Judicious QoS using Cloud Overlays

Osama Haq
Tufts University

Cody Doucette[*]
Raytheon BBN

John W. Byers
Boston University

Fahad R. Dogar
Tufts University

ABSTRACT

We revisit the long-standing problem of providing network QoS to applications, and propose the concept of *judicious* QoS – combining the cheaper, best effort IP service with the cloud, which offers a highly reliable infrastructure and the ability to add in-network services, albeit at higher cost. Our proposed J-QoS framework offers a range of reliability services with different cost vs. delay trade-offs, including: i) a forwarding service that forwards packets over the cloud overlay, ii) a caching service, which stores packets inside the cloud and allows them to be pulled in case of packet loss or disruption on the Internet, and iii) a novel coding service that provides the least expensive packet recovery option by combining packets of multiple application streams and sending a small number of coded packets across the more expensive cloud paths. We demonstrate the feasibility of these services using measurements from RIPE Atlas and a live deployment on PlanetLab. We also consider case studies on how J-QoS works with services up and down the network stack, including Skype video conferencing, TCP-based web transfers and cellular access networks.

CCS CONCEPTS

• **Networks → Overlay and other logical network structures**; **In-network processing**; **Network reliability**; **Data center networks**; *Network protocol design*; *Network measurement*.

KEYWORDS

Cloud Overlays, Quality of Service, Inter-Data Center Networks, Network Coding, Loss Recovery, Latency Sensitive Applications

ACM Reference Format:
Osama Haq, Cody Doucette, John W. Byers, and Fahad R. Dogar. 2020. Judicious QoS using Cloud Overlays. In *The 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20), December 1–4, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3386367.3431318

1 INTRODUCTION

The limitations of IP's best effort service are well-known: it provides no guarantees on latency, packet loss, or bandwidth, which is restrictive, especially for interactive applications such as voice and video conferencing that require quality of service (QoS) support from the network. Despite decades of research in this area, from "first generation" QoS proposals that required in-network changes (e.g., IntServ [25]) to overlay based solutions in the late 1990s and early 2000s (e.g., RON [16], OverQoS [62]), an ideal solution still remains elusive – a solution that can offer the reliability and performance of in-network solutions while being as easy to deploy as overlay-based solutions.

Fortunately, the emergence of the cloud offers us an opportunity to revisit this problem. By the cloud, we refer to a distributed network of data centers (DCs), inter-connected through a private network (e.g., Azure, EC2, Google Cloud). For any communication between two end-points, we can potentially use the cloud as an *overlay*, with DCs acting as an insertion point for in-network services [41, 47]. A cloud-based overlay offers unique opportunities: cloud paths are well-provisioned, offering low jitter and very high reliability. Also, each DC has visibility into many users and applications, so it can act as a unique vantage point for control and insertion of in-network services. On the flip side, using the cloud as an overlay can be costly: cloud providers charge for the use of their resources (e.g., processing, network connectivity), with wide area network (WAN) bandwidth being particularly expensive [41, 43, 69]. Therefore, we argue that the most effective use of the cloud as an overlay is one that does so in a *judicious* manner, in conjunction with the cheaper, best effort Internet paths.

Toward this end, we present the Judicious QoS (J-QoS) framework, which uses the cloud infrastructure to provide enhanced network QoS. The main goal of J-QoS is to provide reliable and timely packet delivery to demanding applications. J-QoS achieves this goal by offering three services with different cost vs. performance trade-offs.

The *forwarding* service is the simplest one: it forwards packets over the cloud overlay, similar to how IP forwards packets on the Internet, but with the additional reliability and lower latency of cloud paths. A potential use case of this service is switching flows with consistently poor Internet paths onto the cloud overlay, similar to VIA [47]. The forwarding service acts as a building block for two new services, which use the storage and processing capability of the cloud, in addition to leveraging the high quality cloud paths.

The *caching* service provides (short term) storage of packets at a DC, leveraging the storage capability of the cloud – functionality missing in IP routers. Unlike traditional caching approaches (e.g., CDNs), we use the cache for fast packet recovery. In our primary use case, a *copy* of the packet is forwarded along the cloud path and cached at a DC close to the receiver. In case of a packet loss on the direct Internet path, the receiver retrieves the lost packet from the DC cache, rather than all the way back from the source. Because typical Internet path loss rates are low ($< 1\%$), this reactive approach used by the caching service saves on egress bandwidth of the cloud compared to the more proactive forwarding service.

Finally, we present a novel *coding* service, which provides the most economical option for protecting against packet losses on

[*] This work was completed when the author was at Boston University.

CoNEXT '20, December 1–4, 2020, Barcelona, Spain
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7948-9/20/12...$15.00
https://doi.org/10.1145/3386367.3431318

the best effort Internet paths, albeit at a slight increase in delay. The coding service builds on top of the caching service while also leveraging both the processing and storage capability of the cloud. It relies on the observation that not all receivers experience a loss at the same time, so instead of caching *all* the original packets at a DC near the receiver, the DC stores a small number of coded packets. To recover a lost packet on the Internet, these coded packets are combined with data packets from other receivers, using an on-demand cooperative recovery process. For wide-area paths, this cooperative recovery process is faster than an end-to-end retransmission from the source. The coding service thus exploits a number of observations and trends: encoding packets across users is made feasible because of cloud's visibility into many concurrent streams, and independent losses on Internet paths. Similarly, cooperative recovery, using other receivers, can be feasible because of the low (and decreasing) latency between end-points and their nearby DCs.

While these services run in the cloud, J-QoS also provides suitable end-point support, which is important for fully leveraging the benefits of these services, including an API to access the services and a receiver-driven loss detection mechanism. The API allows end-point applications to specify their latency budget, allowing J-QoS to choose the lowest cost service that would meet this requirement. The loss detection mechanism runs on the receiver; it predicts losses based on past packet arrivals and proactively undertakes loss recovery with the help of a suitable service (e.g., caching or coding) running on a nearby DC.

We have implemented a prototype of J-QoS that logically sits just below the transport layer providing enhanced reliability on top of IP's best effort service. It seamlessly works with both TCP and UDP based applications (including encrypted traffic) without requiring any application modification, enabling a holistic evaluation of J-QoS along two broad themes: i) the feasibility and benefits of various J-QoS services in providing timely packet delivery, through measurements on RIPE Atlas [6], and a deployment on the public cloud and PlanetLab [57], and ii) interplay of these services with protocols up and down the stack, with the help of case studies.

Our measurements on RIPE Atlas show the feasibility of our proposed services through latency measurements of the public Internet and the cloud overlay paths. For example, we observe that 80% of the nodes can reach their nearest data center within 20ms, resulting in the caching service recovering packets within a quarter of a round-trip time. Our month-long deployment on a public cloud then helps us quantify the wide area performance improvement for PlanetLab paths. For example, our results show that the coding service is able to recover more than 70% of losses, the recovery is typically within half a round-trip time, and the associated overhead of using the cloud judiciously is far less compared to other services (e.g., forwarding).

Through case studies, we also evaluate how J-QoS interacts with protocols up and down the network stack – we show that: i) J-QoS's enhanced packet reliability can improve the user's QoE experience for a Skype video conferencing scenario, ii) J-QoS can speed up short web transfers by avoiding TCP timeouts and congestion avoidance caused by bursty losses, iii) it is feasible to use J-QoS in certain mobile network scenarios prescribed by bandwidth, energy consumption, and latencies to nearby DCs.

Overall, we make the following contributions in this paper.

- A case for the judicious use of cloud overlays, using them in conjunction with the best effort Internet to meet the desired application QoS (§2).
- The design and implementation of the J-QoS framework, which includes multiple reliability services, each using the cloud in a different way, providing different cost-latency trade-offs (§3).
- The design and implementation of a novel coding service that includes a practical, tunable coding module and a receiver-driven protocol for cooperative packet recovery (§3).
- A multi-faceted evaluation of J-QoS, using both network and user level metrics, on diverse networks (RIPE Atlas, PlanetLab, cellular) and applications (video conferencing, short web transfers) (§5).

2 THE CLOUD AS AN OVERLAY

We consider using a cloud overlay as a potential solution to the network QoS problem. Some interactive applications, such as Skype and Google Hangouts, have already migrated their services to at least partial use of cloud relays [47], but there has been little work in studying how to best utilize the cloud for such QoS-sensitive applications. With COVID-19 and the increasing reliance on these applications (e.g., Zoom), a systematic way to use the cloud overlays has become even more important.To approach this problem, we characterize the properties of cloud paths in terms of network conditions and cost, and ask: can the cloud be leveraged in a cost-efficient way to make up for the Internet's performance limitations?

Benefits. There are two key advantages of a cloud-based overlay:

(1) **Improved Performance.** Measurements show that cloud paths are highly reliable, with a typical downtime target of a few minutes per month [36, 42]. Recent studies show that inter-DC paths have an order of magnitude lower loss rate, better latency, and significantly higher bandwidth, compared to public Internet paths [17, 42]. Similar benefits are being extended all the way up to ISP networks, with major cloud operators providing bandwidth-guaranteed pipes between their data centers and customer premises (e.g., Azure ExpressRoute [12], AWS Direct Connect [8]). These advances in the WAN as well as at the last hop are poised to make the *entire* cloud overlay highly reliable with predictable latency between end-points.

(2) **Ability to insert in-network services.** The cloud infrastructure provides the ability to implement in-network services in software in a scalable and fault tolerant fashion, with the help of network function virtualization (NFV) [58, 59]. These in-network services can leverage the storage and processing capabilities of the cloud to help with timely packet delivery, such as caching or coding of packets – functionality that is infeasible to support in today's IP routers.

Cost. Although the cloud as an overlay provides significant benefits, it can be expensive to use, especially due to the high cost of inter-DC bandwidth. Anecdotal evidence, as well as our discussions with operators, suggests that an inter-continental leased line could be an order of magnitude or more expensive compared to a connection to the best effort Internet. This reasoning underlies several recent proposals that try to make efficient use of inter-DC bandwidth in order to reduce their network costs [41, 46, 49, 53].

