RAPID: a RAN-aware Performance Enhancing Proxy for High Throughput Low Delay Flows in MEC Networks

Mamoutou Diarra, Walid Dabbous, Amine Ismail, Brice Tetu, Thierry Turletti

RAPID: a RAN-aware Performance Enhancing Proxy for High Throughput Low Delay Flows in MEC Networks

Mamoutou Diarra[a,b,c], Walid Dabbous[b,c], Amine Ismail[a], Brice Tetu[a] and Thierry Turletti[b,c]

[a] *Ekinops, Sophia Antipolis, France*
[b] *Inria, Sophia Antipolis, France*
[c] *Université Côte d'Azur, Sophia Antipolis, France*

ARTICLE INFO

Keywords:
TCP Congestion and Flow control
Congestion Control Algorithm (CCA)
self-inflicted bufferbloat
5G Enhanced Mobile Broadband (eMBB)
Multi-access Edge Computing (MEC)
Radio Network Information Service (RNIS)
Performance Enhancing Proxy (PEP)
OpenAirInterface (OAI)

ABSTRACT

5G enhanced Mobile broadband (eMBB) aims to provide users with a peak data rate of 20 Gbps in the Radio Access Network (RAN). However, since most Congestion Control Algorithms (CCAs) rely on startup and probe phases to discover the bottleneck bandwidth, they cannot quickly utilize the available RAN bandwidth and adapt to fast capacity changes without introducing large delay increase, especially when multiple flows are sharing the same Radio Link Control (RLC) buffer. To tackle this issue, we propose RAPID, a RAN-aware proxy-based flow control mechanism that prevents CCAs from overshooting more than the available RAN capacity while allowing near optimal link utilization. Based on analysis of up-to-date radio information using Multi-access Edge Computing (MEC) services and packet arrival rates, RAPID is able to differentiate slow interactive flows from fast download flows and allocate the available bandwidth accordingly. Our simulation and experimentation results with concurrent Cubic and BBR flows show that RAPID can reduce delay increase by a factor of 10 to 50 in both Line-of-Sight (LOS) and Non-LOS (NLOS) conditions while preserving high throughput in both 4G and 5G environments.

1. Introduction

Over the last years, Internet content and resources have increasingly been located close to the end users by means such as Content Delivery Networks (CDN) or edge computing. In such a context and as today's cellular networks rely on well-provisioned backhaul, the radio link between the user and the base station, as illustrated in Figure 1, most often becomes the bottleneck [1], mostly due to the inherent characteristics and limitations of cellular Radio Access Networks (RANs). Consequently, if not managed appropriately, such a bottleneck could hinder 5G performances, especially in enhanced Mobile Broadband (eMBB) scenarios.

Moreover, traditional loss-based Congestion Control Algorithms (CCAs), which still govern the most majority of today's TCP traffic, are not designed to appropriately discover cellular RAN bandwidth without introducing high delay or excessive buffering. These algorithms are designed to continuously overshoot the bottleneck link until a packet loss occurs, generally due to buffer overflow. Therefore, in presence of deep buffer like in today's cellular networks, such a technique naturally ends up creating excessive buffering, also known as bufferbloat (as termed by Gettys in [2]). Also, in cellular networks, since data traffic is isolated on per-user basis (thanks to the use of per-user queues and logical tunnels), an excessive buffering caused by a given user cannot propagate to another user in the same cell. In other word, a user can suffer only from a bufferbloat caused by its own flows, hence the term self-inflicted bufferbloat. This phenomenon is observed in the startup and steady states

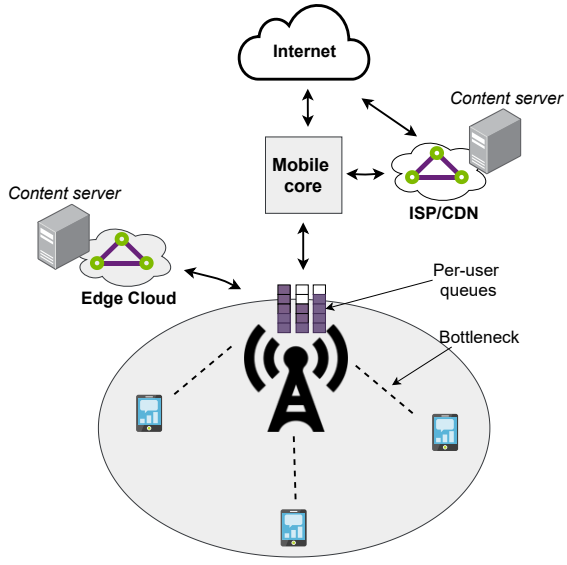


Figure 1: Bottleneck location in cellular networks.

of Cubic, which is currently the most widely deployed loss-based CCA on the Internet [3]. Cubic uses an exponential window adjustment during startup state (meaning that it doubles the amount of bytes in flight every RTT), followed by a Multiplicative-Increase Multiplicative-Decrease (MIMD) window adjustment in the steady state thanks to the use of a cubic function that aggressively increases the amount of bytes in flight after loss events. Considering this way of operation, Cubic is clearly more aggressive than the traditional Additive-Increase Multiplicative-Decrease (AIMD) technique used by the historical NewReno algorithm. However, it is worth noting that, both Cubic and NewReno as

mamoutou.diarra@inria.fr (M. Diarra); walid.dabbous@inria.fr (W. Dabbous); amine.ismail@ekinops.com (A. Ismail); brice.tetu@ekinops.com (B. Tetu); thierry.turletti@inria.fr (T. Turletti)
ORCID(s): 0000-0001-5869-0970 (M. Diarra)

well as most loss-based CCAs tend to have similar behaviors in cellular networks. In fact, since it takes time to overflow the deep buffers (over 10 seconds in some commercial networks as reported in [4]), flows using loss-based CCAs tend to spend most of their time in the startup state, i.e., doing exponential growth, thus rapidly end up exceeding the link actual capacity. In such a context, using a rate-based CCA such as BBR (Bottleneck Bandwidth and Round-trip propagation time) [5] which adapts its sending rate based on the network delivery rate and the minimum round-trip delay, or a cross-layer CCA such as PBE-CC (Physical-Layer Bandwidth measurements, taken at the mobile endpoint) [6] which matches its sending rate with the achievable radio capacity, could theoretically alleviate the issue. However, the self-inflicted bufferbloat issue still persists since these algorithms cannot perform well when competing with loss-based CCAs which keep introducing excessive buffering due to the presence of deep buffers. Other alternatives to end-to-end solutions such as RAN-aware proxies (e.g., milliProxy [7]) have also been proposed in order to mitigate the negative effects of loss-based CCAs, however, these solutions fail to prevent high delay increase in case of multiple flows per user. In general, cross-layer CCAs and existing RAN-aware proxies adapt their sending rate to the RAN bandwidth allocated to the User Equipment (UE) without taking into account the number of active flows sharing this bandwidth in the UE. As a result, the per-user queues (at the base station) grow and cause self-inflicted bufferbloat when multiple flows progress in parallel in the same UE.

Indeed, all the aforementioned limitations along with the diversity of congestion control algorithms on the Internet make it very challenging to address the self-inflicted bufferbloat problem at the CCA level. In such a context, it could be necessary to use a flow-level bandwidth shaping in order to limit the interference between the flows that are using different CCAs. As suggested in [8], a simple solution could be to allocate equal portions of the available bandwidth to the concurrent flows; however such a solution may result in link under-utilization since some flows (e.g., slow or app-limited flows) may require just a small fraction of the total bandwidth. With that in mind, we design and evaluate a solution called RAPID that minimizes RTT increase (regardless of the CCA used by the end servers) and allocates the available RAN bandwidth by taking into account the nature of the active flows in the UE.

This work is an extension of our recent paper [9]. In particular, we further evaluate the proposed RAPID mechanism in a real-world 4G network, not only with TCP flows but also with QUIC and UDP flows, which allows us to better explore the key concepts embodied in its design. Overall, our contributions can be summarized as follows:

- We propose a RAN-aware performance enhancing proxy that intercepts TCP connections and distributes proportionally the available RAN bandwidth among the active flows. On this purpose, we rely on the MEC Radio Network Information Service (RNIS) and on packets arrival rates to estimate the aggregated RAN bandwidth and categorize the concurrent TCP flows in the UE. This approach does not involve the end user unlike existing cross-layer CCAs that introduce computational overhead and additional power consumption at the UE as they require client-side modifications in order to introduce radio information into the TCP header [6, 10].
- We provide an implementation of our solution both for the ns-3 network simulator and for real-world mobile networks by leveraging OpenAirInterface (OAI) [11] and other open source projects. The ns-3 implementation is exploited in order to conduct various performance evaluation experiments with different CCAs (under both LOS and NLOS radio conditions). We run our ns-3 simulations on a cluster of servers[1] and deploy the OAI infrastructure in the R2lab anechoic chamber[2] which is an open wireless experimentation platform. The scripts to reproduce all our scenarios/results as well as the source code of RAPID's implementation in Linux are made available to the community [12].
- We explore the demand-aware fairness concept embodied in the design of RAPID and demonstrate why a bandwidth allocation scheme based on Jain's Fairness Index (JFI) is not adapted for flows using different CCAs.
- We highlight the limitations of current CCAs over next-generation cellular networks and show that RAPID can significantly improve the overall performance in terms of goodput, delay and jitter, even in the presence of unintercepted UDP or QUIC flows.

The remainder of the paper is organized as follows. In Section 2 we present the motivations of this work and shed light on the self-inflicted bufferbloat problem and bandwidth under-utilization issue introduced by traditional CCAs in cellular networks. Section 3 presents the design and implementation details of RAPID in the ns-3 network simulator. Section 4 describes the simulation environment, evaluates the performance of our solution in some mobile edge scenarios and discusses some observed limitations.

Section 5 presents the design and implementation of our solution in Linux along with the 4G OAI experimentation testbed and evaluates its performance using well-known open source tools.

Section 6 explores the related work and finally, we summarize our contributions and future research directions in Section 7.

2. Motivations

The use of per-user queues in cellular networks has almost eliminated the likelihood of flow interactions and CCAs interference between separate users [1]. However,

[1] NEF cluster: https://wiki.inria.fr/ClustersSophia/Clusters_Home.
[2] R2lab Testbed: http://r2lab.inria.fr.

flows of the same user may still use different CCAs, and depending on the design and TCP friendliness of the involved CCAs, a large bufferbloat can be introduced [4, 13]. In fact, the probability of having multiple concurrent flows on the same UE is quite high in today's cellular networks, mostly due to the increasing number of mobile services which create more opportunities for parallel data transfer [14]. This probability becomes even higher when multiple devices are using the same mobile device's Internet connection (i.e., tethering). However, this also increases the probability of self-inflicted bufferbloat since the concurrent data flows on the UE may use different CCAs. This type of bufferbloat is called self-inflicted bufferbloat since it is introduced by the user itself. According to a recent study [3], over 30% of the top 20,000 Alexa websites are using Cubic and roughly 18% are currently based on BBR. This study also reveals that delay-based TCP variants such as Vegas/Veno and other unknown variants are used on 2.8 and 17.6% of the evaluated websites, respectively. Based on these findings, it appears that a large majority of current TCP traffic uses loss-based CCAs which are known to introduce excessive buffering in mobile networks. Furthermore, in mobile networks, due to the use of deep buffers in base stations, recent CCAs such as BBR that attempt to minimize delay increase are outperformed in terms of goodput by loss-based TCP variants when they compete for bandwidth. Other studies show that, in addition to being RTT unfair, BBR also suffers from bandwidth overestimation issues in fast-varying networks [15] and severe throughput collapse due to high delay variations over millimeter wave (mmWave) wireless links [16]. These issues along with the ever-growing number of CCAs that could potentially compete for bandwidth make it very challenging to solve the self-inflicted bufferbloat issue with a new CCA. It is also worth noting that the delay increase that may arise from these issues is particularly harmful for delay-sensitive flows and, therefore, greatly slow down web browsing as reported in [17].

