XRC: An Explicit Rate Control for Future Cellular Networks

XRC: An Explicit Rate Control for Future Cellular Networks

Morteza Kheirkhah*
InterDigital
morteza.kheirkhah@interdigital.com

Mohamed M. Kassem
University of Surrey
m.kassem@surrey.ac.uk

Gorry Fairhurst
University of Aberdeen
gorry@erg.abdn.ac.uk

Mahesh K. Marina
University of Edinburgh
mahesh@ed.ac.uk

***Abstract*— We propose XRC, an explicit rate control algorithm that overcomes the poor performance of commonly used TCP variants in cellular networks. XRC exploits explicit feedback from the radio access network that is aware of the physical, network and transport layer information of all UEs as well as resource distribution policies for users with different traffic characteristics. XRC co-exists fairly with other XRC and non-XRC flows at the wireless and non-wireless bottlenecks while it strictly controls queuing delay within a small threshold. We implement XRC in NS-3 and examine its performance across a range of network loads and dynamics. When competing with CUBIC at a wireless bottleneck, XRC achieves a Jain's fairness index of 99.7% while providing a 3x lower median queuing delay compared to when CUBIC competes with CUBIC in the same setup.**

I. Introduction

Most of today's cellular end-to-end data flows are originated from Transmission Control Protocol (TCP) servers [1]. TCP is desirable for its ability to offer reliable packet delivery and rate control, essential for error-prone cellular links. However, it is known that TCP and its end-to-end variants [2], [3], [4], [5], [6], [7] suffer from notoriously poor performance over cellular networks due to the rapid fluctuation of wireless channel conditions (i.e., milliseconds [8], [9]). The extent of the fluctuation may even be worse with emerging high capacity wireless technologies (*e.g.,* mmWave), which exhibit a volatile signal propagation behavior with high sensitivity to foliage and blockage. Several contributing factors to this fluctuation include user mobility, signal propagation characteristics (slow-fading and path-loss), and interference. When the capacity is suddenly dropped, TCP connections experience severe packet losses that typically lead to exponentially increasing timeouts. Similarly, TCP cannot adapt to utilize sudden increases in capacity quickly, resulting in a waste of radio resources.

On the other hand, as we emerge into 5G, applications/services are expected to vary significantly, from stringent ultra-low latency (*e.g.,* for autonomous cars) to a very high rate (*e.g.,* for video streaming). Some applications even require high rates and low latency simultaneously (*e.g.,* VR/AR). These diverse traffic flows are expected to share the cellular last hop bottleneck. The simultaneous co-existence of these heterogeneous traffic flows in the cellular last hop bottleneck further magnifies the limitations of existing TCP congestion control algorithms, given that they fail to co-exist in the same bottleneck fairly. TCP fairness is a long-lasting non-trivial problem and despite the recent efforts [10], [11], achieving fairness between heterogeneous – in terms of traffic requirements, and used congestion control – flows is still unsolvable.

To address the above challenges, several cellular-specific end-to-end congestion control algorithms have been proposed [8], [9], [12], [13], [11], [10]. Although these solutions outperform standard TCP, they are sub-optimal in being unable to accurately and quickly track capacity changes. This issue is fundamentally challenging to tackle for a highly dynamic environment, especially when a congestion control algorithm only acts upon end-to-end measurements, which are often vulnerable to network jitter, ACK delay and noise. Additionally, end-to-end measurements also require several RTTs before providing stable/meaningful statistics to the sender to adapt the sending rate. On top of that, these schemes have failed to achieve fairness in co-existence between their flows running different applications with distinct requirements.

In this paper, we propose using an explicit feedback approach to update the TCP sender on wireless physical layer capacity changes, thereby enabling accurate adjustment of the congestion window (i.e., at the sender) to track the available capacity. We argue that a base-station-centric explicit feedback approach is more accurate and practical for future cellular networks compared to an endpoint-centric approach (such as PBE-CC [14]) for several reasons. First, users' available wireless capacity changes over a time scale known as the wireless coherence time, which is as small as milliseconds especially, in mobility and high-frequency technologies (*e.g.,* mmWave). Such abrupt changes of the wireless capacity cannot be tracked sufficiently fast at the last hop mobile endpoint. Second, the mobile endpoint is unaware of the radio resources assigned to other users in the same cell, and accordingly, is unaware of the capacity they use. This limited view hinders the congestion control algorithm from achieving a fair share of available resources among the multiple users. Other approaches that rely on eavesdropping on all mobile users' downlink control information (DCI) messages are not practical in 5G networks where these control messages will be encrypted. Third, the physical location of the bottleneck is not at a mobile user, but rather at the base station. Therefore, critical bottleneck statistics (*e.g.,* RLC buffer occupancy) cannot be available at the mobile endpoint. These statistics are crucial for efficient queue management and minimizing the end-to-end latency to meet the stringent requirements of the latency-sensitive traffic.