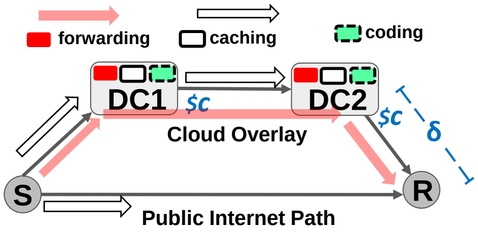


Figure 1: J-QoS Services - An Overview. Forwarding sends packets using the cloud overlay. Caching stores a copy of packet at DC2 while Coding only sends a small number of coded packets across the inter-DC path.

Judiciously Using the Cloud. We argue that in current settings, we only need to rely on the cloud whenever the best-effort Internet cannot provide the desired QoS. For example, an application using an Internet path with 0.01% loss is still getting 99.99% of its packets delivered, so it could potentially make minimal use of the cloud and yet get its desired QoS. To this end, we propose judicious use of cloud resources: leveraging the availability, performance, and other benefits of the cloud only when the best-effort Internet fails to meet the desired QoS. The key to our idea is to not only leverage the performance benefits of cloud paths (which other overlay proposals like VIA [47] propose), but to also judiciously use the storage and processing capability of the cloud, resulting in novel cloud-based services for timely packet delivery.

3 J-QOS DESIGN

J-QoS offers cloud-based reliability services that enhance the best effort service provided by IP. Figure 1 shows the main use-case for the J-QoS services: there is a sender (S) sending latency-sensitive traffic (e.g., voice) to a receiver (R) over a wide-area Internet path (e.g., across continents). Both the sender and the receiver have nearby DCs (DC1 is close to the sender while DC2 is close to the receiver) with a small access latency (δ) and the only cost incurred at the DCs is their egress bandwidth charge, which is denoted by c ($/packet). These DCs run the J-QoS services, which leverage different aspects of the cloud (storage, processing, etc), and offer trade-offs in terms of latency and cost.

The *forwarding* service forwards packets over the cloud overlay, leveraging the reliable inter-DC paths and high egress bandwidth of DCs. In the typical use case of forwarding, the sender forwards the packet to DC1, which forwards it to the receiver using the cloud overlay (via DC2). The resulting packet delivery latency is comparable to the direct Internet path latency, as we show in our evaluation (§5.1). However, it incurs the cloud bandwidth cost of $2c$ (egress bandwidth of both DC1 and DC2).

Our forwarding service use case is similar to VIA[47], which performs selective forwarding and chooses between cloud relays or Internet paths on the basis of performance history. J-QoS, on the other hand, offers other services that trade-off lower costs with slightly higher latencies, as we describe next.

The *caching* service, built on top of the forwarding service, provides on-demand delivery by storing packets at the DC. In the typical case, a copy of the packet is sent on the cloud overlay – from the sender to DC2 via DC1 – but instead of forwarding it all the way to the receiver, it is cached at the DC close to the receiver. In case of a packet loss on the Internet path, the receiver can initiate a pull request to retrieve the missing packet from the nearby DC (i.e., DC2). Compared to the forwarding scenario described above, the caching service can reduce the cost by a factor of two, from $2c$ to c, but at the expense of additional delay which is at least 2δ.

Our *coding* service (Figure 2) uses cloud processing and generates a small number of coded packets at DC1; these coded packets are sent across the inter-DC path and cached at DC2. When a receiver issues a pull request for a missing packet, the DC undertakes a cooperative recovery process with the help of other (nearby) receivers. This service is the extreme point in this design space – it brings the cost down to only $\alpha \cdot c$ (where α is a small constant $\ll 1$). However, it adds some latency by requiring additional delay 4δ once the receiver detects a loss. With latencies to nearby DCs trending smaller over time, and the best effort Internet being sufficient most of the time, the coding service can be a cheaper (but with higher latency) alternative to the caching service, which in turn, is a cheaper alternative to the forwarding service.

To fully benefit from these services, J-QoS also provides suitable end-point support, in the form of a reliability layer that logically sits in between the transport and network layers, thereby enabling support for legacy applications, including both encrypted and non-encrypted traffic. Unlike today's sender-based recovery mechanisms (e.g., TCP), J-QoS includes a receiver-driven recovery protocol (§3.2), which proactively detects losses, and undertakes loss recovery with the help of a nearby DC. Finally, J-QoS employs a simple API and service selection mechanism (§3.3): given an application latency budget, it chooses the cheapest service that can meet the requirement.

The above simplified overview only considers the bandwidth cost for the services, but in reality, the cost depends on the deployment scenario. In §3.4, we discuss how bandwidth costs would dwarf other costs (e.g., processing) if J-QoS services are deployed on a public cloud, and how advanced services like coding would provide significant cost savings over a service like forwarding. Finally, we also discuss how J-QoS services can be used to support a diverse range of scenarios – beyond the primary use case considered in this paper – including support for multicast, partial cloud overlays (where only a single DC is used rather than a full overlay), and selective duplication (§3.5).

3.1 Coding Service

We focus on the design of the coding service as it makes use of both the forwarding and caching services. Our coding service, CR-WAN, uses a full overlay, where multiple senders send a copy of their packets to their nearby DC (Fig. 2). DC1 generates a small number of coded packets, which are sent to DC2 using the inter-DC cloud path. The key aspect of CR-WAN is in how it generates these coded packets – some important considerations include which packets are considered together, what type of coded packets are generated, and at what rate. CR-WAN uses a novel *cross-stream* coding design: coding is done *across* a subset of user streams, which protects against bursty losses or even complete outages on a network path. For example, if (S_1-R_1) experiences an outage, J-QoS undertakes

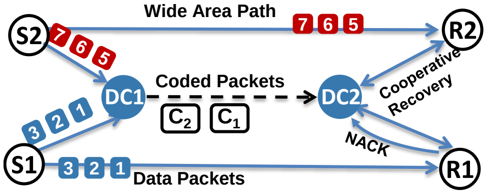


Figure 2: Coding Service Overview. DC1 encodes packets, DC2 stores them and performs recovery upon request.

a *cooperative* recovery process by combining the coded packets at DC2 with the data packets of S_2-R_2 to recover the lost packets.

The cooperative recovery process, however, has its own set of challenges. First, decoding overhead can be high because it requires getting data packets from all other flows in the encoding subset. To ensure that this procedure is invoked only when necessary, CR-WAN also uses *in-stream* coding, whereby it generates a small number of forward error correction (FEC) packets *within* a single user stream, thereby avoiding the potentially costly cooperative recovery for random losses. Second, during cooperative recovery, some packets could be lost or delayed, especially if many streams are involved – we call this the *straggler problem*. J-QoS's cross-stream coding accounts for potential stragglers by generating extra coded packets, thereby treating packets from stragglers similarly to losses on the direct Internet path. We elaborate on how CR-WAN deals with these challenges in §3.1.2.

While CR-WAN provides a cost-effective recovery option, its effectiveness depends on multiple factors, notably the latency and nature of losses on the direct Internet paths, latency between DC to end hosts, the cloud's visibility into concurrent streams, and independent losses across multiple flows. Through CR-WAN's deployment on PlanetLab (§5.2), we shed light on these factors and highlight scenarios where CR-WAN is able to provide unique benefits compared to other loss mitigation techniques (e.g., FEC).

3.1.1 Coding Plan

To specify which packets are appropriate for batching together, we need a coding plan. This plan needs to account for *spatial* and *temporal* constraints while forming a batch of packets over which coding will be applied. By spatial constraints, we mean that only flows with the same destination DC can be considered together for cross-stream coding. For example, if DC1 is in the East US region and is receiving traffic destined for a European DC and an Asian DC, it forms two groups, one for each destination DC. Each flow belongs to one group and DC1 keeps a track of the mapping of flows to groups. Within a group, we pick a further subset of flows based on the arrival timing of their packets to form coding batches.

Temporal constraints restrict packets in a batch to only those packets that arrive within a short interval – this imposes an encoding delay. For in-stream coding, the encoding delay is well-understood (and is considered a limitation of FEC for low bitrate applications) as we need to wait for all packets in a block to arrive before we can generate the FEC packets. However, J-QoS's use of cross-stream coding ensures that encoding delay is typically even lower, because packets from different user streams can arrive within a short time-frame, even if each application individually is generating low bitrate traffic. Finally, our coding module limits the block size (for a given level of protection) and uses timeouts to bound delay.

3.1.2 Coding Rate

Given a batch of data packets arriving at DC1, J-QoS also needs to decide *how many* cross-stream and in-stream coded packets to generate. For both types, the coded packets are created using a block code (for example, Reed-Solomon codes), which allows J-QoS to generate multiple coded packets per batch if desired. Figure 3(a) depicts some of the possible trade-offs, for a batch of 20 packets from four synchronous (for simplicity) flows, A-D. In this depiction, in-stream encoding proceeds horizontally: a single FEC packet (Y_i) is produced for each flow i. Cross-stream encoding proceeds vertically: two cross-stream packets are produced from groups of four packets across flows, i.e., $A2$, $B2$, $C2$, and $D2$ are combined to generate coded packets $X3$ and $X4$.

Coding logically proceeds with two rates: an in-stream encoding rate of $s < 1$ coded packets per within-flow data packets, and a cross-stream encoding rate of $r < 1$ coded packets per data packet, where the data packets are selected among at most k different flows.[1] DC1 must also include information in the coded packets about which flows and sequence numbers are represented, to facilitate later recovery. In our depicted setting, we use $k = 4$, $r = \frac{2}{4}$ and $s = \frac{1}{5}$, but in practice we use fewer coded packets per batch of data packets, with the typical overhead of coded packets <20%.

Coded packets provide protection in multiple ways. In-stream encoding packets protect primarily against random loss, much like traditional FEC. This first line of defense provides faster recovery for random losses. As depicted in Figure 3(b), packet Y_A can recover from the loss of $A3$. Cross-stream encoding, on the other hand, is both much more powerful (it can recover both random and bursty losses), but also incurs a potentially higher delay because of cooperative recovery (§3.2). In Figure 3(d), if some of C's packets are also lost on the direct path, additional protection using more encoding packets could enable recovery at both A and C.