The self-inflicted bufferbloat issue becomes even more significant in mmWave bands mostly due to the inherent properties of high frequencies. According to 3GPP TS 38.104, 5G New Radio (NR) can operate in two distinct frequency ranges, Frequency Range 1 (i.e., FR1 or sub-6GHz) and Frequency Range 2 (i.e., FR2 or 24.25 GHz to 52.6 GHz) which falls in the mmWave region. In fact, frequencies in the mmWave region are prone to high penetration losses due to obstacles blocking the Line-of-Sight. Therefore, in case of NLOS conditions, the base station buffers incoming packets which eventually leads to high delay increase and/or persistent bufferbloat [18].

Our main goal with this work is to mitigate the self-inflicted bufferbloat issues that arise from parallel flows or intermittent radio links using a new approach completely independent of the CCAs in use.

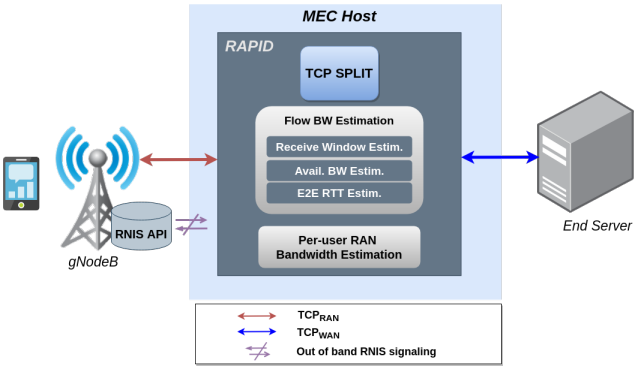


Figure 2: RAPID high-level architecture

3. RAN-aware proxy-based Flow control

In this section, we briefly describe the Radio Network Information Service, which is an essential part of our approach, then we present the high-level design of our RAN-aware proxy called RAPID. After that we show how our proxy leverages actual RAN statistics through the RNIS concept, as well as various transport layer related information in order to categorize flows and properly allocate the available bandwidth. RAPID's main originality w.r.t. solutions presented in Section 6 is twofold. First, it mitigates self-inflicted bufferbloat while preserving high throughput regardless of the characteristics of the CCAs in use or the number of concurrent flows sharing the same per-user queue at the base station. Second, it brings a demand-aware bandwidth allocation scheme which makes it possible to dynamically allocate bandwidth to each individual flow based on their behavior and estimated demand. In other words, it enforces a demand-aware fairness principle among the different flows of the user.

3.1. Radio Network Information Service (RNIS)

RNIS is a service exposed by the ETSI MEC framework that provides real-time radio network related information to applications deployed at the edge (i.e., in the MEC host). As described in the ETSI GS MEC-012 specification [19], the applications that subscribe to this service periodically receive various contextual information about the cells associated to the MEC host, such as: up-to-date radio network information regarding radio network conditions; measurement information related to the user plane; information about the connected UEs in the cells (e.g., information about the UE's radio bearers, channel quality, throughput, resource allocation, etc.); information about the cells Physical Resource Block usage; etc. One of the advantages of such an information, is that it gives the possibility to the edge applications to dynamically adapt their behaviors based on the current RAN conditions.

The use of RNIS in real-world experimentation has been demonstrated by FlexRAN [20], an open-source implementation of RNIS which allows a controller deployed somewhere in the network to collect radio information from some agents collocated with the base stations. FlexRan agents can

report with up to a Transmission Interval Time level (TTI-level) periodicity, detailed RAN statistics to their respective controllers. These statistics mainly consist of: the cell's total Physical Resource Blocks (PRBs), the used sub-carrier spacing, the Radio Network Temporary Identifier (RNTI) of each connected UE, the number of PRBs allocated to each UE, the Channel Quality Indication (CQI) reported by each UE as well as the corresponding Modulation and Coding Scheme (MCS), and the corresponding Transport Block Size (TBS) computed by the base station for each UE.

3.2. Proxy architecture

RAPID is a TCP proxy that seats close to the base station, in the MEC host and that leverages up-to-date radio information exposed by the RNIS service [19], similar to RAVEN [21] and MELD (proposed in our previous work [22]). As illustrated in Figure 2, RAPID relies on several modules organized in layers, meaning that each module provides services to its upper module.

At a basic level, RAPID first splits each incoming end-to-end TCP flow from a UE in two separate connections, referred here as TCP_{RAN} (i.e., TCP connection between the UE and RAPID) and TCP_{WAN} (TCP connection between RAPID and the corresponding end server). After that, it invokes the per-flow bandwidth estimation module in order to compute the bandwidth-delay product (BDP) available to each intercepted flow. Finally, the computed BDPs are used to override the Receive Window (RW) value in the TCP acknowledgements towards the corresponding end servers, thereby limiting the bytes in flight from those servers to the BDPs estimated for their respective flows. It is worth noting that while RAPID has no control over the choice of the end-server CCA, it can still control all the parameters of the connections established in the RAN segment (i.e., TCP_{RAN}). Therefore, by default and in order to prevent excessive buffering in the proxy, TCP_{RAN} relies on a simplified CCA that always sets its congestion window to the estimated flow BDP.

3.3. RAN bandwidth estimation

As illustrated in Figure 2, RAPID periodically receives RAN statistics through the RNIS service. Similar to the statistics reported by FlexRAN agents [20], these statistics include the cell bandwidth, the sub-carrier spacing, the Radio Network Temporary Identifier (RNTI), the number of allocated Physical Resource Blocks (PRBs), the computed Transport Block Size (TBS) and the selected Modulation and Coding Scheme (MCS) information for all the UEs connected to the cell. This information allows RAPID to estimate the bandwidth allocated to each UE by the base station in units of PRBs. However, the number of allocated PRBs does not always reflect the achievable bandwidth since the MAC scheduler at the base station may also take into account the RLC buffer state [23] or the incoming data rate [6] when allocating resources to users. Therefore, in order to estimate the achievable bandwidth, we first calculate the number of unused PRBs, $Pu_{\mathrm{i}}(t)$, by using the total number of PRBs in the cell N_{PRB}, the number of connected UE M and the number of PRBs allocated to each UE $Pa_{\mathrm{i}}(t)$:

$$Pu_{\mathrm{i}}(t) = N_{\mathrm{PRB}} - \sum_{i=1}^{M} Pa_{\mathrm{i}}(t)\,, \tag{1}$$

The expected number of PRBs, $Pe_{\mathrm{i}}(t)$ for a given UE i is then computed as follows:

$$Pe_{\mathrm{i}}(t) = Pa_{\mathrm{i}}(t) + Pu_{\mathrm{i}}(t)\,, \tag{2}$$

With the expected number of PRBs and the selected MCS values, RAPID computes the maximum transport block size at time t, denoted $TBS_{\mathrm{i}}(t)$, for each UE by either using the tables in 3GPP TS 36.213 [24] or the indications in 3GPP TS 38.214 [25]. The former is used for 4G while the latter is used for 5G. In case of 4G, we first use the MCS value (which is a direct mapping of the Channel Quality Indicator reported by the UE) to retrieve the corresponding TBS index (also known as I_{TBS}) from the MCS and I_{TBS} mapping table in [24]. Then the expected number of PRBs together with the I_{TBS} are mapped to the corresponding TBS value using the TBS mapping table (i.e., Table-7.1.7.2.1-1 in [24]). On the other hand, in case of 5G, the theoretical number of information bits that can be transmitted is computed by multiplying the number of sub-carriers in the allocated PRBs (e.g., 12 sub-carriers per PRB in case of 15 kHz sub-carrier spacing) by the modulation order (deduced from the MCS table in [25]) and by the coding rate (also deduced from the MCS table). If the resulting value is inferior or equal to the standardized 3824 bits [25] (which is chosen based on the maximum code block size that can be handled by 5G's channel coding technique without segmentation), then the corresponding TBS is directly fetched from the TBS table in [25] (i.e. Table-5.1.3.2-1), otherwise, the TBS is determined thanks to the standardized 3GPP formula given in [25], which takes into account the channel coding overhead in case of segmentation. In any case, once the necessary mappings are done and the TBS value that corresponds to the expected PRB combined with the reported MCS is found, the highest achievable throughput for a given UE denoted $C_{\mathrm{i}}(t)$ (in unit of bits/s), can be expressed as follows:

$$C_{\mathrm{i}}(t) = \frac{TBS_{\mathrm{i}}(t)}{TTI}\,, \tag{3}$$

where TTI stands for the Transmission Time Interval (e.g., TTI = 125 μs for Numerology μ=3). It is worth noting that this real-time RAN bandwidth estimation makes it possible to detect fast capacity variations as well as bad radio conditions (e.g., NLOS conditions in mmWave), which, to some degree, can cause excessive buffering at the base stations.

3.4. Per-flow bandwidth allocation

The major contribution of RAPID with respect to other existing solutions is its bandwidth allocation scheme. Rather than relying on a static allocation scheme, the bandwidth estimation module distributes the aggregated bandwidth of

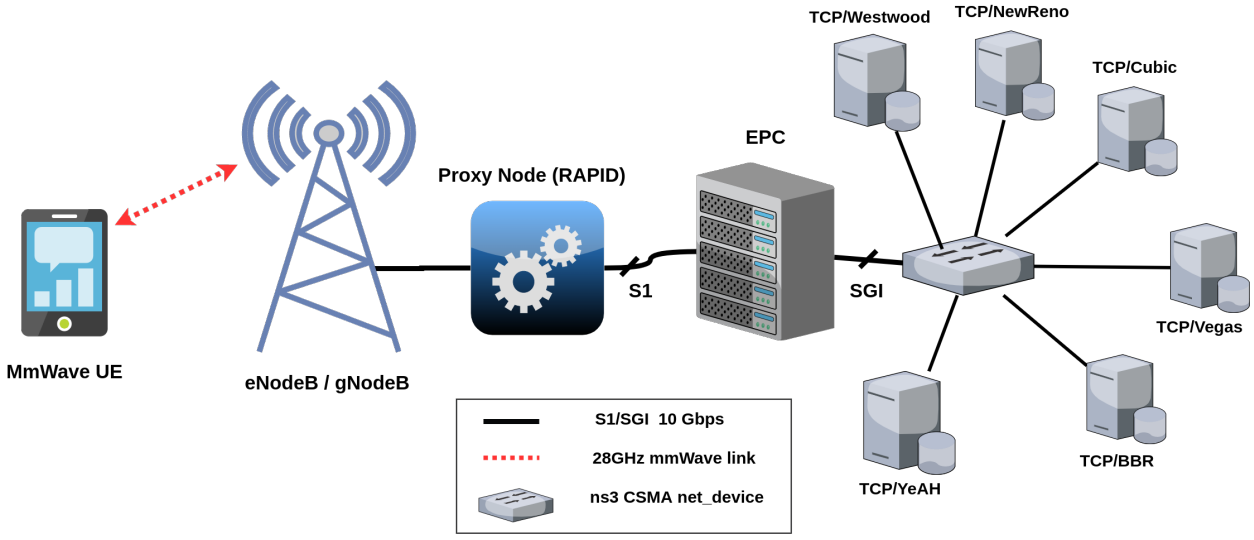


Figure 3: RAPID ns-3 experimentation testbed

the UE according to the requirements of the active flows in terms of bandwidth. On connection start, the initial Receive Window denoted $RW_{ij}(t)$ of a given flow j, from a UE i, is computed by dividing the expected bandwidth estimated for the UE (in $bits/s$) by the number of active flows:

$$RW_{ij}(t) = \frac{C_i(t)}{N} \, RTTmin_{ij}(t)\,, \tag{4}$$

where N is the number of active flows in the UE and $RTTmin_{ij}(t)$ is the minimum RTT of the j^{th} flow on the wired segment (i.e., from RAPID to the original end server) at time t. This initial phase is followed by a dynamic allocation phase after K_{ij} RTT_{WAN}, where the allocated bandwidth changes depending on flow category. We define two distinct categories of flows: slow interactive flows (i.e., browsing, instant messaging, etc.) and fast download flows (e.g., HD streaming, file transfer, Augmented and Virtual Reality AR/VR, etc.). The former are sensitive to delay but require low bandwidth while the latter may require both high bandwidth and reasonable delay variations. In such a context, allocating the same amount of bandwidth to all flows not only penalizes fast download flows (since slow flows only require a small fraction of the bandwidth) but also results in radio link under-utilization and wastage of the bandwidth allocated to the UE. With that in mind, we devise a flow categorization scheme based on a common behavior of TCP flows during startup phase. In fact, since the congestion windows of most TCP flows follow an exponential increase pattern during the initial phase, we can directly compute the congestion window value at different RTTs as follows:

$$cwnd = \begin{cases} MSS * 2^0 & \text{at the } 1^{st}\, RTT \\ MSS * 2^1 & \text{at the } 2^{nd}\, RTT \\ MSS * 2^{(K-1)} & \text{at the } K^{th}\, RTT\,, \end{cases} \tag{5}$$

where MSS is the negotiated Maximum Segment Size. From (5), by replacing $cwnd$ by RW, we can deduce the number of RTTs it takes to reach a congestion window value equivalent to the receive window:

$$RW = MSS * 2^{(K-1)} \implies K = \log_2\left(\frac{RW}{MSS}\right) + 1\,, \tag{6}$$

For the j^{th} flow of the i^{th} UE and at the instant t, eq. 6 becomes:

$$K_{ij}(t) = \log_2\left(\frac{RW_{ij}(t)}{MSS}\right) + 1\,, \tag{7}$$

where K_{ij} is the number of RTTs that the j^{th} flow of i^{th} UE takes in order to reach an amount of bytes in flight equivalent to the receive window RW_{ij}.
Additionally, since we know that the flow is going to reach the receive window RW_{ij} after K_{ij} RTTs (i.e., at instant $t + K_{ij}$), we can also express the arrival rate or incoming throughput the flow is expected to achieve at that time as:

$$R_{ij}(t + K_{ij}) = \frac{RW_{ij}(t)}{RTT_{WANij}}\,, \tag{8}$$

where RTT_{WANij} is the current smoothed RTT of the j^{th} flow on the wired segment (i.e., between the proxy and the end server.