ABC [15] follows a base-station-centric approach. A base station marks each packet with either "accelerate" or "break"

*Part of this article was written by Dr. Morteza Kheirkhah while he was associated with University College London and not InterDigital's employee.

command using Explicit Congestion Notification (ECN), which results in the increase or decrease of the sender's congestion window, respectively. However, due to the nature of the ECN signal, ABC cannot co-exist gracefully with other TCP variants at the same queue. Thus, its traffic should be segregated into a separate queue from other traffic.

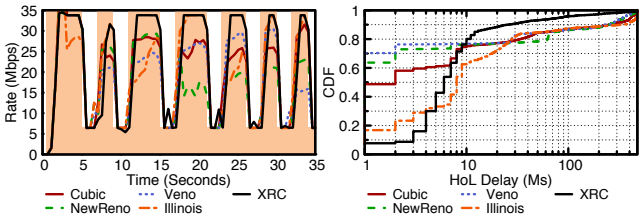
We introduce XRC, an adaptive TCP congestion control that exploits explicit feedback from the cellular's base station. XRC requires two pieces of information from the base station: (1) the rate available to UEs' active flows; and (2) the head of line (HoL) delay of the base station's RLC buffer. The former enables XRC's sender to respond to the capacity changes in millisecond granularity when the bottleneck is at the cellular link. When the bottleneck is on the Internet, XRC incorporates the RLC's HoL delay and continuous end-to-end probing for available capacity with careful end-to-end queuing delay monitoring to adjust the sending rate accurately. Thus, XRC provides efficient utilization of radio resources as the available capacity rapidly fluctuates and achieves fairness across heterogeneous – both in terms of applications and congestion control – competing flows at a shared bottleneck.

Overall XRC aims to achieve the following key goals: (i) full utilization of scarce radio resources; (ii) low queuing delay (i.e., essential to meet the low-latency traffic requirements); and (iii) fair co-existence between heterogeneous competing flows. The key contributions of XRC are:

- A new rate control algorithm harnesses multiple bits of feedback from the base station to the sender to optimize the sending rate. XRC quickly adapts to channel condition changes while maintaining a small queuing delay. In particular, the XRC monitors the end-to-end queuing delay and the RLC HoL delay to force the RLC buffer to drain when the delay is above a threshold.
- A prototype implementation is developed over NS-3 to thoroughly evaluate XRC's performance across a wide range of scenarios and network settings. When competing with CUBIC at a wireless bottleneck, XRC achieves a Jain's index of 99.7% while providing a 3x lower median queuing delay compared to when CUBIC competes with CUBIC in the same setup. Also, when heterogeneous traffic flows (*e.g.,* latency-sensitive, video-streaming) compete at the bottleneck, XRC enables fair co-existence, helping these flows to meet their traffic requirements.

II. Background and Motivation

In this section, we demonstrate how the widely deployed TCP variants such as CUBIC [2], NewReno [4], Illinois [16], and Veno [6] fall short over cellular links. To study the performance of these TCP variants over the time-varying cellular link, we configure NS-3 simulated LTE link with a total bandwidth of $10MHz$. The RLC buffer can hold 128 MTU-sized packets. We randomly change the distance between the UE to the base station to model sudden changes in channel conditions. The scheduler at the base station's MAC layer selects the Modulation Coding and Scheme (MCS) according to the received channel quality indicator (CQI) feedback from the UE. It then allocates a transport block size (TBS) to the UE that dictates the UE's available capacity. As the distance between UE and base station increases, the CQI value drops sharply, and in turn, the MCS value becomes smaller, resulting in a lower TBS allocation by the scheduler. In this setup, UE has only a single flow; thus, the theoretical rate of the UE's flow fluctuates between 6.7Mbps and 34Mbps.



(a) (b)

Figure 1. Performance of TCP variants under time-varying channel condition of the cellular link. Figure 1(a) shows the achieved rate as the available capacity to UE fluctuates (shaded area). XRC constantly achieves close to maximum possible rate while it maintains the HoL delay within a small threshold (10ms).

Figure 1 shows the results. The shaded area in Figure 1(a) is the available rate and the curves are the achieved rate. The corresponding results for the RLC's Head-of-Line (HoL) delays is illustrated in Figure 1(b). CUBIC, NewReno, and Veno respond similarly to bandwidth changes by reducing their rate only when they detect packet losses. Yet, they neither reduce their rate sufficiently to cause the queue to drain quickly as the capacity shrinks nor increase their rate promptly when the capacity becomes available again. Figure 1 highlights the consequence of this poor rate adjustment as CUBIC, NewReno, and Veno occupy a small fraction of the queue, most of the time, which indicates their failure to utilize the available capacity. At other times when they fully utilize the available capacity, they tend to occupy the bottleneck queue exceedingly. For example, at the 90th percentile, all schemes, except XRC, suffer from 400ms HoL delay, an order of magnitude higher than XRC. Illinois has shown slightly better adaptation to capacity changes compared to CUBIC, NewReno, and Veno (Fig. 1(a)), yet it still suffers from high HoL due to its inaccurate sending rate adjustment. On the other hand, XRC precisely responds to capacity fluctuations, fully utilizing the available capacity offered by the base station to UE quickly, while it controls end-to-end delay within its small queuing budget (10ms in this experiment) at 80% of the time.

