Enhancing End-to-End Transport with Packet Trimming

Abdulazaz Albalawi*, Hamed Yousefi†, Cedric Westphal†, Kiran Makhijani†,
J.J. Garcia-Luna-Aceves*
*Department of Computer Science and Engineering, University of California, Santa Cruz, USA
†Futurewei Technologies, Santa Clara, USA
aalbalaw@ucsc.edu, hamed.yousefi@futurewei.com, cedric.westphal@futurewei.com, kiranm@futurewei.com,
jj@soe.ucsc.edu

***Abstract*—A new transport protocol is introduced to increase the responsiveness of the network to congestion. The new transport protocol, QUCO, reacts to congestion by selectively dropping off parts of a payload packet (combined with mitigation mechanisms to handle the loss of part of the payload). This packet trimming scheme greatly reduces the variations in the number of the packets going through the network. This allows to set tighter targets on the number of packets in flight and on the depth of the switch buffers. QUCO has less delay and much less delay variations than TCP. The resulting reduction in jitter is extremely useful, especially for media distribution.**

I. Introduction

The tolerance to latency and packet losses in volumetric media applications like AR/VR or holographic rendering is extremely low. The problem is amplified with larger packet sizes because a greater amount of data is lost. Such scenarios make a compelling case for new means to mitigate packet-drop on bottleneck nodes. One such scheme, qualitative communication, has been proposed by Li et al. [1] (Sec. II). It exploits the fact that multi-media data frames are made up of parts with varying degree of relevance.

This paper proposes QUCO, a set of QUalitative COmmunications transport functions. Our proposal consists of a set of transport control functions for qualitative communications (Sec. III) in which both losses and re-transmissions are mitigated by discarding only some selected parts of the packet. These functions may be integrated with any transport protocol. Our design requires applications to compose packets payloads into several data chunks. For example, the chunks can be different layers of a scalable video stream, so that the base layer is prioritized and the enhancement layers may be omitted; the chunks can also be part of a workload generated using a fountain code, so that the receiver only needs to receive a sufficient number of chunks to decode. The chunks can be different level of information with different priority levels, such that it is preferable to receive the highest-priority chunks in time, rather than wait for the whole packet to be re-transmitted.

One key aspect of this mode of transmission is that the rate of processing packets at the router increases upon a congestion event, which is a desirable property. For instance, consider a canonical example of a packet with two chunks. If the buffer at the router exceeds a threshold (say, half utilization), then the router drops one of the two chunks for the incoming packets. Assuming negligible overhead, this means that the packet processing rate goes from one to two packets per unit of time. Any build up in the queue therefore disappears twice as fast as with a typical transport protocol (Sec. IV). This also implies that we can use smaller buffers in the network and larger packets. As we move to very high bandwidths, there is a trend toward larger (jumbo) packets to provide data transmission at line rates. Whether a packet is small or large, its header needs to be processed, hence larger packets offer certain efficiency gains. However, as a result, each discarded packet results in a larger quantum of data that needs to be re-transmitted than in the past. Our scheme allows for an improved network utilization with large packet sizes.

We make three main contributions: (a) a theoretical analysis of the queue model using QUCO; (b) QUCO-AQM, an AQM algorithm using qualitative packets; (c) an extensive simulation evaluation of the QUCO flows from an end host's view (Sec. V).

Our analysis shows that QUCO achieves lower delays and much lower delay variations than TCP, with full link utilization. In particular, QUCO transmissions have a more predictable buffer utilization, and hence more predictable end-to-end delays. This is important for real-time (multimedia) applications, such as AR/VR [2]–[4] or holographic communications.

We do not provide detailed analysis of different deployments or traffic profiles due to space limitation. Rather, we motivate the use of new techniques as a way to reduce buffer sizes and therefore delays in the network. We view this as a promising avenue for future research (Sec. VI).

II. Related Work

Reducing Internet delays has been a major goal [5] for congestion control protocol [6].

LEDBAT [7] manages congestion by observing one-way congestion delays. It allows background type flows to yield bandwidth to the foreground flows thus achieving high link utilization and predictable flow completion times. Alizadeh et al. [8] proposed DC-TCP in the context of data centers. DC-TCP leverages Explicit Congestion Notification (ECN) [9], a mechanism used to address congestion before it becomes too

severe. In contrast to LEDBAT, QUCO aims to achieve predictable delays on per packet basis, instead of flow completion times. QUCO can be viewed as a form of early congestion notification similar to DC-TCP. However, instead of flipping a bit in a packet header, it trims chunks off the packet.