At this point, with the information at our disposal, we can analyze the behavior of the flow and assign it to a predefined category. Basically, our categorization is based on the assumption that a flow that requires large bandwidth would follow an exponential increase and fully consume the allocated bandwidth by the end of the K_{ij}^{th} RTT, while an app-limited or interactive flow might not. We refer to the period of time between the last modification of the receive window and the end of the $(K_{ij})^{th}$ RTT as the categorization period. To put it another way, at the end of each categorization period, the actual incoming throughput or arrival

rate (noted $r_{ij}(t + K_{ij})$) of a flow that has fully consumed its allocated receive window should match its expected arrival rate ($R_{ij}(t + K_{ij})$, given in eq. 8. In this case, the flow is categorized as a "fast download flow" since it consumed the allocated bandwidth as expected. On the other hand, if the actual arrival rate of the flow at the end of the categorization period is less than the expected value, the flow is considered as a "slow flow", meaning that it doesn't require/need all its allocated bandwidth. However, such a strict threshold might cause some fast flows to be wrongly flagged as slow since it doesn't take into account the time it takes to move the packets from the Network Interface Card (NIC) to the TCP buffer, or the fact that some flows might leave the slow start phase prematurely due to the Hybrid slow start algorithm (HyStart). For all these reasons, we decided to use a more conservative categorization threshold by considering a flow as slow only if its actual arrival rate is less than a certain percentage (set empirically to 80%) of its expected arrival rate.

In any case, a flow categorized as slow, strongly indicates that the previously allocated bandwidth was too high compared to the the flow actual demands. Therefore, for this type of flow, we set the receive window to a value that reflects the flow actual demands. For a given flow, the demand or required bandwidth is computed based on the observed arrival rates during the categorization period. Technically speaking, at each RTT, a sample of the arrival rate denoted $r_{ij}(t)$, is taken and used to compute an exponential weighted moving average value, following an inverse version of the smoothed RTT calculation method in TCP. In fact, in our case, the weights are reversed in order to strongly reflect the increase in throughput over time. The rationale behind this choice is the fact that most CCAs keep increasing their congestion window or sending rate as long as the bottleneck pipe is not full but not the other way around. Thus, following our method, the actual demand of the flow is computed at each RTT until the end of the categorization period using the following expression:

$$\hat{r}_{ij}(t) = \alpha * \hat{r}_{ij}(t-1) + (1-\alpha) * r_{ij}(t)\,, \tag{9}$$

where $\alpha = 1/8$ (as in RFC 6298), $\hat{r}_{ij}(t)$ and $\hat{r}_{ij}(t-1)$ the exponential weighted moving average of the arrival rate calculated during the current RTT and during the previous RTT, respectively. Once the demand or required bandwidth of the slow flow is captured thanks to eq. 9, we decrease its bandwidth allocation, i.e., its receive window in order to match its demand. This is done by multiplying the previous receive window by the ratio of the captured smoothed arrival rate over the expected arrival rate. In other words, the previous receive window is multiplied by a value that roughly indicates the percentage of consumed bandwidth. As a result, we obtain a receive window value that is proportional to the captured demand or smoothed arrival rate. This computation is performed using the following equation:

$$RW_{ij}(t + K_{ij}) = \frac{\hat{r}_{ij}(t + K_{ij})}{R_{ij}(t + K_{ij})}\, RW_{ij}(t)\,, \tag{10}$$

where $\hat{r}_{ij}(t+K_{ij})$ is the exponential weighted moving average of the arrival rates at time $(t + K_{ij})$. It is also important to note that, the use of the exponential moving average of all the past arrival rates also allows to capture irregular throughput variations during the categorization period (i.e., during K RTTs), even though less importance is accorded to very old throughput samples in our case. After each change of receive window, a certain amount of bandwidth is generally released, especially when the change is made after a flow is categorized/recategorize as slow. So, in order to fully exploit the radio capacity and increase the link utilization, the released bandwidth must be reallocated to fast flows which need more bandwidth unlike slow flows. For that, it is important to keep track of the released bandwidth both at flow level and at UE level. Basically, every time a flow is categorized as slow, the released bandwidth at flow level, denoted A_{ij}, is recorded by computing the difference between the previous and the new receive window. Then the available bandwidth at UE level, denoted A_{ij}, is immediately updated by computing the sum of all the released bandwidth from the slow flows belonging to the UE. The following two equations are used to compute the unused bandwidth at flow level, and at UE level, respectively:

$$A_{ij}(t + K_{ij}) = |RW_{ij}(t) - RW_{ij}(t + K_{ij})|\,, \tag{11}$$

$$A_{i}(t + K_{ij}) = \sum_{j=1}^{Z} A_{ij}(t + K_{ij})\,, \tag{12}$$

where A_{ij} is the available (unused) bandwidth at the flow level, A_i is the aggregated unused bandwidth at the UE level and Z is the number of identified slow flows. After these two operations, the available unused bandwidth is evenly distributed among the fast download flows of the UE. So, the receive window of each fast flow is immediately increased with an equal fraction of the available bandwidth as follows:

$$RW_{ij}(t + K_{ij}) = RW_{ij}(t) + \frac{A_i(t + K_{ij})}{N - Z}\,. \tag{13}$$

Equation 13 is also used whenever a flow is categorized as fast. It is important to note that, in this equation, unlike for slow flows, the $RW_{ij}(t)$ value is always recomputed as indicated in eq. 4.

For a given UE, the presented dynamic categorization and allocation operations are repeated in an infinite loop for each intercepted flow, so that the bandwidth allocated to this UE by the base station can be better distributed between its active flows, which naturally leads to a more refined radio link utilization. Also, it is important to note that, even if a fast flow is wrongly classified as a slow flow, it is still allocated a bandwidth that is proportional to its average arrival rate. Therefore, unless the flow has very irregular arrival rates, the misclassification should be automatically corrected in the next categorization round with almost no consequences on the flow's global performance. This is explained by the

fact that, if a misclassified flow is really a fast flow, this flow is very likely to reach at least 80% of its average rate at the end of the subsequent rounds, unless its arrival rates are very irregular, at which case it may take more than one round to correctly solve the misclassification. Indeed, with this method, the allocated radio bandwidth is adequately distributed among the concurrent flows regardless of the CCA in use. However, it is important to note that, even when they are provided with the same receive window values, CCAs with more aggressive growth functions yield higher link utilization than CCAs with very conservative growth since the former reach the target Receive Window faster.

3.5. RAPID's demand-aware fairness

The degree of fairness between two CCAs is typically measured by computing their Jain's Fairness Index (JFI) [26], which gives a number between 0 and 1. A JFI of 1 (i.e., the highest fairness degree), indicates that the two CCAs equally share the bottleneck bandwidth. However, equally sharing the bandwidth does not necessarily lead to real fairness since some CCAs, because of their design, may not be able to fully exploit their fair share, hence penalizing those with higher demands or potential to grow. Furthermore, as indicated in [27], JFI is known for being demand unaware, in the sense that it prohibits any flow from claiming the available bandwidth not used by other competing CCAs (e.g., because of their low demand or design). Therefore, in order to avoid these limitations, we design RAPID with a demand-aware bandwidth allocation goal in mind.

Basically, RAPID continuously monitors the demand of a flow by observing the arrival rate of the incoming packets (from this flow) through periodic categorization periods. The flow is then assigned a bandwidth that is proportional to the captured demand, but at the same time bounded by the available bandwidth and the number of concurrent flows. Upon the next categorization period, the flow is given a new opportunity to grow if the captured demand is not inferior to the previous one. Through this repetitive process, RAPID periodically gives each flow an opportunity to grow depending on its demand and allows sharing the unused bandwidth among the flows with high demands unlike an equal-rate or JFI-based bandwidth allocation scheme.

For instance, let us consider two concurrent flows sharing a 15 Mbps bottleneck link, the first one using Cubic, the second one using a dummy CCA that always sends 2 MSS per RTT. In such a configuration, although it is obvious that the second flow would greatly under-utilize the link, an equal-rate or JFI-based bandwidth allocation scheme would still allocate an equal share of the bandwidth to the two flows. In other words, the first flow would be penalized because of the impairments in the dummy CCA's design. In contrast, RAPID's demand-aware bandwidth allocation scheme would allocate just a small fraction of the available bandwidth to the dummy CCA. All the remaining bandwidth would then be used by the first flow since it has higher demands.

Table 1
Global simulation parameters

Parameters	Values
Carrier Freq.	28 GHz
Bandwidth	200 MHz
Numerology	3
RLC Mode	Ack. (AM)
gNB Height	10 m
UE Height	1.5 m
RLC Buffer	10 MB
MSS	1440 B
Initial Window	10 MSS
RAN Throughput	850 Mbps
Propagation Model	3GPP Urban-Micro (UMI) ns-3 PropagationLossModel
3GPP Channel Scenario	UMI-Street Canyon

Table 2
Mobility and blockage parameters

Parameters	Values
User speed	1 m/s
Propagation Model	mmWave3GPP Buildings PropagationLossModel
Number of buildings	8
Building sizes[2]	Av. Jean Médecin, Nice, France

4. Evaluation of RAPID with NS-3

In this section, we first describe the implementation of our proxy in ns-3 [28]. Then, we evaluate its efficiency under some relevant scenarios that reproduce the issues mentioned in Sections 1 and 2. We repeat each scenario 10 times and show the observed results with a 95% confidence interval.