III. XRC Design

XRC is an explicit rate control algorithm designed to utilize the available wireless capacity while maintaining the queuing delay within a small threshold. In particular, XRC has the following objectives:

- To track wireless channel conditions quickly and accurately, especially suited to wireless technologies (*e.g.,* 4G, 5G and Beyond) ensuring high link utilization across a wide range of network dynamics;

- To maintain the end-to-end queuing delay below a threshold that can meet the stringent latency requirements of emerging ultra-low latency applications;
- To support heterogeneous flows – in terms of traffic requirements and congestion control – by fair co-existence with at both cellular and Internet bottlenecks.

XRC has two components, (i) the **XRC Sender**, a novel congestion control algorithm that operates at the transport layer, and (ii) **XRC Agent**, a distributed subsystem that resides at the PDCP layer with a set of API functions distributed over the RLC and MAC layers of the 4G/5G stack. The XRC Agent leverages the MAC layer scheduling information to estimate the fair capacity-share for each active UE flow. As the scheduling decision changes every Transmission Time Interval (TTI), which is 1ms in 4G, the XRC Agent estimates the capacity at millisecond granularity. This estimate can be performed at microsecond granularity in the case of 5G-NR due to a shorter TTI. The Agent also extracts the HoL delay from the RLC layer, which represents the last hop queuing delay. XRC Agent explicitly sends these measurements to the XRC Sender by piggybacking them on the available ACK packets. The Sender can then detect the bottleneck's location and instantaneously increase/decrease its sending rate to utilize the available capacity while avoiding a high queuing delay.

A. XRC Agent and Deployment Feasibility

The next-generation cellular networks adopt an open and flexible RAN architecture. The core idea is to decompose the RAN into a chain of virtualized functions that can be distributed and run over the cloud environment or commercial-off-the-shelf (COTS) hardware. In the new architecture [17], the virtualized RAN functions can be distributed among a Centralized Unit (CU), Distributed Unit (DU), and Radio Unit (RU), as shown in Figure 2. This functional split and its variant dynamic logical split options (i.e., options 1-7) provide the flexibility required to handle the diversity of traffic requirements and deployment scenarios. It also allows mobile operators to add new functions/services to their network without hardware modifications. In light of this, we envision the deployment of the XRC Agent as part of the RAN virtualization, where the XRC Agent represents a set of virtualized monitoring functions that do not intervene or change the decision of the actual 4G/5G stack control modules but rather monitor statistics and decisions, and report them to the XRC Sender. In particular, the XRC Agent leverages MAC layer scheduling decisions to estimate the capacity assigned to each UE and the free resources at TTI granularity. It then calculates the capacity available to a UE as a function of the TBS (Transport Block Size) assigned by the scheduler and the Modulation and Coding Scheme (MCS) observed on that UE, together with the fair share of the free resource blocks. The XRC Agent avoids over-estimating the cellular link capacity by excluding the MAC layer retransmissions from this capacity calculation. Specifically, with every scheduling decision, the Agent counts only the new transport blocks identified by New Data Indicator (i.e., NDI = 1) flag in a given TTI. The observed

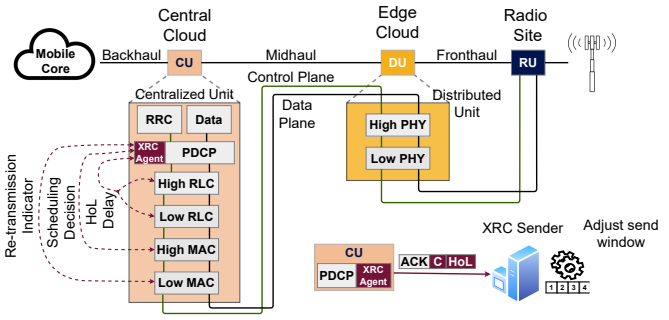


Figure 2. An overview of XRC deployment scenario in a 5G RAN functional split architecture. The XRC Agent determines the available capacity for each active UE flow, and the HoL delay from the MAC and RLC layers. This is sent to the XRC Sender, which adjusts the send rate of its flows.

capacity is averaged over the most recent 40 TTI (i.e., four frames in case of 4G) and sent to the PDCP layer where the XRC's Agent calculates the expected fair-share capacity as the ratio between the capacity per UE to the total active flows at the UE:

$$C_i = \frac{C}{af} \tag{1}$$

where C_i is per-flow capacity, af is the UE's active flows and C is the UE's capacity. The XRC Agent at the PDCP layer identifies the active flows by monitoring the packet inter-arrival time. Each flow is marked as inactive if the inter-arrival time of its packets exceeds a certain threshold (e.g., 200ms). This ensures that application-limited flows, such as video, with an ON/OFF traffic pattern, would be treated as inactive flow once they are in their OFF period. XRC considers these flows as active when they start sending traffic again. This ensures that the available capacity for the UE is distributed efficiently across flows with non-continuous traffic patterns. Finally, when ACK packets are present at the PDCP layer, the Agent queries the XRC's API at the RLC layer to retrieve the current HoL delay of the RLC buffer, and then explicitly sends this with the estimated fair-share of the capacity to the XRC Sender by piggybacking on the available ACK packets, as illustrated in Figure 2.

