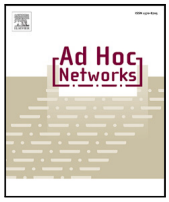
Ad Hoc Networks 164 (2024) 103599

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc



Exploiting stream scheduling in QUIC: Performance assessment over wireless connectivity scenarios

Fátima Fernández [a], Fátima Khan [b], Mihail Zverev [a], Luis Diez [b,*], José R. Juárez [a], Anna Brunstrom [c], Ramón Agüero [b]

[a] Ikerlan Technology Research Center, Arizmendiarrieta 2, Arrasate/Mondragón, 20500, Spain
[b] Universidad de Cantabria, Plaza de la Ciencia s/n, Santander, 39005, Spain
[c] Karlstad University, Frödingshöjd 28, Karlstad, 65637, Sweden

ARTICLE INFO

Keywords:
QUIC
Real-time applications
Wireless networks
Stream multiplexing
Priority-based schedulers

ABSTRACT

The advent of wireless technologies has led to the development of novel services for end-users, with stringent needs and requirements. High availability, very high throughput, low latency, and reliability are all of them crucial performance parameters. To address these demands, emerging technologies, such as non-terrestrial networks or millimeter wave (mmWave), are being included in 5G and Beyond 5G (B5G) specifications. mmWave enables massive data transmissions, at the expense of a more hostile propagation, typical for high frequency bands. Consequently, the inherent instability of the physical channel significantly affects the upper layers of the protocol stack, resulting in congestion and data losses, which might strongly hinder the overall communication performance. These challenges can be addressed not only at the link layer, but at any affected layer. QUIC is a new transport protocol designed to reduce communications latency in many ways. Among other features, it enables the use of multiple streams to effectively manage data flows sent through its underlying UDP socket. This paper introduces an implementation of priority-based stream schedulers along with the design of a flexible interface. Exploiting the proposed approach, applications are able to set the required scheduling scheme, as well as the stream priorities. The feasibility of the proposed approach is validated through an extensive experiment campaign, which combines Docker containers, the ns-3 simulator and the Mahimahi framework, which is exploited to introduce realistic mmWave channel traces. The results evince that an appropriate stream scheduler can indeed yield lower delays for time-sensitive applications by up to 36% under unreliable conditions.

1. Introduction

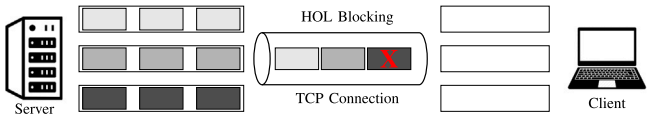
Novel wireless networks, in particular 5G and Beyond 5G (B5G) systems, enable an evolution in mobile communications, especially in terms of new applications and services. These technologies allow high transmission rates (enhanced mobile broadband, eMBB) and low latency (ultra reliable and low latency communications, URLLC). In addition, they also foster the massive interconnection of devices generating and collecting information (massive machine-type communications, mMTC). This has helped to the quick spread of the Internet of Things (IoT) paradigm [1].

The next evolution of IoT, exploiting upcoming mobile networks, is the intrinsic support of real-time applications, such as haptic communications, autonomous driving, interactive virtual education, or free-viewpoint video (*i.e.* streaming video from a drone, allowing the projection of an enlarged view beyond the current viewpoint), among others [2].

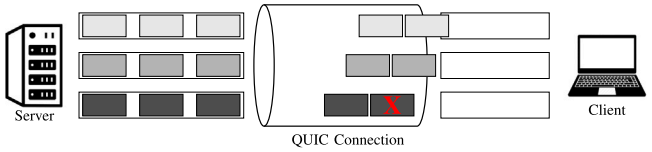
One of the keys to meet these demanding requirements is upgrading the underlying network technologies. Greater bandwidths can be achieved using mmWave technology, which consists of switching the communications to GHz frequency range [3]. This allows for massive data transmissions at high data rates and low latency, making it ideal for a wide range of IoT applications, including smart cities, intelligent transportation, healthcare, innovative industries, among others [4]. At the same time, non-terrestrial networks based on Low Earth Orbit satellites can greatly reduce latency for distant end-points, apart from increasing network availability [5]. The new technologies suit well both traditional and emerging applications. Unmanned Aerial Vehicles (UAVs) could be used to monitor the environment with embedded sensors, to offload communications from congested terrestrial networks, or to provide emergency assistance [6]. For instance, in the field of agriculture, drones can collect data and send it to LEO satellites, as part of a control network, to finally send it to the cloud [7].

* Corresponding author.
E-mail address: ldiez@tlmat.unican.es (L. Diez).

https://doi.org/10.1016/j.adhoc.2024.103599
Received 19 June 2024; Accepted 14 July 2024
Available online 12 August 2024



(a) HTTP/2 stream multiplexing with head-of-line Blocking.



(b) QUIC stream multiplexing without head-of-line delays.

Fig. 1. Stream multiplexing in HTTP/2 and QUIC.

Furthermore, real-time applications have very strict time requirements that can be achieved with the combination of high-speed mobile technologies such as 5G [8,9], suitable network topologies, and appropriate communication protocols, among others.

QUIC is a new transport layer protocol that could support real-time applications [10]. One of its main goals is to reduce communications latency. To achieve that, QUIC does not only include an efficient information loss recovery mechanism, but it also implements multiplexing to avoid head-of-line (HOL) blocking. HOL blocking strongly affects Transmission Control Protocol (TCP), the widespread transport protocol used by almost all applications that need reliable data transmission, such as the popular HTTP/2 application layer protocol. In TCP all information flows go through its socket. In QUIC, information flows are organized in streams, which are multiplexed over time. Fig. 1 illustrates how a loss of a single data packet affects both TCP and QUIC. Since TCP has only one data flow, the loss recovery mechanism delays the whole connection (Fig. 1(a)), while in QUIC this only impacts the corresponding stream (Fig. 1(b)). Stream multiplexing can also have a positive impact on application performance. For this reason, the standard does not define a specific stream management mechanism but suggests that the application can prioritize its data based on the active streams, [10, Section 2.3].

In real-time applications, some flows can be more time-sensitive than others. To efficiently manage QUIC streams, those flows might need to be prioritized, since they could be critical for many processes of the aforementioned IoT applications. In this work we propose the use of QUIC stream multiplexing and stream transmission scheduling, based on user-defined priorities, to ensure low latency for time-sensitive traffic. We present a two-fold, complementary evaluation. We first assessed in [11] the performance of the proposed policies over various underlying connectivity conditions, with different loss rates. In the present study we evaluate the impact of very dynamic channel fluctuations, as those that characterize the behavior of mmWave links.

In summary, we extend the analysis carried out in [11], whose main contributions are again presented in this paper. In such previous paper, we proposed the integration of new stream schedulers in a real QUIC implementation (in GO programming language). These included baseline solutions, such as Weighted Fair Queuing (WFQ), as well as an absolute-priority strategy, which aims at ensuring low delay for critical traffic flows. The schedulers were validated using a methodology that combined real nodes (Docker containers) with the `ns-3` simulator, which was used to emulate different connectivity conditions. This methodology was exploited to assess the latency of a particular IoT scenario, which entails real traffic, using traces that capture control messages from a controller to an UAV. In this paper we extend the aforementioned validation by proposing a new scheduling solution and emulating other link layer technology. In particular we consider multi-gigabit mmWave links, exploiting the Mahimahi framework instead on `ns-3`. All in all, the main contributions of this work are:

- A dynamic stream scheduling policy, which based on instantaneous channel conditions switches between the different scheduling solutions, initially proposed in [11].
- A detailed analysis and validation of all considered stream schedulers.
- We use a new evaluation setup using Mahimahi framework [12]. We study the behavior of the proposed scheduling policies over highly dynamic wireless channels, using mmWave links.
- All the code has been made available in a public git repository,[1] to ensure reproducibility of the presented results and experiments.

The remainder of this paper is structured as follows. Section 2 discusses related work, positioning our paper and pointing out its main contributions. Section 3 describes the integration of the scheduling mechanisms within a real QUIC implementation. Section 4 presents the tools used to build the simulation setups, and discusses the obtained results, which are used to compare the behavior of the proposed scheduling approaches. The conclusions are outlined in Section 5, which also provides an outlook of our future work.