Trimming methods have been proposed for data centers before, such as "cut payload" (CP) [10] and NDP [11]. The goal in these prior schemes is to achieve fast re-transmissions for data center workloads. However, they trim the whole payload and only keep the header. This approach is too drastic at Internet scale. Buffer sizing [12]–[14] and reducing the buffer size [15] has been studied previously as well.

These prior works established that the buffer depth should be $C \times RTT/\sqrt{N}$, where C is the link capacity, and N is the number of concurrent flows. Large buffers, and buffer-bloat in particular, have introduced unwanted latency in the network [16]. We contend that QUCO can reduce buffer depth to reduce latency due to congestion. Abdullayev et al. [17] proposed a mechanism that groups consecutive MPEG-2 TS (transport stream) packets with the same priority into a single IP packet. When network conditions deteriorate, large IP packets are fragmented into smaller packets, which results in increasing the probability of reception. QUCO generalizes this approach to other applications.

The trade-off QUCO makes is between losing some information and achieving predictable delays with shallow buffers while minimizing re-transmissions. Our work is inspired from packet-wash by Li et al. [1], which uses the concept of selectively dropping chunks in a packet based upon some congestion criteria. However, packet-wash does not specify congestion management aspects.

III. Design of the QUCO Algorithm

The tolerance to packet losses in volumetric multi-media applications is extremely low. The problem is amplified with larger packet sizes since a greater amount of data is lost. Such data-intensive applications can afford to sacrifice a part of the transmitted data for a more predictable end-to-end data rates.

A. Objective: In order to preserve smooth data delivery, QUCO design needs to support nearly fixed-rate transmission. According to Mathis' equation $T_{max} = (MSS/RTT) * (1/\sqrt{p})$, where throughput T_{max} will be steady when the packet loss probability p is low and the maximum segment size (MSS) and round-trip times (RTT) are constant. Thus, our design goal is to maintain nearly constant T_{max} by achieving quasi-steady state in the bottleneck network node without requiring re-transmission and without causing bufferbloat latency.

We leverage packet trimming for this purpose (Sec. III-C). Of course, dropping chunks off the payload of a packet is not the end-all solution to congestion. It is an alternate outcome of the quasi-steady state decision that would have otherwise degraded T_{max}. An application formats data into chunks. Then the network is able to selectively remove the least-significant part(s) of the chunks (Sec. III-B).

Secondly, we still need to adjust the sending rate through feedback. As in TCP, we use a congestion window. The receiver sends a chunk negative acknowledgment (NACK) to indicate congestion to the sender (Sec. III-D).

QUCO achieves improved burst tolerance, high throughput, and low latency, with support for commodity shallow-buffered forwarding nodes. To this end, the network and end-points cooperate to react in proportion to the extent of congestion. By setting up aggressive congestion thresholds in the forwarding nodes and mild reaction at the end-points, QUCO avoids intermittent packet losses (and retransmissions).

Each QUCO operation is performed cooperatively. It requires, a) a packetization function on the source side, b) a trimming-aware queuing discipline in the network, and c) QUCO−NACK congestion notification on the receiver as shown in Fig. 1. These operations seamlessly plug into any reliable transport protocol machinery.

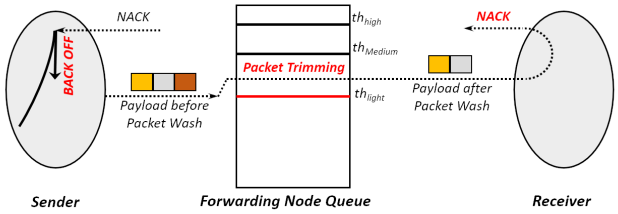


Fig. 1: QUCO: a 3-chunk example. The sender, network, and receiver react based on the level of congestion.

B. Packetization: Packetization takes place on the source node in the application layer. A qualitative packet is composed of different pieces of information (referred to as chunks) with different significance. The application layer creates a stream of chunks with different priorities. There are two cases to consider.

In the first case, the application can tolerate to lose chunks of lower priority (for instance, some speech or video encoded bits may be less important than other). This translates naturally into creating the chunks that compose the packet payload.