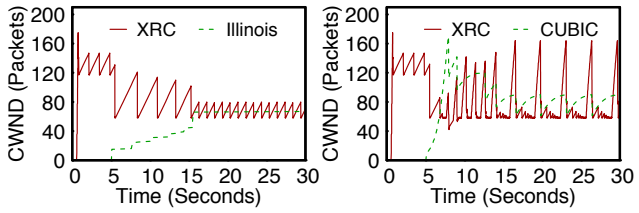
B. XRC Algorithm

Upon reception of an ACK with explicit feedback, XRC calculates a target congestion window ($twnd$) as follows:

$$twnd = R_{exp} * RTT_{min} \tag{2}$$

where R_{exp} is the explicit advertised rate by the network and RTT_{min} is the minimum observed RTT since connection startup. In an ideal condition, $twnd$ should maximize throughput with no queuing delay. However, adjusting the congestion window (CWND) blindly to $twnd$ is not practical in today's Internet because there can also be another end-to-end bottleneck in the Internet portion of the path, which is not reflected in $twnd$. As a result, monitoring the queuing delay is crucial to detecting such a bottleneck.

The XRC Sender estimates the queuing delay (σ) in every window of data as the maximum of the average queuing delay



(a) (b)

Figure 3. XRC co-exists with non-XRC flows. (a) XRC competes with Illinois (delay-based scheme), (b) XRC competes with CUBIC (loss-based scheme)

and the RLC HoL delay (HoL_{exp}) as follows:

$$\sigma = \max\{\sum_{j=1}^{N} RTT_j * \frac{1}{N} - RTT_{min}, HoL_{exp}\} \quad (3)$$

where σ is the end-to-end average queuing delay estimation, RTT_j is jth RTT sample in a window, N is the total number of RTT samples, and HoL_{exp} is the explicit HoL delay of the RLC buffer. The RLC HoL delay ensures a correct estimation of the queuing delay when no accurate RTT_{min} has yet been observed. This might be the case when XRC's flow arrives at a wireless bottleneck that is already carrying other flows (e.g., CUBIC, NewReno).

Detection of the bottleneck location. XRC operates differently depending on the detected location of the bottleneck link. To correctly identify the location of the bottleneck link, we leverage both the end-to-end queuing delay measurements and the explicit RLC HoL delay feedback. When the RLC HoL delay is below the queuing budget and the observed end-to-end queuing delay exceeds the RLC HoL delay by more than the queuing budget, XRC deduces that the bottleneck link is in the Internet portion of the path. On the other hand, when the RLC HoL delay is larger than the queuing budget, XRC assumes the bottleneck is at a cellular link.

Congestion Avoidance. XRC computes a weight factor (α), which dictates the aggressiveness of the XRC Sender. The value of α is large when the congestion window (CWND) is lower than the estimated target window (defined in Equation 2), as well as the queuing delay (σ) is smaller than the queuing budget (λ). That ensures a rapid increase (i.e., within 2 RTTs) to the CWND until it reaches the estimated target window, ($twnd$). Once the estimated target window is reached, XRC increases the CWND slowly (i.e., one segment per RTT). XRC resets the CWND to the estimated target window when the end-to-end queuing delay exceeds the pre-defined queuing budget. XRC behaves similar to standard TCP (α=1), when it detects an Internet bottleneck, which continues until the sender reaches the target rate, during which it sends with a steady rate until a packet is dropped or the estimated queuing delay becomes smaller than the budget.

Unfairness mode. When XRC flow competes with non-XRC flows at a wireless bottleneck, it might experience a high queuing delay (i.e., when competing with loss-based flows). In such conditions, XRC increases slowly for N number of RTTs (*e.g.,* N=10) before it enters the unfairness mode, during which α becomes temporarily large until the competing flows slow down due to packet losses. Finally, XRC follows standard TCP's loss recovery and slow start mechanism.

To highlight key behaviors of XRC, we setup an experiment where a XRC flow competes with a non-XRC flow at a single UE. The non-XRC flows join the bottleneck link after 5 secs with the slow start threshold value of one segment (i.e., the non-XRC flow goes to the congestion avoidance phase immediately after its connection startup). We run the experiment for CUBIC and Illinois. The results in Figure 3 demonstrates the following: (1) XRC's behavior using an empty pipe (Fig. 3(a) between 0-5 secs). XRC first reaches its $twnd$ quickly and then starts to slowly probe for path capacity until the observed queuing delay reaches the queuing budget, at which point the CWND is reset to $twnd$. A sawtooth wave results from this cycle; (2) XRC's behavior when competing with a flow that is slow in ramping up to its fair capacity-share (Fig. 3(a) between 5-15 secs). In such scenario, XRC's probing mechanisms takes longer to reach the queuing budget and thus, it tends to avoid the link underutilization caused by slow non-XRC flow (Illinois); (3) XRC's behavior when competing with a flow that respects its fair-share of the capacity (Fig. 3(a) between 15-30 secs). This highlights the perfect co-existence of XRC with a delay-based scheme. We expect a similar outcome when XRC competes with BBR; and (4) XRC's behavior when competing with a flow that seeks to violate its fair-share of capacity (Fig. 3(b) between 5-30 secs). In such a scenario, XRC tries to maintain a low-queuing delay, producing periodic packet losses that enforce the competing flow (*e.g.,* CUBIC) to slow down.