2. Background

The QUIC protocol was first proposed by Google Inc. [13] and it has been recently standardized and published by the Internet Engineering Task Force (IETF) as a series of "Request For Comments" (RFC) documents [10,14–16]. QUIC was created with the aim of reducing the download time of HTTPS traffic, without losing security and reliability [17]. It is implemented over UDP and it supports congestion control and loss recovery mechanisms [16]. QUIC also includes a number of features to tackle some of the challenges that TCP cannot successfully overcome, such as: reduction of connection establishment latency, head-of-line blocking delay, and protocol ossification. The implementation at the application layer and protocol version management allows the inclusion of other useful features, which were not included in the original QUIC specification. In order to provide security between two endpoints of the network, QUIC uses the TLS 1.3 protocol [15]. As a result, it is possible to reduce the connection start-up time, by transmitting application data during connection establishment. In addition, QUIC is transparent to middleboxes, which consider its encrypted packets as UDP traffic. For this reason, middleboxes cannot be optimized for QUIC traffic, which facilitates the deployment of new QUIC versions, thus preventing protocol ossification.

Among the many extensions that can be added to QUIC, it is important to highlight the unreliable datagram extension. By default, QUIC sends reliable data in STREAM frames [10]. Some applications, especially those having strict real time requirements, need unreliable data transmission to avoid delays induced by retransmissions. UDP alone offers this feature, but UDP datagrams are not congestion-controlled or encrypted, unlike the new QUIC DATAGRAM frame added in [18].

As mentioned above, QUIC is organized in streams, which can be created by any endpoint. They are independent units of information that can be unidirectional or bidirectional [10]. They can send data interleaved with other streams, without guaranteeing the byte order of the streams, and can be canceled when needed. QUIC allows streams to operate simultaneously. Furthermore, an arbitrary amount of data can be sent in any stream, depending on the flow control constraints and stream boundaries. However, QUIC does not define how the scheduling policy should behave when several streams run over the same connection. In this sense, this work arises as a response to the flexibility that QUIC provides in terms of its scheduling policy implementation, taking into account the requirements of the application traffic.

To the best of the authors' knowledge, very few works have analyzed the stream scheduler within QUIC. The vast majority of the

[1] Schedulers in `quic-go`: https://github.com/fatimafp95/quic-go_scheduler/tree/mod.

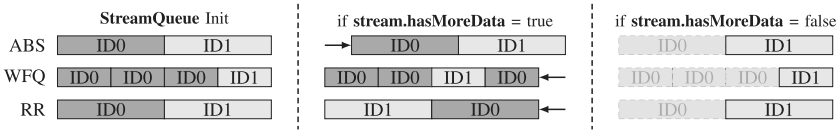


Fig. 2. Representation of each scheduler functionality in each state of the stream queue management.

schedulers found in the literature aim at other aspects of the connection, rather than multi-streaming, and, in particular, multipath. Transport layer multipath (MP) consists of the simultaneous use of multiple network interfaces. MP and stream schedulers tackle different challenges: MP sends data through different and surely heterogeneous network paths, while streams without MP are traversing the very same path. An MP extension for QUIC is currently under development [19]. In addition, it is worth mentioning that some works reveal that stream-awareness would actually improve the behavior of MP scheduling. Shi et al. proposed a priority-based online stream scheduling mechanism for MPQUIC [20]. Path scheduling was performed according to stream properties. The authors showed that the proposed scheduler could reduce the download time in an environment where paths might have very different characteristics. Viernickel et al. also provided a stream-to-path scheduling policy for MPQUIC, showing a reduction of the HOL effect, and reducing the time required to establish subflows [21]. Other works compare MP schedulers without taking into account how the streams are handled within the transport protocol [22,23]. MP is out of this paper's scope, which focuses on multi-streaming over a single path.

Chiariotti et al. [24] also studied the use of multiple streams over one path. They only focus on how to map application data to the underlying streams. The authors assume that applications define the relationship of the data in each stream to the requirements of the corresponding service. Their scheduler sought to maximize this correlation at the receiver node. They evaluate their approach using two particular use cases: inter-vehicular and haptic communications. Hervella et al. compared the performance of QUIC over realistic satellite networks with that shown by TCP [25]. They modify the ns-3 network simulator to emulate satellite links with different characteristics. They also carry out an analysis of the impact of multi-streaming in QUIC, using the default Round Robin scheduler. Their results evince that this scheduling policy does not yield lower delays. In addition, Cui et al. also used multiple streams in QUIC to carry MPEG-DASH protocol (DASH) in [26]. In this case, the scheduler takes the decision based on the deadline for each video segment that is adaptively sent through each stream, or on the buffer state. If the segment is not received within the required time or the playback buffer is not full, the multi-stream scheduler switches the transmission mode to a simplex mode of operation, receiving the first requested segment, and pausing the remaining streams. The authors also used ns-3 to conduct the experiments. The results evince that DASH over QUIC (DASH+) with multi-stream achieves higher throughput and better video quality compared to alternative approaches. However, the authors did not modify the scheduling policies of the QUIC protocol. It is worth highlighting the work by Rozen-Schiff et al. [27], since they actually modify these policies. They proposed a QUIC module called AQUA that features a WFQ scheduler to distribute bandwidth according to stream requirements for metaverse applications.

Another set of papers that analyze the use of multiple streams and their scheduling within the same QUIC implementation we use herewith (`quic-go`), are mentioned hereinafter. Hervella et al. conducted a comparative study on the performance of TCP and QUIC transport protocols over realistic satellite networks [28]. The authors proposed a novel methodology that combines real implementations, leveraging virtualization techniques, with simulations to conduct systematic and repeatable experiments. The default operation the ns-3 framework is modified to incorporate the dynamic characteristics of satellite communication links, particularly LEO communications. Their comprehensive assessment spans various setups, including different operating frequency bands and packet buffer lengths. While the main focus was evaluating both protocols within dynamic environments, the assessment also analyzed the impact of QUIC's multi-streaming capability. They concluded that the default stream scheduler that `quic-go` includes, the well-known Round Robin (RR) may be suboptimal. Afterwards, Khan et al. further extended the assessment, analyzing the behavior of other stream scheduling policies [29,30]. They considered a more complex application where streams are prioritized based on application data generation rate. For this, scheduling algorithms were implemented at the application layer, defining a buffer for each stream. Using this approach, they analyzed WFQ, Fair Queueing (FQ) and they proposed three new schedulers based on Lyapunov's theory [31]. These novel schedulers are focused on equalizing the occupancy and the accumulated sojourn time at the aforementioned buffers. The authors showed that Lyapunov based scheduling policies fairly distribute both the buffer occupancy and the delay between streams, without jeopardizing the throughput, even under very high load situations, and highly dynamic connectivity situations.

Some applications have very strict delivery time requirements. Examples of such applications are the aforementioned inter-vehicular and haptic services, drone applications, or even gaming. The so-called Tactile Internet, where haptic applications are considered, requires round-trip times lower than ~10 ms [9]. UAV control traffic has very strict requirements for delay and reliability. In 2018, Ferranti et al. predicted that 78% of the total traffic in 2021 would be from video services [32]. This is actually strengthened by mobile technologies, as they enable the interconnection of more devices, including UAVs. When a drone carries different types of traffic, such as video and control flows, it is important that the latter arrives at its destination with the lowest possible delay. In this work, we propose to exploit multiple streams in QUIC as a means to send the control traffic over a separate stream, prioritizing it over the stream carrying bulk data.

We consider an appropriate stream scheduler for the QUIC implementation `quic-go` [33]. The default behavior in this implementation reflects the traditional Round Robin (RR) policy. This algorithm equally distributes the resources on a first-come, first-served basis. It means that no priority is assigned to any stream, even if it carries sensitive data. We propose two priority-based schedulers: weighted fair queuing (WFQ) and absolute prioritization (ABS). On the one hand, WFQ assigns a specific weight to each stream, which corresponds to the respective time fraction. On the other hand, absolute prioritization allocates all time to the highest priority stream as long as it has data to transmit. It is worth noting that, as a consequence of the selected QUIC implementation, prioritization covers both transmission and retransmission queues of a stream. Finally, a third algorithm is proposed which takes into account the network conditions to adapt the stream scheduling switching dynamically from RR to ABS and vice-versa.