In the second case, all bits are equal. If chunks are dropped, a recovery mechanism is necessary in this case. Two such mechanisms are possible. One mechanism consists of acknowledging the received chunks, instead of the received packets, and re-transmitting the missing chunks. This introduces a re-transmission delay, which is only one RTT, because the receiver gets the rest of the packet and therefore knows what to re-transmit immediately, without waiting for a time-out to expire. Another mechanism entails creating the chunks at the source using network coding or a fountain code, so that any combination of, say, k chunks at the receiver is good to decode the payload.

The chunks may be encrypted separately to preserve the integrity of the packet. The authors in [1] proposed a header format to inject priority/significance factors, checksum and boundaries of chunks.

C. Packet Trimming: Trimming operation will be performed in the forwarding (bottleneck) node. The forwarding nodes treat qualitative packets differently under congestion, i.e. they may selectively drop chunks in a packet in response to the network congestion. The forwarding nodes do not access a chunk's content, but only drop the chunk when necessary.

The forwarding nodes need to describe their own AQM thresholds beyond which the chunks cannot be further dropped as the packet would become useless. Other solutions, where the information required for a forwarding node on how to trim a packet can be programmed for a flow out of band or be decided by the individual forwarding node, are also possible and beyond the scope of this paper.

Under constrained network conditions, forwarding nodes immediately react by (1) trimming the packet payload based on the current buffer occupancy, and (2) modifying the metadata of the qualitative packet header to ensure that receiver knows about the number of dropped chunks, so it can notify the source. The source can react accordingly.

D. Congestion Control: A stream of qualitative packets adapts itself to some extent to congestion. The source node still need to be notified about the chunk drops to react accordingly. For this purpose, ECN bit may be used but it does not provide the extent of congestion. In our design, the receiver is responsible for providing feedback through chunk NACKs, source nodes then manage their CWNDs less aggressively based on number of chunks NACKed.

1) QUCO-AQM Algorithm: We employ a simple active queue management scheme that is aware of qualitative packets. Without loss of generality, we consider three thresholds associated with different level of buffer occupancy (and thus potential congestion) in each buffer: $th_{light}, th_{medium}, th_{high}$. One may consider three priority levels in each qualitative packet: Gold, Silver, and Bronze. The forwarding node drops the Bronze chunk if buffer occupancy exceeds, say, 50% (th_{light}), both Silver and Bronze chunks if exceeding, say, 75% th_{medium}, and the entire payload -but not the header- if exceeding, say, 90%. Clearly, when the buffer occupancy is less than the minimum threshold, no action is required. Fig. 1 shows how the router drops one chunk of the incoming packet's payload as the buffer occupancy exceeds the light congestion threshold. Algorithm 1 gives a pseudo-code description of how each forwarding node reacts the congestion in proportion to the extent of congestion.

Algorithm 1: QUCO in a router — a 3-chunk example

```
Initialize th_light, th_medium, and th_high
for each arriving data packet P do
    if buffer utilization ≥ th_light then
        if buffer utilization < th_medium then
            PW(P, 1 chunk)
        else if buffer utilization < th_high then
            PW(P, 2 chunks)
        else
            PW(P, 3 chunks)    //only keep header
    Enqueue P
```

QUCO fairly spreads the impact of congestion over a larger number of flows. Thus, instead of dropping a packet and accepting the next one, both packets may see their payload cut in half. This is fair and has the added benefit of notifying more flows of the congestion occurring inside the network.

It is also important to preserve fairness among different flows at each forwarding element which makes a decision to drop chunks from the qualitative packets. One mechanism to achieve this is to increase the QoS level of the packet after QUCO trimming operation, by increasing the ToS to the next higher value.

2) Receiver: Like TCP, QUCO uses both positive and negative acknowledgements. Unlike TCP, NACK as in Algorithm 2 is a request for easing the network load and reducing congestion through rate adaptation at the senders (see Fig. 1), rather than a request for re-transmissions. Some applications may tolerate receiving degraded packets; however, the receiver may ask for the re-transmission of a dropped chunk if it so wishes, both are supported. The partial dropping of a packet is a warning that congestion is occurring. The receiver triggers an adaptive congestion control by notifying the sender about the network conditions. Receiving a more degraded packet (i.e., a packet with more dropped chunks) at the receiver means higher congestion in the network.