IV. Evaluation

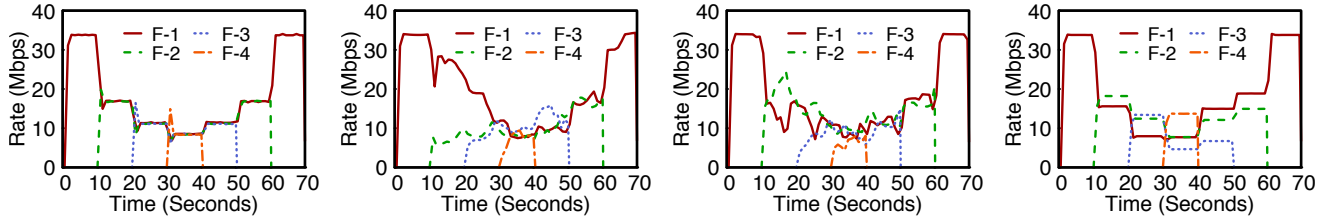
In this section, we evaluate XRC by focusing on the key design objectives as follows:

- **Fairness and Convergence:** we examine XRC's fairness when competing with other XRC flows with the range of RTTs and sharing the same wireless bottleneck.
- **Efficiency and Co-existence:** we highlight the XRC's performance when it competes with heterogeneous flows, including (1) capacity-seeking flows; (2) video-streaming flows with constant bit-rate and ON/OFF traffic patterns; and (3) latency-sensitive flows.

Experiment setup. We implemented XRC in NS-3, and emulated the cellular link using a modified LTE module in NS-3. The end-to-end minimum RTT is configured with $38ms$. This includes $8ms$ RTT between the base station and UE on the air interface and $30ms$ RTT between the sender and the base station. The queue budget is $10ms$. For the cellular link, we configured a single base station with a total bandwidth of $10MHz$ (50 Physical Resource Blocks), and the RLC buffer size of 128 MTU-sized packets.

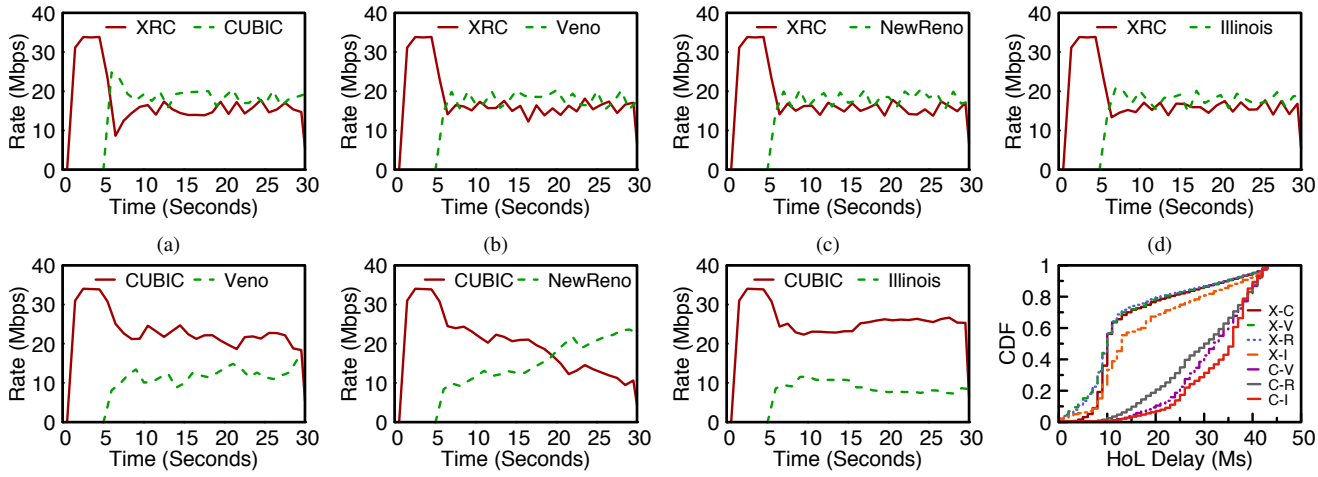
A. Fairness and Convergence

Multiple TCP flows sharing the same bottleneck at a cellular link should compete fairly with one another. In the light of that objective, we compare how fast different TCP variants,



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

Figure 4. (a) Convergence and fairness property of XRC flow with different minimum RTTs. Figures (b) CUBIC, (c) NewReno, and (d) Illinois show the same experiment but use different congestion control algorithms. XRC's flows quickly converge to their fair share as the bottleneck's dynamic changes.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 5. Achieved rate when (a) XRC co-exists with CUBIC (b) XRC co-exists with Veno, (c) CUBIC co-exists with NewReno, and (d) CUBIC co-exists with Illinois. XRC enables equal distribution of cellular-link resources between flows. X, C, I, and V represent XRC, CUBIC, Illinois, and Veno, respectively.