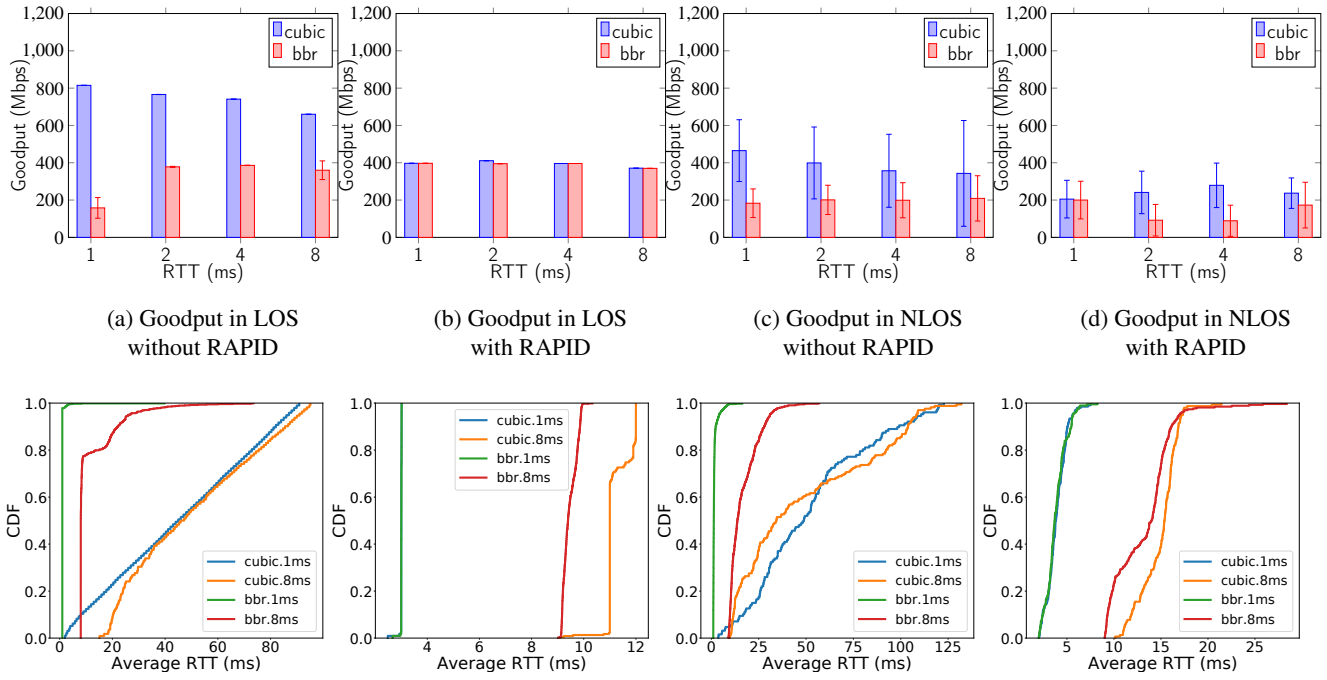
4.1. ns-3 experimentation testbed

The ns-3 mmWave module paves the way for the simulation of end-to-end 5G environments. Besides providing large bandwidths in the millimeter wave band for eMBB scenarios, it also enables the simulation of low-latency communications thanks to a customizable sub-carrier spacing (i.e., supporting various numerologies) in the RAN. With that in mind and in order to evaluate the performance of RAPID, we implement all the functional modules illustrated in Figure 2 in a proxy device collocated with a mmWave base station in ns-3. The proxy is connected to the core network via a 10 Gbps link in order to simulate a real-world fiber backhaul. Since Cubic and BBR are the dominant CCAs today [3], we simulate two TCP sources based on these algorithms as illustrated in Figure 3. Other simulation parameters are detailed in Tables 3 and 2.

4.2. MEC scenarios

In this paper, we focus on evaluating the performance of RAPID in mobile edge settings since it is more suitable

[2]Google Earth™ is used to estimate the size of buildings.



(a) Goodput in LOS without RAPID

(b) Goodput in LOS with RAPID

(c) Goodput in NLOS without RAPID

(d) Goodput in NLOS with RAPID

(e) RTT increase in LOS without RAPID

(f) RTT increase in LOS with RAPID

(g) RTT increase in NLOS without RAPID

(h) RTT increase in NLOS with RAPID

Figure 4: Fast eMBB downloads scenario.

for throughput-intensive ultra-low latency applications (e.g., AR/VR, tactile Internet, etc.) that require both high throughput (e.g., from 100 Mbps to a few Gbps) and low latency (e.g., from 1 to 10 ms) [16]. Furthermore, although several studies have been conducted on the evaluation of recent CCAs in 4G and 5G mmWave networks, to the best of our knowledge, there is no significant work on their evaluation in mobile edge and very low latency settings. Therefore, in our different experimentation, we consider only end-to-end RTT in the range of 1 to 10 ms.

In order to evaluate the performance of RAPID in different RTT and flow configurations, we consider a main single user scenario consisting of one UE receiving simultaneously several flows from different end servers. Depending on the characteristics of the concurrent flows (i.e., sheer download or slow/interactive downloads), such a basic scenario can highlight both the self-inflicted bufferbloat issue introduced by loss-based CCAs and the unfairness in terms of bandwidth when different CCAs compete in deep buffer environments. Based on our main scenario, we define two sub-scenarios that highlight the impact of RAPID's flow-level bandwidth allocation scheme on the global achievable performance in terms of goodput and delay:

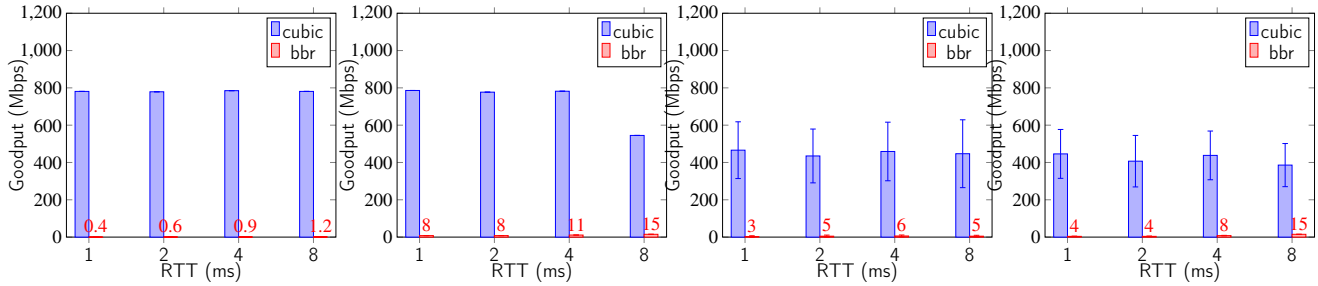
Fast eMBB downloads:[3]This first scenario aims to showcase the efficiency of RAPID in mitigating the interference between different CCAs (i.e., Cubic and BBR). To that end, the UE downloads simultaneously 200 MB from two TCP servers, one using Cubic and the other BBR, in both LOS and NLOS conditions. The obtained goodput and the delay variations are observed for each flow in various end-to-end RTT configurations.

Mixed Fast and App-limited/Slow downloads: In this scenario, we highlight the negative effects of self-inflicted bufferbloat on app-limited flows (i.e., slow flows). The UE receives simultaneously a continuous flow (e.g., streaming) that uses Cubic and a slow flow (e.g., web browsing) that uses BBR. The slow flow is paced at 16 Mbps in order to simulate a continuous web browsing activity with 2 MB page sizes and a Page Load Time (PLT) of 1 second, which corresponds to the limit of user's real-time perception [30]. We then evaluate the effect of bufferbloat on web browsing for different RTT values and radio conditions by monitoring the bandwidth share of the slow flow, which should remain around 16 Mbps for a reasonable real-time perception.

Note that the NLOS conditions during our experiments are simulated by reproducing the positions, sizes and heights of the buildings in Jean Medecin Avenue (a famous busy avenue in Nice, France).

Also, it is worth noting that, the reason we decided to stick with a single user, is simply because evaluating RAPID in a multiple-user scenario would not bring any additional information about its performance, since the users are already isolated from each other by the base station. Instead, such a scenario would just show how efficient is the base station in distributing the available radio resources among the users.

[3]The average web page size on the Internet is around 1.5 MB [29].

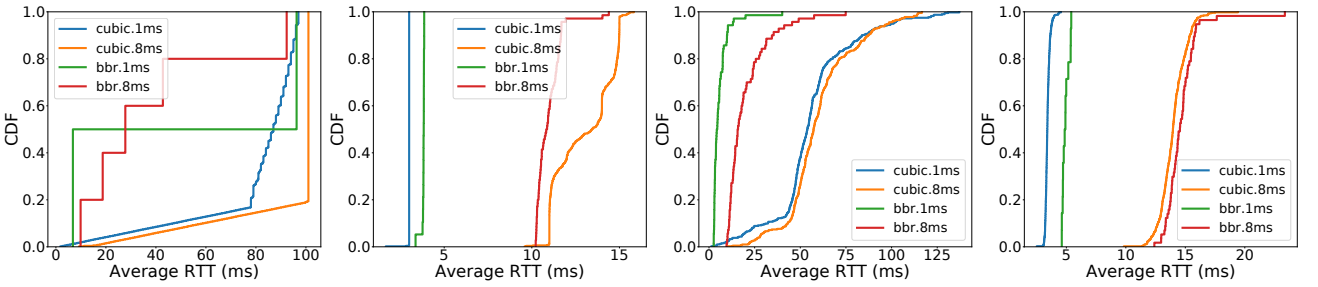


(a) Goodput in LOS without RAPID

(b) Goodput in LOS with RAPID

(c) Goodput in NLOS without RAPID

(d) Goodput in NLOS with RAPID



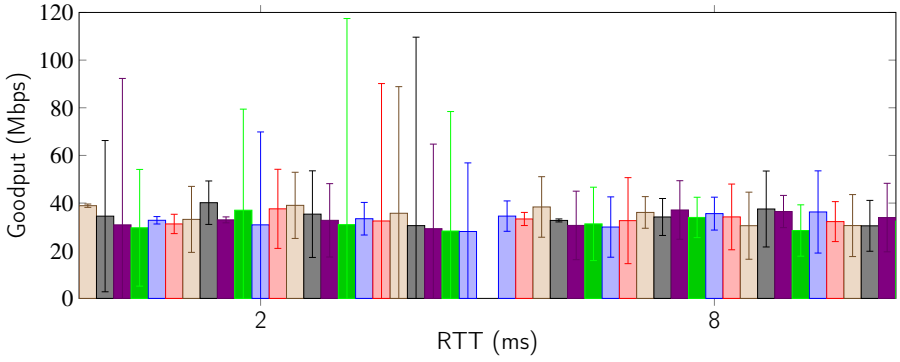
(e) RTT increase in LOS without RAPID

(f) RTT increase in LOS with RAPID

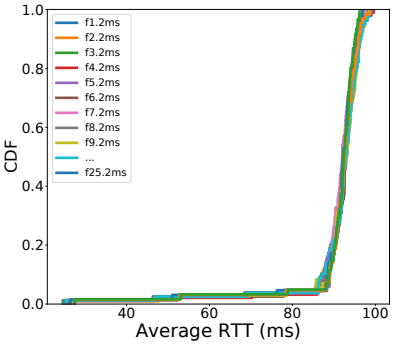
(g) RTT increase in NLOS without RAPID

(h) RTT increase in NLOS with RAPID

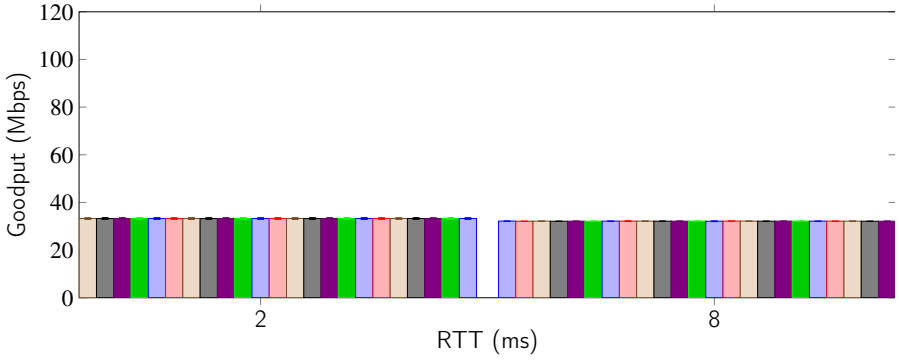
Figure 5: Mixed Fast and App-limited/Slow downloads scenario.



