Empowering QUIC with Connection Delegation to Save Energy

Vany V. Ingenzi
UCLouvain
vany.ingenzi@uclouvain.be

Louis Navarre
UCLouvain
louis.navarre@uclouvain.be

Paul-Edouard Bertrand Van Ouytsel
UCLouvain
paul-edouard.bertrand@uclouvain.be

Julien Montavont
University of Strasbourg
montavont@unistra.fr

Maxime Piraux
UCLouvain
maxime.piraux@uclouvain.be

Olivier Bonaventure
UCLouvain & WELRI
olivier.bonaventure@uclouvain.be

Abstract

The end-to-end principle is fundamental to the design of the Internet. It mandates that hosts participating in a transport connection must be able to process packets continuously. This principle conflicts with the energy-saving objectives of battery-powered wireless devices, which benefit from short, intermittent usage patterns.

In this paper, we propose a QUIC delegation mechanism that enables a client to move an existing connection to a trusted third party by transferring the connection state. The third-party host assumes responsibility for the connection and data exchange on behalf of the original client, allowing the client to enter a power-saving mode. We design and implement it on two different QUIC implementations to validate its interoperability. Our evaluation shows that QUIC Connection Delegation can save up to 55 % of energy when the smartphone can leverage existing sleep mode techniques, and 25 % considering background traffic and an active display.

1 Introduction

The TCP/IP protocol suite was designed under several assumptions [1]. At that time, energy consumption was not a concern, and minimizing energy was not an initial objective of the Internet architecture. Endpoints could not make any assumptions about the time of packet reception; therefore, they must remain active at all times. The situation changed with the introduction of battery-powered and wireless devices. Several techniques have been proposed to optimize the energy consumption of wireless networks [2, 3]. However, these energy-saving techniques often interact with the fundamental behavior of the Internet protocol suite, yielding only incremental improvements.

Modern smartphones support idle power modes, such as Android's Doze [4]. The smartphone OS monitors the device usage and decides when to put the platform to sleep. When an Android phone is in Doze mode, the system restricts apps' access to the network and CPU-intensive services. Chen et al. [5] highlights the energy cost of background activities and idle states, showing that Wi-Fi scans and beaconing alone account for roughly 17 % of a device's daily energy consumption. Researchers and the Wi-Fi industry have developed and deployed several power-saving techniques that allow Wi-Fi devices to put their network interfaces to sleep while waiting for frames [6]. In a nutshell, the Wi-Fi access point buffers frames intended for a sleeping device. When the device wakes up, it informs the access point, which delivers the buffered frames. This works when a few frames are waiting, but it is not suitable for reliable transport protocols such as TCP and QUIC that adhere to the end-to-end principle [7] and incorporate congestion control schemes that adjust the instantaneous throughput according to the network conditions. When such connections are active, devices must process each packet and return acknowledgments to allow the transfer to continue.

In this paper, we propose *Connection Delegation*, a novel approach and radical shift to reduce the energy consumption of battery-powered wireless devices, **by allowing an endpoint to delegate one or more QUIC connec-**

tions to a trusted delegate in its network. We build upon two key observations:

1. From an energy perspective, wireless media (Wi-Fi and Cellular) are significantly more efficient when transferring data at high rates than at low rates [8, 9];
2. In modern domestic networks, the wireless LAN's throughput is hindered by the access network.

We elaborate on these observations further in Section 2. When the transport connection has a low throughput, the endpoint of the connection (in our example, a smartphone) moves the connection to another, non-battery-powered device, which continues the transfer while the smartphone goes to sleep. We refer to the second device as a *delegate*. At the end of the transfer, the delegate wakes up the smartphone and transfers all received data at a higher throughput (observation 2), thereby reducing the smartphone's energy consumption (observation 1). While this mechanism is protocol-agnostic, we focus on QUIC [10] to leverage its migration feature, which is already widely deployed [11], so that our solution is entirely transparent to the server.

Our results show that when the server throughput is lower than the Wi-Fi bandwidth, smartphones performing screen-off HTTP/3 downloads benefit increasingly from delegation: the longer the transfer, the greater the energy savings. For instance, QUIC Delegation reduces the smartphone's total energy consumption by 23 J (55 %) on a 95 MB transfer, while introducing an average additional delay of 9 seconds (17 %). These savings occur because (i) the delegated transfer completes at a higher throughput, thereby reducing the time the device spends in its high-power download state. Additionally, (ii) when the device finishes the transfer, it enters an idle mode that is significantly less power-consuming than when it is transmitting. Such gains are particularly relevant for large background transfers, such as software updates or cloud storage synchronization, which typically occur during user inactivity.

Contributions. We make the following contributions:

- We design ***QUIC Delegation***, allowing a host to delegate an ongoing QUIC connection to a trusted delegate.
- We implement QUIC Delegation in Cloudflare *quiche* [12] and evaluate the benefits of the approach on smartphones in a realistic testbed. We show that **it reduces device total energy consumption by up to** 55 % during screen-off HTTP/3 downloads, and up to 20 % while using the screen.
- We specify the QUIC Delegation mechanism (e.g., how to (de)serialize the QUIC state to/from the delegate) to enable adoption by any QUIC stack. We implement in another open-source stack, *aioquic* [13].

We will release all software artefacts, including our two implementations, upon publication of this paper. This work does not raise any ethical issues.

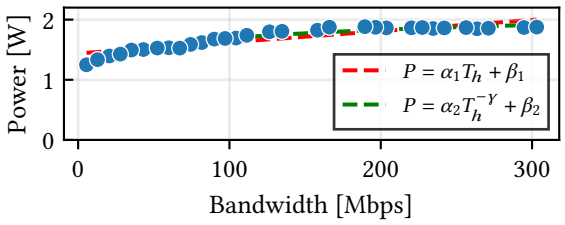


Figure 1: *Power consumption during transmission.*

Figure 2: *Power consumption and energy efficiency of Pixel7a during data transmission on Wi-Fi with various bandwidths. On both figures, we display a linear model [8, 14] (red) and the generic model proposed by [15] (green).*

2 Motivation

Section 2.1 reviews the factors influencing the energy consumption of smartphones during data transfers over Wi-Fi. We focus on smartphones due to their massive adoption, but this approach also works with other battery-powered devices. Our results, as well as the literature, indicate that *the faster the data transfer, the better the energy efficiency*. Section 2.2 shows that the throughput across the Internet rarely matches that in Wi-Fi networks. Section 2.3 details the Connection Delegation mechanism, building upon these two hypotheses.

2.1 Energy consumption of mobile devices

Power consumption in smartphone communications has been extensively studied [16, 5, 14, 15, 8]. Gupta et al.

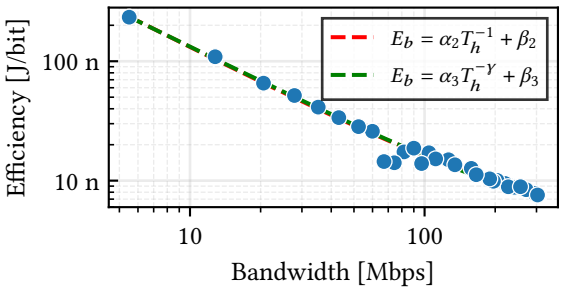


Figure 3: *The energy per bit during transmission.*

Figure 4: *Power consumption and energy efficiency of Pixel7a during data transmission on Wi-Fi with various bandwidths. On both figures, we display a linear model [8, 14] (red) and the generic model proposed by [15] (green).*

[16] report that during a 720p YouTube streaming, Wi-Fi accounts for 42-65 % of the total device power consumption, depending on the device model. Friedman et al. [14] analyze the trade-offs between Wi-Fi and Bluetooth for file transfers, showing that Wi-Fi is generally more energy-efficient due to its higher data-per-power ratio. They also observe an increasing linear relationship between Wi-Fi power consumption and throughput, noting that *the power consumption of a smartphone Wi-Fi chip heavily depends on the time it is active, and only slightly on the provided throughput*. Sun et al. [15] propose a generic model for estimating energy efficiency as a function of data rate, demonstrating higher efficiency at higher throughput. Other works have proposed similar linear models between power consumption and throughput [8].