3. Stream scheduler implementation

The QUIC implementation we use [33] is based on the IETF specifications [10,15,16], including the unreliable datagram extension [18]. It is worth mentioning that this implementation has been following the changes proposed during the QUIC specification process. Furthermore, Crochet et al. evaluated in 2021 the performance of various QUIC implementations, and the most successful servers were `quinn` and

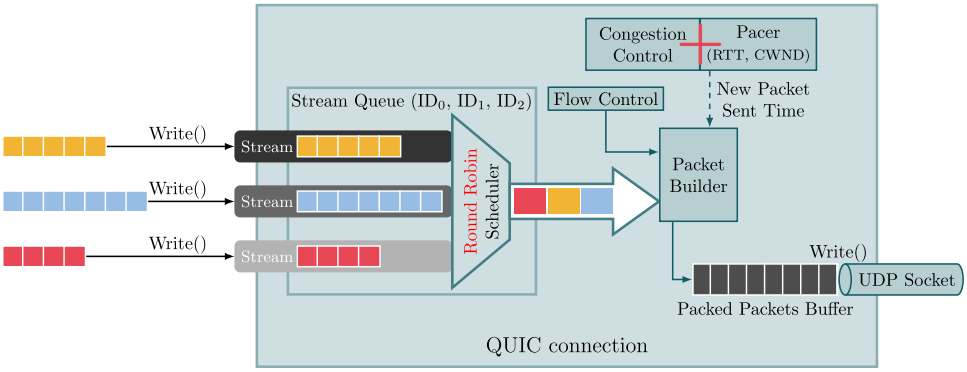


Fig. 3. Diagram of all interfaces in `quic-go` related to stream management.

`quic-go`, while `quic-go` was the best choice at the client side [34]. In addition, the QUIC interoperability matrix[2] depicts several implementations with many interoperable features. Again, `quic-go` stands out as one of the implementations that has the most features working correctly, even with other implementations.

As already mentioned, the only stream scheduling policy found in the original implementation is Round Robin. Before depicting the schedulers that we have implemented, it is important to understand the operation of the stream management interface in `quic-go`. Fig. 3 illustrates an implementation diagram of all interfaces related to the stream management. When application data is ready to be sent, one or more streams can be opened, where the data is written. Each stream i has an assigned identifier (ID_i) that is queued in the stream queue. Then, the interface, depending on the scheduler, selects data chunks from the stream at the front of the queue to build the QUIC packet. This packet is then assembled and transmitted according to the congestion control,[3] flow manager, and pacer. The pacer determines the appropriate transmission time based on the RTT and the congestion window to avoid a packet burst that might cause congestion and losses [16, Section 7.7]. Then, the packet is written in the UDP buffer.

In the following lines, we depict how the stream scheduling functionality is designed and implemented in `quic-go`. The interfaces to start a QUIC session and to interact with the streams are defined in the *interface.go* file. This includes the functions used to open new streams and to configure the connection (*Config* structure) needed for both QUIC server and client.

`quic-go` stream management mostly consists in handling a stream queue. We thus modify the queue management mechanisms to implement new stream schedulers: absolute prioritization (ABS) and weighted fair queuing (WFQ). Fig. 2 illustrates the default queue management and the modifications we introduce. In this example, we have assigned a higher priority to the stream with the id "ID0". This means that messages sent in this stream will be processed first. At the beginning, the stream queue is filled with the IDs (*StreamQueue Init.*). For WFQ validation, we have allocated 75% transmission time to "ID0" and 25% to "ID1". This is achieved by replicating the corresponding streams within the queue. Then, the stream queue is checked, according to the scheduler type and whether it has more data to send (*stream.hasMoreData*). After taking data for transmission from the first stream in the queue, and if the stream still has data to send, it returns to the queue. For WFQ and Round Robin, the stream is moved to the end of the queue, while for ABS it remains in the head. If there is no more data to send, the stream is not sent back to the queue and all of its replicas are removed.

The default behavior for stream processing in `quic-go` is based on the Round Robin (RR) algorithm, as shown in Algorithm 1. When QUIC opens one or more streams that have data to transmit, they are considered as active streams (*activeStreams*). The implementation allocates them at the end of the queue and starts processing them, according to the moment they were opened (lines 1–3). The streams are then assessed in terms of their construction and whether they have data to transmit. Next, the availability of more stream data is checked (*hasMoreData*). If there is more data to send, the corresponding stream ID (*streamID*) is placed at the end of the stream queue, otherwise, it is removed from the active stream queue (lines 10–14).

Algorithm 1 RR algorithm in `quic-go` for stream management.

```
Require: activeStreams, streamID, streamQueue, packet
 1: if activeStreams[streamID] ≠ 0 then
 2:     streamQueue = append(streamQueue, streamID)
 3: end if
 4: numActiveStream = length(activeStreams)
 5: streamCount = 0
 6: while streamCount < numActiveStream do
 7:     id = streamQueue[0]
 8:     streamQueue = streamQueue[1:END]
 9:     hasMoreData = packet.FillWithStreamData(id)
10:     if hasMoreData then
11:         streamQueue = append(streamQueue, id)
12:     else
13:         delete(activeStreams, id)
14:     end if
15:     if packet.isFull then
16:         break while loop
17:     end if
18:     streamCount++
19: end while
```

In order to implement a flexible interface to use multiple schedulers, we extend the QUIC configuration with both the scheduler type and the priority level for each stream (*SchedulerType* and *StreamPrio* list, respectively). Based on this, any application using this enhanced QUIC implementation could easily configure the stream manager features.

We redesign and adjust how the stream queue is managed and processed, by the *AddActiveStream* and *AppendStreamFrames* functions in the *framer.go* file, to include ABS and WFQ schedulers. *AddActiveStream* updates the stream queue when a stream becomes active (newly opened, new data written, or flow control update). The new streams are added at the end of the queue. Depending on the selected scheduler, we modify this behavior to sort the queue based on the streams' priority levels as depicted in Algorithm 2. The stream queue is further managed in *AppendStreamFrames*, which is called every time a

[2] https://interop.seemann.io/, last visit: 12/05/2024.

[3] The only congestion control algorithm implemented in `quic-go` is CUBIC [35].

Algorithm 2 ABS and WFQ algorithm for priority-based queuing (function addActiveStream).

```
Require: SchedulerType, streamQueue, streamMapPrio, streamID, StreamPrio
 1: switch SchedulerType do
 2: case ABS:
 3: lenQueue = length(streamQueue)
 4: streamMapPrio[streamID] = StreamPrio[lenQueue−1]
 5: if lenQueue > length(StreamPrio) then
 6:     priority, streamIDFound = streamMapPrio[streamID]
 7:     if not streamIDFound then
 8:         streamMapPrio[streamID] = 0
 9:         priority = 0
10:     end if
11:     StreamPrio = append(StreamPrio, priority)
12: end if
13: currentPosition = lenQueue−1                    ▷ New stream position
14: newPriority = StreamPrio[currentPosition]
15: while currentPosition ≥ 0 do
16:     if newPriority ≥ StreamPrio[currentPosition] then
17:         correctPosition = currentPosition
18:     end if
19:     currentPosition−−
20: end while
21: StreamPrio.insertAt(newPriority, correctPos)
22: streamQueue.insertAt(streamID, correctPos)
23: StreamPrio = StreamPrio[0:lenQueue−1]
24: streamQueue = streamQueue[0:lenQueue−1]
25: case WFQ:
26: prio = 1
27: knownPrio, streamIDFound = streamMapPrio[streamID]
28: if streamIDFound then
29:     prio = knownPrio
30: else
31:     if length(StreamPrio) > 0 then
32:         prio = StreamPrio[0]
33:         StreamPrio = StreamPrio[1:END]
34:     end if
35:     streamMapPrio[streamID] = prio
36: end if
37: replicate = prio
38: while replicate > 0 do
39:     streamQueue = append(streamQueue, streamID)
40:     replicate−−
41: end while
42: end switch
```

Algorithm 3 Stream management depending on whether more data is available (function appendStreamFrames).

```
Require: SchedulerType, hasMoreData, streamQueue, streamID,
    StreamPrio
 1: if SchedulerType == ABS then
 2:     if hasMoreData then
 3:         streamQueue=append(streamID, streamQueue)
 4:     else
 5:         StreamPrio = StreamPrio[1:]
 6:     end if
 7: else if SchedulerType == WFQ then
 8:     if hasMoreData then
 9:         streamQueue = append(streamQueue, streamID)
10:     else
11:         position, found = streamQueue.find(streamID)
12:         while found do
13:             delete(streamQueue, position)
14:             position, found = streamQueue.find(streamID)
15:         end while
16:     end if
17: end if
```

new QUIC data packet is built. By default, this function takes the first stream in the queue, uses its data to build a STREAM frame and, if there is more data, it returns the stream to the end of the queue, resulting in Round Robin behavior. To implement other schedulers, we modify this operation as presented in Algorithm 3.