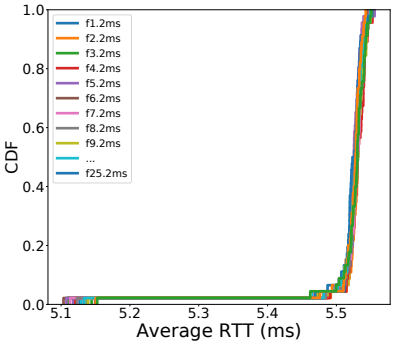
(a) Goodput distribution of 25 Cubic flows without RAPID



(b) Corresponding RTT increase without RAPID

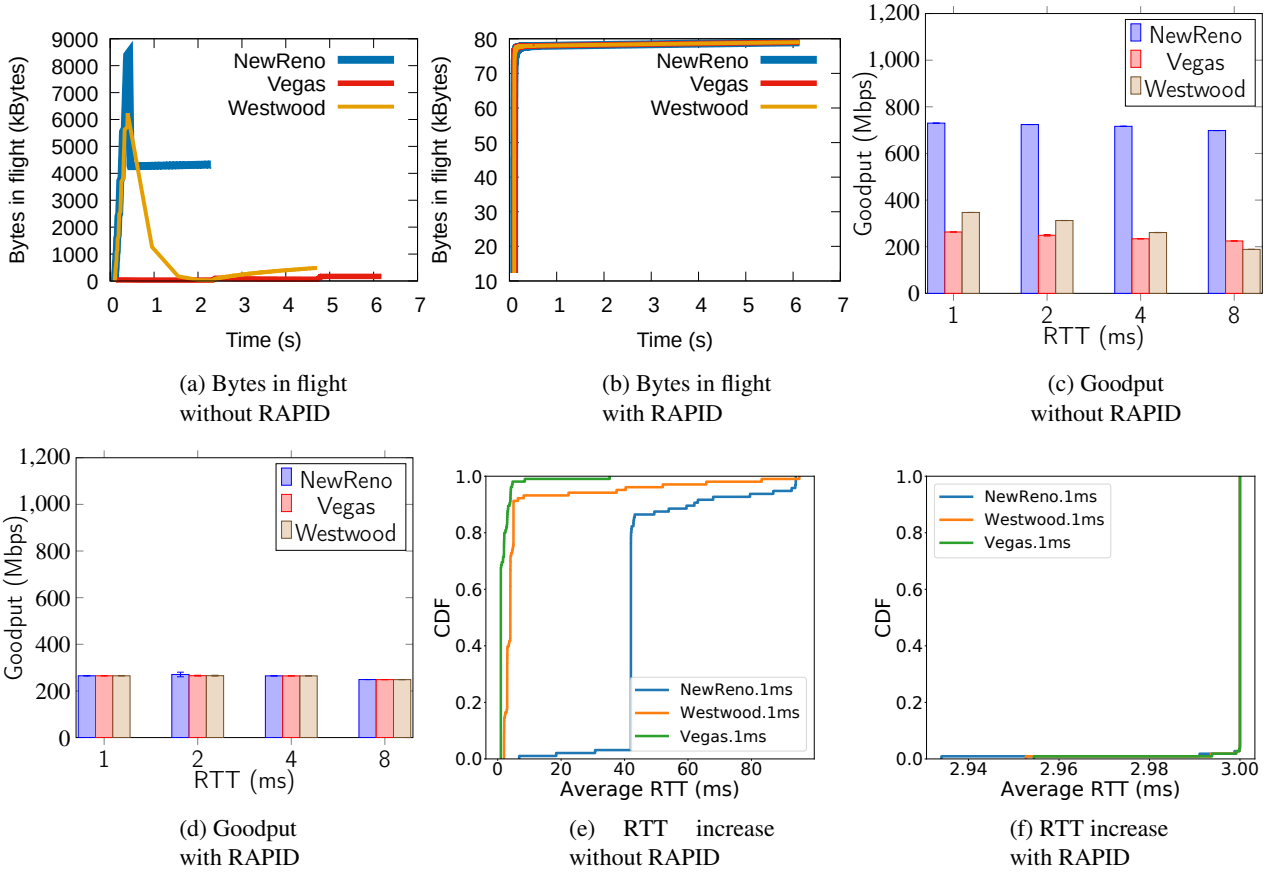


(c) Goodput distribution of 25 Cubic flows using RAPID



(d) Corresponding RTT increase with RAPID

Figure 6: Fast-download with 25 Cubic flows progressing in parallel in the same UE



(a) Bytes in flight without RAPID

(b) Bytes in flight with RAPID

(c) Goodput without RAPID

(d) Goodput with RAPID

(e) RTT increase without RAPID

(f) RTT increase with RAPID

Figure 7: Concurrent NewReno, Vegas and Westwood flows

4.3. Results

Figures 4a, 4c, 4e, 4g show a comparison of both goodput and end-end latency when RAPID is not used in the fast eMBB downloads scenario. At first, it can be seen that Cubic grabs more bandwidth than BBR in all the evaluated configurations regardless of the radio conditions. This outcome was indeed expected since several studies show that loss-based CCAs outperform BBR in terms of goodput in deep buffer environments [5, 31, 32]. Although most of these studies claim that the degree of fairness in this situation only depends on the bottleneck buffer size, our results show that the end-to-end RTT value greatly affects the bandwidth shares of the two CCAs in mobile edge settings. As illustrated in Figures 4a and 4c, the bandwidth share of BBR appears to increase as the end-to-end RTT increases while Cubic follows an opposite pattern. This phenomenon is simply due to the Hystart and startup phases of Cubic and BBR, respectively. Basically, the two CCAs always start with an exponential growth; however, in case of very small RTT (e.g., 1 ms), the BDP of the link is quickly reached. For instance, with the following parameters: initial congestion window = 10 MSS; RAN throughput = 850 Mbps and RTT = 1 ms, the bytes in flight for each flow exceed the BDP of the link just after 4 RTTs (i.e., more than 2 BDPs of inflight data is maintained). However, since BBR's bottleneck bandwidth estimation window is around six to ten RTTs by default [5], it greatly underestimates the available bandwidth due to the large accumulation of data at the RLC buffer introduced by Cubic (more than 5 BDPs of inflight data at 10 RTTs). On the other hand, as the end-to-end RTT increases, it takes more RTTs to reach the BDP of the link, which allows BBR to get a better estimate of the bottleneck bandwidth (i.e., before Cubic introduces large buffering) but causes Cubic to leave the exponential increase phase prematurely. In fact, Cubic's Hystart exits the slow start phase after a certain increase in RTT [33], meanwhile BBR's startup phase continues until it introduces more than two BDPs of inflight data [15]. Due to RTT increase, if BBR does not observe a lower minimum RTT (RTTmin) during a 10 seconds window, it enters a phase called probeRTT. In this phase the number of bytes in flight is reduced to 4 Maximum Segment Size (MSS) at least for 200 ms in order to capture the new RTTmin. During this period, Cubic continues to grab more bandwidth as BBR decreases its sending rate, which explains why loss-based CCAs eventually outperforms BBR in deep buffer environments. However, for short to medium flows that finish before BBR's probeRTT phase takes place (i.e., for download duration around 10 s), as we increase the end-to-end RTT, BBR bandwidth share increases (even in very deep buffer environments) while Cubic share decreases due to the Hystart behavior. It can be seen from Figures 4e and 4g that excessive buffering causes

a significant increase in RTT (from 100 to over 5000%) for both BBR and Cubic especially in case of NLOS conditions. Furthermore, it is important to note that if the bandwidth is shared equally between the two flows, they should complete around the same time since they are downloading the same amount of data (i.e., 200 MB). In other words, the combined download time should be equivalent to the time it takes to download 400 MB over a 850 Mbps link, which is normally around 4 seconds. However, Figure 4a clearly shows that the Cubic flow always completes earlier (since it exhibits higher goodput) while the BBR flow takes much longer. For instance, in the 1 ms RTT configuration, the BBR flow takes at least 8 seconds in order to download 200 MB, thereby, doubling the combined download duration of the two flows (i.e., from 4 to 8 seconds). Our subsequent experiments demonstrate that RAPID avoids this issue, allowing for a combined download time of 4 seconds instead of 8 seconds.

RAPID provides both flows with relatively fair bandwidth shares under LOS conditions as illustrated in Figure 4e and the RTT increase due to Cubic overshooting becomes almost negligible (see Figure 4f) thanks to RAN-aware flow control. However, in case of NLOS conditions, although RAPID reduces the self-inflicted bufferbloat by a factor of 50 as illustrated in Figure 4h, BBR is not able to fully exploit its allocated bandwidth share, as it is subject to throughput oscillations in case of fast variations in delay [16]. Figures 5a through 5h show the test results for the mixed fast/app-limited downloads scenario. In the cases where RAPID is not used (Figures 5a, 5c), it can be seen that the app-limited flow is unable to obtain its desired bandwidth share (i.e., 16 Mbps) due to the fast download flow that overshoots more than the BDP. As a result, the slow flow is limited at less than 1 Mbps in LOS conditions and around 5 Mbps in NLOS. The increase in BBR goodput in case of NLOS is simply due to Cubic's sending rate decrease caused by packet losses. In fact, even after Cubic halves its congestion window, the goodput achieved by the slow flow is still 3 times less than the desired goodput. As a result, the PLT for a 2 MB web page is around 3 to 16 seconds. Moreover, as depicted in Figure 5e, over 8000% end-to-end RTT increase can be observed for the slow flow in some cases. Such a large delay increase is unacceptable both for traditional web browsing and throughput-intensive ultra-low latency applications.
On the other hand, in the cases where RAPID is used (Figures 5b and 5d), the app-limited flow is allocated on average 11 Mbps in LOS and 8 Mbps in NLOS. Furthermore, unlike in the first scenario where RAPID equally shares the available bandwidth between the two download flows, here, the slow flow is automatically detected thanks to Equation 8 and is allocated a Receive Window proportional to its average data rate using Equation 10. As shown in Figures 5b and 5d, this mechanism allows the fast download flow to exploit all the available bandwidth not used by the app-limited flow while remaining bounded by the actual RAN capacity. This not only improves the link utilization but also reduces the RTT increase for both flows by a factor of 10 to 50. Overall, thanks to RAPID flow categorization and bandwidth allocation schemes, the PLT is reduced by 90% (i.e., from 16 s to 1.5 s).

Besides evaluating RAPID's behavior under basic network conditions, we also consider other parameters that may affect its performance in real-world deployments:

Scalability: In commercial mobile networks, there is no limitations on the number of flows a user can maintain in parallel. Therefore, in order to be efficient in such networks, RAPID must exhibit good performances regardless of the number of active flows per user. Figure 6 illustrates this behavior by showing the bandwidth allocation and delay distribution of 25 Cubic flows progressing in parallel in the same UE for both 2 ms and 8 ms end-to-end RTT configurations. From this figure, it can be seen that, when RAPID is not used, a large degree of variability is observed between the goodput of the Cubic flows. This is due to packet losses (from an RLC buffer overflow) occurring at different moment at each new run. On the other hand, with RAPID, each flow gets the same goodput and the large variability is not observed anymore, simply because, by making sure the available radio capacity is never exceeded, RAPID completely prevents packet losses that arise from an RLC buffer overflow.

Packet loss: In most cases and as Equation 7 assumes, packet loss in the backhaul network is generally negligible. This owes to the fact that, in commercial deployments, backhaul link dimensioning is done in such a way that the peak data rate or at least the average data rate of the cell is supported [34]. After all, if the backhaul link limits the cell data rate, there would be no point for the operator to invest in large frequency spectrum. On the other hand, packet loss may still occur in the RAN segment because of bad radio conditions or buffer overflow. The latter is avoided when using the RAPID's flow control mechanism while the former is handled by RAPID's simplified CCA which always adapts its sending rate to the available RAN capacity regardless of random losses.