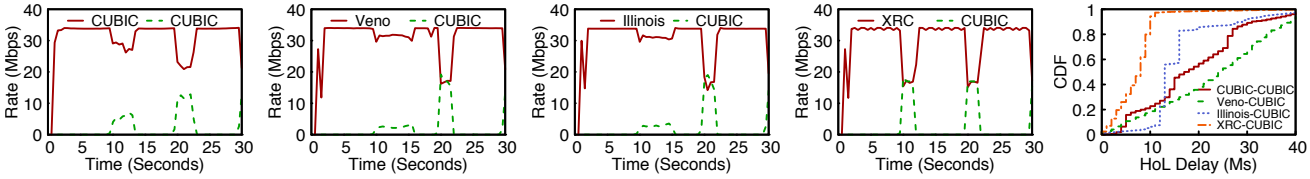
including XRC, converge to their fair-share of capacity as flows join and leave the bottleneck. We investigate this by considering F number of XRC flows (e.g., F1 - F4 for four flows) of different round-trip propagation delays (with RTT differences of up to 30ms), sharing a cellular bottleneck. Initially, 4 XRC flows to arrive at the UE with an inter-arrival time of $10secs$, making the base station a bottleneck. After $70secs$, the flows leave the bottleneck again with a $10secs$ gap between each flow's departure. Figure 4 shows the results. XRC (Fig. 4(a)) quickly converges to the fair share of the bottleneck capacity with Jain's fairness index of 99.5% when competing with four simultaneous flows with different RTT (100% is ideal). Both CUBIC (Fig. 4(b)) and NewReno (Fig. 4(c)) suffer from slow convergence when competing flows have different RTT; It is expected to see that CUBIC performs better than NewReno under such conditions due to its time-based cubical window increase function, however, due to the small BDP (Bandwidth Delay Product) across the flows in this setup, NewReno performs slightly better than CUBIC because it can ramp up faster. The behavior is reflected in the observed Jain's fairness index for both the CUBIC (85.71%) and NewReno (86.74%) flows. Illinois (Fig. 4(d)) suffers a severe RTT fairness problem due to imprecise estimation of the bottleneck capacity when competing with other flows. Illinois tends to converge and stay in an unfair state. Instead, XRC flows share the bottleneck capacity equally as flows join and leave; XRC converges to its fair share immediately while controlling the RLC's HoL delay within the budget of $10ms$ at 99% of the time (due to space we did not show this result). Finally, we make a similar observation across all the schemes by repeating this experiment with flows with the same RTT.

B. *Efficiency and Co-existence*

A XRC flow should share resources fairly with non-XRC flows sharing a wireless bottleneck. To that end, we conduct several experiments where XRC competes with other TCP variants carrying either homogeneous or heterogeneous traffic to XRC. In the following, we first define our flow types (e.g., capacity-seeking, low-latency, and video-streaming) and then walk through our experiments.

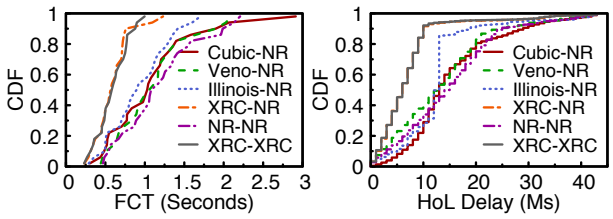
Capacity-seeking flow. It is represented by a long-lived TCP flow (*e.g.,* file download) that uses the bottleneck link for the duration of the simulation.

Video-streaming flow. Video traffic typically requires high capacity, but due to the widely used DASH protocol, exhibits an ON/OFF traffic pattern over TCP. This way, the flow has a periodic cycle of 10-secs wherein the first 2-secs burst packets with a constant bit rate of $17Mbps$ (i.e., representing a high-quality video chunk), then stops sending. In the next 8-secs, it sends only if all the sent packets during the 2-secs ON period are delivered successfully. This 10-secs cycle is repeated for the period of the simulation.



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

Figure 6. With different traffic types (red curves represent file download traffic, and green curves represent video-streaming traffic) XRC shows immediate response and successfully slows the rate of the file download flow to enable a fair share of resources.



(a) (b)

Figure 7. (a) Comparing the flow completion time (FCT) of low-latency short flows, and (b) the HoL delay at the RLC buffer. NR represents NewReno.