To avoid breaking QUIC connection calls to the data write method, we provide priorities in a list (*StreamPrio*) through the *Config* structure. At the time of stream selection, `quic-go` works with a stream queue (*streamQueue*), which is synchronized with the active stream list (*activeStreams*). If all stream data has been sent, or if the stream is blocked, its ID will be removed from the stream queue. When such stream returns to the queue, it needs to recover the priority level assigned to it. For this purpose, we define an ID-priority map (*StreamMapPrio*), which is populated during priority list consumption. Map population is defined on line 4 for ABS and lines 30–36 for WFQ in Algorithm 2.

The default `quic-go` stream manager takes the first ID from the stream queue, with the queue being filled on a first-come, first-serve basis. For absolute prioritization, we sort the stream queue following the *StreamPrio* list (Algorithm 2: lines 13–24). It establishes which stream should be processed first, *i.e.*, sending all of its data (keeping the stream flow control limitation). Afterwards, the stream ID is removed from the stream queue, until it receives more data to transmit. Then the stream with the next highest priority level is processed likewise. WFQ scheduler distributes resources on a weighted basis. Our implementation emulates this behavior by interpreting *StreamPrio* as a list of weights. Every new stream ID is replicated as many times as indicated in the head of *StreamPrio* list (Algorithm 2: lines 37–41).

After building a STREAM frame for a new data packet, the default Round Robin scheduler puts the corresponding ID back at the end of the queue (Algorithm 1, lines 1–12). Following the same logic, ABS scheduler takes the ID from the head of the stream queue, but instead of appending it at the end of the queue, ABS puts it back to the head (Algorithm 3, lines 2–4). If the stream has no data left, it does not return to the queue and its priority level is removed from the *StreamPrio* list (Algorithm 3, line 5). WFQ and RR behaviors are rather different. For each stream that has more data to transmit, its ID is added to the end of the queue (for RR, line 14 of Algorithm 1, and line 9 for WFQ in Algorithm 3). If the stream has no more data to send, RR does not return it to the stream queue (line 16 in Algorithm 1). This behavior applies as well to WFQ, except that in this case we also need to remove all of the depleted stream's replicas from the queue (Algorithm 3, lines 11–14).

Finally, combining two of the previous schedulers a novel policy is proposed that aims to guarantee certain requirements of the data streams. It would be specially appealing in scenarios with highly varying capacity where a static configuration could not render the best behavior. On the one hand, when the communication capacity is high the prioritization would induce delays in the non-prioritized streams even if it is not really necessary. On the other hand, when the communication capacity degrades the prioritization would be necessary to satisfy the requirements of the priority streams. In order to estimate the communication capacity shared among the different data streams we use the congestion window (CWND) and smoothed RTT (sRTT) variables. It is worth noting that both parameters can be measured at the QUIC layer, so guaranteeing the feasibility of this approach. In this sense, sRTT is a variable within `quic-go` that corresponds to an exponentially weighted moving average of the round-trip time samples [16, Section 5.3]. These metrics are updated once an ACK event occurs, facilitating bandwidth estimation. The capacity estimation is calculated as the ratio of the CWND in bits to the RTT in seconds. Following `quic-go` implementation, this ratio is scaled with a correction factor of 5/4 to ensure that the actual bandwidth is not under-utilized [36]. Then, upon the definition of a target shared capacity sBW, the proposed scheduler dynamically switches between ABS and RR according to the

Algorithm 4 Logic to enable the adaptability of the scheduler (included in appendStreamFrames).

Require: *cwnd*, *sRTT*, target
1: $sBW = (5/4) \cdot (cwnd \cdot 8)/(sRTT)$
2: **if** sBW > target **then**
3: SchedulerType = "RR"
4: **else**
5: SchedulerType = "ABS"
6: **end if**

channel conditions. As can be seen in Algorithm 4, if the estimated bandwidth is below the target then ABS will be employed to guarantee the early reception of the prioritized stream (line 3). Conversely, if the target is below the estimated bandwidth, the scheduler will be switched to RR to prevent the starvation of other streams, as detailed in line 5 of Algorithm 4. It is worth mentioning that the application has only to select this scheduler (*SchedulerType* = "`adapt`") and specify the stream with the highest priority and its target bandwidth.

4. Results

4.1. Environment setup

To carry out the experiments, we first exploit the ns-3 discrete-event simulator[4] and Docker containers.[5] Docker containers are connected through ns-3, which emulates the characteristics of the underlying connectivity, by altering bandwidth and delay parameters (Fig. 4a). Additionally, the loss rate can be also tweaked to consider different conditions. We connect two containers that exchange real application traffic. Each container hosts an application consisting of a client and a server that use QUIC. These applications are isolated from each other on separate networks, running on their corresponding operating systems, with their own network stack, and interacting with a dedicated network device. To establish the network topology, each container is connected to a bridge that is linked to a tap device and then to a special ns-3 *NetDevice*. These *NetDevices* connect each emulated node (Docker container) to a network router. The two network routers are connected by a point-to-point link (*PointToPointNetDevice()* in ns-3), which is used to emulate different networking technologies by modifying bandwidth and round-trip time. Furthermore, connecting links to edge routers have very high capacity and zero delay, ensuring the bottleneck is on the network.

The bandwidth and the round-trip time during the experiments are 1 Mbps and 40 ms, respectively. We also use different packet drop rates to assess the performance under both reliable and unreliable link conditions (0%, 5%, and 10%).

We extend the evaluation setup with a configuration that allows end-to-end evaluation of communication conditions of high capacity and variability. It is worth mentioning that ns-3 emulation capability presents some limitation when configured with high communication capacity. In those cases, ns-3 is not able to ensure timely packet delivery acting as a bottleneck. In order to overcome this limitation, we have used the Mahimahi [12] link emulator fed with capacity traces to ensure multi-gigabit communications capacity, as depicted in Fig. 4b. The capacity traces are generated from a mmWave scenario in ns-3 [37] where a server and a client communicate sending UDP traffic in saturation. During the communication the reception time of every UDP packet and the number of received bytes are recorded, and that trace is used to configure Mahimahi. More information about the generation of traces can be found in [38]. On top of the capacity traces, Mahimahi allows us to define additional link delay and buffer size. In Section 4.3 we will analyze the performance of the adaptive scheduler over a scenario where a user moves between two buildings in a street canyon, getting closer to a base station at a constant walking speed. Tables [38, Tables 1, 2] provide more details about the configuration that was used to obtain the traces used afterwards.

4.2. Performance assessment

Streams in QUIC can be used to send a single message, *i.e.* transmitting the objects of a web page in separate streams, but can be also mapped to information flows, thus sending multiple messages over time, for instance, messages that might be sent to the same MQTT topic in an IoT environment [39].

To carry out the first experiment, we use traces obtained from a real communication between a controller and a drone over a 5G network. The information of these traces comprises Micro Air Vehicle Link (MAVLink3[6]) packets over UDP. We thus consider two IoT nodes with the Docker containers and the network is created by means of the ns-3 simulator. The messages exchanged are generated by replicating the information provided by the traces: we use each packet's timestamp and payload length. We establish one QUIC connection between the IoT nodes and we then open ten streams, emulating ten different data flows, to better assess the benefits of the prioritization capacity of the proposed schemes. One stream was assigned to the control traffic (drone–controller traces) and the remaining streams to lower priority background traffic. We emulated this non-essential traffic with bulk transmissions, to mimic an intense usage of the underlying QUIC connection. We assign higher priority to control traffic through the extended QUIC configuration introduced in Section 3. Then, we analyze the latency-sensitive control traffic, measuring the time that elapses between the moment the message was written on the stream socket and when it was completely delivered to the receiver. WFQ was configured to allocate 25% of the transmission time to the priority stream and the remaining time to the nine other streams. This particular configuration (weights) is just meant to validate the correct operation of the implementation. In upcoming research, we will exploit this framework to develop optimal scheduling policies, by adapting the particular weights to the characteristics of the scenario and traffic patterns.