For space limitation reasons, we mainly focus in this paper on scenarios involving Cubic and BBR as they are currently the two main CCAs in use on the Internet. Nevertheless, we have also evaluated RAPID with other well-known CCAs such as NewReno, Vegas [35], Yeah [36] and Westwood [37] that still control a large portion of today's Internet traffic. For instance, a subset of these additional experiments are shown in Figure 7 where we evaluate a simple Fast-Download scenario in LOS condition involving NewReno, Vegas and Westwood in a 1 ms RTT configuration. As expected, our experiments show that RAPID allows delay-based TCP variants like Vegas (known for their very poor performances when competing with loss-based CCAs) to achieve similar goodput as NewReno or Westwood, while avoiding high delay increase. Figures 7a and 7b show, for a single run, the bytes in flight of the 3 flows as they progress in parallel on the same UE. It can be seen from Figure 7d that RAPID equally shares the total bandwidth (i.e., 850 Mbps) between the 3 flows. All the flows get a similar completion time (around 6 s) as illustrated in Figure 7b and the overall

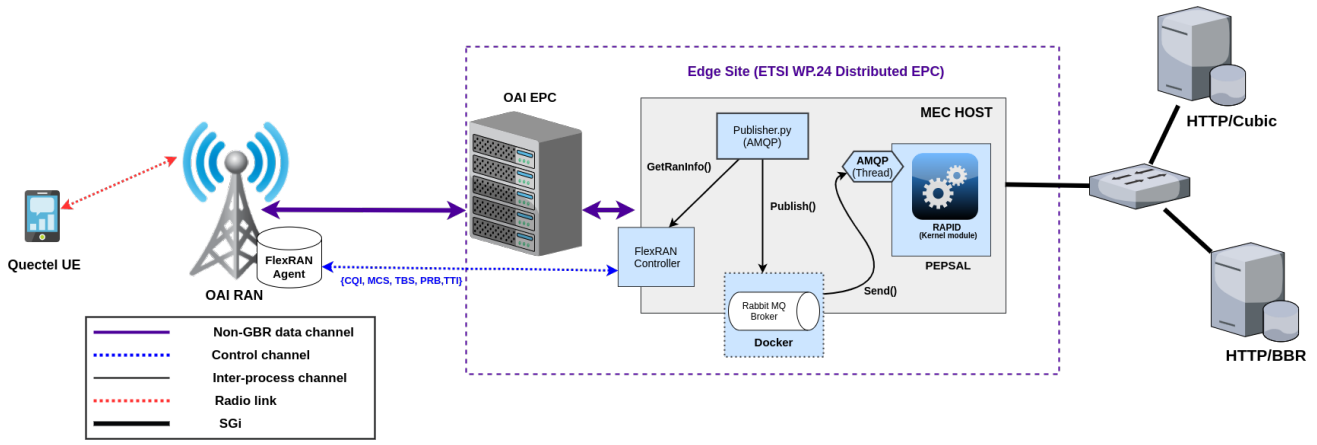


Figure 8: RAPID OAI experimentation testbed

delay increase is reduced by a factor of 10 to 90, see Figures 7e and 7f. As illustrated in Figure 7a, without RAPID, NewReno gets by default the lowest completion time (around 2s) because of its aggressive congestion growth function. In other words, it always maintains a high number of bytes in flight which negatively impacts the other competing flows, and in particular Vegas, as the latter is delay-based. As a result, NewReno always grabs a larger share of the total bandwidth (see Figure 7c) at the cost of over 7000% increase in delay (i.e. from 1 ms to 80 ms) and with a high degree of unfairness to the other flows. Thanks to RAPID, a certain degree of fairness is maintained between the competing flows. Despite their different designs, all the 3 CCAs are imposed a similar number of bytes in flight, which allows them to obtain the same share and to finish approximately at the same time, as illustrated in Figures 7b and 7d. Note that other additional scenarios involving the aforementioned CCAs as well as the results shown in this section can be reproduced with ns-3 using the codes and scripts that we have made publicly available [12].

4.4. Bandwidth overhead

To work properly, RAPID requires an out-of-band control channel in the backhaul as shown in Figure 2. This control channel is necessary in order to allow the continuous retrieval of radio information, therefore, its requirements in terms of bandwidth depend both on the size of the retrieved radio information and the configured polling period. In our simulations, we use a polling period at subframe granularity (i.e., radio information is fetched every ms) and the size of the retrieved information per UE including TCP and IP headers is around 170 bytes. In other words, each UE introduces an overhead of 1 to 1.5 Mbps in the control channel, which is close to the FlexRAN overhead in real world deployments [20]. We believe that such an overhead that accounts only for 0.13% of the achievable throughput (i.e., 850 Mbps in our case) and 10^{-8}% of a typical 10 Gbps backhaul, is negligible compared to the resulting 150% and 3600% BBR's goodput increase in fast-download and web browsing scenarios, respectively (see Figures 4b and 5b) as well as the overall end-to-end delay reduction in the range of 80 to 97%, see Figures 4h and 5h.

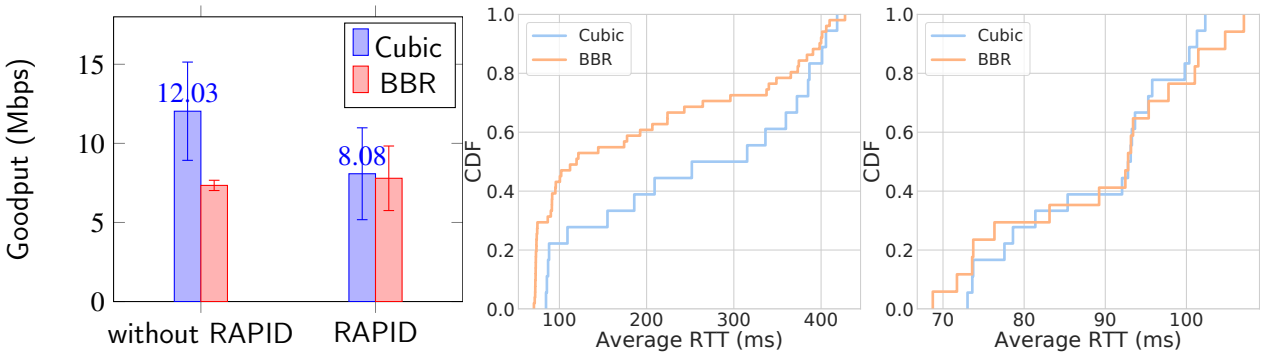
4.5. Discussion

Our simulation results demonstrate that RAPID significantly mitigates the self-inflicted bufferbloat issue in mobile networks, while preserving near optimal link utilization (Figures 4b, 5b). The idea is to impose a certain level of fairness on the concurrent flows depending on their nature and on the available radio bandwidth. However, even if all the flows are assigned the same Receive Window, their performance in terms of goodput and link utilization still depends on the design of their respective CCA. Indeed, depending on their growth functions, different CCAs will reach a given number of bytes in flight at different times. Therefore, CCAs with faster growth functions will yield better link utilization. In addition to that, it is worth noting that upcoming 5G and 6G networks are expected to deliver throughput in the range of several gigabits per second. So, with the design and growth functions of the current CCAs, it would be virtually impossible to reach such a high throughput in a couple of RTTs. Therefore, it becomes necessary to rethink congestion control approaches in order to maintain high link utilization with minimum latency, which is very challenging as indicated in [38], mostly because of the highly intermittent nature of mmWave capacity.

In such a situation, we believe that providing flow-level RTT control and bufferbloat mitigation with proxy-based solutions such as RAPID will significantly simplify the design of future CCAs by allowing them to focus only on fast convergence.

5. Evaluation of RAPID with OAI

In this section we first describe the design and implementation of our proxy as a Linux kernel module based on well-known open-source projects. Then, we showcase its efficiency when integrated into a 4G OAI infrastructure by conducting various experiments. Finally we discuss the observed limitations and propose some workarounds.



(a) Bandwidth allocation (b) RTT increase without RAPID (c) RTT increase with RAPID

Figure 9: Goodput and RTT increase in OAI-4G with and without RAPID

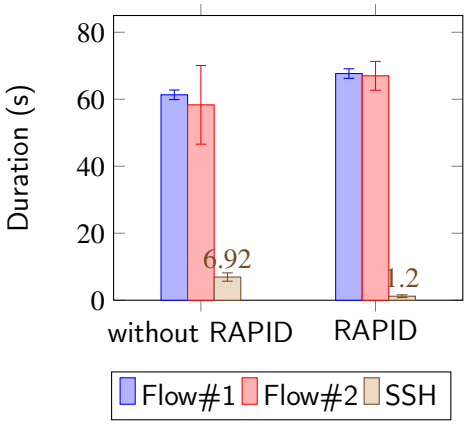


Figure 10: Short flow duration during fast download

5.1. Design and implementation

All the software components used in the implementation of RAPID as well as the way they interact with each other are illustrated in Figure 8. Basically, our Linux implementation is built on top of two open-source projects, namely pepsal [39] and FlexRAN [20]. Pepsal is an open-source Performance Enhancing Proxy (PEP) that splits incoming TCP connections, as such, it naturally replaces RAPID's TCP split functional module (illustrated in Figure 2). FlexRAN on the other hand brings the Radio Network Information Service (RNIS), which is exploited via Advanced Message Queuing Protocol (AMQP) so that the pepsal process can periodically receive real-time radio information. The latter information is then cleaned and sent via ioctl to the RAPID kernel module which enforces the per-flow bandwidth allocation mechanism on the intercepted TCP connections.

5.2. Experimentation and results

The experimentation setup illustrated in Figure 8 shows how we deploy RAPID along with an OAI 4G infrastructure. Also, it is important to note that this deployment method is by default compliant with any commercial 4G/5G networks. While our architecture is based on the "ETSI WP.24 Distributed EPC approach" [40], where the Evolved Packet Core (EPC) is collocated with the MEC host at the edge site, it is worth noting that RAPID can also be evaluated following the "Bump in the wire approach" [40], in which the MEC platform is located between the base station and the mobile core. Table 3 describes the characteristics of our experimentation setup in the R2lab anechoic chamber. With these parameters, we assess the following key aspects of RAPID using *wget* and *iperf* generated traffic:

Table 3
OpenAir experimentation parameters

Parameters	Values
LTE mode	FDD
LTE bandwidth	5 MHz
Number of PRBs	25
RLC AM/UM buffer size	1 MB
UE	5G Quectel RM500Q-GL
End-to-End RTT	50 ms
Max. RAN Capacity	16 Mbps
Backhaul/s1 BW.	1 Gbps

Demand awareness and RTT reduction: We validate the efficiency of these features by reproducing the two scenarios presented in Section 4 under LOS conditions. The Fast eMBB Download is reproduced by launching two *wget* traffic, each simultaneously downloading a 60 MB file from two different HTTP servers that respectively use Cubic and BBR as CCAs. Similarly, the second scenario is reproduced by running two *wget* traffic in parallel using Cubic, followed later by a short SSH connection under BBR which displays a 500 kB text file using the Linux *cat* command. Figure 9 and 10 shows the results of these experiments for 10 successive runs with 95% confidence interval.

In the first scenario, we observe that RAPID provides as expected a high level of fairness between Cubic and BBR despite the respective differences of the two CCAs. In the normal OAI deployment (i.e., without RAPID), Cubic grabs almost all the available bandwidth aggressively while BBR exploits only the remaining 25% (at least until the Cubic flow ends) as illustrated in Figure 9a. This causes the average RTT

to grow from 50ms to over 400 ms. Such a high increase in RTT not only penalizes BBR in terms of goodput, but also in terms of delay since the BBR flow exhibits average RTT values around 300 ms in the 80th percentile (see Figure 9b). On the other hand, when RAPID is deployed along with OAI, it can be observed from Figures 9a and 9c that Cubic and BBR achieve an average goodput of 8 and 7.8 Mbps respectively while maintaining RTTs below 100 ms in the 90th percentile. These results show that RAPID can allow in sheer download scenarios, over 4 times or 75% RTT reduction in a real-world 4G network while maintaining a fair bandwidth allocation regardless of the differences between the concurrent CCAs.

In the second scenario, from the results illustrated in Figure 10, we observe that the short SSH traffic takes on average 7 seconds in the standard deployment while it only takes 1 second with RAPID. In other words, RAPID enables 85% or 7 times reduction in the short flow completion time. This is in fact possible thanks to the low buffer occupancy resulting from RAPID's bandwidth allocation scheme and the demand awareness feature which allows RAPID to allocate enough bandwidth to the SSH flow during its lifetime. This experiment demonstrates that RAPID is effective in real-life scenarios where the user maintains short and long flows at the same time. For instance a user may want to surf on social media while a heavy software update is progressing in the background.

Impact of background UDP traffic: RAPID has been designed to only intercept TCP traffic, therefore, protocols based on UDP such as QUIC or RTP are not intercepted and can affect RAPID's per-flow bandwidth allocation scheme. As described in Section 3, RAPID estimates the demand of a flow by using successive categorization periods upon which the average arrival rate of incoming packets is compared to the expected throughput (i.e., estimated radio bandwidth divided by the number of connections). It is important to detail the three reasons that can prevent the flow from reaching this expected throughput: (1) the flow has a low demand (e.g., Web browsing flow); (2) the capacity of the backhaul is less than the current radio capacity allocated to the UE; (3) one or several concurrent data traffic (most likely unintercepted) exceed the capacity of the backhaul link. Suppose we are using a sheer download TCP flow, in this case we can eliminate (1) but also (2) since this condition is never met in commercial networks [34]. Likewise, the likelihood of observing (3) is close to zero in case of Constant-Bit-Rate (CBR) UDP flows such as Voice over IP (VoIP) traffic since their requirements in terms of bandwidth are very low (e.g., from 64 to few hundreds *kbps* [41]). Such flows would even benefit from a significant decrease in delay and jitters thanks to the very low buffer occupancy enabled by RAPID. However, UDP flows that mimic the behavior of TCP such as QUIC could potentially overshoot the backhaul link depending on their CCAs. In such a situation, RAPID would continue to maintain a radio BDP worth of bytes in flight as long as the concurrent QUIC flow does not exceed the BDP of the backhaul. Once this occurs, RAPID would reduce the bandwidth allocated to the TCP flow (at the end of each categorization period) since it would no longer be able to reach the expected arrival rate.