To ensure a similar hypothesis in our environment, we measure the energy consumption of the Google Pixel 7a, which we will use throughout all of our experiments. Figure 8 presents our experimental setup. The device performs HTTP/3 downloads from a server in our campus network through an 802.11 5 GHz Wi-Fi access point. We vary the available network bandwidth by applying shaping at the server's egress and perform 20-second-long transfers; higher bandwidth experiments result in larger download sizes.

Energy efficiency. Figure 1 shows the power consumption as a function of the bandwidth. The Figure shows a correlation between the throughput and the power consumption, aligning with the results proposed by previous work [8, 14, 15]. We observe a 51.6 % increase in power consumption (from 1.24 W to 1.88 W) when the bandwidth increases by 5405 % (from 5.5 to 303 Mbps). The power consumption of the smartphone slightly increases with the bandwidth. In contrast, the potential power savings due to the shorter transfer duration are much more significant[1].

Figure 3 shows the energy efficiency (J/bit), calculated as the total device energy consumption divided by the overall data transferred. We observe an inverse proportional decrease in energy per bit, consistent with previous work [8, 14, 15]. Energy efficiency improves significantly at higher throughput: at 303 Mbps, the smartphone consumes 31× less energy per bit (7.4 nJ) compared to 5.5 Mbps (233 nJ). We fit a reciprocal function of the linear model shown in Figure 1 (red) with the general model proposed by Sun et al. [15] (green).

Wi-Fi chip power consumption. The hypothesis from Friedman et al. [14] states that, from an energy-saving perspective, *faster transmissions are always better in the presence of idle power*. To assess this conclusion in our setting, we downloaded a 12.5 MB file[2] on our Google Pixel 7a (following the same setup) at $\{5, 20, 100, 200\}$ Mbps, while monitoring power consumption over a 30 s window. Figure 5 reports the total energy consumed by the device, as measured by the Monsoon HVPM (High Voltage Power Monitor), together with per-component energy consumption obtained from the ODPM (On-Device Power Monitor) rails [17]. Since the ODPM rails do not account for all power consumption and present conversion inefficiencies at high power consumption [18], we also include the difference with the Monsoon (*Diff Monsoon*).

As expected, for a fixed download size, higher throughput leads to lower energy consumption: from 30 J at 5 Mbps to 9 J at 200 Mbps. The Figure also highlights that substantial saving comes from the Wi-Fi chip (*WiFi*). Because faster transfers complete more quickly, the device *potentially*[3] enters a low-power mode sooner. All together, the results from Figure 3 and 5 confirm our first hypothesis: **higher throughput leads to a greater en-**

[1] For a fixed transfer size, increasing the throughput by 100× should lead to a ∼ 100× shorter duration.

[2] We chose this size to limit the 5 Mbps experiment to a 30 s transfer.

[3] Whether the device enters low-power mode also depends on the operating system.

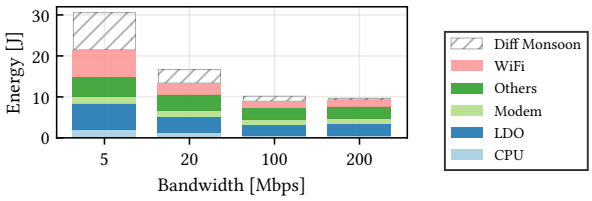


Figure 5: *The total energy consumption of a 12.5 MB with a measurement window of 30 seconds.*

ergy efficiency since the device (notably Wi-Fi chip) must stay active for a shorter time.

2.2 Achievable server throughput

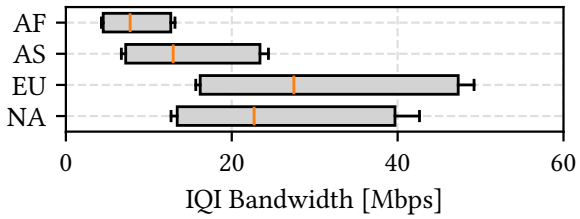


Figure 6: *Cloudflare IQI Bandwidth from January to July of 2025.*

From an energy efficiency viewpoint, the previous section shows that a smartphone should transfer data at the highest possible throughput. However, the Internet is a shared infrastructure, and servers often limit their throughput for various reasons [19, 20], thereby decreasing the potential energy-saving benefits. To illustrate this, we analyze Cloudflare's Internet Quality Index (IQI) [21], which represents the measured throughput of real connections, such as web browsing. Figure 6 shows the 25^{th}, 50^{th}, 75^{th} percentiles of IQI downloads across Africa (AF), Asia (AS), Europe (EU), and North America (NA) for the first 7 months of 2025. We observe that the median download throughput is 7, 12, 27, and 22 Mbps for Africa, Asia, EU, and NA, respectively. Similarly, for the 75^{th} percentile, throughputs are respectively 12, 23, 47 and 39 Mbps. Hence, **only a fraction of Internet users can exchange data with Internet servers at a throughput matching the highest available Wi-Fi speed** (100-253 Mbps [22]), thus confirming our second hypothesis.

2.3 The case for Connection Delegation

Building upon the two key observations (Section 2.1 and 2.2), this paper introduces the concept of *QUIC Connection Delegation*, as Figure 7 illustrates. A Connection Delegation is a three-step process. **Delegation.** A client can redirect the processing of an established connection to a trusted delegate. To do so, the host *serializes* its connection state (and optionally data that needs to be transmitted) and sends it to the delegate, then *can* enter a low-power mode to save energy. **Transfer.** The delegate fully operates the transport connection *in place of* the client, using the serialized state, and continues the transfer at limited throughput with the server (hypothesis 2). *QUIC Connection Delegation* supports both downloads and uploads. During this transfer, the client is asleep. **Retrieval.** Once the transfer is complete, the delegate *wakes up* the client, which retrieves the connection state and downloads the data at a significantly higher throughput (hypothesis 2). Hence, it reduces the total amount of time its wireless interface is awake, ultimately leading to energy savings (hypothesis 1). The following Section details the QUIC Connection Delegation mechanism, which does not require any server modification thanks to QUIC's *connection migration*.

3 Delegating a QUIC connection

QUIC [10] is a modern, reliable transport protocol running atop UDP. It is used to support HTTP/3 [23], which provides the same semantics as HTTP/2 [24]. QUIC combines the security of TLS 1.3 and the modern reliability mechanisms of TCP. QUIC packets can be seen as *frame containers*. Each frame type embeds the logical parts of the protocol. QUIC hosts do not retransmit lost packets; instead, they send the (reliable) lost frames in new packets with increased packet numbers. While TCP uses the 4-tuple to identify a connection, QUIC uses Connection Identifiers (CIDs) included in the QUIC header. On each connection, each QUIC host allocates one or more CIDs that the peer can use to identify the connection. This mechanism allows a QUIC host to change its source address without breaking the established connection using the *Connection Migration* procedure illustrated in Figure 15 (Section A). To migrate, a host initiates a *path probing* phase, ensuring the peer that migration is