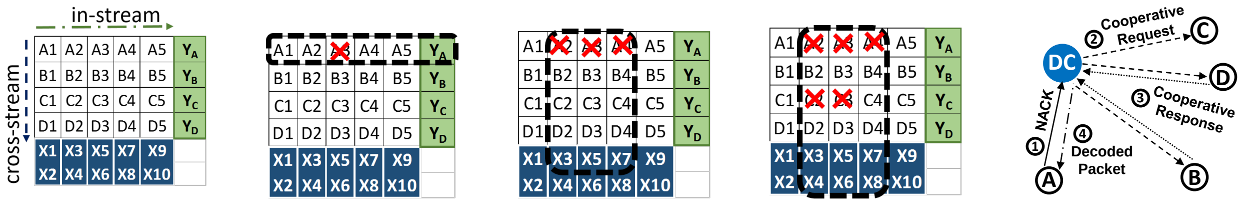
Coding Algorithm Overview. The DC1 cross-stream encoding algorithm[2] groups packets based on their destination DC. Within these groups, packets are selected to be encoded together in an online manner since grouping flows together requires the packet inter-arrival times and lifespans of the flows to be similar. DC1 utilizes timer-based queues with strict flow occupancy limits to deal with varying packet inter-arrival times. In case a receiver goes offline, the encoding process chooses another flow and can generate multiple coded packets to handle the straggler problem during recovery.

3.2 Receiver-Driven Recovery Protocol

For the caching and coding services, J-QoS uses a receiver-driven recovery protocol: the onus is on the receiver to quickly detect packet loss and undertake recovery with the help of its nearby DC. The key challenge in loss detection is how to make a fast,

[1] We deviate from the standard notation of block coding theory, where k data elements are encoded to generate a block of size n, yielding $(n - k)$ coded packets. Data rate and timing constraints may require us to encode before k packets are available.

[2] The details of the J-QoS encoding algorithm can be found in the Appendix A.



(a) cross-stream and in-stream encoding. (b) Y_A protects flow A (in-stream). (c) X's protect flow A (cross-stream). (d) X's protect A and C (cross-stream). (e) Cooperative Recovery.

Figure 3: Coded Packet Generation and Recovery.

accurate prediction of whether a packet is lost (and thus needs to be recovered with help from the nearby DC).

The caching and coding services provide (short term) storage of packets inside a DC. For any packet using these services, there should be an associated *timeout* value and an *identifier* that can be used to retrieve that packet. These concepts are well known in the context of prior proposals that support in-network caching (e.g., NDN [45], XIA [39], etc.) or indirection-based architectures (e.g., i3 [61]). For the use cases considered in this paper, in-memory caching of packets is sufficient, but other scenarios could benefit from longer term storage of the packets (e.g., DTN [33], SlackStack [29]). Similarly, any unique identifier schema (e.g., XIDs [39], URIs [45], etc.) can be used, although for convenience our prototype uses unique packet sequence numbers.

Loss Detection. In J-QoS, the receiver detects a loss if either a gap in sequence numbers is detected (the simple case) or a timer expires for the next expected packet. Setting a suitable timeout value – low enough for fast recovery, but high enough not to cause spurious timeouts – requires learning and predicting packet arrival times. While this opens up the possibility of using machine learning algorithms, our current design uses a simple two-state Markov model that works well for our workloads. The model uses a *small* timeout value for packets arriving within a burst (i.e., sub-RTT scale), and a *long* timeout value across packet bursts or application sessions. These values are chosen based on previously observed inter-arrival times of packets and the J-QoS loss detection module switches back and forth between them accordingly.

Once a NACK is sent to the nearby DC, the recovery depends on the service being used. In the case of caching, recovery is simple, as the data packet can be transmitted to the requesting receiver. CR-WAN's recovery is often more involved, as it may need to undertake *cooperative* recovery.

Cooperative Recovery. Figure 3(e) shows the steps involved in the cooperative recovery protocol. After receiving a NACK from receiver A (step 1), DC checks that there are sufficient cross-stream coded packets to conduct the recovery process. If so, DC then sends cooperative requests to relevant receivers, since they have the data packets needed to decode the missing packets (step 2). The J-QoS module at the receivers stores the data packets for a few (e.g., 1-2) RTTs and purges them after this time.

DC then processes any incoming cooperative recovery responses from the solicited receivers (step 3). By tracking responses, DC can tabulate the number of cooperative responses for each recovery event. For each loss, once the number of responses is equal to $k-1$, then recovery is possible. DC then decodes the lost packets and sends them to the receiver (4). Depending on the number of cross-stream coded packets, DC may only require a few of the receivers to respond in a timely fashion, thereby ignoring stragglers (such as C in Figure 3(e)) that can cause delay in recovery. Since recovery is time-sensitive, the protocol fails silently if not enough coded packets or cooperative recovery responses are received within a set deadline. We discuss these conditions under which recovery is not possible in Section 5.2.

3.3 J-QoS End-to-End Workflow

J-QoS services not only require adequate end-point support in the form of loss detection and recovery but also a way for applications to specify their demands. We now describe how J-QoS framework selects and switches between services according to application demands and changing network conditions.

API and Service Selection. Applications registering with J-QoS use a simple `register(...)` API to express their latency budget and target destination. Based on the budget, J-QoS selects the lowest cost service that can meet this requirement. As our services operate on a continuous spectrum, with CR-WAN being the cheapest and forwarding being the expensive, the J-QoS service selection module picks the cheapest service as long as it can meet the latency budget. In our evaluation, we show that these services operate in different delay regions and can be mapped to a latency demand (§ 5.1). J-QoS end hosts also use `register(...)` to bootstrap with the membership management module running on a central DC. This module relays the nearest DC location as well as the latency information required to compute service delays to the registering end host.

For packet routing, we currently employ next-hop forwarding based on destination addresses. Given the small scale of the overlay network in J-QoS, the next hop decision is simple and made in a centralized fashion. The next hop could be another J-QoS service, an endpoint (e.g., the receiver), or a multicast group.

Delay Computation. The J-QoS service selection module uses the destination of the application flow to calculate the service delay. Some of the delays, such as the latency between `DC1-DC2` are pre-computed and stored at each end host during initial bootstrap. Other delays, such as `S/R-DC` latency (δ) and `S-R` latency are initially

v_{dc}	VM instance price (\$/hr)	i	Ingress
f_{dc}	# of flows sharing the VM	e	Egress
t_j	Time duration of flow j	p	Internet Packet loss rate
b_{dc}	Bytes sent/received at DC	r	Cross-stream coding rate
λ	Unit bandwidth price (\$/GB)	s	In-stream coding rate
Δ_N	Total NACK bytes	Δ_{crsp}	Coop response bytes
Δ_{creq}	Coop request bytes	B_{fwd}	Bandwidth cost of fwd
B_{cache}	Bandwidth cost of caching	B_{cod}	Bandwidth cost of coding
C_{cod}	Compute cost of coding	$C_{fwd-cache}$	Compute cost of fwd & cache

Table 1: Notations for the cost model.

	DC1	DC2
B_{fwd}	$b\lambda_{i_{dc1}} + b\lambda_{e_{dc1}}$	$b\lambda_{i_{dc2}} + b\lambda_{e_{dc2}}$
B_{cache}		$(b + \Delta_N)\lambda_{i_{dc2}} + bp\lambda_{e_{dc2}}$
B_{cod}	$b\lambda_{i_{dc1}} + b(r + s)\lambda_{e_{dc1}}$	$(b(r + s) + \Delta_N + \Delta_{crsp})\lambda_{i_{dc1}}$ $+(bp + \Delta_{creq})\lambda_{e_{dc2}}$
$C_{fwd-cache}$	$v_{dc1}/f_{dc1} \cdot t_j$	$v_{dc2}/f_{dc2} \cdot t_j$
C_{cod}	$\alpha \cdot (v_{dc1}/f_{dc1} \cdot t_j)$	$\alpha \cdot (v_{dc2}/f_{dc2} \cdot t_j)$

Table 2: J-QoS Cost Model. Service cost is the sum of bandwidth (B) and compute (C) cost at DC1 and DC2.

assumed to be average values based on the latency of existing end-points communicating with their nearby DCs. The delay values are updated once communication starts between the end-points.

Feedback. Finally, the service selection decision is communicated to the J-QoS sending module so it can route the packets accordingly. The service selection mechanism receives the packet delivery statistics from the receiver and can decide to upgrade to an improved service, if the existing service is not meeting the application's latency demand. As our caching and coding services mask packet loss from the sender, J-QoS can use this feedback information to indicate congestion to higher layers that may require this information (e.g. TCP).

3.4 Deployment Model and Cost

Deployment model. We believe a J-QoS-like service can potentially be deployed by the cloud infrastructure provider, an ISP, or a third-party service provider who pays the cloud operator only for the infrastructure usage. Users can explicitly opt into using J-QoS if they require a more reliable packet delivery service – for example, implemented as a paid service or added value proposition. One recent example of a similar paid service is the network tier service by Google Cloud [9], which offers packet delivery using either standard transit ISP (at lower cost) or Google's own network (at higher cost) between its DCs [5].

The deployment model may also have implications on the practical usage of J-QoS design. For example, an ISP or cloud provider with full control over its resources (e.g., using SDN [43]) may prioritize the inter-DC traffic for even higher resilience and lower delay, leading to even higher recovery efficiency. An ISP providing the service can also reduce sender overhead by duplicating the packets in the core as well.

Deployment Cost. We use a relatively simple model to estimate the cost of J-QoS services for a single flow (Tables 1 & 2). This model estimates the compute and bandwidth cost of each service; we do not consider storage cost, as J-QoS services store packets in the local (currently free) storage of the cloud virtual machine (VM).

The *compute* cost of a VM instance in a DC is shared among all the flows traversing it. The coding service requires encoding and decoding of packets, and hence has a higher cost (represented by α) compared to the other services.

The *bandwidth* cost (B) of a single flow at a DC is the sum of the ingress and egress bandwidth cost at that DC. For the forwarding service, the same amount of bytes pass through both DCs; for the caching service, DC2 only sends packets in case of Internet packet loss; and the coding service utilizes cross and in-stream encoding to reduce egress cost at DC1; however, cooperative recovery adds overhead at DC2. Finally, the bandwidth costs could be different across DCs as cloud providers typically have different prices for different regions.