Algorithm 2: QUCO at the receiver, upon lost chunk

```
for each packet arrival do
    if chunk(s) have been dropped then
        Send back a NACK/ECN for the degraded quality, with a reference
          to the dropped chunk;
    else
        Send back an ACK;
```

3) Sender: In normal network conditions, QUCO senders behave as TCP senders. They maintain congestion windows, limiting the total number of unacknowledged chunks that may be in transit end-to-end. The senders initiate in slow start, where the windows grow quite aggressively. The congestion window size will be increased by one with each ACK received, effectively doubling the window size each RTT. In congestion avoidance state ($CWND > ssthresh$): (1) the increase in window follows TCP (i.e., linearly at the rate of $\frac{1}{CWND}$ on each new ACK, means 1 every RTT), and (2) the window size decreases (by a constant factor β) in proportion to the level of buffer occupancy. This allows the sender to decrease its sending rate as a response to potential congestion in the network unlike loss-based congestion control protocols. The approach is shown in Algorithm 3. For example, the sender can back off by β=10% (slight reduction as shown in Fig. 1), 25%, and 50% (like TCP), in light, medium, and high congestion scenarios, respectively, in congestion avoidance state.

This is how QUCO maintains low queue length, while still ensuring high throughput. Unlike a TCP sender which can only detect the existence of congestion, a QUCO sender can react to the actual extent of the encountered congestion, due to feedback in QUCO NACKs. Re-transmissions may still be required; whether or not a packet is re-transmitted is up to the application, which is beyond the scope of this paper.

Algorithm 3: QUCO at the sender — a 3-chunk example

```
Initialize β_light, β_medium, and β_high;
CWND ← 1
ssthresh ← ∞
for each ACK do
    if CWND ¡ ssthresh then
        CWND ← CWND + 1     //Slow Start
    else
        CWND ← CWND + (1 / CWND)
for each NACK do
    if 1-chunk drop then
        ssthresh← max(β_light× CWND, 1)
    else if 2-chunk drop then
        ssthresh← max(β_medium× CWND, 1)
    else if 3-chunk drop then
        ssthresh← max(β_high× CWND, 1)
    CWND ← ssthresh
```

QUCO adjusts its window size as TCP if the packet is dropped as a whole. Therefore, if QUCO is not supported, it defaults back to TCP. If QUCO and TCP co-exist in a network, it is necessary to put them in different queues as they react differently to congestion and the concept of packet trimming is unapplicable to TCP packets.

E. Discussion: As discussed above, QUCO trades off between losing some information and achieving predictable delays with shallow buffers while minimizing re-transmission.

QUCO adds some overhead, namely each packet is composed of chunks with some associated meta-data. However, for a relatively limited number of chunks within a packet, this is not a significant addition, especially for larger payload sizes or larger frames. There is a computational overhead as well, but QUCO is involved only upon congestion. Namely the cost of trimming the packets always serves a purpose.

QUCO can be associated with some differentiated levels of QoS. Actually, Li [1] suggests increasing the QoS level of a packet after a trimming operation. This is to ensure that a packet that has been trimmed is less likely to be dropped later on. In general, QUCO packets could be given a better than Best Effort level of QoS due to the willingness of the sender to gracefully reduce the congestion over the network.

IV. QUCO-based Queuing Model

We motivate the benefit of dropping only part of the packets when the network gets congested. In essence, QUCO increases the per-packet processing rate at the cost of losing some information in the packet. Current networks are stretched to high utilization during peak hours. We would like to consider a utilization where the incoming rate is higher than the capacity.

For illustrative purposes, we consider a packet composed of two chunks. When QUCO is applied to the packet and one chunk is dropped at first, then the processing rate for a packet at the router is doubled. Therefore, twice as many packets can be forwarded on the congested link (of course, with a loss of half the data in the original packet). While we only present analytical models for two-chunk packets, the models can be generalized to the case of multiple chunks per packet.

Fluid Model Approximation: Consider a fluid system in which requests arrive as a process of rate λ packets per second and packets are processed at the server with a rate μ packets per second. Each packet is composed of two chunks of equal size. If $\lambda > \mu$ then the queue builds up in the server's buffer. When the system reaches a threshold T (say, when the buffer is half full), it drops one chunk of all new incoming packet. The *per-packet* processing rate increases to 2μ.

This system will stay empty if $\lambda < \mu$. The system will overflow if if $\lambda > 2\mu$, given that 2μ is the maximum processing rate. The system will converge to the threshold T and keep the buffer half full if $\mu \leq \lambda \leq 2\mu$. Consequently, if the incoming rate is between μ and 2μ, the system operates with a buffer of constant size B, where B is the buffer utilization threshold, and the wait time for a task is simply B/C, where $C = 1/\mu$ is the link capacity. The three modes of operation of such a system are therefore to have: (a) an empty buffer ($\lambda < \mu$), (b) a buffer at the threshold T ($\mu \leq \lambda \leq 2\mu$), or (c) an overflowing buffer.