Fig. 5 uses a boxplot representation of the message delay for each scheduler. As can be seen, results show a notable dispersion, due to the large variability of the message lengths, specifically with Round Robin. The operation of this scheduler assumes that the application sends the information immediately, while the stream manager takes data from each stream at a regular and equal pace. Hence, the information might likely be queued, waiting to be transmitted. Some of the prioritized messages arrive at the queue when their stream is closer to the transmission, thus reducing the delay for such messages, while other messages need to traverse the entire queue. The performance of the absolute prioritization scheduler is better than RR. The proposed scheduler consistently achieves lower latency for the prioritized traffic. The gain is not remarkable due to the message lengths. QUIC data manager fills a packet with data from the stream that is at the top of the active stream queue. If the stream has no more data, it is removed from the queue. The maximum length of control messages is 242 bytes, so they always fit into a single QUIC packet, and the performance of the schedulers is similar.

The behavior of WFQ is hampered by the computational load of queue management with this scheduler, which delays the transmission of data from the next queued stream. For this use case, prioritization with applications that open multiple streams for the transmission of multiple data flows can reduce latency despite the delay introduced by scheduling operations.

[4] A discrete-event network simulator for Internet systems, https://www.nsnam.org/ version 3.35.

[5] Docker containers for virtualization, https://www.docker.com/ version 20.10.17.

[6] MAVLink3 is a communication protocol for small unmanned vehicles.

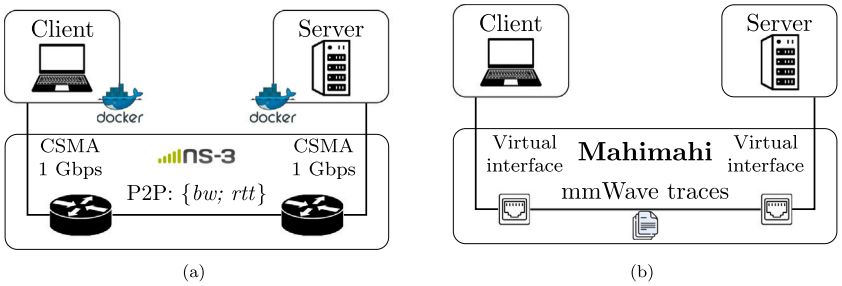


Fig. 4. Schematic illustration of the two evaluation setups. (a) is based on Docker containers and the ns-3 simulator; (b) is based on the Mahimahi framework, to load mmWave traces.

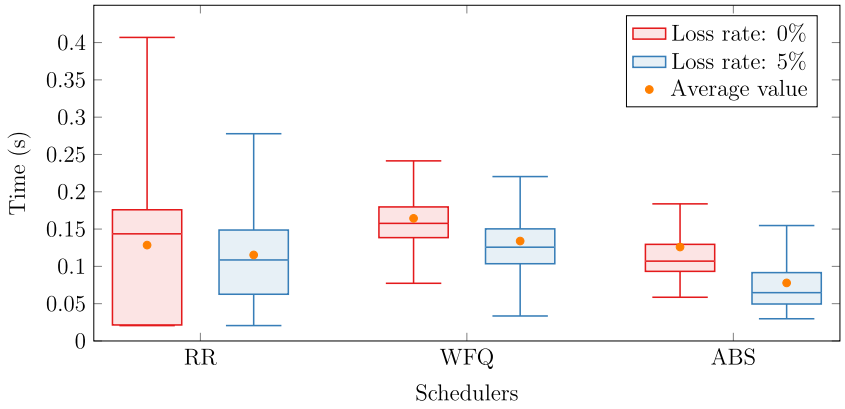
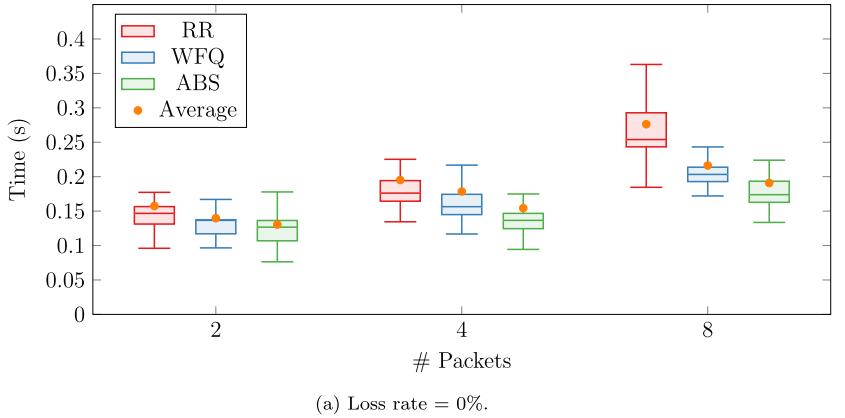
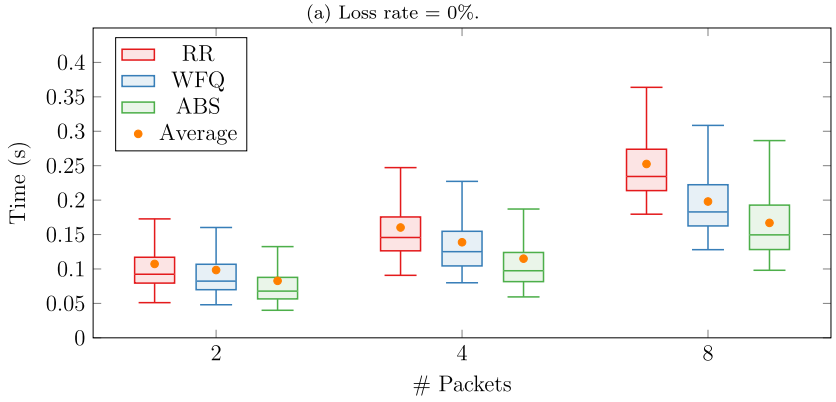


Fig. 5. Control traffic messages latency using each scheduler with 10 streams.



(a) Loss rate = 0%.



(b) Loss rate = 5%.

Fig. 6. Message latency for the prioritized stream.

Table 1
Relative gain (in terms of delay) of WFQ and ABS schedulers, compared to legacy RR.

#	WFQ			ABS		
Pkts	0%	5%	10%	0%	5%	10%
2	0.113	0.083	0.167	0.171	0.229	0.288
4	0.085	0.134	0.157	0.210	0.282	0.301
8	0.217	0.216	0.229	0.309	0.339	0.360
16	0.130	0.176	0.210	0.307	0.340	0.376

Table 2
Scenario setup B1 from [38, Table 1,2,3].

Link features	
Buffer length	$\{0.5, 1, 2, 5\} \times$ BDP
Delay [ms]	{0, 5, 10, 20}
sBW [Mbps]	{10, 20, 50, 100, 200, 500}
Application	
# Streams	2
File size per stream	1GB

For a more in-depth analysis of these new proposals, we send longer messages on the prioritized stream. We create one QUIC connection and open two streams. We increase the length of the prioritized messages to ensure each of them corresponds to more than one single packet. More specifically, we transmit messages of 2, 4, and 8 packets. Fig. 6 shows the observed delay of this approach, using the same network parameters. As was done in the previous experiment, WFQ allocates 75% of the time resource to the priority stream. It can be seen that as the message length increases, priority-based stream scheduling becomes more beneficial since the gain is more remarkable. When two QUIC packets are generated, shorter times are handled when using prioritization schemes. The transmission time is reduced by approximately 20 ms using ABS without losses. This difference increases with the length of the message. When the message embraces 8 packets, a reduction of at least 50 ms can be seen in the case of WFQ with respect to RR, and around 80 ms if absolute prioritization is used. The same behavior is also seen for the error-prone channel (5%, Fig. 6(b)), and a difference of almost 85 ms is observed between ABS and Round Robin when 8 packets are generated.

To complement the results of Fig. 6, Table 1 shows the relative improvements of both WFQ and ABS over the RR scheduler for different message lengths and link qualities. As can be seen, the use of advanced scheduling policies, together with the multi-streaming capacity featured by QUIC can yield relevant performance gains, especially with larger messages (#pkts ≥ 2).