Latency-sensitive flow. It is represented as a short-lived TCP flow with random sizes ranging from $10KB$ to $2MB$. During the period of simulation, we generated 50 non-overlapping short flows.

Capacity-seeking vs. Capacity-seeking flows In this setup, an XRC flow carrying capacity-seeking traffic initially arrives at the UE at 0.5 secs and then after 5 secs another non-XRC flow carrying a similar traffic pattern arrives, after which the two flows compete for the remainder of the simulation. We repeat the experiment with the first flow using CUBIC. Figure 5 shows the results. As illustrated in Figure 5(a)-5(d), XRC perfectly co-exists with all other TCP variants without damaging their performance, while CUBIC failed to co-exist in all the scenarios (Fig. 5(e)-5(g)) particularly with Illinois, which is a delay-based congestion control. Figure 5(h) shows the RLC HoL delay in each experiment. Almost 70% of the time in all experiments XRC controls the queue occupancy to be within a small queuing budget of 10ms. In addition, at the 50th and 90th percentiles XRC reduced the RLC HoL delay by more than 50% across all experiments compared with using CUBIC. Thanks to XRC's unfairness mechanism that actively controls the HoL delays within a small threshold by slowing down competing flows as they try to bloat the buffer.

Capacity-seeking vs. video-streaming flows. In this simulation, a TCP flow carries file download traffic which competes with a CUBIC flow carrying video content. Figure 6 shows the results. XRC shows immediate response and successfully slows down the rate of the file download traffic as the video-streaming traffic bursts its packets into the network. When the video traffic enters its OFF period, XRC immediately increases its sending rate, fully utilizing the available capacity. On the other hand, other schemes fail to slow down and consequently the video traffic causes severe packet losses across all competing flows. As a result, the video chunk takes longer than 2 secs to be delivered with non-XRC flows, degrading the quality of experience for users watching the video content. Figure 6(e) illustrates that XRC consistently maintains (at 99% of the time) the end-to-end queuing delay below its budget of 10ms, almost 4x lower than other schemes.

Capacity-seeking vs. low-latency flows A capacity-seeking flow competes with 50 non-overlapping low-latency flows using NewReno. Each low-latency flow starts a transfer to the UE with a gap of 5 secs. Figure 7(a) and 7(b) show the results for the FCT of low-latency flows and the HoL delay at the RLC buffer dedicated to the UE. When the capacity-seeking flow is XRC, the FCT of short flows (NewReno) is reduced by 50% at both the 50th and 90th percentiles compared to other schemes. With XRC the maximum FCT of low-latency flows is 3x and 1.4x lower than CUBIC and Illinois, respectively. Overall, Illinois performs slightly better than CUBIC, NewReno and Veno due to its delay-based control. This can also be seen in Figure 7(b), where Illinois bloats the bottleneck buffer less than other schemes, except XRC, which controls the buffer within its queuing budget of 10ms. This figure also shows the results of a setup where both flows of latency-sensitive and capacity-seeking are XRC. This demonstrates that XRC allows different applications with distinct requirements from the network to co-exist gracefully.

V. Discussions and Future Plan

Multipath congestion control. A transport layer protocol that exploits multiple paths between source and destination has been an active area of research [18], [19], [20], [21], [22], [23], [24], [25]. MPTCP [20] divides a TCP flow into multiple subflows. Since those subflows may take different paths, MPTCP shifts traffic between its subflows to avoid congested paths. This can ensure high performance and improved resilience to network failure, primarily when a mobile device uses multiple access technologies simultaneously (e.g., 4G/5G, WiFi, and Satellite). We plan to expand XRC in the context of MPTCP.

Access Traffic Steering, Switching, and Splitting (ATSSS). The 3GPP has defined in Release 16 [26] a new 5G functionality called ATSSS that manages WiFi and 5G convergence. ATSSS supports MPTCP as an underlying transport protocol to deliver diverse traffic (e.g., UDP, TCP, and Ethernet) between UE and UPF (User Plane Function). UPF is located in the 5GC (5G core) network. From UPF, a single path TCP may be utilized to deliver the UE traffic to its destination. ATSSS operates over MA-PDU (Multi-Access PDU) session, which allows a single PDU session to be established between UE and 5GC (i.e., UPF), carrying UE's traffic across both 3GPP and non-3GPP access. For the latter case, a gateway is utilized

(either TNGF or N3IWF, depending on whether the access is trusted or not [26]). In such cases, XRC Agent could reside in gNB, N3WIF/TNGF, and/or UPF. We plan to study XRC in the context of the 3GPP's ATSSS framework.

Real prototype and its challenges. In this paper, we have prototype XRC within the NS-3 simulator and examined its performance against several other TCP congestion control algorithms available in NS-3. We are currently working on a real prototype with srsRAN and Linux Kernel supporting several state-of-the-art congestion controls. We will report on these in the near future.