We used this model in our evaluation (§5.1.1) to compare the actual costs of using the J-QoS services under a range of scenarios. The key takeaways are: i) the bandwidth cost dwarfs the compute cost, so the coding service is the cheapest option; ii) current pricing schemes, which include free ingress bandwidth and expensive egress cost in certain regions, are helpful for the coding service, but the coding service remains a better choice even if we relax these conditions; and iii) the coding service remains the cheapest option for typical loss rates, but for persistently high loss rates (or outages) it is more economical to move to caching or forwarding.

Currently our model does not provide support for storage costs and excludes the cost of duplication at the sender. We discuss how to expand this model to include more use cases in section §7.

3.5 Other Use Cases

We now discuss a number of other use cases involving the forwarding and caching services, how applications might use these services (e.g., selective duplication), and cost models.

Forwarding. For even higher reliability, we can use *both* the Internet and the cloud overlay: the sender can use the forwarding service to transmit a copy of the packet to the receiver. This use case is most beneficial for mission critical applications like financial transactions [3]. In contrast, a *partial overlay* (single DC) scenario may not offer the benefits of reliable inter-DC paths, but it costs less, and can still benefit from the high egress bandwidth of the DC. This use case can be extended to support a multicast scenario: the sender sends its stream to the cloud forwarding service which forwards it to the multicast group, again leveraging its high egress bandwidth. This scenario can be useful for applications like video streaming and software distribution.

Caching. The basic caching use case described earlier could be extended to support a *hybrid multicast*, which provides a cheaper alternative to the cloud-based multicast that uses the forwarding service exclusively. In hybrid multicast, the sender uses the public Internet to send its stream to all the receivers; a copy of the stream is sent to the nearby DC where it is cached. If a receiver fails to receive a packet, it goes to the DC and retrieves it.

The caching service is also useful for mobility scenarios, providing an on-path caching/rendezvous point for mobile-hosts, similar to Internet architectural proposals like NDN [45], XIA [39], and i3 [61]. For example, a mobile sender only sends packets to a DC, where they are cached. The receiver, whenever it is online, pulls the packets from the nearby DC rather than requiring the mobile sender to come online and retransmit the packets.

Selective Duplication. J-QoS can also support scenarios where applications may want only some packets to be duplicated and sent through the J-QoS services. Some examples of such packets could be an I-frame for video streaming, important user actions for gaming or AR applications, and the last packet of a window for short TCP transfers [34]. Such selective duplication can provide a lower overhead alternative to the full duplication option described earlier, and can be used in scenarios where sources have limited bandwidth or a limited budget for packet recovery. We evaluate benefits of selective duplication in §5.4.

4 J-QOS PROTOTYPE

The J-QoS prototype [4] is implemented in C++ and operates in user space. Our implementation uses UDP for forwarding application traffic, coded packets, and cooperative recovery packets, and uses TCP for control channel traffic between the endpoints and the data centers. J-QoS can intercept outgoing packets for a specific application using iptables [13] and the NetFilter library [64] or it can act as a local proxy receiving data from applications.

Our prototype uses Reed-Solomon codes to encode and decode application data using the open-source zfec [66] library, and uses 25ms for the small timer and RTT for the long timer. Finally, we tune the parameters related to coding (coding rate, timers, and queues) on a per-application basis, depending on the application's characteristics and requirements.

Coding Parameters. For cross-stream coding, we use a default of two cross-stream coded packets ($r = 2/k$) to mitigate the effects of stragglers and protect against bursty losses and outages. In practice, we bound k to a moderate value ($k <= 10$ in our evaluation), since larger values add significant overhead in the cooperative recovery process. When more than k flows use J-QoS concurrently at an ingress DC, the DC organizes them into subgroups of at most k flows per group.

For in-stream coding, we find that for interactive applications – where the average frame rate is 10-15 fps and the average frame is composed of 2-5 packets [63] – it is best suited to send an in-stream packet for each frame ($s = \frac{1}{5}$), although that results in relatively higher overhead, so applications with a low cost budget can choose to fall back to cross-stream coding only. The in-stream encoding overhead is less for applications that send back-to-back packets, such as TCP flows, where a single coded packet can be sent for an entire TCP window (e.g., $s = \frac{1}{16}$ or $s = \frac{1}{32}$).

5 EVALUATION

We perform a multi-tiered evaluation with the goals of answering: (1) How feasible are J-QoS services in the real world (§5.1)? (2) How effectively does J-QoS coding service recover packets within a time budget for wide-area paths (§5.2)? (3) How does J-QoS perform in the contexts of challenging application (§5.3), transport (§5.4), and network (§5.5) requirements?

5.1 Feasibility of J-QoS Services

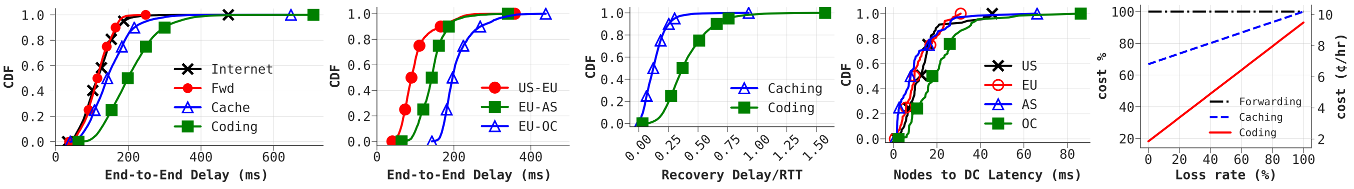
Our goal is to evaluate the feasibility of J-QoS services using latency data from hosts around the world.

Methodology. We use the RIPE Atlas testbed [6] and Amazon Web Services (AWS) [7] data centers to measure latencies of public Internet and cloud overlay paths. In our scenario, RIPE Atlas anchor nodes are senders and probe nodes are receivers. We use AWS data centers near senders and receivers to form a full (2-DC) cloud overlay. Overall, we measure 22K paths spanning four continents: US, EU, Asia, and Oceania [4].

We measure the following latencies for each pair of RIPE Atlas nodes: δ (S/R-DC), x (DC1-DC2), and y (S-R). We compute delay for forwarding service as $x + \delta_{S-DC1} + \delta_{R-DC2}$, caching as $y + 2\delta_{R-DC2} + \Delta$ and coding as $y + 2\delta_{R-DC2} + 2\delta_{R'-DC2} + \Delta$. Note that, for caching and coding, we include y, the one-way latency from S to R. We use Δ to represent the delay of a caching/coding pull request if it reaches DC2 before the desired packet arrives from the sender to DC2. We also represent cooperative recovery delay as $\delta_{R'-DC2}$. In this case, we compute it as the maximum R-DC2 latency of five random nodes in the same region.

J-QoS services can meet latency budget of applications. Figure 4(a) shows the end-to-end packet latency for J-QoS services as well as the direct Internet latency for all paths. We make three observations. First, using the (indirect) cloud overlay does not inflate latency compared to using the direct Internet: for a majority of the paths, the forwarding service has a latency similar to the Internet paths. Second, Internet delivery has a long tail compared to the forwarding service, confirming earlier findings by Microsoft [47] that some Internet paths are persistently low quality, and it is better to completely switch to a cloud overlay for such paths. The forwarding service can be a great fit for such cases. Third, we also observe that packet delivery, using the caching service, takes up to 200ms for 85% of the paths (50% for coding). This is acceptable delay for many latency-sensitive applications that require timely packet delivery [1, 38, 55]. Therefore, for these set of paths, both caching and coding can provide similar benefits as the forwarding service but at much lower costs. These results add to the growing evidence that cloud overlays can be feasible for a diverse range of end-to-end scenarios (e.g., cloud middleboxes [59], web transfers [23], etc.).

Service delays vary across regions. Figure 4(b) shows the region-wise end-to-end packet delivery latency for the caching service. We observe that, in the US-EU region, packet delivery takes less than 200ms for 95% of the paths. We ascribe this to the short one-way latency between the majority of the nodes in the US and EU regions as well as low values of δ in the EU region. It also means that, for US-EU paths, caching and coding can provide similar benefits to the applications as the forwarding service. On the other hand, due to high one-way latency on EU-OC paths, only 50% of the paths deliver packets within 200ms, requiring the rest of the paths to rely only on forwarding. We also find that other paths (not shown e.g., US-AS, US-OC) follow a similar pattern as EU-AS paths. For these paths, stricter delay requirements will result in a subset of the paths using the caching service. Region-wise coding service delays (not depicted) follow a similar pattern, albeit shifted slightly to the right.



(a) (b) (c) (d) (e)

Figure 4: J-QoS service feasibility. a) End-to-End packet delivery latency is within acceptable range. b) Caching service latency varies across regions. c) Most packets are recovered within 0.5xRTT. d) Latency to nearest DC is small. e) J-QoS service costs from OC to EU, Coding beneficial for typical loss rates.

These results show that based on the delay demand and the end hosts region, we can potentially select a suitable service.

On-demand recovery delays. A traditional retransmission-based recovery from the source takes at least one RTT, whereas J-QoS's caching and coding services retrieve the missing packet from a nearby DC. We compare this difference in delay by plotting the recovery latency as a fraction of the RTT.

Figure 4(c) shows that 75% of the time, these services can recover packets within 0.5xRTT of the direct Internet path. We also observe a clear separation between service recovery times, e.g., 90% of the time, caching service can recover packets within 0.25xRTT whereas coding service can only recover packets 20% of the time. Most of the coding service benefits are in the 0.25xRTT to 0.5xRTT range; we also confirm this in our real world deployment (§5.2).

End host to DC latency (δ) is small. J-QoS services rely on end hosts having low latency to their nearest DC, so we now focus on these latencies. Figure 4(d) shows the value of δ for RIPE Atlas probes in different regions (i.e., EU, Asia, Oceania, US). We observe that 50% of paths have δ less than 10ms in Europe and Asia. We also observe that 20% of paths have δ higher than 20ms. While Oceania region has relatively high δ compared to the other regions, we observe that 75% of probes can be reached within 25ms. Depending on the application's latency budget, paths with high δ can still utilize coding and caching services.