4.3. Evaluation of an adaptive scheduler in a mmWave environment

As mentioned before, the evaluation of the adaptive scheduler is conducted over a highly dynamic scenario, comprising mmWave links, characterized by high capacity and variability. In all scenarios, an application transmits a 1 GB file over each stream, resulting in saturation conditions until the transmission is complete, mimicking a bulk transfer. We study the performance of the adapting scheduler under different configurations, as defined in Table 2. The mmWave channel presents an average throughput of 244.6 Mbps, with instantaneous (measured every 10 ms) minimum and maximum values of 0.94 and 3703.79 Mbps, respectively. Over such communication channel we study the performance of the adaptive scheduler under different configurations of the target throughput (sBW), link delay and buffer size. As can be observed in Table 2, the buffer size is defined in multiples of the bandwidth-delay-product (BDP) which is obtained multiplying the average capacity mentioned above and the link delay configured. Firstly, we study the evolution of the congestion window and throughput in a configuration where the buffer length equals to

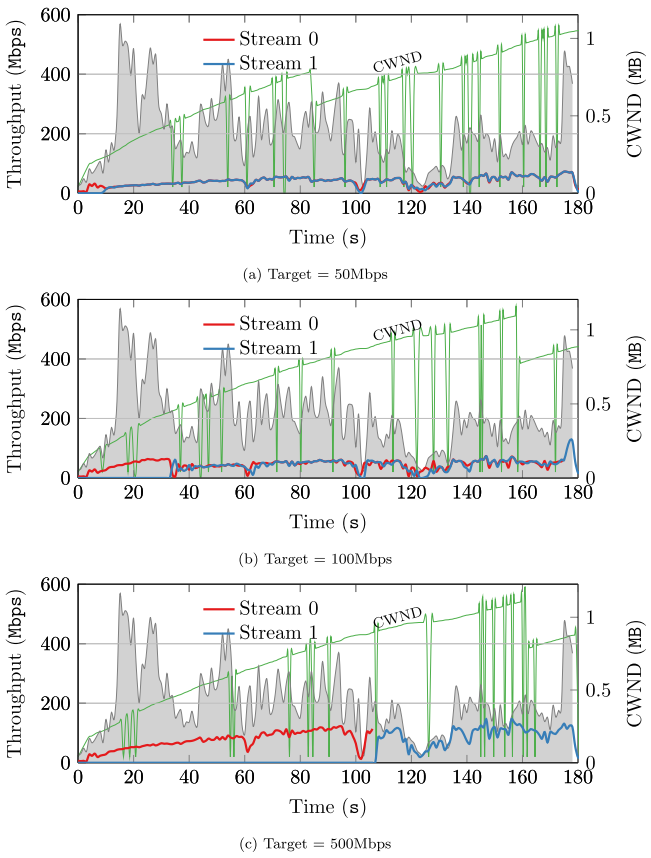


Fig. 7. Evolution of the throughput and congestion window upon different target throughput. The light gray shadow at the background represents the channel capacity variability. Link delay and buffer size are set to 20 ms and 5 BDP, respectively.

5 ·BDP and the delay of the underlying connectivity is set to 20 ms, using QUIC with two streams and the adaptive scheme, which allows switching between ABS and RR. Fig. 7 shows the results that were observed. It also includes, using a light gray shadow at the background, the channel capacity variability. For the sake of visibility, the channel capacity is plotted using the average capacity measured during 1 s. In this sense, the figure does not represent capacity variations happening in a higher pace, such as the maximum capacity of above 3 Gbps. Additionally, we also varied the target threshold from a rather low value (50 Mbps), where RR is predominantly used, to a more strict target (500 Mbps), where ABS prioritizes one of the selected streams.

As illustrated in Fig. 7, when the threshold is lower, see Fig. 7(a), than the estimated bandwidth and so the average rate, the RR scheduler is mostly used. It is important to highlight that the download completion time of each stream remains approximately the same. Secondly, as evinced by Fig. 7(b), when we increase the threshold to 100 Mbps, we observe, at the beginning of the simulation, that a lower congestion window leads to a reduction in the estimated rate, prompting the use of the ABS scheduler. Finally, Fig. 7(c) shows that when we increase the threshold beyond the average throughput, the bulk transfer of the highest priority stream is sent first, followed by the others. This behavior is a consequence of the ABS scheme being employed, as a result of an estimated bandwidth lower than the target. This phenomenon can be attributed to the presence of a larger delay, fixed at 20 ms, which leads to increased RTTs and so lower estimated bandwidth.

Furthermore, we study the completion time under different configurations to ascertain the impact of varying variables in the same scenario. Fig. 8 shows the impact of varying the target throughput, resulting in a reduction of the completion time, which is particularly

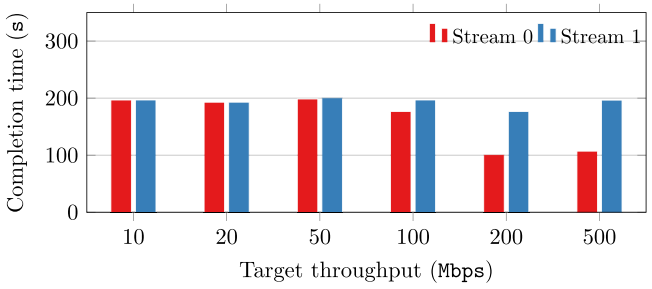


Fig. 8. Impact of the target throughput over the completion time. Link delay and buffer size are set to 20 ms and 5 BDP respectively.

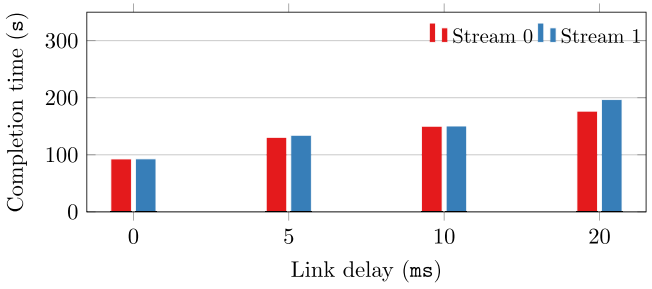


Fig. 9. Impact of the link delay over the completion time. Target throughput and buffer size are set to 100 Mbps ms and 5 BDP respectively.

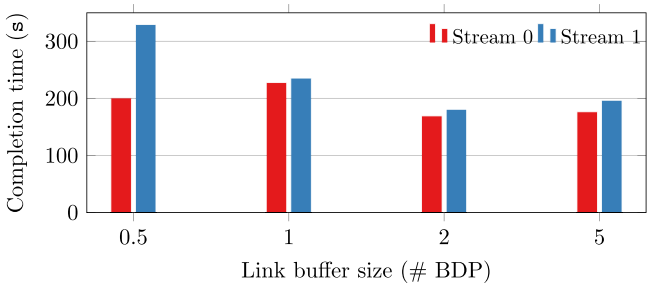


Fig. 10. Impact of the link buffer size over the completion time. Target throughput and link delay are set to 100 Mbps ms and 20 ms respectively.

relevant for the first stream, due to the use of the ABS scheme with a strict target.

Then, Fig. 9 shows the results that were observed when we fixed the target throughput and buffer length, while varying the link delay; in this case we see an increase in the completion time, as the estimated bandwidth had been reduced. Finally, in Fig. 10, we modified the BDP, by sweeping the buffer length. As it was increased, the estimated bandwidth was larger, resulting in a reduction in the overall completion time.

We can thus conclude that, at a reasonable cost, the proposed strategy of shifting between scheduling policies yields the behavior that was expected. We use metrics that are accessible from the QUIC layer, thus ensuring the feasibility of the proposed solution, and we use them to decide the scheduler that guarantees the expected performance. When the threshold is strict, the ABS is heavily used, granting more resources to the highest priority stream, resulting in a better performance (shorter completion time). On the other hand, when the threshold is looser, the proposed solution fosters the RR scheduler, avoiding the starvation of the non-priority stream, even if the ones having a highest priority are satisfied (see Figs. 9 and 10).

5. Conclusions

In this paper, we designed and implemented a new stream manager, including three scheduling strategies, to prioritize traffic in QUIC. We used a GO-based implementation (`quic-go`) in which we added the possibility for the user to choose the scheduler, as well as the priority assigned to each stream. All the code has been made available in a public git repository.