A crucial aspect in real deployment of XRC is related to delivering explicit feedback from the XRC Agent to XRC Sender. There are several ways to deliver such a signal. One common approach is to use the IP or TCP option to piggyback the explicit feedback on ACK packets [27]. However, in today's Internet, there are middle-boxes with the tendency of removing the TCP option from the TCP stream [28]. That said, it is common practice to host applications and services in the network edge close to users (e.g., via CDN servers) to minimize latency. This limits the exposure of XRC to non-cooperatives routers and middle-boxes. Another approach would be sending the explicit feedback on a separate connection between the XRC Agent and the XRC Sender. We plan to examine these deployment options and their practical implications over our real prototype.

The XRC Agent at PDCP collects statistics and measurements from the RLC and MAC layers. When NG-RAN utilizes the functional split with option 6 (see Fig. 2), the PDCP, RLC, and MAC layers are not distant from one another, so the communication latency between them is negligible. However, when the DU and CU are split via Option 2 (a preferred option by 3GPP), the PDCP layer is only located at CU, and it communicates with other layers down the radio protocol stack at DU over the F1 interface. In the latter case, there would be a communication latency, and it merely depends on the link technology used between DU and CU. This may not be crucial for the XRC because the XRC Agent calculates the available capacity and reports to the XRC Sender over a specific time window. That said, we plan to study the latency involved between DU and CU and its potential impacts on the XRC operations in real scenarios.

VI. Conclusion

In this paper, we proposed XRC, an adaptive congestion control algorithm that exploits explicit base station's feedback. XRC achieves fair co-existence between flows with heterogeneous traffic requirements while sharing the bottleneck with various congestion control algorithms. Our NS-3 simulation results show the superior performance of XRC compared to the state-of-the-art schemes (both delay-based and loss-based).

References

[1] E. Atxutegi, F. Liberal, H. K. Haile, K.-J. Grinnemo, A. Brunstrom, and A. Arvidsson, "On the use of TCP BBR in cellular networks," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 172–179, 2018.

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[3] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[4] S. Floyd and T. Henderson, "RFC2582: The NewReno Modification to TCP's Fast Recovery Algorithm," 1999.

[5] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[6] C. P. Fu and S. C. Liew, "TCP Veno: TCP enhancement for transmission over wireless access networks," *IEEE Journal on selected areas in communications*, vol. 21, no. 2, pp. 216–228, 2003.

[7] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *SIGCOMM*, 1994.

[8] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg, "Adaptive Congestion Control for Unpredictable Cellular Networks," in *SIGCOMM*, 2015.

[9] W. K. Leong, Z. Wang, and B. Leong, "TCP Congestion Control Beyond Bandwidth-Delay Product for Mobile Cellular Networks," in *CoNEXT*, 2017.

[10] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-learning congestion control," in *NSDI*, 2018.

[11] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *NSDI*, 2018.

[12] S. Park, J. Lee, J. Kim, J. Lee, S. Ha, and K. Lee, "Exll: an extremely low-latency congestion control for mobile cellular networks," in *CoNEXT*, 2018.

[13] H. Lee, J. Flinn, and B. Tonshal, "RAVEN: Improving interactive latency for the connected car," in *MobiCom*, 2018.

[14] Y. Xie, F. Yi, and K. Jamieson, "PBE-CC: Congestion Control via Endpoint-Centric, Physical-Layer Bandwidth Measurements," in *SIGCOMM*, 2020.

[15] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "ABC: A Simple Explicit Congestion Controller for Wireless Networks," in *NSDI*, 2020.

[16] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6-7, pp. 417–440, 2008.

[17] 3GPP, "Study on new radio access technology: Radio access architecture and interfaces (Release 14)," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.801, 2017, version 14.0.0.

[18] M. Zhang, J. Lai, A. Krishnamurthy, L. L. Peterson, and R. Y. Wang, "A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths," in *ATC*, 2004.

[19] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM ToN*, vol. 14, no. 5, pp. 951–964, 2006.

[20] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *SIGCOMM*, 2011.

[21] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *INFOCOM*, 2016.

[22] M. Kheirkhah and M. Lee, "AMP: An Adaptive Multipath TCP for Data Center Networks," in *IFIP Networking*, 2019.

[23] M. M. Kassem, M. Kheirkhah, M. K. Marina, and P. Buneman, "WhiteHaul: An Efficient Spectrum Aggregation System for Low-Cost and High Capacity Backhaul over White Spaces," in *MobiSys*, 2020.

[24] M. Kheirkhah, I. Wakeman, and G. Parisis, "Short vs. Long Flows: A Battle That Both Can Win," in *SIGCOMM*, 2015.

[25] M. M. Kassem, M. Kheirkhah, M. K. Marina, and P. Buneman, "WhiteHaul: White Space Spectrum Aggregation System for Backhaul," in *MobiCom*, 2020.

[26] 3GPP, "Access Traffic Steering, Switching and Splitting (ATSSS); Stage 3; (Release 16)," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 24.193, 2021, version 14.4.0.

[27] V.-H. Tran and O. Bonaventure, "Beyond socket options: making the Linux TCP stack truly extensible," in *IFIP Networking*, 2019.

[28] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is It Still Possible to Extend TCP?" in *IMC*, 2011.