The waiting time for a stable system is either 0 or B/C. Because the system has constant buffer size, the jitter is 0. Further, the outgoing packet rate is equal to the link capacity after filling up the buffer. This implies that the fraction α of packets that are shrunk is equal to:

$$(1-\alpha)\lambda + \alpha/2\lambda = \mu \qquad \text{or} \qquad \alpha = 2(1-\mu/\lambda) \tag{1}$$

Note that a similar fluid TCP model would have the same buffer occupancy of 0 for $\lambda \leq \mu$. For $\lambda > \mu$, the delay would increase until the buffer is filled up (at rate μ/λ) then drop off to half and repeat the process again. The waiting time for a packet $\lambda > \mu$ will be $3/4B'$ where B' is the buffer size and the delay will be spread uniformly over ($B'/2$, $B0$). Setting B to be less than $3/4B'$ ensures that the packet trimming approach has a lower delay and a much lower jitter than the TCP fluid model.

M/D/1 Model Approximation: Instead of a fluid model, we consider that packet arrivals occur according to a Poisson process with rate λ. For simplicity, packet lengths are assumed to be identical (prior to any trimming operation), which results in processing times that are deterministic. Packets are served immediately if they arrive in an empty system, or put in a queue if the server is busy processing a packet.

We assume the following policy: if the queue holds fewer than or equal to B tasks, then the packet is processed with rate μ (and mean service time $1/\mu$). If the queue holds more than B tasks, then the service rate is 2μ. The arrival for any state is simply for any state, while the service rate is μ for states 0 to B, and 2μ for states $B+1$ to k. Define by p_k the probability that the buffer holds k packets and let $\rho = \lambda/\mu$.

Define by π_k^μ to be the stationary distribution of the M/D/1 queue with arrival rate λ and service rate μ. This can be recursively computed, cf for instance [18]. A coupling argument shows that the system is stable for $\lambda < 2\mu$ and that for $\lambda > \mu$, once the buffer occupancy goes beyond B, the system behaves exactly as the M/D/1 queue with rate 2μ. This means that we can compute p_k for $k \geq B$ as $\pi_{(k-B)}^{2\mu} P(L > B)$ where L

is the buffer occupancy. An upper bound on the mean buffer occupancy is therefore:

$$L = B + \frac{1}{2}\left(\frac{\rho^2/4}{(1-\rho/2)}\right) \tag{2}$$

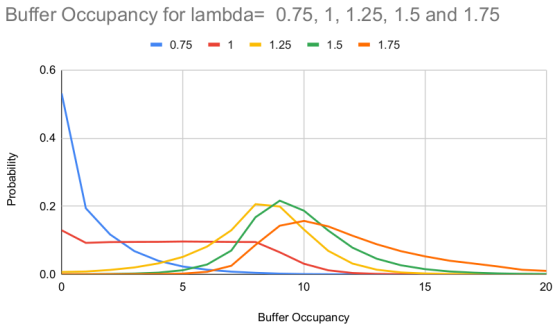


Fig. 2: Buffer occupancy probability for a QUCO system with 2 chunks per packet and buffer depth of 20 packets, threshold at 50% occupancy

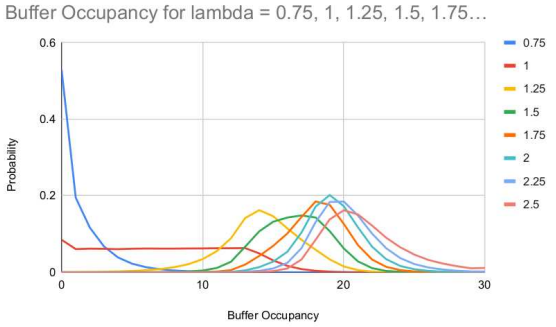


Fig. 3: Buffer occupancy probability for a QUCO system with 3 chunks per packet and buffer depth of 30 packets, thresholds at 50% and 66.7% occupancy

Because the system is stable, the computation of α from Eq. 1 still applies for $\mu < \lambda < 2\mu$. Figs. (2) and (3) show the probability distribution of the buffer occupancy, with a clear peak at the target threshold value. We can observe a fast drop-off in the probability of a buffer occupancy over the highest threshold.

This indicates that QUCO limits the jitter and is therefore suited for real-time applications and real-time media in particular.