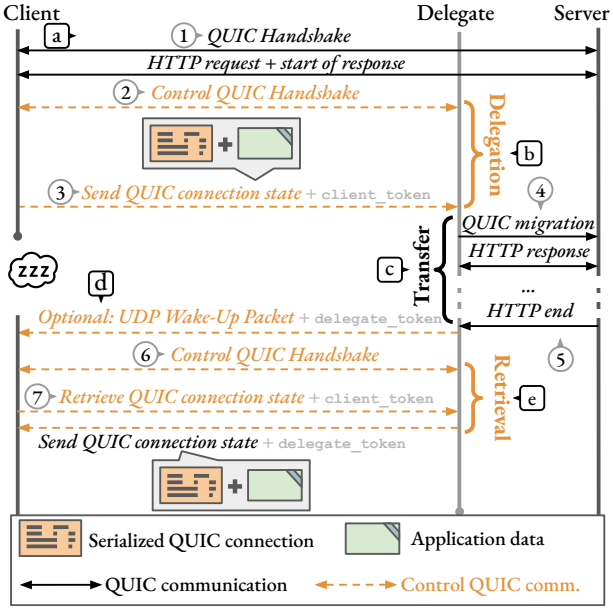


Figure 7: *A client starts a QUIC connection with a server. As the transfer is slow, it moves the connection to a delegate that continues the data transfer while the client enters power-save mode. The client wakes up and interacts with the delegate to retrieve the connection state and all data exchanged.*

not an attempt to attack a third-party [10]. Once verified, the whole connection migrates to the new path, stopping packet exchange on the older one.

To enable QUIC clients to sleep while slow data transfers execute, we propose a technique that enables such a client to **delegate the execution of a connection to a trusted host** that we call a **delegate**. We assume that the client fully trusts the delegate. For example, the delegate of a smartwatch could be a computer or Wi-Fi access point owned by the same user. We leverage QUIC's connection migration to delegate the client side of the connection to a delegate. Hence, connection migration is the ***only*** server-side requirement for our solution. This standard feature is already widely deployed [25]. Later we discuss how QUIC clients could delegate connections to less trusted devices by using application-layer encryption, similar to the approach proposed by Kosek et al. [26].

QUIC Connection Delegation. Figure 7 illustrates our QUIC connection delegation mechanism, which we further detail in the following subsections. We first consider an example and focus on the main steps highlighted by letters (a) to (d). In this example, ((a)) a smartphone initiates a long file download with a server. To preserve its battery during the long download, the client sends the HTTP/3 request to the server and, after receiving some bytes (e.g headers), opts to delegate the established connection to a trusted delegate. The process requires the client to write the application data (if any) to the QUIC layer, then stops processing, sending, or receiving incoming packets on the QUIC connection, and *serializes the connection state*. Then, ((b)) it contacts the *delegate* and *sends the serialized connection state*. At this point, the smartphone is no longer an active endpoint of the QUIC connection, and it can put its wireless interface in a power-save mode to preserve its battery. ((c)) The *delegate* uses the *connection state to derive the state of the connection*, then it performs a QUIC connection migration to the server. Once the connection migration finishes, the delegate fully handles the client side of the connection: it sends and receives application data with the server in place of the client. The connection delegation is seamless to the server as it only sees a standard migration caused by a change in the client's IP address. The delegate ensures reliable delivery and respects flow control limits and congestion control as any QUIC client [10]. In the example from Figure 7, ((d)) the delegate wakes the client once the HTTP/3 response body is fully received. ((e)) The smartphone contacts the delegate to retrieve the QUIC connection and all the received data. If the client retrieves the connection earlier than the end of the transfer, it can issue a new connection migration to the server to become the transfer's endpoint device again.

The next subsections thoroughly answer the following questions: (3.1) How to represent the state of a QUIC connection? (3.2) How can a client safely delegate a QUIC connection? (3.3) How can a client safely resume a delegated connection?

3.1 QUIC connection state

The QUIC protocol specification [10, 27, 28] contains many requirements regarding the behavior of QUIC clients and servers but does not explicitly specify the state composing a QUIC connection. This abstraction emphasizes QUIC's behavior and lets implementers architect their implementations appropriately to their requirements.

To cope with the diversity of QUIC stacks, we propose a *portable QUIC connection state* format. This format represents the binary state of an ongoing QUIC connection from the perspective of a single host. This is similar to Linux's `TCP_REPAIR` feature [29], this format captures the state of an ongoing QUIC connection in a portable binary representation. However, unlike TCP, the QUIC state machine requires more than just sequence numbers, packets in the queue, and options. To demonstrate the feasibility of delegating connections between different QUIC stacks, we implement it in two very different QUIC implementations: `quiche` (`commit` dfeaa589), a production-ready implementation written in Rust and developed by Cloudflare, and `aioquic` (`commit` 9bc1e43), a minimal implementation of TLS 1.3, QUIC and HTTP/3 written in Python. These QUIC implementations maintain different types of state information, however they fully implement the QUIC protocol, including connection migration. Due to space limitations, we summarize the information retained in our QUIC connection state representation. The details are provided in Section B.

Packet Number Spaces. We store information including the received packet numbers that need to be acknowledged, the set of already received packet numbers or the largest packet number received in the Application data packet Number space.

Transport parameters. QUIC transport parameters [10] are exchanged during the connection handshake to define key protocol settings, such as the maximum UDP payload size or acknowledgment delays, ...

Cryptographic Materials. We store the cryptographic secrets and the cipher suite negotiated during the handshake to enable packet encryption and decryption on the delegate. This ensures that the delegate derives the same encryption materials as the client. We transmit the cryptographic secrets, rather than just the derived keys, to allow the delegate to compute new secrets such as `client_token`, `delegate_token`, and perform key update if requested by the server.

Connection IDs. Connection IDs (CIDs) uniquely identify a QUIC connection on a peer [10]. These identifiers are crucial, as each $1-RTT$ packet contains a Destination Connection ID (DCID) to associate each packet with the corresponding connection at the receiver. When storing CID information, we record all non-retired CIDs from each peer, along with their reset tokens and sequence numbers. Connection delegation requires at least two active CIDs.

Flow and Congestion control. We store all the information required to manage the Stream and connection level flow controls on the delegated connection. QUIC endpoints perform congestion control per path [28]. Since QUIC hosts must reset their congestion control state upon connection migration [10], we do not store congestion control and Path MTU discovery information.

Our prototype encodes the QUIC connection state using MessagePack [30], an efficient, compact, and widely supported format that supports mappings with keys of any data type, aligning well with connection state needs. From empirical measurements, we observe that the size of the connection state when delegating is around 2.5 kB, without connection data.

Summary: Our portable QUIC connection state enables capturing a snapshot of an ongoing QUIC connection at any time after the handshake, from the perspective of a client. It is lightweight, complete, and fully compliant with QUIC version 1.

3.2 Transferring QUIC connection state

The connection delegation mechanism transfers the connection state between the client and the delegate. We use QUIC and HTTP/3 for this transfer (ⓑ and ⓔ), as illustrated on Figure 7.

To start the delegation process, the client initiates *control connection* with the delegate (②). Our prototype leverages HTTP/3 to authenticate both peers on this connection. The client authenticates the delegate using the certificate supplied during the connection handshake. This certificate can be pre-installed on both hosts prior to QUIC delegation, similar to SSH. The delegate can authenticate the client using Mutual TLS [31] or other HTTP/3 authentication schemes [32, 33, 34].

When this control connection is established, the client serializes the state of its connection with the server to the delegate (③), along with application data that must be transferred to the server. We name the initial connection to the server the *delegated connection*. At this point, the client immediately stops processing all packets received from the server. Otherwise, those packets could modify the connection state and jeopardize the connection that the delegate tries to establish. Finally, the client provides

a client_token to the delegate. The client derives this token using the TLS Exporter [35] with 0 as label from the exporter master secret of the initial connection with the server. This client_token acts as a proof of ownership of the initial connection. Since it is sent over a QUIC connection, a passive attacker cannot observe this token.