We evaluated QUIC performance with absolute prioritization (ABS), Weighted Fair Queuing (WFQ) and the default Round Robin (RR) schedulers in two complementary scenarios. The first one considered time-sensitive traffic using real communication traces from a drone-controller. This traffic is prioritized and competes with other background flows. To carry out experiments and compare the performance of the proposed techniques, the methodology exploited in this first setup entails both the `ns-3` simulator and Docker containers. Using the same testbed, a second scenario has been considered, focusing on the impact of varying message lengths. Analyzing both scenarios, we observe that the gain of our proposed scheme strongly depends on the size of the prioritized message. Applications that prioritize flows with relatively long messages (multiple QUIC packets), exhibit much lower delays than the baseline scheduler. To address the unstable performance of wireless links, we proposed a new adaptive scheduler, which shifts between different solutions based on the current situation. To evaluate its performance we defined a different scenario, based on mmWave technology, which has a relevant role in 5G and B5G systems, in environments with interference. We fed the traces of bandwidth variation in such scenario to the Mahimahi emulator, to evaluate the improvements brought to the QUIC performance thanks to the new scheduling policy.

As a first assessment of the stream scheduler, this paper demonstrates the feasibility and relevance of appropriate stream management, according to the particular requirements of the applications. By using a suitable scheme, the latency of prioritized messages can be significantly reduced. According to the definition of the ABS scheduler provided in this work, it was specifically conceived to be used for highly prioritized streams, such as those coming from applications and services with critical data flows, for instance, an unmanned vehicle exchanging navigation control information with its operator. These services do not normally require high traffic loads, so it is not likely that they would block other streams. In any case, we could extend the behavior of the proposed strategy, by including a mechanism to increase the priority of particular streams, if packets have been waiting for long times. Additionally, as a second assessment, we have assessed the performance of an adaptive scheduling policy over a mmWave environment, which is known for its high variability. The proposed scheme is able to adjust its behavior to the channel conditions, based on two parameters (estimated bandwidth, and the RTT) that are accessible at the QUIC layer, to guarantee the feasibility, adhering to a specified target threshold and prioritization configuration. The experiments, carried out over a scenario with different buffer lengths and delays, evince that the proposed scheme is able to improve the performance, according to an indicated target value. More stringent configurations would prioritize certain streams, whereas less strict thresholds distribute resources equally. Additionally, we have seen that an increase of the buffer length leads to an overall completion time reduction. On the other hand, when suffering from longer communication delays, the proposed scheme fosters the ABS scheduler, to avoid hindering the performance of the highest priority streams.

In our future work, we will look at the interplay between multi-stream scheduling with both multi-path and congestion control mechanisms. We will also consider the integration of additional schedulers, in particular looking at strategies that consider the delay as the main parameter. We will study adapting their behavior to the corresponding application and services, as well as to their particular requirements. In this sense, the adaptative approach that has been proposed in the paper is a first step. In the future we plan to broaden the potential configuration possibilities, and to consider AI techniques to establish the optimum setup, depending on the particular circumstances.

CRediT authorship contribution statement

Fátima Fernández: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Fátima Khan:** Data curation, Formal analysis, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Mihail Zverev:** Conceptualization, Formal analysis, Methodology, Writing – review & editing. **Luis Diez:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Supervision, Visualization, Writing – review & editing, Writing – original draft. **José R. Juárez:** Investigation, Supervision. **Anna Brunstrom:** Conceptualization, Methodology, Supervision. **Ramón Agüero:** Conceptualization, Formal analysis, Investigation, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The authors appreciate the funding of the Industrial PhD Program from University of Cantabria (2020 Call). This work has been partially supported by the Spanish Government (Ministerio de Economía y Competitividad, Fondo Europeo de Desarrollo Regional, FEDER) with the project SITED: Semantically-enabled Interoperable Trustworthy Enriched Data-spaces (PID2021-125725OB-I00), the Government of Cantabria, Fondo Europeo de Desarrollo Regional (FEDER): Grants for research projects with high industrial potential of technological agents of excellence for industrial competitiveness (TCNIC) program (2023/TCN/002), as well as by the Basque Government through the Elkartek research program under the SONETO project (Ref. KK-2023/00038). The work of Anna Brunstrom was partially supported by the Knowledge Foundation of Sweden under project grant 20220072-H-01.

References

[1] S. Li, L.D. Xu, S. Zhao, The internet of things: a survey, Inf. Syst. Front. 17 (2) (2015) 243–259.
[2] G.P. Fettweis, The tactile internet: Applications and challenges, IEEE Veh. Technol. Mag. 9 (1) (2014) 64–70.
[3] M. Okano, Y. Hasegawa, K. Kanai, B. Wei, J. Katto, TCP throughput characteristics over 5G millimeterwave network in indoor train station, in: 2019 IEEE Wireless Communications and Networking Conference, WCNC, IEEE, 2019.
[4] V. Dang, Benefits of 5G millimeter-wave communication in IoT applications, 2022.
[5] X. Lin, S. Rommer, S. Euler, E.A. Yavuz, R.S. Karlsson, 5G from space: An overview of 3GPP non-terrestrial networks, IEEE Commun. Stand. Mag. 5 (4) (2021) 147–153.
[6] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, C. Bettstetter, Drone networks: Communications, coordination, and sensing, Ad Hoc Netw. 68 (2018) 1–15, URL https://www.sciencedirect.com/science/article/pii/S1570870517301671. Advances in Wireless Communication and Networking for Cooperating Autonomous Systems.
[7] S. Kota, G. Giambene, Satellite 5G: IoT use case for rural areas applications, in: Proceedings of the Eleventh International Conference on Advances in Satellite and Space Communications-SPACOMM, 2019, pp. 24–28.
[8] N. Akshatha, K.H.J. Rai, H. MK, R. Ramesh, R. Hegde, S. Kumar, Tactile internet: Next generation IoT, in: 2019 Third International Conference on Inventive Systems and Control, ICISC, 2019, pp. 22–26.
[9] A. Ebrahimzadeh, M. Maier, R.H. Glitho, Trace-driven haptic traffic characterization for tactile internet performance evaluation, in: 2021 International Conference on Engineering and Emerging Technologies, ICEET, 2021, pp. 1–6.
[10] J. Iyengar, M. Thomson, QUIC: A UDP-based multiplexed and secure transport, 2021, http://dx.doi.org/10.17487/RFC9000, URL https://www.rfc-editor.org/info/rfc9000. RFC 9000.
[11] F. Fernández, M. Zverev, L. Diez, J.R. Juárez, A. Brunstrom, R. Agüero, Flexible priority-based stream schedulers in QUIC, in: Proceedings of the Int'L ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, in: PE-WASUN '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 91–98.
[12] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, Mahimahi: accurate record-and-replay for HTTP, in: Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, in: USENIX ATC '15, USENIX Association, USA, 2015, pp. 417–429.
[13] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, Z. Shi, The QUIC transport protocol: Design and internet-scale deployment, in: Proc. of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17, Association for Computing Machinery, 2017, pp. 183–196.
[14] M. Thomson, Version-independent properties of QUIC, 2021, http://dx.doi.org/10.17487/RFC8999, URL https://www.rfc-editor.org/info/rfc8999. RFC 8999.
[15] M. Thomson, S. Turner, Using TLS to secure QUIC, 2021, http://dx.doi.org/10.17487/RFC9001, URL https://www.rfc-editor.org/info/rfc9001. RFC 9001.
[16] J. Iyengar, I. Swett, QUIC loss detection and congestion control, 2021, http://dx.doi.org/10.17487/RFC9002, URL https://www.rfc-editor.org/info/rfc9002. RFC 9002.
[17] M. Bishop, HTTP/3, 2022, http://dx.doi.org/10.17487/RFC9114, URL https://www.rfc-editor.org/info/rfc9114. RFC 9114.
[18] T. Pauly, E. Kinnear, D. Schinazi, An unreliable datagram extension to QUIC, 2022, http://dx.doi.org/10.17487/RFC9221, URL https://www.rfc-editor.org/info/rfc9221. RFC 9221.
[19] Y. Liu, Y. Ma, Q.D. Coninck, O. Bonaventure, C. Huitema, M. Kühlewind, Multipath Extension for QUIC, Internet-Draft draft-ietf-quic-multipath-04, Internet Engineering Task Force, 2023, URL https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/04/. Work in Progress.
[20] X. Shi, L. Wang, F. Zhang, B. Zhou, Z. Liu, Pstream: Priority-based stream scheduling for heterogeneous paths in multipath-QUIC, in: 2020 29th International Conference on Computer Communications and Networks, ICCCN, 2020, pp. 1–8.
[21] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, R. Steinmetz, Multipath QUIC: A deployable multipath transport protocol, in: 2018 IEEE International Conference on Communications, ICC, 2018, pp. 1–7.
[22] P.S. Kumar, N. Fatima, P. Saxena, Performance analysis of multipath transport layer schedulers under 5G/B5G hybrid networks, in: 2022 14th International Conference on COMmunication Systems & NETworkS, COMSNETS, 2022.
[23] A. Rabitsch, P. Hurtig, A. Brunstrom, A stream-aware multipath QUIC scheduler for heterogeneous paths, in: Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC, EPIQ '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 29–35.
[24] F. Chiariotti, A.A. Deshpande, M. Giordani, K. Antonakoglou, T. Mahmoodi, A. Zanella, QUIC-EST: A QUIC-enabled scheduling and transmission scheme to maximize VoI with correlated data flows, IEEE Commun. Mag. 59 (4) (2021) 30–36.
[25] C. Hervella, L. Diez, F. Fernández, N.J. Hernández Marcano, R. Hylsberg Jacobsen, R. Agüero, Realistic assessment of transport protocols performance over LEO-based communications, in: Proceedings of the 19th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, in: PE-WASUN '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 91–98.
[26] C. Cui, Y. Lu, S. Li, J. Li, Z. Ruan, DASH+: Download multiple video segments with stream multiplexing of QUIC, in: 2022 Tenth International Conference on Advanced Cloud and Big Data, CBD, 2022, pp. 66–72.
[27] N. Rozen-Schiff, A. Navon, I. Pechtalt, L. Bruckman, Y. Boyuan, AQUA: Adding bandwidth allocation to QUIC for metaverse multi-stream applications, in: 2023 IEEE International Conference on Metaverse Computing, Networking and Applications, MetaCom, 2023, pp. 762–768.
[28] C. Hervella, L. Diez, F. Fernández, N.J. Hernández Marcano, R. Hylsberg Jacobsen, R. Agüero, Realistic assessment of transport protocols performance over LEO-based communications, in: Proceedings of the 19th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks, in: PE-WASUN '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 91–98.
[29] F. Khan, C. Hervella, L. Diez, F. Fernández, N.J.H. Marcano, R.H. Jacobsen, R. Agüero, Realistic assessment of transport protocols performance over LEO-based communications, Comput. Netw. 236 (2023) 110008, URL https://www.sciencedirect.com/science/article/pii/S138912862300453X.
[30] F. Khan, F. Fernández, L. Diez, R. Aguero, Exploiting QUIC multi-streaming over NTN: Delay-based scheduling policies, in: GLOBECOM 2023 - 2023 IEEE Global Communications Conference, 2023, pp. 5433–5438.
[31] M.J. Neely, Stochastic Network Optimization with Application to Communication and Queueing Systems, Springer International Publishing, 2010.