V. Evaluation

We evaluate the performance of QUCO using the NS-3 simulator. We consider a single-flow scenario to show how forwarding nodes and end-points (sender and receiver) perform; and a multiple-flow scenario to show QUCO's fairness. We also vary the buffer depth to verify that QUCO offers a lower and more predictable delay with smaller buffers.

The preliminary results are compared to TCP New RENO (TCP for short). We compare to New RENO because it is an up-to-date version of TCP (70% of Internet traffic) and is even used in QUIC (15%). Therefore it is a valid benchmark. ECN-based protocols, on the other hand, are known to be difficult to tune to make a meaningful comparison [19], and are not widely deployed [20]. We think that QUCO can also outperform ECN-based protocols as they do not react to congestion immediately inside the network, but only after one RTT.

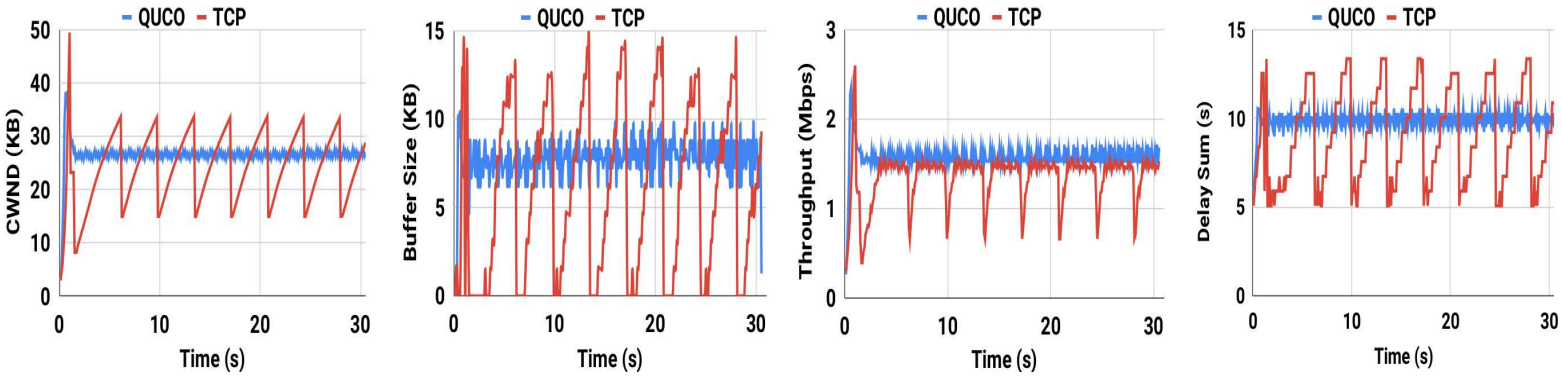
A. Single-Flow Scenario Our first experiment shows the performance over a simple network consisting of a single path of four nodes with a single source at one end, sending data to a single receiver at the other end. The source and sink are connected via a 1.5Mbps bottleneck link. The buffer size of the routers at the bottleneck link is set to be equal to the bandwidth delay product (BDP), namely 15KB. The file size to be transmitted is set to 6MB. Eack packet is 1500B, and can carry three chunks each of size 500 bytes with three levels of priority for QUCO. For TCP, the router's queue is set as a tail-drop. QUCO uses a simple active queue management scheme where the number of chunks to be dropped at the queue depends on three thresholds (light, medium, and high congestion).

Figs. 4a and 4b show both the sender's CWND and bottleneck's buffer occupancy for TCP and QUCO.For TCP, packet drop is the only way to detect congestion. Therefore, the sender increases its sending rate until the buffer overflows. This will either result in a timeout or a fast re-transmit event. Then the sender halves its congestion window. This process drains the queue empty for a while until it gets filled up again, and the same cycle repeats. This affects the TCP throughput and end-to-end delay for packets as shown in 4c and 4d, respectively.

However, QUCO senders can detect not only the existence of congestion in the network but also the level of congestion. Once the queue exceeds the light congestion threshold, QUCO starts dropping low priority chunks. This will cause the receiver to send NACK of the missing chunk to the sender (per Algorithm 2)), allowing the sender to reduce its congestion window proportionally to the congestion reported in the NACK. Since there is only one flow, the level of congestion in the network never exceeds the low congestion threshold, (50% in Fig. 4b). As a result, the queue never empties in Fig. 4a. Thus, QUCO maintains the packet sending rate while also maintaining high throughput (Fig. 4c) and low end-to-end delay (Fig. 4d). The figures also confirm a much lower jitter and predictable delay for QUCO.