After receiving the delegated connection state, (④), the *delegate* resumes the delegated connection and performs a QUIC connection migration to bind it to its own address. The delegate continues the connection on behalf of the client, thus processing incoming QUIC packets, sending acknowledgments, and buffering/uploading application data (⑤). All these packets modify the state of the delegated QUIC connection on the delegate.

In our design, a client can resume a previously delegated connection in two ways, further detailed in Section 3.3. In Figure 7, the *delegate* signals the client that the transfer on the delegated connection has finished. For this, it sends a UDP packet to the client. To prevent attacks, the delegate proves that it knows the exporter master secret of the initial connection by sending the output of the TLS Exporter computed with 1 as a label. This delegate_token is in the UDP packet sent to the client.

Upon reception of this packet, the client verifies the value of the delegate_token by applying the TLS Exporter. The client then opens a new *control connection* with the delegate (⑥) and sends its client_token to prove that it owned the initial connection (⑦). The delegate answers with its delegate_token to prove its ownership of the delegated connection. At this point, the delegate stops processing the delegated connection, serializes its updated state, and sends it to the client on the control connection. Upon reception of the delegated connection state and application data, the client deserializes the original connection. It can resume then the connection with the server and initiate a new connection migration.

3.3 Signalling for connection resumption

Once the data transfer has finished, the delegate must return the control of the QUIC connection to the client. Our prototype currently supports two different mechanisms for connection resumption.

Timer Based. In this approach, the client specifies an *abstention period*, during which it commits to contact the delegate to retrieve the connection. The delegate, in turn, retains the connection state for a maximum duration of $2 \times$ *abstention period*. The delegate can discard the connection state at the end of this period without client retrieval. Additionally, if the requested *abstention period* exceeds the delegate's predefined threshold, the delegate may refuse the delegation request. To ensure seamless resumption, the client should select an *abstention period* shorter than the max_idle_timeout transport parameter negotiated during the initial handshake with the server.

Wake-Up Packet. In this mechanism, the client specifies an IP address and UDP port number in the HTTP/3 headers when initiating the control connection, allowing the delegate to contact it as needed. The delegate sends the delegate_token to this port, prompting the client to resume the connection upon receiving the expected payload. The delegate employs an exponential back-off strategy, retrying up to three times if the client does not respond. If the connection remains unclaimed, the delegate discards its state. While Wake-Up packets currently utilize UDP, alternative implementations may leverage low-power, out-of-band protocols such as IEEE 802.15.4 or Bluetooth.

Trade-off. A client must provide the necessary information for at least one resumption mechanism; otherwise, the delegate must reject the delegation request. The delegate can autonomously select the most appropriate method if both mechanisms are supported. The timer-based approach minimizes resource consumption by allowing the client to enter a low-power state during the *abstention period*. However, this method prevents real-time notifications from the delegate, limiting the client's ability to respond to unexpected connection events. Contrarily, the Wake-Up packet mechanism enables immediate client reaction, making it better suited for scenarios where timely reconnection is critical, such as premature connection termination by the server.

3.4 Extending QUIC delegation

The QUIC Delegation mechanism allows the delegate to access the application data. It is possible to reduce the level of trust that our delegation mechanism imposes on delegates by using two levels of encryption keys. In brief, the server and the client exchange two sets of security keys. The first set is used to encrypt and authenticate the QUIC packet headers and the control information. This

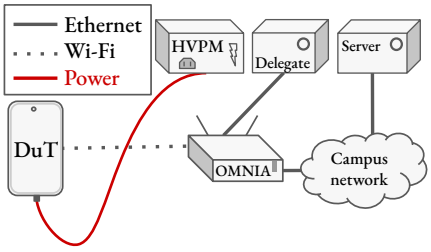


Figure 8: *Experimental testbed.*

Sc.	DUT ↔ Del.	{Del., DUT} ↔ Server	Bg. Traf.
#1	No limit	22 Mbps	No
#2	[22-132] Mbps	22 Mbps	No
#3	No limit	22 Mbps	Yes

Table 1: *Experimental scenarios using the setup from Figure 8.*

set is shared by the client when delegating a connection. The second set is used to encrypt and authenticate the payload of the QUIC frames and is not shared with the delegate. Due to space limitations and because this design requires protocol extensions on servers, we discuss it in more detail in this paper [36]. Our proposed QUIC delegation mechanism can also be applied to servers [36]; however, it requires the use of Multipath QUIC [37].

4 Evaluation

In this section, we leverage our implementation in Cloudflare's `quiche` due to its high performance [38] to evaluate their performance in our lab environment.

4.1 Experiment methodology

Environment. Figure 8 illustrates the environment we use across all our evaluations. The Device-under-Test (DuT) is a Google Pixel 7a smartphone (Tensor G2 SoC, 8 GB RAM, 4385 mAh Li-ion battery), running Android 14. The device supports Wi-Fi 802.11 a/b/g/n/ac/e. All experiments were carried out over the 5 GHz interface. Background services and unused radios (cellular, Bluetooth) were disabled to minimize confounding effects, unless otherwise specified. On the phone, we removed all preinstalled applications and services, so no network traffic is outside our control. The device is powered via USB by a High Voltage Power Monitor (HVPM), as the Pixel 7a has a non-removable battery. The delegate is an intel NUC[4] wired to the same Wi-Fi network as the DUT. The server is in our university's data center, and the campus network is shared, which explains the potential variance in some measurements.

Scenarios. We evaluate our solution under three scenarios represented in Table 1. In the first scenario, we measure the benefits of QUIC delegation when the throughput between the smartphone and the server is significantly lower than that of the Wi-Fi network. Then, we investigate QUIC Connection Delegation while varying the maximum bandwidth of the Wi-Fi network. Scenario #3 introduces background traffic, showing that our solution still presents benefits even if the smartphone does not completely enter a low-power mode when delegating. Given that Android only enters sleep mode when the device screen is turned off, we turn it off in the first two Scenarios. The last Scenario uses an active screen. We simulate bandwidth restrictions using `tc` on the delegate for the Wi-Fi bandwidth and the server for the server throughput. For each set of parameters, we perform both an end-to-end (E2E) HTTP/3 download, where the client downloads directly from the server, and a *delegated* HTTP/3 download, where the transfer is delegated. Throughout all experiments, we set the server throughput to 22 Mbps, as indicated by the results in Figure 6. All transfers are file-to-memory (the client doesn't write to disk) to avoid interference with disks or storage devices. We focus on downloads due to space limitations. Uploads provide similar results [36].

Power Measurement. We measure the power consumption with the Monsoon High Voltage Power Monitor [39]. We also rely on the On-Device Power Monitor (ODPM) rails [17] to gain insight into the power consumption of individual components. We use both tools since previous works [16, 18] have shown that the sum of the ODPM rails diverges from the battery counters or total device power consumption under heavy workload.

Metrics. For each measurement, we compare the energy consumed during an end-to-end (E2E) data transfer with the energy consumed during a delegated transfer. Figure 9 illustrates the client's perceived events, which we use to describe the following metrics:

(i) *Idle/Standby energy consumption* [J] is the energy

[4] Intel Corporation , running Linux kernel 6.11

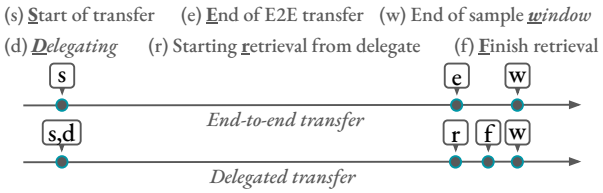


Figure 9: *Events that take place on the client.*

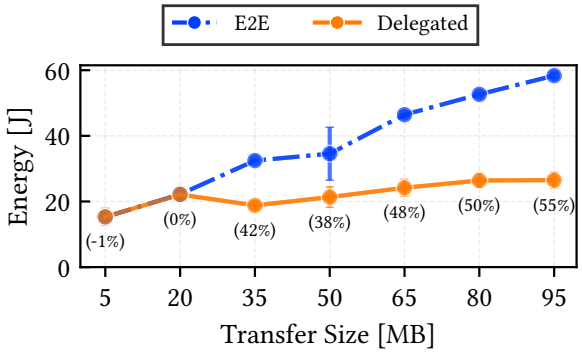


Figure 10: *Energy consumption during a window of 50 seconds in Scenario #1 (Table 1).*

consumption when the device is not transferring data (E2E: [e] → [w]; Delegated: [d] → [r] + [f] → [w]).