5.1.1 Cost Feasibility

We now turn our attention to cost and use our cost model to estimate J-QoS cloud cost for a single user. Our main goal is to understand where each of the J-QoS services provides the most benefits in proportion to cost.

Methodology. We consider a Skype-like application which sends data at 1.5Mbps [10] for 1 hour with 20% coding overhead ($r = \frac{2}{10}$). For our analysis, we consider Amazon (AWS) and use an on-demand compute-intensive VM (8vCPUs, 15GBMem) and the cheapest bandwidth prices for data transfer.

Compute & bandwidth pricing. We observe that compute requirements for the coding service is approximately 2.6× more than that of the forwarding service[3]. In our setting, with current AWS pricing, the median compute cost for a single flow is 0.19¢/hr for the forwarding/caching services and 0.51¢/hr for the coding service. We also observe that the cloud bandwidth pricing varies significantly across regions. For example, US and EU DCs have low egress cost (2-5¢/GB) for both inter DC and Internet destinations compared to other DCs in Asia, South America, and Africa (8-14¢/GB).

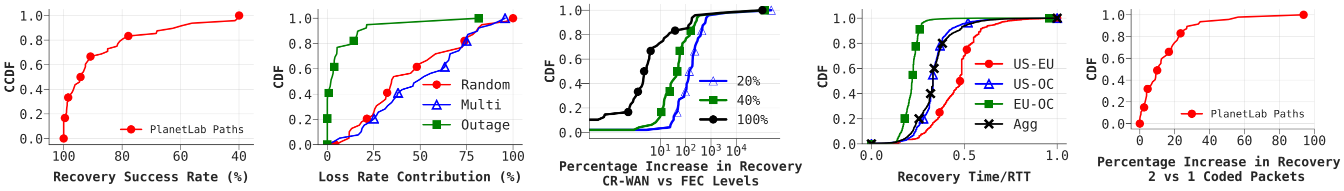
We estimate the cost of our services for various regions at different loss rates and make the following observations.

Coding provides significant benefits under certain settings. Figure 4(e) shows the normalized and actual cost of J-QoS services from OC to EU with the forwarding service as the baseline. We observe that the coding service provides significant cost benefits compared to the caching and forwarding services. This is due to the high inter-DC egress cost of OC region (9.8¢/GB). In this case, CR-WAN incurs a DC1 egress cost of 1.32¢/hr, whereas caching and forwarding cost 6.62¢/hr. With the current pricing, these benefits exist for all the flows originating from Asia, Oceania, South America, and Africa and terminating at the US and EU.

Coding is beneficial for typical Internet loss rate. While the previous result is an example of a region with high DC1 and low DC2 egress pricing, CR-WAN benefits vary for regions with low DC1 and/or high DC2 egress pricing. We observe that the loss threshold for these regions varies from 10% to 50%, i.e., coding remains the primary service until loss rates are greater than this threshold. This means that for typical Internet loss rates (<1%), coding will provide cost benefits in *all* of the regions, irrespective of where the flow originates and terminates. However, for persistent high loss rates or complete outage scenarios, a flow using caching or forwarding is more economical.

Sensitivity Analysis. Although the current cloud pricing model provides free ingress, this can change in the future. Therefore, we also evaluate the feasibility of J-QoS services by factoring ingress cost into our model. We use varying prices from $0.01/GB to $0.1/GB, in accordance with current and past cloud prices [2]. We observe that for typical Internet loss rates, at maximum ingress price, the average cost of the forwarding service doubles, whereas the cost of the coding service is around 40% of the new forwarding cost. We also observe that overall service costs become very high. In the above scenario, the cost of forwarding increases from 10¢/GB to 23¢/GB, whereas the cost of coding increases from 1.8¢/GB to 9.5¢/GB. Hence we observe similar patterns as discussed earlier, with service costs being high. Overall the coding service remains

[3] We determine this based on our prototype evaluation – Appendix B.



(a) (b) (c) (d) (e)

Figure 5: CR-WAN's performance on PlanetLab paths.(a) CCDF of successfully recovered packets. (b) Loss episode contribution to loss rate on paths with greater than 80% recovery. (c) Percentage increase in CR-WAN recovery rate vs FEC with 20%, 40%,and 100% packet overhead on direct path (note x-axis). (d) Packet recovery times as ratio of direct path RTT. (e) Percentage increase in recovery rates using 2 cross coded packets per batch versus 1.

cheaper than the caching and forwarding services but the increase in price might make these services infeasible for some users.

We also evaluate scenarios where the changes in the network pricing might result in the coding service cost to become too high. If, in the future, the ingress cost from the Internet increases to \$1/GB, the cost of the coding service will be same as the forwarding service. However, this scenario is harder to envision, as cloud providers rely on their customers to upload large amounts of data for various tasks (e.g., storage, analytics).

5.2 CR-WAN Deployment and Evaluation

We have evaluated J-QoS services under various controlled settings, verifying their ability to handle different uses which we described earlier. In this section, we evaluate our coding use case, CR-WAN, with its deployment and evaluation on the PlanetLab testbed. We select this service because it builds on the other services and adds the most delay in terms of packet recovery, representing the worst case scenario for the effectiveness of J-QoS services.

Experimental Setup. We ran CR-WAN as a service on five different DCs of Microsoft Azure [11], located in US (East, West), EU, Asia, and OC, for over a month. We use F1 type virtual machines, which are compute-optimized with 2.4 GHz single core and 2 GiB RAM. We evaluate 45 PlanetLab wide area paths spanning four different continents [4].

We run a simple constant bitrate application on the PlanetLab nodes. To observe long-term time-averaged behavior without overloading the paths, we use ON/OFF periods with Poisson OFF times and constant ON times. In each ON interval, we send packets for 5 minutes with 10ms frequency; we set the mean OFF time to be 55 minutes. DC1 relays the start of each ON interval to senders using a separate control channel, thereby ensuring that senders are (loosely) synchronized. We use $r = 2/6$ and $s = 1/5$ as our coding parameters. Given the high churn rate of PlanetLab nodes, the total samples collected from each path varies. Typically, we recover 500-800 samples per path, which translates to 3-5 weeks of measurement collection. Our wide-area evaluation makes five key findings, summarized below:

Most losses happen on wide-area links and CR-WAN is able to recover them. CR-WAN is able to recover 78% of all packets that are lost on the PlanetLab paths. Loss rates on these paths are relatively high: up to 0.9% loss, with 40% of paths having a loss rate greater than 0.1%. Overall, we lose 0.02% packets in our experiment and we consider any packet that takes longer than one RTT to recover as a lost packet. As we discuss later, most of the packets that CR-WAN is unable to recover are lost on the access paths. If we ignore those losses, CR-WAN's packet recovery goes up significantly. Figure 5(a) elaborates on the above results – it shows a CCDF of the fraction of successfully recovered packets (i.e., those lost packets that are recovered within one RTT) for all PlanetLab paths. Most paths experience high recovery (low unrecovered packet rate) – overall, 82% of paths successfully recover more than 80% of lost packets.

CR-WAN's coding is able to handle a wide range of loss patterns. We next zoom into the loss patterns to understand what types of losses are being recovered by CR-WAN. Figure 5(b) shows a CDF of loss episode patterns observed on PlanetLab paths that have greater than 80% packet recovery (82% of total paths). We look at the burst length of the loss episode and classify them as Random (single packet loss), Multi-Packet (2-14 packets), and Outage (>14 packets). We observe all three types of loss patterns on the chosen paths. While random and multi-packet bursts contribute more towards the loss rate, outages are not uncommon on these paths. Our data shows that 45% of paths see outages that last from 1 to 3 seconds. Our recovery rates show that J-QoS service can handle multiple types of burst lengths, quickly.

Most access losses can be recovered using existing techniques. While access losses (between source-DC1 and DC2-receiver) are not the main focus of J-QoS, we look at their loss characteristics to see whether well-known techniques can be used to recover such losses. Our results show that around 98% of such losses occur on source-DC1 paths and that a significant fraction, 90%, of loss bursts are single packet losses and can be recovered using simple retransmissions (ARQ) or other simple redundancy-based techniques (e.g., [34]) at the edges (i.e., between the end-points and the DCs). In future work, we plan to augment J-QoS to incorporate this observation.

CR-WAN vs. On-Path FEC schemes. To compare CR-WAN with traditional, on-path FEC packet recovery schemes, we perform a what-if analysis on the probes sent on the direct PlanetLab paths. Our goal is to compare CR-WAN with sending different numbers of FEC packets on the direct path. We divide the probes into 5 packet bursts and treat each burst as either all data or all FEC packets. We then compute recovery success rates for 20% ($s = \frac{1}{5}$), 40% ($s = \frac{2}{5}$), and 100% ($s = \frac{5}{5}$) FEC overhead. We also assume that, for CR-WAN, access losses can be recovered using existing techniques.

Figure 5(c) shows the percentage increase in recovery rates for all the paths using CR-WAN, compared to different levels of FEC. We observe that even at 100% overhead (full duplication), 90% of the paths had at least one loss episode that could have been recovered using CR-WAN but not with on-path, 100% FEC overhead. Further, 10% of the paths observe more than 160% improvement in recovery rates with CR-WAN compared to full, on-path duplication. These are paths that experience long burst of losses or outages that cannot be recovered using FEC on the direct path. For the 20% overhead setting, 100% increase in recovery rate is seen by 70% of the paths. This result shows that there exist paths for which CR-WAN's cross-stream coding is more effective in recovering from outages and bursty losses compared to traditional, on-path FEC based schemes.

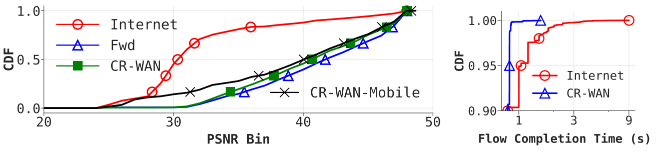
CR-WAN's loss recovery is usually fast. We next look at packet recovery time using CR-WAN, which Figure 5(d) depicts for paths in different regions. We show our recovery times as a ratio of direct public Internet path RTT between the source and destination. We note that 95% of packets are recovered within 0.5 × RTT. As expected, we observe faster recovery for paths with higher absolute latency on the direct public Internet path. For example, on low RTT paths between the US and EU (110-130 ms), we see higher recovery times as a proportion of RTT, but in terms of absolute latency, 90% of packets are retrieved within 75 ms. We also observe that receiver-DC2 RTTs on these paths vary significantly. For example, the RTT between receivers in the EU and their nearest data center varies from 16-70 ms (μ = 28 ms). However, as cloud providers continue to strive towards reducing their latency to end-users [27], we expect CR-WAN recovery times to continue to improve over time.