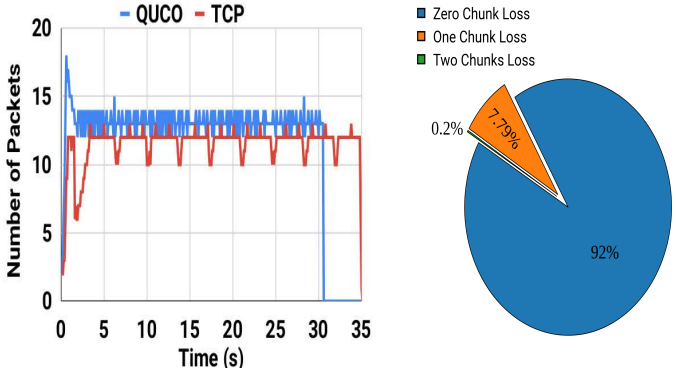
Fig. 5a shows the performance from the receiver's point of view. The improved performance comes at the cost of 1-chunk loss for only 7% of the packets (and 2-chunk loss for less than 0.2% as shown in Fig. 5b), which is tolerable in many applications. This clearly demonstrates the main strength of QUCO for applications that need to receiving part of the data in time, while maintaining high throughput and low end-to-end delay.

B. Multiple-Flow Scenario We evaluate QUCO's fairness under a multiple-flow scenario. We use a dumbbell topology with two senders and two receivers connected via a link with



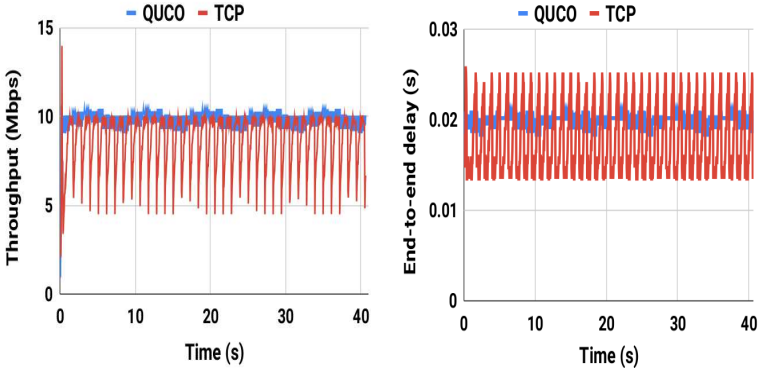
(a) Sender's CWND Size **(b)** Router's Buffer Occupancy **(c)** Throughput **(d)** End-to-End Delay Sum

Fig. 4: Single-flow scenario



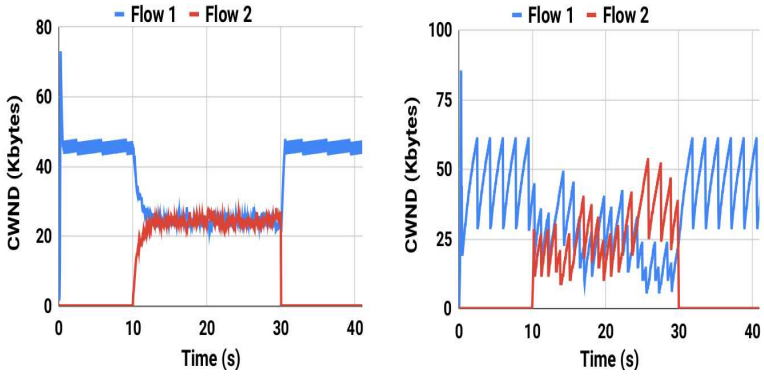
(a) Packets received vs time **(b)** Chunk losses

Fig. 5: Goodput at the Receiver



(a) Throughput **(b)** Delay

Fig. 7: Shallow buffer



(a) QUCO CWND size **(b)** TCP CWND size

Fig. 6: Multiple-flow scenario

10Mbps capacity. The queue size is set to the BDP, which is 30KB. Sender 1 sends data first. Once it fully utilizes the link, sender 2 starts sending data.

QUCO trims packets based on the congestion. This will cause sender 1 to back off, allowing sender 2 to increase its congestion window as shown in Fig. 6a. By adopting a less aggressive window reduction with QUCO in-network operations, the QUCO senders reach fairness state faster than TCP.

TCP flows reach fairness when they become synchronized [21]. However, synchronization for TCP can take a while. This is shown in Fig. 6b as both TCP flows struggle to synchronize even though they have the same RTT. This affects the application's performance.