(ii) *Active energy consumption* [J] reflects the energy consumption when the device is transferring data (E2E: [s] → [e]; Delegated: [s] → [d] and [r] → [f]).

(iii) *Window energy consumption* [J] represents the energy consumption of the device during a fixed period of time ([s] → [w] for both E2E and delegated). This metric highlights the *real* energy consumption. After a transfer, the smartphone still consumes power in *idle* mode; hence, we must determine if the energy savings provided by our solution are not overshadowed by the *Idle/Standby energy consumption*.

4.2 Energy savings when delegating

We evaluate the energy savings achieved through delegation during HTTP/3 downloads under Scenario #1. Figure 10 presents the total *Window energy consumption* of a Pixel 7a during a period of 50 seconds. Each point of the Figure represents the mean of five repetitions, with error bars indicating the standard deviation.

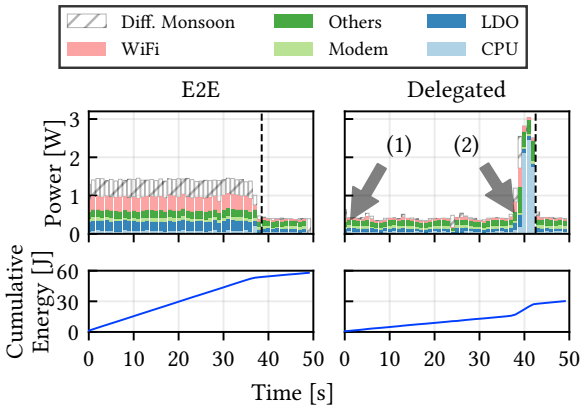


Figure 11: *Device power profile during a* 95 MB *in Scenario #1 (Table 1). The vertical line represents the client's perceived end of the transfer.*

Energy savings are higher for longer transfers. We observe that transfer sizes below 20 MB do not yield significant energy savings. For tiny file sizes (e.g., 5 MB), the overhead of Connection Delegation introduces an energy penalty. However, larger downloads incur longer transfer durations, thereby exposing the benefits of connection delegation. This trend increases with the file size: starting from 35 MB, we observe energy savings of 13 J (42 %), reaching up to 31.49 J (55 %) with a 95 MB transfer. Due to the device's 250 MB heap size limit, we only tested up to 200 MB, where energy savings reached 75 J (50 %) on 130 second sampling window (Section C in Appendix).

Profiling the energy savings. To fully understand *where* the energy savings come from, we profile the E2E and Delegated connections for the 95 MB transfer, over a window of 50 s in Figure 11. The dashed vertical line shows the end of the transfer on each side. The upper subfigures show the instant power consumption, while the bottoms represent the cumulative energy consumption. In the end-to-end case, the device constantly consumes ∼1.39 W until the end of the transfer (∼38 s). Then, it consumes ∼0.37 W during the remaining 12 s in an idle state. While transmitting, the Wi-Fi card consumes the main proportion of the power, accounting for approximately 24 % according to the ODPM.

On the right side, the delegated transfer moves the connection after receiving the HTTP response from the server ([l]), realizing that the transfer will be long. Once the client delegates its connection, it enters a low-power mode

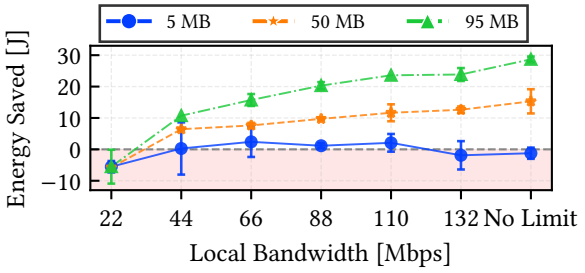


Figure 12: *Energy saved in Scenario #2 (Table 1). The* No limit *bandwidth represents results from Section 4.2.*

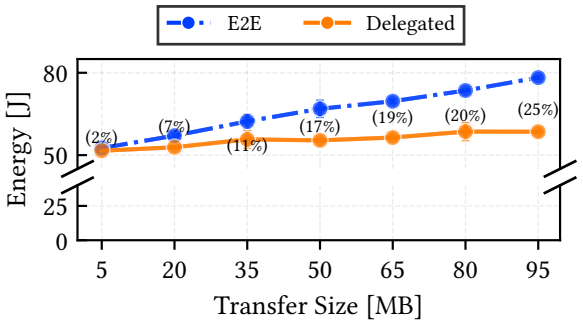


Figure 13: *Energy consumption in the presence of background traffic with a 50 second window in Scenario #3 (Table 1).*

and simply waits for the *Wake-up packet* while the delegate receives the content from the server. During this period, the device consumes approximately 0.41 W for 38 s. At the end of the transfer, the delegate sends a Wake-Up packet to the client (2). Then the client retrieves the data on the local Wi-Fi network (we measured a 309 Mbps throughput) for 2.4 seconds, with an average power consumption of 2.1 W. During the retrieval phase of ∼6 s, we observe that the CPU is the major contributor to energy consumption. In addition to retrieving its connection, the client must also *de-serialize* its state. The cost of this operation increases with the quantity of data buffered on the delegate (i.e., the file size), since the received data is serialized alongside the connection state. We investigate the overhead of QUIC delegation in Section 4.4.

Takeaway: When the server throughput is significantly lower than that of local Wi-Fi network, Connection Delegation considerably saves the overall device's energy consumption. For large files, the CPU overhead incurred when retrieving the connection state may become a bottleneck due to the de-serialization process. Future work will consider removing the application data from the serialized state and transmitting it from the delegate to the client alongside the state.

4.3 When is it beneficial to delegate?

Impact of the local Wi-Fi bandwidth. The energy savings presented in Section 4.2 are primarily attributed to the difference in bandwidth between the Wi-Fi network and the server throughput. Figure 12 illustrates how changing this bandwidth disparity impacts the energy savings for different file sizes. We vary both the transfer size (5, 50, 95 MB) and the Wi-Fi bandwidth (multiples of the server throughput 22 Mbps) while measuring the device's total energy consumption over a time window of 100 s.

When the local and remote bandwidths are equal, delegation consumes $\sim 1.2\times$ (5 J) more energy than an end-to-end transfer. Indeed, the device uses the same energy to retrieve data from the delegate as it does in the equivalent end-to-end transfer, with the additional overhead introduced by delegation. When the Wi-Fi bandwidth is $2\times$ the server throughput, energy savings begin to emerge with larger transfer sizes, thereby increasing the observed savings. For transfer sizes as small as 5 MB, Connection Delegation never brings significant savings or even consumes more energy due to the mechanism's overhead. For larger file sizes, we observe linear energy savings, which is consistent with our expectations.