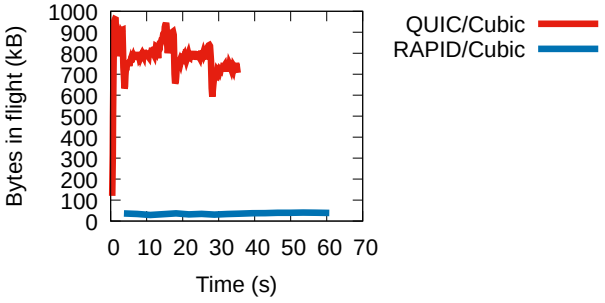
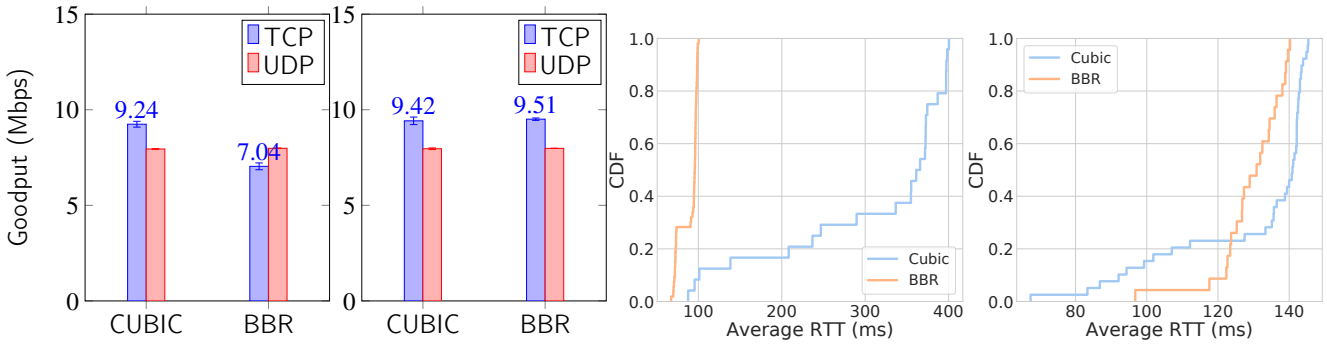


Figure 11: RAPID/Cubic competing with QUIC/Cubic

RAPID would eventually increase the bandwidth allocation of the TCP flow when the QUIC flow backs off due to a buffer-overflow or an RTO expiration. Figure 11 illustrates this mechanism in case of two competing TCP and QUIC flows using both a loss-based CCA. As shown in this figure, RAPID forces the TCP flow to operate around the radio BDP after the backhaul BDP is exceeded. Such a behavior allows the TCP flow to grab its fair share on average at the cost of delay increase, even when competing with aggressive long-lasting QUIC flows. Furthermore, it is worth noting that the latter explanation holds true only for long-lasting QUIC flows using aggressive CCAs. In cases where the competing QUIC flows use a less aggressive or delay-aware CCA such as BBR, RAPID allows the intercepted TCP flow to outperform QUIC both in terms of goodput and delay. In fact, this second scenario is the most likely to occur on the Internet since Google plans to use BBR on all its QUIC and TCP traffic [42].

Thus, in order to validate the aforementioned behaviors, we evaluate RAPID's performance under constant bit rate UDP traffic and also under loss-based (i.e., Cubic) and model-based (i.e., BBR) QUIC traffic. For the CBR UDP scenarios we rely on *iperf* in order to generate an 8 Mbps UDP traffic competing with a *wget*-generated TCP flow and then we run another experiment which involves a 150 kbps UDP traffic also competing with a *wget*-generated TCP flow. We then modify the CCA used by the TCP flow in order to observe the impact of the CBR flow on Cubic and BBR. For the scenarios involving QUIC, we rely on picoquic in order to generate QUIC flows that use Cubic and BBR. The goal of these experiments is to show that RAPID performs well both under aggressive and low bit-rate UDP traffic. This allows CBR traffic such as VoIP or streaming flows to run successfully alongside any TCP flow without experiencing a significant increase in delay and jitter. In fact, VoIP flows are considered of best quality only if the one-way latencies are below 130 ms [43]. Our experiments show that this threshold is exceeded in the wild when an aggressive TCP download is progressing in parallel. More specifically, we observe the



(a) UDP vs TCP without RAPID

(b) UDP vs TCP with RAPID

(c) RTT increase without RAPID

(d) RTT increase with RAPID

Figure 12: Goodput and RTT in OAI-4G under background UDP traffic

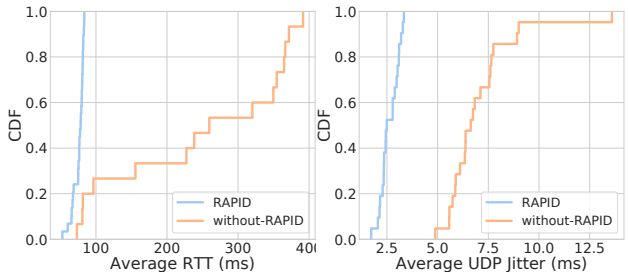


Figure 13: Average RTT and UDP Jitter under 150 kbps CBR

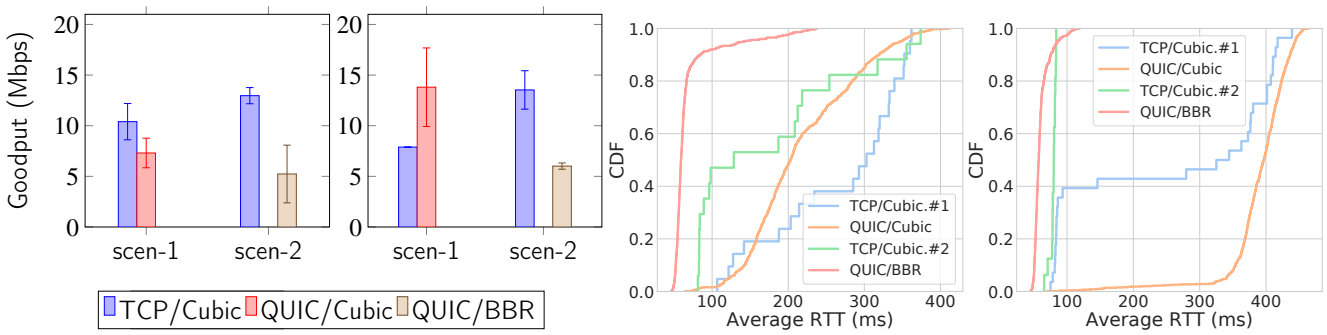
following phenomenons when a background UDP traffic is introduced.

First, for the 8 Mbps CBR traffic, we observe several performance issues in the standard OAI deployment (strongly driven by the CCA in use). As illustrated in Figure 12a, when RAPID is not used, we observe that the TCP flow exceeds its 8 Mbps fair share when competing with an 8 Mbps UDP flow, which, in case of a loss-based CCA indicates an aggressive growth behavior. This situation causes a tremendous RTT increase, from 50 to over 350 ms (as shown in Figure 12c), which, naturally hinders the quality of any concurrent VoIP traffic. On the other hand, for the same experiment, BBR shows a relatively lower performance in terms of goodput. It can be seen from Figure 12a that the BBR flow cannot exceed 7 Mbps on average even though the UDP flow is limited at 8 Mbps. In other words, BBR is not able to exploit the remaining 1 Mbps not used by the UDP flow. While this behavior prevents delay increase as shown in Figure 12c, it also enforces low link utilization since over 12% of the remaining bandwidth is not exploited. This low link utilization issue is actually due to incorrect BBR measurements of RTTmin in the OAI environment. Since BBR is prone to "throughput collapse" when the delay variation is high or when the minimum RTT estimate is too small [16], the observed low throughput was indeed expected from the fast-varying radio environment of OAI.

However, after enabling RAPID, we observe significant improvements for both CCAs. As expected, RAPID allows Cubic and BBR to fully exploit the remaining bandwidth at the cost of a relatively small delay increase as shown in Figures 12b and 12d. While Cubic benefits from over a five time decrease in delay, BBR exhibits both low delay and over 35% increase in goodput. Unlike the previous experiment, here BBR is not affected by the fast delay variations in the radio segment. In fact, thanks to the TCP split feature, the BBR control loop remains in the wired segment, i.e., between RAPID and the end-server which allows BBR to get correct RTTmin estimates. Furthermore, the congestion window in the radio segment, computed based on the estimated BDP (from on eq. 10 or 13) prevents large buffering at the base station. These results demonstrate that RAPID can share the available bandwidth with a UDP traffic constantly exploiting half of the bandwidth without introducing a considerable delay increase.

Similarly, the results obtained from the 150 kbps CBR experiment, illustrated in Figures 13 and 13, show that RAPID enables significant decrease in delay and jitter for time-sensitive flows. Besides allowing the UDP flow to reach its required goodput (i.e., 150 kbps), over six times jitter reduction and 5 times delay reduction are observed. We owe it to the low buffer occupancy enforced by RAPID on the intercepted flow, thus preventing the aggressive TCP download from degrading the performance of the delay-sensitive UDP flow, which can be seen here as a basic VoIP flow.

The experiments with background QUIC traffic show that RAPID's intercepted TCP flow grabs at least its fair share of the bandwidth in the best case scenario (i.e., when the QUIC flow is not aggressive), and manage to exploit at least 30% of the bandwidth in the worst case scenario (i.e., when the QUIC flow is too aggressive). On the other hand, as shown in Figures 14a and 14b, the QUIC flow exhibits overall poor performance when it uses BBR, whether with or without RAPID.This performance penalty was expected given BBR's default behavior in response to delay increase or decrease in the network delivery rate. In fact, from a design point of view, the main goal of BBR is to operate near



(a) QUIC vs TCP without RAPID

(b) QUIC vs TCP with RAPID

(c) RTT increase without RAPID

(d) RTT increase with RAPID

Figure 14: Goodput and RTT under Cubic/BBR QUIC traffic

the Kleinrock operating point, i.e., making sure the sending rate matches the bottleneck delivery rate and the RTT stays around the minimum RTT value. As a result, when a BBR flow shares the bottleneck with another flow that has already filled the bottleneck pipe, the BBR flow observes a delay increase. Therefore, it decreases its sending rate as an attempt to match the reduced delivery rate and to rediscover the minimum RTT value. Basically, in such a situation, the goodput that can be achieved by the BBR flow depends on how much delay or degree of buffering the concurrent flow can introduce. This phenomenon is well reflected on Figures 14a and 14b, from which it can be seen that the goodput achieved by the QUIC/BBR flow when RAPID is used to limit the buffering introduced by the TCP flow is slightly higher than the goodput it achieves when RAPID is not used, i.e., when the TCP flow introduces much more buffering. In any case, it is important to note that the BBR flow does not adopt such a behavior when it is intercepted by RAPID. It rather achieves roughly the same goodput as Cubic (see Figure 9a), because RAPID always makes sure the bottleneck capacity is adequately distributed between all the intercepted flows. However in this scenario, RAPID is completely unaware of the QUIC flow. As a result, RAPID enforces the intercepted TCP flow to always operate at the radio BDP, regardless of the CCA used by the QUIC flow. Because of this, as illustrated in Figure 14b, the intercepted TCP flow gets lower goodput when the QUIC flow sends more than the radio BDP (i.e., in case of QUIC/Cubic) and gets higher goodput when the QUIC flow sends less than the radio BDP (i.e., in case of QUIC/BBR). On the other hand, in terms of overall delay increase, we observe that the intercepted TCP flow starts experiencing considerable delay increase as soon as the QUIC flow exceeds the bandwidth of the RAN. The delay continues to grow as the QUIC flow increases the amount of byte in flight. In other words, the overall delay increase becomes less significant as soon as the QUIC flow stops. This is well illustrated in Figures 14c and 14d, where a significant decrease in delay is observed at lower percentiles (e.g., 40th) for the intercepted TCP flow competing with QUIC/Cubic (i.e., for TCP/Cubic.#2). It is important to note that RAPID always tries to minimize its delay increase, even without being explicitly aware of the background QUIC traffic. This behavior is illustrated in Figure 11, where it can be observed that the amount of bytes in flight for the intercepted TCP flow decreases as the QUIC flow increases its sending rate. Overall, these results demonstrate that RAPID does not penalize QUIC flows (although they are not intercepted), but rather increases the performance of competing TCP flows.