C. Shallow-Buffer Scenario The goal of this evaluation is to highlight QUCO ability to achieve high throughput and low latency compared to TCP when the switches have shallow buffers. With the use of in-network trimming operations at intermediate switches and with the senders reacting in proportion to the level of congestion, QUCO is able to operate with shallow buffers without loss of throughput as shown in Fig. 7a. QUCO also maintains a low delay without any under-utilization of the buffer compared to TCP as shown in Fig. 7b. This is because TCP reacts to congestion in general instead of the level of congestion as in QUCO. By dropping the window in half, TCP causes under-utilization of the buffer, leading to a loss in throughput.

VI. Conclusions and Next Steps

We argue that dropping chunks off a packet is a potentially beneficial approach to address network congestion, as long

as either the dropped chunks are dispensable or a mitigation mechanism (re-transmission, network coding) is in place. Dropping chunks avoids dropping entire packets and increases the per packet processing rate during congestion events. Because the end points still receive some information, they can react quicker to congestion events. This of course requires the packets to be modified to support shedding chunks, either by having different levels of information within a packet(say, important, helpful, and "nice to have" chunks within a packet); or by having the chunks encoded using some fountain code to allow transmission of a stream of chunks until enough are received at the end point to recover the original data.

The introduction of "qualitative packet" is promising, and we described QUCO as a protocol taking advantage of this idea. Our simulation results show a greater delay predictability and much lower delay variations with QUCO, even in the case of smaller buffers.

REFERENCES

[1] R. Li et al, "A framework for qualitative communications using big packet protocol," *ACM SIGCOMM NEAT*, pp. 22–28, 2019.

[2] C. Westphal, "Challenges in Networking to Support Augmented Reality and Virtual Reality," in *IEEE ICNC*, Jan. 2017.

[3] D. He, C. Westphal, and J. Garcia-Luna-Aceves, "Network Support for AR/VR and Immersive Video Application: A Survey," in *ICETE SIGMAP*, July 2018.

[4] D. He, C. Westphal, and J. Garcia-Luna-Aceves, "Joint rate and FoV adaptation in immersive video streaming," in *Proceedings of the ACM SIGCOMM Workshop on Virtual Reality and Augmented Reality Network*, pp. 27–32, 2018.

[5] B. Briscoe at al, "Reducing internet latency: A survey of techniques and their merits," *IEEE Communications Surveys Tutorials*, 2016.

[6] M. Polese et al, "A survey on recent advances in transport layer protocols," *CoRR*, vol. abs/1810.03884, 2018.

[7] S. Shalunov et al, "Low extra delay background transport (LEDBAT)," *RFC*, vol. 6817, pp. 1–25, 2012.

[8] M. Alizadeh et al, "Data center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[9] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP." IETF RFC3168, 2001.

[10] P. Cheng et al, "Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Centers," in *Usenix NSDI*, 2014.

[11] M. Handley et al, "Re-architecting Datacenter Networks and Stacks for Low Latency and High Performance," in *ACM SIGCOMM*, 2017.

[12] G. Appenzeller et al, "Sizing router buffers," in *ACM SIGCOMM*, 2004.

[13] A. Vishwanath et al, "Perspectives on router buffer sizing: Recent results and open problems," *ACM CCR*, vol. 39, 2009.

[14] A. Dhamdhere and C. Dovrolis, "Open issues in router buffer sizing," *SIGCOMM Comput. Commun. Rev.*, vol. 36, Jan. 2006.

[15] M. Enachescu et al, "Routers with very small buffers," in *IEEE INFOCOM'06*, 2006.

[16] N. Cardwell et al, "BBR: Congestion-based congestion control," *ACM Queue*, 2016.

[17] J. Abdullayev et al, "A Dynamic Packet Fragmentation Extension to High Throughput WLANs for Real-Time H264/AVC Video Streaming," in *CFI'15*.

[18] L. Kleinrock, *Theory, Volume 1, Queueing Systems*. New York, NY, USA: Wiley-Interscience, 1975.

[19] M. Kwon and S. Fahmy, "PTCP increase/decrease behavior with explicit congestion notification (ECN)," in *IEEE ICC*, April 2002.

[20] A. Mandalari et al, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile," *IEEE Communications Magazine*, 2018.

[21] E. Altman et al, "Fairness analysis of TCP/IP," in *IEEE Conf. on Decision and Control*, Dec 2000.