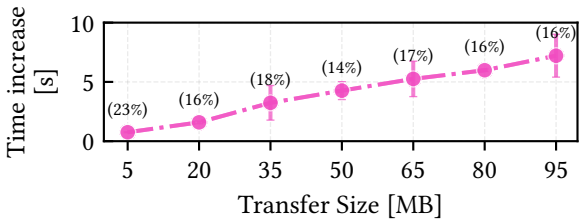
Takeaway: Connection Delegation brings energy savings when the Wi-Fi throughput is $\geq 2\times$ higher than the server throughput. For small file sizes (e.g., 5 MB), the solution's overhead compensates for the energy savings.

Impact of background traffic. Figure 13 represents the energy consumption with background transfer. We run an `iperf3` TCP client in the background with a Rx throughput of 160 kbps, similar to a medium-quality audio bitrate [40]. Additionally, we hold the Android wake lock, preventing the smartphone from entering Doze power mode [4]. As a consequence, the device's screen remains on and constantly consumes approximately 0.14 W. Although the smartphone does not enter a low-power mode, we notice a difference between its active power consumption (1.8 W) and idle power consumption (0.84 W). In this scenario, Figure 13 reports that del-

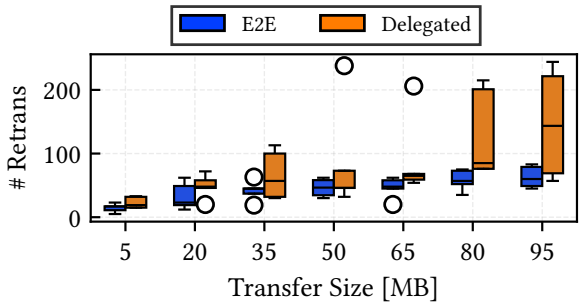
egation provides energy savings of 12 J and 21 J for transfer sizes of 55 MB and 95 MB, respectively. In proportion, this represents an energy savings of up to 25 %, considering that the screen remains active. These energy savings are lower than those observed in Scenario #1 because the smartphone does not enter a low-power mode.

Takeaway: Despite the presence of background traffic and of an active display, QUIC Connection Delegation still saves up to 25 % of the device's total energy consumption for a 95 MB download.

4.4 Time Overhead and Server-side Impact



(a) *Delegation overhead in seconds from the client perspective.*



(b) *Server-side retransmitted packets.*

Figure 14: *Impact of delegation in Scenario #1 (Table 1).*

Time overhead. QUIC delegation involves additional steps compared to a regular end-to-end connection, as explained in Section 3. We evaluate this time overhead by subtracting the connection time from the time it took (i) for the client to send the connection state to the delegate; (ii) to actually retrieve the connection state from the delegate; (iii) (de)serialize the QUIC connection state. Figure 14a reports the additional time duration in seconds and as a proportion of the total connection time in annotations. The results depict a roughly linear increase in time overhead because longer E2E transfers accumulate more data, which requires proportionally more time to retrieve. However, relative to the total connection duration, the overhead remains constant $\sim$19 % with respect to the file size, since the difference in throughput remains identical between the server throughput and the Wi-Fi network. Again, we observe that for a small transfer size (5 MB), the overhead of delegation is relatively higher due to the short transfer time.

Server-side impact. QUIC Connection Delegation is transparent to the server, which only needs to support connection migration [10]. Nevertheless, migration may impact the connection performance as observed from the server's viewpoint. First, the client stops processing packets while it delegates a connection, inducing packet losses. Second, since the delegate is a different device, its RTT may differ.

Figure 14b shows that delegated transfers trigger more retransmissions compared to E2E transfers. An inspection of the server's QLOGs [41] showed that these additional retransmissions typically occur during the delegation (Figure 7(b)). There is a significant variance in the measurements, which we attribute to a change in network conditions within our campus network. For future work, we could leverage QUIC's flow control to limit the data the server can send, and rely on the delegate to update this limit once the connection completely moved. After a connection migration, the RTT and congestion state is reset since the server cannot make any assumption on statistics regarding the *new path*.

5 Related Work

Wake on LAN (WoL) [42] is a technology allowing devices to be woken up via their network interface after entering sleep mode. When a device enters a low power consumption mode (such as an ACPI sleep state), selected interfaces continue to listen for incoming data. If a "magic packet" is received, the interface generates an interrupt to indicate that the device should leave sleep mode. This technology was initially proposed for Ethernet and then ported to wireless interfaces (WoWLAN). Wake On Lan (WoL) differs from Connection Delegation as it only allows the entire device to sleep until a specific packet arrives, whereas delegation allows an entire connection to be moved. Wake On Lan could delegate es-

tablished QUIC connections when entering sleep mode. Additionally, WoL has no energy savings if the device is active, such as Scenario #3 (Table 1), whilst delegation still offers energy savings in this scenario.

In the pursuit of energy savings in networked devices, Christensen et al [43] introduced the concept of "sleep proxy". This concept has then been further explored in numerous other papers [44, 45, 46]. In general, a sleep proxy refers to a network entity capable of sending packets on behalf of an unavailable host, in such a way that the host still appears to be online. This allows the host to enter a low power consumption mode without impacting the rest of the network. The proposed sleep proxies mainly process ARP, ICMP and TCP packets, but do not consider secure protocols using TLS or QUIC.

6 Discussion

Connection or application delegation. We designed our Delegation mechanism at the transport layer for two reasons. First, QUIC includes a connection migration feature [10], allowing for seamless transition between the client and its delegate without requiring any server change. Second, by controlling the transport connection state directly, we ensure that the client's device can enter its sleep mode when it hands off its connection to the delegate, thereby saving energy. Instead of moving the QUIC endpoint, alternatives could leverage Multiplexed Application Substrate over QUIC Encryption (MASQUE) [47]. MASQUE enables the encapsulation of multiple streams or datagram flows within a QUIC connection, making it a potential candidate for application-layer delegation. Future work will compare the benefits of our solution to MASQUE proxies [48, 49] in terms of energy savings. We will also compare our mechanism with previous approaches related to *sleep proxies* [43, 44, 45, 50].

Connection delegation when the Wi-Fi network is the bottleneck. Connection Delegation brings energy savings when the Wi-Fi bandwidth is significantly higher than the server's throughput. Recent studies have found that Wi-Fi becomes the bottleneck when the access link reaches $\geq$200 Mbps [51]. We did not extend our testbed to evaluate our solution in such conditions. However, Section 4.3 gives insight when the Wi-Fi bandwidth is equivalent to that of the access link: connection delegation does not save energy in this context. As such, we expect our solution to be counterproductive when the server throughput is faster than the Wi-Fi network.

Cellular networks. Based on the measurements of Narayanan et al. [9] over 5G networks, we observe that cellular communication consumes more energy than Wi-Fi. This motivates our future work, as we expect QUIC Connection Delegation to be more beneficial using cellular networks compared to our measurements in Section 4.

Delegation strategies and trusting the delegate. In this work, the client hands off its connection state to the delegate for the remainder of the download. In future work, we will introduce custom policies that allow applications to fine-tune their delegation/retrieval strategy with the delegate. Moreover, as described in Section 3, we will consider security mechanisms to ensure trust between the client and the delegate within the same LAN.

Beyond saving energy. While this work highlights the benefits of QUIC Connection Delegation in saving energy on a battery-powered device, this mechanism can also bring benefits considering other metrics. Future work will consider more use cases, such as allowing applications to determine when and how to delegate, or delegate some application functions to the delegate (e.g., application-level keep-alive could be handled by the delegate itself).

7 Conclusion

With the ever-growing use of battery-powered devices, such as smartphones or smartwatches, for accessing Internet content, designing energy-saving methods to extend the devices' battery life has become a significant challenge. While earlier works propose incremental improvements [2, 3, 52], we suggest a **radical shift to the end-to-end principle**. This paper introduces the *QUIC Connection Delegation* mechanism, which enables a client to *delegate* a transport connection to a trusted device, allowing it to continue communication with a remote server on the client's behalf. The client can later retrieve the connection data from the delegate. Delegation is beneficial when the server throughput is smaller than the bandwidth between the client and its delegate.