[32] L. Ferranti, F. Cuomo, S. Colonnese, T. Melodia, Drone cellular networks: Enhancing the quality of experience of video streaming applications, Ad Hoc Netw. 78 (2018) 1–12, URL https://www.sciencedirect.com/science/article/pii/S157087051830204X.
[33] L. Clemente, M. Seemann, A QUIC implementation in pure Go, 2022, URL https://github.com/quic-go/quic-go. (Accessed 20 February 2023).
[34] C. Crochet, T. Rousseaux, M. Piraux, J.-F. Sambon, A. Legay, Verifying QUIC implementations using ivy, in: Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC, EPIQ '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 35–41.
[35] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, R. Scheffenegger, CUBIC for fast long-distance networks, 2018, http://dx.doi.org/10.17487/RFC8312, URL https://www.rfc-editor.org/info/rfc8312. RFC 8312.
[36] A QUIC implementation in pure go: quic-go/internal/congestion/pacer.go, 2024, https://github.com/quic-go/quic-go/blob/master/internal/congestion/pacer.go#L25. [Online; (Accessed 15 May 2024)].
[37] M. Mezzavilla, S. Dutta, M. Zhang, M.R. Akdeniz, S. Rangan, 5G MmWave module for the ns-3 network simulator, in: Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '15, ACM, New York, NY, USA, 2015, pp. 283–290.
[38] S. Jeddou, L. Diez, N. Abdellah, A. Baina, R. Agüero, On the performance of transport protocols over mmwave links: Empirical comparison of TCP and QUIC, IEEE Open J. Commun. Soc. 4 (2023) 2596–2608.
[39] A. Banks, E. Briggs, K. Borgendale, R. Gupta, MQTT Version 5.0, standard, Organization for the Advancement of Structured Information Standards (OASIS), 2019.

Fátima Fernández received the degree in telecommunication technologies engineering and the M.Sc. degree in telecommunication engineering from the University of Cantabria, Santander, Spain, in 2017 and 2019, respectively. She also received the Ph.D. from the University of Cantabria in 2024 in collaboration with the Ikerlan Technology Research Centre. She is currently a smart connectivity researcher at Ikerlan. In 2019, she was a Fellow Researcher with the Communications Engineering Department, University of Cantabria. Her area of work is mainly centered around transport layer protocols, wireless networks and Internet of Things applications.

Fátima Khan received her B.Sc. and M.Sc. from University of Cantabria in 2021 and 2023 respectively. Since 2022 she is a researcher with the Communications Engineering Department at the University of Cantabria. Her research focuses on transport layer optimization in non-terrestrial-networks (NTN) for 5G and B5G architectures.

Mihail Zverev received the degree in telecommunications engineering from University of Cantabria, Santander, Spain, in 2014, and the M.Sc. degree in project management from La Salle, University of Ramón Llull, Barcelona, in 2017. He received his Ph.D. from the University of Cantabria with the collaboration of the Ikerlan Technology Research Centre in 2023. He is currently a smart connectivity researcher at Ikerlan.

Luis Diez received his M.Sc. and Ph.D. from University of Cantabria in 2013 and 2018 respectively. He is currently Assistant Professor at the Communications Engineering Department in that University. He has been involved in different international and industrial research projects. His research focuses on future network architectures, resource management in wireless heterogeneous networks, and IoT solutions and services. He has published more than 40 scientific and technical papers in those areas, and he has served as TPC member and reviewer in a number of international conferences and journals. As for teaching, Dr. Diez has supervised 15 B.Sc. and M.Sc. Thesis, and he teaches in courses related to cellular networks, network dimensioning and service management.

José R. Juárez Rodríguez is a Spanish telecommunication engineer with Ph.D. in Computer Science. Currently, he is a smart connectivity senior researcher at Ikerlan. Experience with R&D projects in the distributed computing area. Good technical background on wireless technologies such as GSM, GPRS, UMTS, HSDPA, Bluetooth, Wifi, WiMAX, LTE, RFID, NFC, etc. and different programming languages such as Java, .NET, C, C++ and Pascal. Deep knowledge in unix and databases administration, networking and ethic hacking.

Anna Brunstrom received a Ph.D. and M.Sc. in Computer Science from College of William & Mary, VA, in 1996 and 1993, respectively. She received a B.Sc. in Computer Science and Mathematics from Pepperdine University, CA, in 1991. She joined the Department of Computer Science at Karlstad University (KAU), Sweden, in 1996, where she is currently a Full Professor and Head of the Distributed Intelligent Systems and Communications Research Group (DISCO). She is also a part-time Distinguished Researcher in the Institute of Software Engineering and Technologies (ITIS) at University of Malaga, Spain. She has a background in distributed systems, but her main area of work over the last years has been in computer networking with a focus on Internet architectures and protocols, techniques for low latency communication, multi-path communication and design and performance evaluation of mobile systems including 5G/6G.

Ramón Agüero Calvo received his M.Sc. in Telecommunications Engineering (1st class honors) from the University of Cantabria in 2001 and the Ph.D. (Hons) in 2008. He is currently a Professor at the Communications Engineering Department in that university. His research focuses on future network architectures, especially regarding the (wireless) access part of the network and its management. He has published more than 230 scientific papers in such areas and he is a regular TPC member and reviewer on various related conferences and journals. Ramon Aguero serves in the Editorial Board of IEEE Communication Letters (Area Editor since 2023), IEEE Open Access Journal of the Communications Society, IEEE Systems Journal, Wireless Networks (Springer). Dr. Aguero has supervised 7 PhDs and more than 80 B.Sc. and M.Sc. thesis. He is the main instructor in courses dealing with Networks, and Traffic Modeling, both at B.Sc. and M.Sc. levels. Since 2016, he is the Head of the IT Area (deputy CIO) at the University of Cantabria.