a clean Python environment created with `venv`. For example, this is how this can be done:

- `python3 -m venv .venv`
- `source .venv/bin/activate`

The htsim can then be installed by running the following command in the `./sim` directory:

- `make clean && cd datacenter/ && make clean && cd .. && make -j 8 && cd datacenter/ && make -j 8 && cd ..`

Artifact Execution

Once the artifact has been installed using the steps above, the results of the paper can be reproduced by running a series of scripts from the root folder of the project. In particular, the user can run:

- `./sc25_figX.sh` where `X` is the figure number.

Such a script will run all the necessary experiments to reproduce the figures present in the paper.

The run time for the simulations depends on the number of generated flows, the size of the topology, and the degree of the network load. Specifically, with a realistic workload (Figures 10 and 11 in the paper), it can take quite some time as the number of generated flows and the network load are high. Table 3 estimates the time required for running the different scripts, while the list below summarizes in detail the effect of each script:

- `sc25_fig1.sh`: Figure 1 produces a plot showcasing how network communication can be mostly latency bound or bandwidth bound as the RTT of the network changes and for different message sizes.
- `sc25_fig3.sh`: Figure 3 illustrates that MPRDMA + BBR and Gemini fall short in efficiently achieving bandwidth fairness, while Uno provides fast convergence to fairness during a mixed incast scenario with intra-DC flows mixed with inter-DC flows.

Script	Description	Estimated runtime
sc25_fig1.sh	Effect of large network latency	1 minute
sc25_fig3.sh	Simulating Uno's fairness during mixed incast	5 minutes
sc25_fig4.sh	Simulating the effect of phantom queues on queue size and FCTs	5 minutes
sc25_fig8.sh	Simulating Uno's performance during different incast scenarios	30 minutes
sc25_fig9.sh	Evaluating Uno's performance during a permutation workload	10 minutes
sc25_fig10.sh	Simulating different degrees of load with realistic traffic	24 hours
sc25_fig11.sh	Simulating distinct $\frac{inter-DC\ RTT}{intra-DC\ RTT}$ with realistic workload	24 hours
sc25_fig12.sh	Simulating Uno under different queue sizes	1 hour
sc25_fig13.sh	Simulating UnoRC under different scenarios	10 minutes

Table 3: Bash scripts for running experiments and their corresponding run times. Note that the estimates are an upper bound on modern hardware.

- `sc25_fig4.sh`: Figure 4 shows the effect of phantom queues for intra-DC traffic. In particular, phantom queues help reduce the tail FCT of intra-DC flows while keeping queues empty but not under-utilized (only by a controlled amount).
- `sc25_fig8.sh`: Figure 8 showcases more in detail how Uno is both fair and has good completion times when running incast with traffic coming from different sources (only inter-DC, only intra-DC and mixed).
- `sc25_fig9.sh`: Figure 9 showcases the performance of Uno compared to other state-of-the-art algorithms when running a permutation workload.
- `sc25_fig10.sh`: Figure 10 tests Uno against state-of-the-art techniques, such as Gemini, under different realistic workloads (*i.e.,* Google web search and Alibaba's WAN traffic) and distinct degrees of load.
- `sc25_fig11.sh`: Figure 11 tests Uno and other baselines under realistic workloads and different inter-DC to intra-DC RTT ratios.
- `sc25_fig12.sh`: Figure 12 focuses on the performance of Uno when changing queue sizes, simulating much larger inter-DC queues.
- `sc25_fig13.sh`: Figure 13 focuses on the load balancing performance of Uno under different workloads and compares it to other state-of-the-art load balancer algorithms.

Note that from the list above, some figure numbers are missing as those figures don't showcase results but just schematic representations of our work.

Finally, we also provide a script to run and reproduce some of the results with the least amount of time required. In particular, experiments for Figures 1, 3, 4, 8, 9, and 13 will be executed. This can be done by running:

- `./sc25_quick_validation.sh`

Artifact Analysis (incl. Outputs)

Once all (or a subset) of the data has been regenerated, it is now possible to study the results and hence reproduce the results of the paper. Such plots will be generated in different sub-folders of `artifact_results/` depending on the considered experiment and Figure.

For each sub-folder, we report a quick analysis of the generated data and plots:

- `fig1/`: The sub-folder contains the bottom plot of the figure used for Figure 1 of the paper.
- `fig3/`: This sub-folder includes the three files used to generate Figure 3. Each figure represents a different congestion control algorithm.
- `fig4/`: This sub-folder includes two plots used to generate Figure 4: one plot showcase the evolution of the queues with and without phantom queue while the third plot showcases average and 99th FCTs with and without phantom queues.
- `fig8/`: This sub-folder includes 2 files used in Figure 8. One file focus on the `cwnd` evolution of Uno under different incast scenarios, while the other file focuses on comparing Uno's performance to the other congestion control algorithms.
- `fig9/`: This sub-folder includes two files used in Figure 9. In particular, we evaluate Uno's performance during a permutation scenario while also changing the degree of oversubscription across the datacenters.
- `fig10/`: This sub-folder includes two files including the bar charts in Figure 10 showcasing Uno's performance with real datacenter traces.
- `fig11/`: This sub-folder includes the bar charts of Figure 11.
- `fig12/`: This sub-folder includes one file showcasing Uno's performance when changing the inter-DC switch queue sizes.
- `fig13/`: This sub-folder includes 3 files used to generate Figure 13, focusing on UnoRC's performance in a number of scenarios.

We note that all our contributions (C_1-C_3) are present and relevant in all of the results showcased here.