Finally, we observe two systematic reasons contributing to the tail in the recovery time (Figure 5(d)): delay in detecting and recovering a loss (e.g., due to delayed NACKs) and delay in arrival of coded packets at DC2. Overall, the percentage of recovered packets that fall outside of a reasonable time budget value is low and only accounts for roughly 1% of the recovered packets.

Recovery time is improved due to straggler protection. Last, we show the benefit of using extra cross-stream coded packets to provide protection against stragglers during cooperative recovery. Figure 5(e) shows the performance gains using two cross-stream coded packets per batch, as opposed to one. We observe that with adequate protection of two packets per batch, 60% of paths see greater than 10% improvement in recovery rates. We also observe that the recovery times decrease by at least 50 ms for 70% of the recovered packets (not shown) – in some instances, the difference is some stragglers that take several seconds. This further justifies our choice of default parameter values for PlanetLab paths.

5.3 Case Study: Skype Performance

We run J-QoS services under Skype's video conferencing scenario to measure their interaction with a popular, interactive application. We focus on the performance of Skype in wide-area settings where outages occur (similar to ones described earlier in our wide-area evaluation). To do so, we leverage the cloud path to run the video conference in three experiments. First, we examine how the video quality degrades during an outage along a public Internet path used by Skype. We then duplicate *all* Skype packets over a cloud path (J-QoS's forwarding service) to show that such a path can indeed make up for lost packets during outages. Finally, we use CR-WAN to selectively transmit coded packets over the cloud path and perform recovery at the receiver.



(a) (b)

Figure 6: a) PSNR scores of a set of video conferences, b) Tail of TCP flow completion times (note y-axis scale).

Testbed and Measurement Procedure. We use a similar testbed to that used by Zhang et al. [70], in which clients communicate using Skype's video conferencing service. We connect clients running Skype for Linux 4.3 in a LAN, and emulate wide area path characteristics such as latency, packet loss rate, and jitter.

We use Skype's screen sharing mode to transmit a pre-recorded video that closely represents the normal motions of human interaction during a video conference. We then compare the quality of each received video against the reference video by converting all videos to raw (uncompressed) format, and compute objective QoE scores on a frame-by-frame basis using VQMT [40]. Although objective video quality metrics are not as reliable as subjective metrics given by users (such as Mean Opinion Score), they are sufficient to approximate the quality of the video on a frame-by-frame basis. We show the scores of each frame in a CDF to approximate the quality of each video in aggregate.

Use of the forwarding service enables higher QoE. Figure 6(a) shows the video quality results as we vary the network conditions and paths used. When a 30 second outage occurs along the Internet path, Skype's built-in FEC mechanism is insufficient to maintain an acceptable level of QoE. The video quality degrades with pixelation and frozen video, and the number of frames with poor PSNR scores significantly increases. Due to the high availability of the cloud path, when we use the forwarding service during the 30-second Internet path outage, virtually all packets reach the destination, preserving the video quality (similar to an Internet path with a 0% loss rate). This shows that Skype is amenable to using J-QoS services running in tandem with it to correct losses on its direct public Internet path.

CR-WAN achieves similar QoE as forwarding but with less bandwidth. When running Skype over CR-WAN, we disable in-stream coding on the cloud path (s = 0), since Skype uses its own FEC techniques on the Internet path to recover lost packets [63]. To use cross-stream coding, we inject three ~200 Kbps background UDP flows whose packets are coded with Skype packets at DC1 at a rate of $r = 1/4$, with $k = 4$. Figure 6(a) shows that CR-WAN achieves a similar level of QoE compared to using the forwarding service. We also observe that CR-WAN uses much less bandwidth. In our experiments, J-QoS sent just 13.4% as many packets and 13.6% as many bytes as did the forwarding service.

5.4 Case Study: TCP Performance

We now evaluate the performance of TCP if it is used over J-QoS. Our goal is to understand how the additional reliability provided by J-QoS interacts with TCP's own reliability and congestion control mechanisms, and whether it can provide any additional benefits. We also evaluate whether we can use J-QoS services for only some (selective) TCP packets rather than all the packets. We focus on TCP short flows as they are latency sensitive and do not require high throughput.

Experimental Setup. Our experimental setup is inspired by a similar experiment conducted by Google to evaluate different loss mitigation techniques for their web transfers [34]. Using Emulab [65], we emulate the same topology and loss model as used in the Google study: we consider a 200 ms RTT between end hosts and loss probabilities of 0.01 for losing the first packet in a burst and 0.5 for each subsequent loss. We pick CR-WAN for our analysis as it incurs the highest delay out of all the J-QoS services. We consider a client-server scenario, in which a client sends a 12B request and receives a 50 KB response from the server. The RTT between server/client-DC paths is 30 ms with an RTT of 200 ms on the DC1-DC2 path. We make 10K requests each for TCP and TCP over J-QoS.

J-QoS reduces tail latency for lossy short flows. Figure 6(b) shows TCP's flow completion times with and without J-QoS. We observe that TCP suffers from a long latency tail that goes up to 9 seconds, whereas J-QoS reduces the tail significantly. Our analysis shows that TCP is able to recover from most of the losses (using SACK), but there are some losses which are problematic for TCP, and hence cause the long tail. Such losses typically occur at the start of the connection, e.g., SYN-ACK(s), or at the very end. Such losses cause TCP to timeout, and successive losses mean that these timeout values could become huge, resulting in the long tail for TCP. J-QoS is able to reduce flow completion times by quickly recovering these losses. As soon as a packet is recovered by J-QoS, our TCP client sends an ACK to the server, effectively hiding the loss, and avoiding TCP timeouts.

Selective duplication can yield some benefits. When full duplication at the source is infeasible – due to limited access bandwidth or applications with high bitrates – we can use J-QoS only for selected packets. To demonstrate the feasibility of such a strategy (and its potential benefits), we modify our TCP experiment and only duplicate SYN-ACK packets. We observe that selective duplication reduces tail completion time by 33% (83% with full duplication). Other examples of such duplication can include I-frames for video streaming, important user actions for gaming or AR applications, and the last packet of a window for short TCP transfers [34].

5.5 Case Study: Mobile Networks

Some of the J-QoS services make assumptions that can be challenged in mobile networks, as mobile settings have different bandwidth, power, and latency characteristics. We pick CR-WAN for our analysis as it subsumes other services, in terms of its overhead. Our findings suggest that while it seems feasible to run CR-WAN on mobile hosts, it may be best to use selective duplication to avoid extra overheads.

Duplicating traffic can be feasible. The bandwidth provided to cellular devices can vary greatly [67] – our survey of major US carriers shows users can typically expect 2-5 Mbps uplink bandwidth. Therefore, we consider whether the most bandwidth-intensive part of CR-WAN – the duplication of traffic to the cloud path at the sender – works within the link rates of mobile networks.

We modified our Skype testbed (§5.3) to tether the sending host to a mobile device connected to an LTE network, and observed that the overall bandwidth required by CR-WAN to duplicate a Skype video stream was 1.5 Mbps and well within the uplink bandwidth afforded by the LTE network (~5.0 Mbps). However, in general the recommended bandwidth for HD video calls in Skype is 1.5 Mbps [10], so duplicating that traffic could reach the capacity of uplinks in some networks. We also tested how CR-WAN affects other ongoing transfers on the device, and found that the transfer time for 5 MB files over WhatsApp is not affected by CR-WAN running simultaneously.

For data-intensive uses, J-QoS may need to utilize the forwarding service so that packets are not duplicated. Alternatively, mobile applications might selectively duplicate packets when using caching or coding and the Internet path performance fails to meet requirements. While current access capacity limits the use of J-QoS services, cellular bandwidth is expected to increase in future with 5G networks.

Duplicating traffic has negligible impact on power consumption. We tested the effect of duplicating a traffic stream on the battery life of the device. We ran 20 minute trials of Skype video calls, with and without cloud path duplication. We observed that in both cases the battery drain was ~20 mAh, highlighting that the extra overhead of CR-WAN has negligible impact on battery life.

Recovery can be feasible despite latency issues. Mobile networks also suffer from greater end-to-end latency and jitter [67]. We conducted a short study to quantify this effect by pinging three major cloud providers (Amazon, Microsoft, and Google) 1,000 times using different mobile networks: Verizon's LTE network (east coast) and T-Mobile's LTE network (both east and west coasts). The median ping times to each provider was typically in the range of 50-60 ms, but the 50%-90% RTTs to each cloud provider was in the range of approximately 50-100 ms.

These latencies could be problematic for mobile receivers, as the effect of greater latency is multiplied during recovery, especially during the coding service's recovery process. Despite this, our mobile Skype testbed was able to recover packets during an outage (Figure 6(a)) because the application is able to adapt to a greater end-to-end delay, as long as the delay is consistent. In addition, due to increased jitter, correcting random packet losses may be difficult for interactive applications, but can likely be mitigated for other applications (such as web transfers) using in-stream coding. Finally, with cellular latencies expected to go down with the rise of 5G networks, recovery delays will become smaller in future.

6 RELATED WORK

J-QoS connects to and benefits from a large body of prior work. We comment on key pieces from the literature that are most relevant to our study.