5.3. Limitations

Despite showing overall better performance with respect to standard TCP (i.e., end-to-end TCP), RAPID presents a minor limitation due to the inherent characteristics of real-world radio environments. More specifically, the random and fast delay variations in the radio segment makes tricky the measurement of the minimum RTT between the mobile and the proxy. We suspect that RTTmin is underestimated for most flows due to either low traffic load at the beginning of the connection, which is a well-known characteristic of TCP Slow Start algorithms, or because of the periodicity of the Scheduling Request (SR), which tells how often the base station allocates uplink resources to the UE. Since we use RTTmin in the computation of the flow BDP in the radio segment, the related flows suffer from low link utilization.

In order to mitigate this issue, we use a smoothed RTTmin value that takes into account the minimum RTT estimated during each categorization period. With this method, the very low RTTmin values observed at the beginning of the connection are discarded and the RTTmin estimate gets better as the connection continues. We also set a relatively high value for the initial congestion window on the RAN segment so that RAPID can quickly reach the highest allowed CWND value (i.e., the estimation of the RAN BDP) and directly deliver all the packets received from the original server. However, even though the smoothed RTTmin mechanism significantly improves the link utilization, we can still observe a slight bandwidth wastage as shown in Figure 9a, which approximately accounts for 1% of the overall capacity (i.e., around 150 kbps).

6. Related work

Authors in [7] proposed milliProxy, a performance enhancing proxy for 5G mmWave scenarios that regulates the sending rate using a variable Flow Window (FW). The FW is computed using the estimated RAN data rate (derived from channel quality information), the RLC buffer occupancy and the estimated RTT. Although milliProxy significantly mitigates bufferbloat in case of single flow, its FW computation scheme is not adapted for multi-flow scenarios.

In-band throughput guidance (TG) was introduced in [8]. This approach allows a functional element (TG provider) residing in the RAN to include RAN throughput information in the TCP header. This throughput information can then be exploited by a TCP server to regulate its sending rate. However, the authors indicate that, in case of multiple flows per bearer, each flow is assigned an equal share of the bearer capacity, which is not sufficient to ensure fairness since flows may have different requirements.

LCTCP (Link-Coupled TCP) proposed in [44] for 5G networks relies on a out-of-band signaling of queue occupancy information between the base station and the end server. The transmitted information is used by the server to adjust its sending rate so that the amount of packets queued at the base station is always aligned with the throughput and delay requirements of the application. However, although LCTCP outperforms conventional CCAs in some scenarios, it cannot deliver both high throughput and low delay simultaneously.

The authors in [45] proposed a TCP proxy for multi-connectivity enabled 5G mmWave network that speeds up congestion window growth and prevents RLC buffer overflow. The former is achieved by separating congestion events of the radio from wired segments while the latter relies on receive window modifications based on an estimation of the available proxy buffer space. Despite improving the download time in some scenarios, the proposed scheme is not adapted for bufferbloat mitigation. Indeed, the proxy buffer size is static and does not consider the actual RAN state, which could result in large delay increase especially in the mmWave band.

A proxy-based flow aggregation TCP proxy for GPRS was proposed in [46]. The solution proposed relies on a static estimation of the bottleneck BDP, which is not adequate for cellular networks because of the highly varying nature of the radio conditions.

7. Conclusion

In this paper we highlight the limits of per-user flow isolation in mitigating self-inflicted bufferbloat issues in case of multiple flows per user and intermittent mmWave links. By using up-to-date RAN statistics from MEC Radio Network Information Service and analyzing the behavior of active flows, the RAPID solution prevents CCAs from exceeding the RAN capacity, thus, drastically reducing the delay increase regardless of the radio conditions while preserving good link utilization.

Our experiments with 4G OAI validate the simulation results and show that RAPID not only offers significant performance improvement in MEC cellular networks, but also mitigates BBR's limitations in fast-varying radio environments, without requiring any modifications or software patches at the server. We also demonstrate that our solution does not penalize QUIC flows, although they represent only 7% of today's Internet traffic (according to Google [47]). Through our results we show that RAPID enables traditional loss-based CCAs such as Cubic or Reno, which are known to introduce high delay increase in deep buffer environments, to outperform BBR in terms of goodput while maintaining comparable and even lower delays in certain scenarios.

To the best of our knowledge, RAPID is the first attempt to mitigate self-inflicted bufferbloat and interference between CCAs at the flow level. We plan to further assess its performance over a real 5G deployment in the near future.

References

[1] H. Haile, et al., End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks, Computer Networks 186 (2021).

[2] J. Gettys, Bufferbloat: Dark buffers in the internet, IEEE Internet Computing 15 (2011) 96–96.

[3] A. Mishra, et al., The great internet TCP congestion control census, ACM POMACS 3 (2019) 1–24.

[4] H. Jiang, et al., Tackling bufferbloat in 3G/4G networks, in: ACM IMC, 2012, pp. 329–342.

[5] N. Cardwell, et al., BBR: Congestion-Based Congestion Control, ACM Queue 14 (2016) 20 – 53.

[6] Y. Xie, et al., PBE-CC: Congestion Control via Endpoint-Centric, Physical-Layer Bandwidth Measurements, in: ACM SIGCOMM, 2020, p. 451–464.

[7] M. Polese, et al., milliProxy: A TCP proxy architecture for 5G mmWave cellular systems, in: ACSSC 2017, 2017, pp. 951–957.

[8] A. Jain, et al., Mobile throughput guidance inband signaling protocol, 2015, pp. 1–16.

[9] M. Diarra, W. Dabbous, A. Ismail, T. Turletti, Ran-aware proxy-based flow control for high throughput and low delay embb, in: Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2021, pp. 41–50.

[10] Z. Zhong, et al., Performance evaluation of CQIC and TCP BBR in mobile network, in: Proc. of ICIN, 2018, pp. 1–5.

[11] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, C. Bonnet, Openairinterface: A flexible platform for 5g research, ACM SIGCOMM Computer Communication Review 44 (2014) 33–38.

[12] M. Diarra, et al., RAPID, 2021. URL: https://github.com/madi223.

[13] R. Robert, et al., Behaviour of common TCP variants over LTE, in: GLOBECOM 2016, 2016, pp. 1–7.

[14] J. K. Nurminen, Parallel connections and their effect on the battery consumption of a mobile phone, in: IEEE CCNC 2010, 2010, pp. 1–5.

[15] F. Chiariotti, et al., BBR-S: A low-latency BBR modification for fast-varying connections, IEEE Access 9 (2021) 76364–76378.

[16] R. Kumar, et al., TCP BBR for Ultra-Low Latency Networking: Challenges, Analysis, and Solutions, in: IFIP Networking, 2019, pp. 1–9.

[17] Z. Wang, et al., Why are web browsers slow on smartphones?, in: ACM HotMobile, New York, NY, USA, 2011, p. 91–96.

[18] M. Zhang, et al., Transport layer performance in 5g mmwave cellular, in: 2016 IEEE INFOCOM WKSHPS, 2016, pp. 730–735.

[19] ETSI, Multi-access edge computing (mec); radio network information api, in: GS MEC 012 V2.1.1, 2019.

[20] X. Foukas, et al., FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks, in: ACM CoNEXT, 2016.
[21] D. Sabella, et al., A Hierarchical MEC Architecture: Experimenting the RAVEN Use-Case, in: IEEE VTC Spring, 2018.
[22] M. Diarra, et al., Cross-layer loss discrimination algorithms for MEC in 4G networks, in: Proc. of IEEE HPSR '21, 2021.
[23] S. Monikandan, et al., A review of mac scheduling algorithms in lte system, Int. J. Adv. Sci. Eng. Inf. Technol 3 (2017) 1056–1068.
[24] 3GPP, LTE Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures, in: TS 36.213 version 15.2.0, 2018.
[25] 3GPP, 5G; NR; Physical layer procedures for data, in: TS 38.214 version 16.2.0, 2020.
[26] R. Jain, et al., A quantitative measure of fairness and discrimination for resource allocation in shared computer systems, DEC Research Report TR-301 (1984).
[27] R. Ware, et al., Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms, in: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 17–24. URL: https://doi.org/10.1145/3365609.3365855. doi:10.1145/3365609.3365855.
[28] T. R. Henderson, et al., Network simulations with the ns-3 simulator, in: ACM SIGCOMM demo, volume 14, 2008.
[29] T. Johnson, et al., Desktop and mobile web page comparison: characteristics, trends, and implications, IEEE Communications Magazine 52 (2014).
[30] F. F.-H. Nah, A study on tolerable waiting time: how long are web users willing to wait?, Behaviour & Information Technology 23 (2004) 153–163.
[31] M. Hock, et al., Experimental evaluation of BBR congestion control, in: IEEE ICNP, 2017, pp. 1–10.
[32] Y.-J. Song, et al., BBR-CWS: Improving the inter-protocol fairness of BBR, Electronics 9 (2020) 862.
[33] S. Ha, et al., Taming the elephants: New TCP Slow Start, Computer Networks 55 (2011) 2092–2110.
[34] E. Metsälä, J. Salmelin, Planning and Optimizing Mobile Backhaul for LTE, 2015, pp. 129–237. doi:10.1002/9781118924655.ch5.
[35] L. S. Brakmo, et al., TCP Vegas: New techniques for congestion detection and avoidance, in: ACM SIGCOMM, 1994, pp. 24–35.
[36] A. Baiocchi, et al., YeAH-TCP: yet another highspeed TCP, in: Proc. PFLDnet, volume 7, 2007, pp. 37–42.
[37] S. Mascolo, et al., TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, in: ACM MOBICsOM, New York, NY, USA, 2001, p. 287–297.
[38] M. Zhang, et al., Will TCP work in mmWave 5G cellular networks?, IEEE Communications Magazine 57 (2019) 65–71.
[39] C. Caini, R. Firrincieli, D. Lacamera, Pepsal: a performance enhancing proxy designed for tcp satellite connections, in: 2006 IEEE 63rd Vehicular Technology Conference, volume 6, 2006, pp. 2607–2611. doi:10.1109/VETECS.2006.1683339.
[40] F. Giust, G. Verin, K. Antevski, J. Chou, Y. Fang, W. Featherstone, F. Fontes, D. Frydman, A. Li, A. Manzalini, et al., Mec deployments in 4g and evolution towards 5g, ETSI White paper 24 (2018) 1–24.
[41] S. V. Bhanu, R. Chandrasekaran, V. Balakrishnan, Effective bandwidth utilization in ieee802. 11 for voip, arXiv preprint arXiv:1005.0952 (2010).
[42] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, I. Swett, J. Iyengar, V. Vasiliev, V. Jacobson, Bbr congestion control: Ietf 100 update: Bbr in shallow buffers, in: Proc. IETF-100, 2017.
[43] M. Kassim, R. A. Rahman, M. A. A. Aziz, A. Idris, M. I. Yusof, Performance analysis of voip over 3g and 4g lte network, in: 2017 International Conference on Electrical, Electronics and System Engineering (ICEESE), 2017, pp. 37–41. doi:10.1109/ICEESE.2017.8298391.
[44] J. D. Beshay, et al., Link-coupled tcp for 5g networks, in: IWQoS 2017, 2017, pp. 1–6.
[45] G. Lee, et al., Simulation study of TCP proxy in multi-connectivity enabled 5G mmWave network, in: ICTC 2019, 2019, pp. 865–869.
[46] R. Chakravorty, et al., Using tcp flow-aggregation to enhance data experience of cellular wireless users, IEEE JSAC 23 (2005) 1190–1204.
[47] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, et al., The quic transport protocol: Design and internet-scale deployment, in: Proceedings of the conference of the ACM special interest group on data communication, 2017, pp. 183–196.