Connection Delegation relies on two key observations: (i) from an energy viewpoint, it is more efficient to transmit data at the highest possible throughput over wireless media (Wi-Fi and cellular); and (ii) while modern Wi-Fi offers $\geq$200 Mbps throughput, domestic access networks

typically limit Internet access at <50 Mbps. We design and implement Connection Delegation in QUIC [10], and measure the benefits and limits of the approach using file transfers in various scenarios with a Google Pixel 7a smartphone. QUIC Connection Delegation can save up to 55 % of energy when the smartphone can leverage existing sleep mode techniques [4], and 25 % considering background traffic and an active display.

While this paper introduces the Connection Delegation principles to save energy on battery-powered devices, we believe that this mechanism can be applied to other use cases. To motivate the research community to explore the benefits and limitations of this approach, we will release the source code of our two implementations, based on Cloudflare *quiche* [12] and *aioquic* [13]. The support for QUIC Delegation required 2769 and 2732 lines of code in *quiche* and *aioquic* respectively, excluding interoperability scripts. As part of our future work, we will analyze how applications can be tuned to leverag the benefits of Connection Delegation, measure our solution in cellular networks, and deploy the solution in real networks.

References

[1] David Clark. The design philosophy of the darpa internet protocols. In *Symposium proceedings on Communications architectures and protocols*, pages 106–114, 1988.

[2] Shilpa Devalal and A Karthikeyan. Lora technology-an overview. In *2018 second international conference on electronics, communication and aerospace technology (ICECA)*, pages 284–290. IEEE, 2018.

[3] Marina Petrova, Janne Riihijarvi, Petri Mahonen, and Saverio Labella. Performance study of ieee 802.15. 4 using measurements and simulations. In *IEEE Wireless Communications and Networking Conference, 2006. WCNC 2006.*, volume 1, pages 487–492. IEEE, 2006.

[4] Optimize for Doze and App Standby | App quality | Android Developers — developer.android.com. `https://developer.android.com/training/monitoring-device-state/doze-standby`, . [Accessed 26-09-2025].

[5] Xiaomeng Chen, Abhilash Jindal, Ning Ding, Yu Charlie Hu, Maruti Gupta, and Rath Vannithamby. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 40–52, 2015.

[6] Erfan Mozaffariahrar, Fabrice Theoleyre, and Michael Menth. A survey of wi-fi 6: Technologies, advances, and challenges. *Future Internet*, 14(10): 293, 2022.

[7] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2 (4):277–288, 1984.

[8] Swetank Kumar Saha, Pratik Deshpande, Pranav P Inamdar, Ramanujan K Sheshadri, and Dimitrios Koutsonikolas. Power-throughput tradeoffs of

802.11 n/ac in smartphones. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 100–108. IEEE, 2015.

[9] Arvind Narayanan, Xumiao Zhang, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Xiaoxuan Zhang, Denis Rybkin, Zhengxuan Yang, Zhuoqing Morley Mao, et al. A variegated look at 5g in the wild: performance, power, and qoe implications. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 610–625, 2021.

[10] J. Iyengar (Ed.) and M. Thomson (Ed.). QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000 (Proposed Standard), May 2021. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9000.txt.

[11] Aurélien Buchet and Cristel Pelsser. An analysis of quic connection migration in the wild. *SIGCOMM Comput. Commun. Rev.*, 55(1):3–9, April 2025. ISSN 0146-4833. doi: 10.1145/3727063.3727066. URL https://doi.org/10.1145/3727063.3727066.

[12] Cloudflare. Savoury implementation of the QUIC transport protocol and HTTP/3. https://github.com/cloudflare/quiche.

[13] aioquic. https://github.com/aiortc/aioquic.

[14] Roy Friedman, Alex Kogan, and Yevgeny Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(7):1363–1376, 2012.

[15] Li Sun, Ramanujan K Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. Modeling wifi active power/energy consumption in smartphones. In *2014 IEEE 34th international conference on distributed computing systems*, pages 41–51. IEEE, 2014.

[16] Agrim Gupta, Adel Heidari, Avyakta Kalipattapu, Ish Kumar Jain, and Dinesh Bharadia. 3 w's of smartphone power consumption: Who, where and how much is draining my battery? In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 2248–2250, 2024.

[17] Power Profiler | Android Studio | Android Developers — developer.android.com. https://developer.android.com/studio/profile/power-profiler, . [Accessed 17-09-2025].

[18] Édouard Guégain, Rémy Raes, Noé Chachignot, Clément Quinton, and Romain Rouvoy. Androwatts: Unpacking the power consumption of mobile device's components. In *2025 IEEE/ACM 12th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 71–81. IEEE, 2025.

[19] Bruce Spang, Shravya Kunamalla, Renata Teixeira, Te-Yuan Huang, Grenville Armitage, Ramesh Johari, and Nick McKeown. Sammy: Smoothing video traffic to be a friendly internet neighbor. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 754–768, 2023.

[20] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 404–417, 2017.

[21] Glossary · Cloudflare Radar docs — developers.cloudflare.com. https://developers.cloudflare.com/radar/glossary/#iqi, . [Accessed 25-08-2025].

[22] David Newell, Philip Davies, Russell Wade, Perry Decaux, and Mak Shama. Comparison of theoretical and practical performances with 802.11 n and 802.11 ac wireless networking. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 710–715. IEEE, 2017.

[23] M. Bishop (Ed.). HTTP/3. RFC 9114 (Proposed Standard), June 2022. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9114.txt.

[24] M. Thomson (Ed.) and C. Benfield (Ed.). HTTP/2. RFC 9113 (Proposed Standard), June 2022. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9113.txt.

[25] Aurélien Buchet and Cristel Pelsser. An analysis of quic connection migration in the wild. *ACM SIGCOMM Computer Communication Review*, 55(1):3–9, 2025.

[26] Mike Kosek, Benedikt Spies, and Jörg Ott. Secure middlebox-assisted quic. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2023.

[27] M. Thomson (Ed.) and S. Turner (Ed.). Using TLS to Secure QUIC. RFC 9001 (Proposed Standard), May 2021. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9001.txt.

[28] J. Iyengar (Ed.) and I. Swett (Ed.). QUIC Loss Detection and Congestion Control. RFC 9002 (Proposed Standard), May 2021. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9002.txt.

[29] J. Corbet. Tcp connection repair. LWN, https://lwn.net/Articles/495304, 2012.

[30] Sadayuki Furuhashi et al. Messagepack. https://msgpack.org, 2021.

[31] Hongying Dong, Yizhe Zhang, Hyeonmin Lee, Kevin Du, Guancheng Tu, and Yixin Sun. Mutual tls in practice: A deep dive into certificate configurations and privacy issues. In *Proceedings of the 2024 ACM on Internet Measurement Conference*, pages 214–229, 2024.

[32] J. Reschke. The 'Basic' HTTP Authentication Scheme. RFC 7617 (Proposed Standard), September 2015. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7617.txt.

[33] M. Jones and D. Hardt. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750 (Proposed Standard), October 2012. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6750.txt. Updated by RFC 8996.

[34] R. Shekh-Yusef (Ed.), D. Ahrens, and S. Bremer. HTTP Digest Access Authentication. RFC 7616 (Proposed Standard), September 2015. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7616.txt.

[35] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc8446.txt.

[36] Anonymous Author. *Anonymized title*. Phd thesis, Anonymous, 2024.

[37] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-16, Internet Engineering Task Force, August 2025. URL https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/16/. Work in Progress.