Overlay Networks and Internet Architectures. Our work is inspired by *overlay* networks that improve availability by using detour points, e.g., RON [16], OverQoS [62], one-hop source routing [37], Spines [15], etc. Our use of the cloud as an overlay creates unique opportunities and challenges. For example, we can do sub-RTT recovery, but to minimize cost, we have to send an additional small number of recovery packets. Recently, there have been proposals that make the case for using cloud as an overlay to improve QoS of interactive applications [18, 41, 47], TCP-based applications [21, 26], and provide Network-as-a-Service [49]. Schemes like VIA [47] improve performance by routing *all* of a user's traffic through the overlay path. VIA uses the performance history of a path between endpoints to decide between cloud relay and Internet. It also dynamically selects the best relay out of a set of top-k options before sending the data. J-QoS, on the other hand, uses the nearest data centers to route a user's traffic. While we take into account the cost of using cloud infrastructure and propose different use cases that use cloud storage and processing (e.g. coding across streams, caching), other schemes [18, 21, 26] *only* route traffic through cloud overlays.

Caching. Our caching service is similar in spirit to various Internet architecture proposals that do in-network caching, e.g., NDN [45], i3 [61], XIA [39]. A DC that stores packet for later delivery can be thought of as a rendezvous point as in i3 [61], or as a fallback host like XIA [39]. Traditionally, CDNs are also used to store and deliver content. Our scheme is analogous to a commercial live streaming CDN solution [51], which considers similar coding techniques as CR-WAN to increase the reliability of inter-CDN paths. In CR-WAN, we use nearby endpoints to recover the lost packets, whereas a CDN node already has all the required data for recovery. Furthermore, our scheme can also be deployed on CDNs provided the quality of inter-CDN paths is same as the inter-DC paths. RPT([38]) is also relevant as it uses caching for loss recovery – it uses on-path content aware routers to compress and decompress packets on every hop. Our use of caching is unique as we use it *only* for packet recovery: we use nearby off-path DCs to storage packet and receivers pull lost packets from the nearby cache. Further, we only store a flow's packet for very short times (1-2 RTTs) whereas typical caches (e.g., CDNs) store content for longer duration and serve multiple users.

Coding. Traditionally, network coding techniques have seen widest use in the context of wireless networks [35, 50]. J-QoS applies cross-stream coding on wide area Internet paths and uses it to recover lost packets. FEC based coding schemes have also been used in different contexts over the last several decades. The most relevant work to our scheme is Maelstrom [19], which uses an FEC-based technique to reduce packet loss on lambda networks. Maelstrom's layered interleaving provides additional protection against bursty losses, but at the expense of higher decoding delay, which limits its use for highly interactive applications. Also, unlike Maelstrom, the coded and data packets are sent on *different* paths, with very different properties.

Reliable and Low Latency Wide Area Communication. Finally, we share the goals of recent proposals that call for *low latency and high reliability* for wide area communication [22, 52, 56, 60]. Some recent proposals focus on improving performance of TCP short flows [28, 31] using different techniques (e.g. per packet timestamps, early retranmissions). We, however, use the nearest DC to recover lost packets within a fraction of a path's RTT.

Data Center Networking. Our work complements the large body of work on data center networking. This includes application of software defined networking (SDN) to such environments (e.g., SWAN [43], B4 [43]), techniques that meet specific workload needs (e.g., application deadlines [48, 69]), scheduling policies [30, 32], and the use of duplication [20, 44]). Similarly, studies on inter-DC measurements [24, 54, 69] have mainly focused on inter-DC *bandwidth*. J-QoS's use of inter-DC paths is complementary to these prior efforts.

7 FUTURE WORK

Cost Model and API. Our cost model excludes the endpoint to cloud bandwidth costs. To extend this model, we need to understand how different types of endpoints connect with the cloud and how bandwidth is billed on these interconnections. For example, enterprises use virtual private interconnections to connect with their nearest cloud locations and pay according to 95th percentile metering. We also foresee expanding the model to include various storage offerings by the cloud providers, including in-memory caching and bucket-based storage. In the future, we plan to extend our model to incorporate these use cases and also expand our API to provide bi-directional support to the applications [14].

Multi-cloud Overlays. In J-QoS, we form a cloud overlay by using a single cloud provider's inter data center network. In the future, we plan to extend the J-QoS framework to incorporate DCs of other providers in a muti-cloud like fashion. Multi-cloud strategies are increasingly becoming popular as they provide more flexibility in choosing data centers, reduce reliance on a single provider, and offer high reliability on inter-cloud paths [68].

8 CONCLUSION

J-QoS seeks to connect two complementary interests: the *pull* of existing (and burgeoning) applications and their demand for better user experience, and the *push* of DC technology that makes cloud services more accessible to the edge than ever before. The key idea behind J-QoS is to use the cloud paths in a judicious manner, in order to provide reliability services to applications with different latency requirements. We view J-QoS as a promising step toward providing application and network architects with new insights into how to judiciously leverage the cloud.

ACKNOWLEDGEMENTS

We thank our anonymous shepherd, the CoNEXT reviewers, and Tufts NAT lab members for their constructive feedback on this work. This work was supported by NSF CNS under award numbers 1815046 and 1815016.

REFERENCES

[1] 2002. ITU G.1010 : End-user multimedia QoS categories. https://www.itu.int/rec/T-REC-G.1010-200111-I.

[2] 2011. AWS News Blog - AWS Lowers its Pricing Again! https://amzn.to/2B93dtW.

[3] 2016. NetScaler SD-WAN | Packet Duplication. https://www.citrix.com/blogs/2016/09/22/netscaler-sd-wan-the-packet-duplicator/.

[4] 2018. Rethinking Network QoS in the Cloud Era. http://sites.tufts.edu/networkqos/.

[5] 2019. GCP Network Service Tiers. https://cloud.google.com/network-tiers/.

[6] 2019. RIPE Atlas Measurement Platform. https://atlas.ripe.net/.

[7] 2020. Amazon AWS. http://aws.amazon.com.

[8] 2020. AWS Direct Connect. https://aws.amazon.com/directconnect/.

[9] 2020. Google Cloud. https://cloud.google.com/.

[10] 2020. How much bandwidth does Skype need? https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need.

[11] 2020. Microsoft Azure. http://azure.microsoft.com/.

[12] 2020. Microsoft Azure ExpressRoute. https://azure.microsoft.com/en-us/services/expressroute/.

[13] 2020. netfilter/iptables project. https://www.netfilter.org/projects/iptables/index.html.

[14] Tooba Ahsen and Fahad Dogar. 2019. A case for a richer, bi-directional interface between augmented reality applications and the network. In *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion*. 59–60.

[15] Yair Amir and Claudiu Danilov. 2003. Reliable communication in overlay networks. In *Proc. IEEE DSN 2003*.

[16] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. 2001. Resilient Overlay Networks. In *Proc. ACM SOSP*.

[17] Todd W Arnold, Ege Gurmericliler, Arpit Gupta, Matt Calder, Georgia Essig, Vasileios Giotsas, and Ethan Katz-Bassett. 2020. (How Much) Does a Private WAN Improve Cloud Performance?. In *IEEE INFOCOM*.

[18] Amy Babay, Emily Wagner, Michael Dinitz, and Yair Amir. 2017. Timely, reliable, and cost-effective internet transport service using dissemination graphs. In *Proceedings of ICDCS*.

[19] Mahesh Balakrishnan, Tudor Marian, Ken Birman, Hakim Weatherspoon, and Einar Vollset. 2008. Maelstrom: Transparent Error Correction for Lambda Networks. In *Proc. USENIX NSDI*.

[20] Hafiz Mohsin Bashir, Abdullah Bin Faisal, M Asim Jamshed, Peter Vondras, Ali Musa Iftikhar, Ihsan Ayyub Qazi, and Fahad R Dogar. 2019. Reducing tail latency using duplication: a multi-layered approach. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 246–259.

[21] Aran Bergman, Israel Cidon, Isaac Keslassy, Noga H. Rotman, Michael Schapira, Alex Markuze, and Eyal Zohar. 2018. Pied Piper: Rethinking Internet Data Delivery. *CoRR* abs/1812.05582 (2018). arXiv:1812.05582 http://arxiv.org/abs/1812.05582

[22] Debopam Bhattacherjee, Sangeetha Abdu Jyothi, Ilker Nadi Bozkurt, Muhammad Tirmazi, Waqar Aqeel, Anthony Aguirre, Balakrishnan Chandrasekaran, P. Brighten Godfrey, Gregory P. Laughlin, Bruce M. Maggs, and Ankit Singla. 2018. cISP: A Speed-of-Light Internet Service Provider. *CoRR* abs/1809.10897 (September 2018).

[23] Debopam Bhattacherjee, Muhammad Tirmazi, and Ankit Singla. 2017. A Cloud-based Content Gathering Network. In *Proceedings of the 9th USENIX Conference on Hot Topics in Cloud Computing (HotCloud'17)*. USENIX Association, Berkeley, CA, USA.

[24] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. 2019. TEAVAR: Striking the Right Utilization-Availability Balance in WAN Traffic Engineering. In *ACM SIGCOMM*.

[25] Robert Braden, David Clark, and Scott Shenker. 1994. Integrated Services in the Internet Architecture: An Overview. RFC 1633.

[26] Chris X Cai, Franck Le, Xin Sun, Geoffrey G Xie, Hani Jamjoom, and Roy H Campbell. 2016. CRONets: Cloud-Routed Overlay Networks. In *Proc. ICDCS*.

[27] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. 2013. Mapping the expansion of Google's serving infrastructure. In *Proc. ACM IMC*.

[28] Yuchung Cheng, Neal Cardwell, and Nandita Dukkipati. 2017. *RACK: a time-based fast loss detection algorithm for TCP*. Internet-Draft draft-ietf-tcpm-rack-02. IETF Secretariat.

[29] Fahad R. Dogar. April 2018. Towards Slack-Aware Networking. *SIGCOMM Comput. Commun. Rev.* (April 2018).

[30] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. 2014. Decentralized Task-aware Scheduling for Data Center Networks. In *Proc. ACM SIGCOMM*.

[31] Nandita Dukkipati, Neal Cardwell, Yuchung Cheng, and Matt Mathis. 2013. *Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses*. Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-01. IETF Secretariat.

[32] Abdullah Bin Faisal, Hafiz Mohsin Bashir, Ihsan Ayyub Qazi, Zartash Uzmi, and Fahad R Dogar. 2018. Workload adaptive flow scheduling. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*. 241–253.

[33] Kevin Fall. 2003. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03* (Karlsruhe, Germany). ACM, New York, NY, USA, 27–34.

[34] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. 2013. Reducing web latency: the virtue of gentle aggression. In *Proc. ACM SIGCOMM*.

[35] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. 2006. Network Coding: An Instant Primer. *SIGCOMM Comput. Commun. Rev.* 36, 1 (Jan. 2006), 63–68.

[36] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proc. SIGCOMM*.

[37] P Krishna Gummadi, Harsha V Madhyastha, Steven D Gribble, Henry M Levy, David Wetherall, et al. 2004. Improving the Reliability of Internet Paths with One-hop Source Routing. In *Proc. USENIX OSDI*.

[38] Dongsu Han, Ashok Anand, Aditya Akella, and Srinivasan Seshan. 2012. RPT: Re-architecting Loss Protection for Content-Aware Networks. In *Proc. NSDI*. USENIX.

[39] Dongsu Han, Ashok Anand, Fahad R Dogar, Boyan Li, Hyeontaek Lim, Michel Machado, Arvind Mukundan, Wenfei Wu, Aditya Akella, David G Andersen, et al. 2012. XIA: Efficient Support for Evolvable Internetworking. In *Proc. USENIX NSDI*.

[40] P Hanhart and R Hahling. 2013. Video Quality Measurement Tool (VQMT).

[41] Osama Haq and Fahad R. Dogar. 2015. Leveraging the Power of the Cloud for Reliable Wide Area Communication. In *Proc. ACM Hotnets*.

[42] Osama Haq, Mamoon Raja, and Fahad R. Dogar. 2017. Measuring and Improving the Reliability of Wide-Area Cloud Paths. In *Proc. WWW*.

[43] 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 *Proc. SIGCOMM*.

[44] Ali Musa Iftikhar, Fahad Dogar, and Ihsan Ayyub Qazi. 2016. Towards a redundancy-aware network stack for data centers. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 57–63.

[45] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking named content. In *Proc. CoNEXT*.

[46] Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, and Ishai Menache. 2016. Dynamic Pricing and Traffic Engineering for Timely Inter-Datacenter Transfers. In *Proc. SIGCOMM*.

[47] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A. Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, and Hui Zhang. 2016. Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection. In *ACM SIGCOMM*.

[48] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. 2014. Calendaring for wide area networks. In *Proc. ACM SIGCOMM*.

[49] P. Kathiravelu, M. Chiesa, P. Marcos, M. Canini, and L. Veiga. 2018. Moving Bits with a Fleet of Shared Virtual Routers. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*.

[50] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Medard, and Jon Crowcroft. 2006. XORs in the Air: Practical Wireless Network Coding. In *Proc. SIGCOMM*.

[51] Leonidas Kontothanassis, Ramesh Sitaraman, Joel Wein, Duke Hong, Robert Kleinberg, Brian Mancuso, David Shaw, and Daniel Stodolsky. 2004. A transport layer for live streaming in a content delivery network. *Proc. IEEE* 92, 9 (2004), 1408–1419.

[52] Vasileios Kotronis, George Nomikos, Lefteris Manassakis, Dimitris Mavrommatis, and Xenofontas Dimitropoulos. 2017. Shortcuts Through Colocation Facilities. In *Proceedings of the IMC*.

[53] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. 2011. Inter-datacenter Bulk Transfers with Netstitcher. In *Proc. SIGCOMM*.

[54] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: comparing public cloud providers. In *Proc. ACM IMC*.

[55] K. Matsuzono, H. Asaeda, and T. Turletti. 2017. Low latency low loss streaming using in-network coding and caching. In *IEEE INFOCOM 2017*.

[56] Simon Peter, Umar Javed, Qiao Zhang, Doug Woos, Thomas Anderson, and Arvind Krishnamurthy. 2014. One tunnel is (often) enough. In *Proc. ACM SIGCOMM*.

[57] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. 2002. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proc. ACM HotNets*.

[58] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. 2012. Design and Implementation of a Consolidated Middlebox Architecture. In *Proc. NSDI*.

[59] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. Rollback-Recovery for Middleboxes. In *Proc. SIGCOMM*.

[60] Ankit Singla, Balakrishnan Chandrasekaran, P Godfrey, and Bruce Maggs. 2014. The Internet at the Speed of Light. In *Proc. ACM HotNets*.
[61] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. 2002. Internet indirection infrastructure. In *Proc. SIGCOMM*.
[62] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy H Katz. 2004. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS.. In *Proc. USENIX NSDI*.
[63] Jue Wang. 2010. *ChitChat: Making video chat robust to packet loss.* Ph.D. Dissertation. Massachusetts Institute of Technology.
[64] Harald Welte and Pablo Neira Ayuso. 2020. NetFilter. http://www.netfilter.org.
[65] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2003. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2003), 255–270.
[66] Zooko Wilcox-O'Hearn. 2008. Zfec 1.4. *Open source code distribution: http://pypi.python.org/pypi/zfec* (2008).
[67] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *USENIX NSDI*.
[68] Bahador Yeganeh, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2020. A First Comparative Characterization of Multi-cloud Connectivity in Today's Internet. In *International Conference on Passive and Active Network Measurement*. Springer, 193–210.
[69] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. 2015. Guaranteeing deadlines for inter-datacenter transfers. In *Proc. EuroSys*.
[70] Xinggong Zhang, Yang Xu, Hao Hu, Yong Liu, Zongming Guo, and Yao Wang. 2012. Profiling Skype video calls: Rate control and video quality. In *IEEE INFOCOM*.

Algorithm 1: Coding algorithm at DC1.

```
   def in_stream_qs[]
   def cross_stream_qs[][]
   dc1_process(pkt, flow_id):
      // (1) In-stream coding.
1     q = in_stream_qs[flow_id]
2     q.push(pkt)
3     if q.isFull() then
4         in_coded_pkts = encode(q)
5         send(dc2_id, in_coded_pkts)
      // (2) Cross-stream coding.
6     dc2_id = extract_dc2_id(flow_id)
7     q_index = next_round_robin_q(flow_id)
8     q = cross_stream_qs[dc2_id][q_index]
      // Find a queue that doesn't have a packet from this flow.
9     initial_q = q
10    while q.contains(flow_id) do
11        q_index = next_round_robin_q(flow_id)
12        q = cross_stream_qs[dc2_id][q_index]
          // If we've tried all q's, empty the first by encoding or
            discarding.
13        if q == initial_q then
14            if q.size() > 1 then
15                cross_coded_pkts = encode(q)
16                send(dc2_id, cross_coded_pkts)
17            else
18                q.clear()
19            break
20    q.push(pkt)
21    if q.isFull() then
22        cross_coded_pkts = encode(q)
23        send(dc2_id, cross_coded_pkts)
```

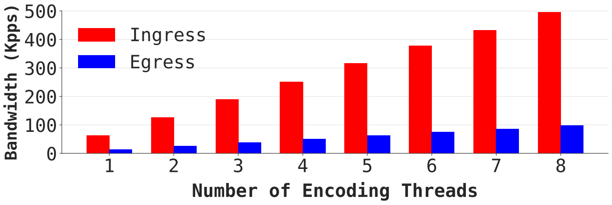


Figure 7: Throughput of the J-QoS prototype scales linearly with the number of encoding threads. Eight threads can handle up to 500Kpps.

A CODING ALGORITHM

DC1 follows Algorithm 1, which captures the task of encoding across multiple flows at once. DC1 maintains two sets of queues: one set for in-stream encoding (one set per flow), and a set for cross-stream encoding (one set per *k*). When a packet arrives, it is copied and pushed into one queue of each type.

Lines 1-5 check whether the relevant in-stream queue has reached a threshold, and if so, create coded packets and send them to DC2. For cross-stream coding, DC1 first selects the set of queues destined for the same DC2, and then chooses the individual queue in round-robin order (lines 6-8). DC1 avoids placing multiple packets from the same flow in the same cross-stream queue; if there already exist packets from the same flow in all queues, then DC1 processes the oldest queue. If there is only a packet from the flow in question, then the old packet is evicted and discarded, since sending cross-stream packets with only packets from a single stream reduces its effectiveness (lines 9-19). Once the packet is pushed into a cross-stream queue, if a threshold is reached, then coded packets are generated and sent to DC2 (lines 20-23).

Timing constraints pose a challenge to this algorithm. If one flow is much faster than all other flows, DC1 cannot hold back recovery data from the faster flow to wait to make full recovery packets. Therefore, we create a timer for each in-stream and cross-stream queue (not shown in Algorithm 1). On expiry of a queue timer, DC1 encodes all packets in the queue and sends them to DC2.

B PROTOTYPE SCALABILITY

We benchmark the performance of our CR-WAN prototype, which is the most computationally intensive service of J-QoS. We focus on its most computationally expensive part: the encoding algorithm performed at DC1. Our goal is to measure how efficiently CR-WAN can process and encode packets as the system scales to many concurrent streams. For each flow, we configure CR-WAN to generate a single coded packet per every five data packets. We use Dell Poweredge R430 servers on Emulab, and each server is equipped with two 2.4 GHz 8-core processors with two threads each, for a total of 32 hardware threads.

We first determine the maximum throughput achievable at DC1 in packets per second. Measuring packets (instead of bits) is the appropriate measurement granularity because the encoder operates over entire packets. We observe that a single J-QoS thread can handle up to 170 Kpps for forwarding and caching service. For context, assuming an average packet size of 512 bytes, 170 Kpps is enough for performing one-way processing for ~460 simultaneous HD Skype video calls [10].

We find that a single encoding thread can handle around 65 Kpps, which is equivalent to ~175 simultaneous HD Skype video calls. At this rate, the bottleneck is the generation of coded packets from data packets. In terms of cost, a single flow using coding service costs 2.6× more than the one using forwarding.

We then increase the number of DC1 encoding threads as we increase the number of senders, and load balance the streams to the different encoding threads. We rate limit each sender to 65 Kpps – the empirical maximum rate that can be processed by a single (sender, encoder) pair. Figure 7 shows that the processing power scales linearly with the number of encoding threads: up to ~500 Kpps with eight encoding threads. This shows that CR-WAN is amenable to parallelism and can be deployed in software to handle a large number of users.