[38] Marcel Kempf, Benedikt Jaeger, Johannes Zirngibl, Kevin Ploch, and Georg Carle. Quic on the fast lane: Extending performance evaluations on high-rate links. *Computer Communications*, 223:90–100, 2024.

[39] "Moonsoon Solutions". High voltage power monitor. https://www.msoon.com/high-voltage-power-monitor, 2024.

[40] Bit rate - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Bit_rate#MP3. [Accessed 01-10-2025].

[41] Robin Marx, Luca Niccolini, Marten Seemann, and Lucas Pardue. qlog: Structured Logging for Network Protocols. Internet-Draft draft-ietf-quic-qlog-main-schema-12, Internet Engineering Task Force, July 2025. URL https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema/12/. Work in Progress.

[42] Ken Christensen, Pedro Reviriego, Bruce Nordman, Michael Bennett, Mehrgan Mostowfi, and Juan Antonio Maestro. Ieee 802.3 az: the road to energy efficient ethernet. *IEEE Communications Magazine*, 48 (11):50–56, 2010.

[43] Kenneth J Christensen and Franklin 'Bo' Gulledge. Enabling power management for network-attached computers. *International Journal of Network Management*, 8(2):120–130, 1998.

[44] Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Victor Bahl, and Rajesh Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *Networked Systems Design & Implementation (NSDI)*, 2009.

[45] Joshua Reich, Michel Goraczko, and Aman Kansal. Sleepless in seattle no longer. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[46] Miguel Jimeno, Ken Christensen, and Bruce Nordman. A network connection proxy to enable hosts to sleep and save energy. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 101–110, 2008. doi: 10.1109/PCCC.2008.4745133.

[47] David Schinazi. The MASQUE Proxy. Internet-Draft draft-schinazi-masque-proxy-06, Internet Engineering Task Force, July 2025. URL `https://datatracker.ietf.org/doc/draft-schinazi-masque-proxy/06/`. Work in Progress.

[48] What's new in Cloudflare: MASQUE now powers 1.1.1.1 & WARP apps, DEX now generally available with Remote Captures — blog.cloudflare.com. `https://blog.cloudflare.com/masque-now-powers-1-1-1-1-and-warp-apps-dex-available-with-remote-captures/`, . [Accessed 02-09-2025].

[49] Apple. iCloud Private Relay Overview. `iCloudPrivateRelayOverview`. [Accessed 02-09-2025].

[50] Karthikeyan Arunachalam, YoungKi Hong, Wonbo Lee, and Jamsheed Manja Ppallan. Low power tcp for enhanced battery life in mobile devices. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[51] Ranya Sharma, Nick Feamster, and Marc Richardson. A longitudinal study of the prevalence of wifi bottlenecks in home access networks. In *Proceedings of the 2024 ACM on Internet Measurement Conference*, pages 44–50, 2024.

[52] Miguel Jimeno, Ken Christensen, and Bruce Nordman. A network connection proxy to enable hosts to sleep and save energy. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 101–110. IEEE, 2008.

A QUIC connection migration

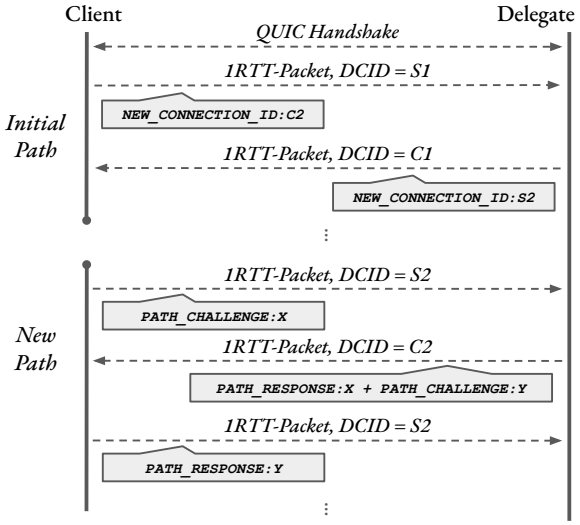


Figure 15: *An example of the information exchanged when a client initiates a QUIC connection migration.*

Figure 15 provides an illustration of the main steps of a QUIC connection migration.

B QUIC connection state

This section provides additional details on the contents of the QUIC connection state exchanged between a client and its delegate. It complements Section 3.1.

Packet Number Spaces. A packet number space is the context in which QUIC packets can be processed and

acknowledged. QUIC uses three packet number spaces: Initial, Application data, and Handshake [10], we only keep the Application data as the other are used only during handshake. Each packet sent over a QUIC connection has a unique packet number in the packet number space. Within this packet number space, we store (i) the received packet numbers that need to be acknowledged; (ii) the largest packet number received; (iii) a *timestamp* indicating when the largest packet was received; (iv) the next packet number to use when transmitting; and finally (v) the set of already received packet numbers.

Transport parameters. QUIC transport parameters [10] are exchanged during the connection handshake to define key protocol settings, such as the maximum UDP payload size or acknowledgment delays, ... These parameters are encoded as a hashmap, where the keys correspond to their standardized names in the QUIC specification, and the values adhere to the data types specified by the RFCs. Including this information ensures the delegate adheres to the negotiated protocol extensions, maintaining compatibility and compliance with the original connection.

Cryptographic Materials. Packet protection in QUIC [27] relies on cryptographic keys derived through TLS 1.3 [35]. During the TLS handshake, endpoints negotiate cryptographic algorithms [35] and TLS exposes four cryptographic keys to QUIC: Initial, Early-data, Handshake, and Application data ($1-RTT$) keys. Since connection delegation occurs after the QUIC handshake between the client and the server, only the $1-RTT$ keys are relevant. We store the corresponding cryptographic secrets and the cipher suite negotiated during the handshake to enable packet encryption and decryption on the delegate. This ensures that the delegate derives the same encryption materials as the client. We transmit the cryptographic secrets, rather than just the derived keys, to allow the delegate to compute new secrets such as `clien t_token`, `delegate_token`, and others.

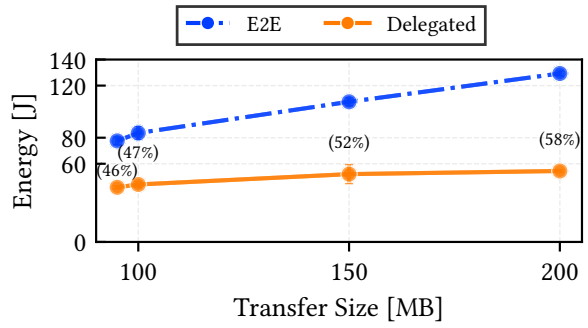
Connection IDs. Connection IDs (CIDs) uniquely identify a QUIC connection on a peer [10]. Beyond the CIDs exchanged in the initial packets, endpoints can receive additional CIDs throughout the connection. These identifiers are crucial, as each $1-RTT$ packet contains a Destination Connection ID (DCID) to associate each packet with the corresponding connection at the receiver. When storing CID information, we record all non-retired CIDs from each peer, along with their reset tokens and sequence numbers. Finally, we store the `active_connec tion_id_limit` for each endpoint to ensure that connection ID restrictions are consistently enforced throughout the connection.

Streams and Flow Control. Streams are the ordered byte-stream abstraction over which a QUIC connection transports application data [10], and they are regulated by flow control. Since the delegate must also transfer application data, we represent the byte-stream abstraction as ordered range buffers for each stream. Therefore, we include each stream's receive and send buffers within the connection state. For the receive side, this represents data received by the QUIC stack that has not yet been delivered to the application. For the send side, the data is written to the QUIC stack but has not yet been sent or acknowledged by the peer. We also add control information such as flow control and `FIN` flags. This preserves the ordered nature of the data in each stream and enables the delegate to send and receive information without disrupting the QUIC abstraction.

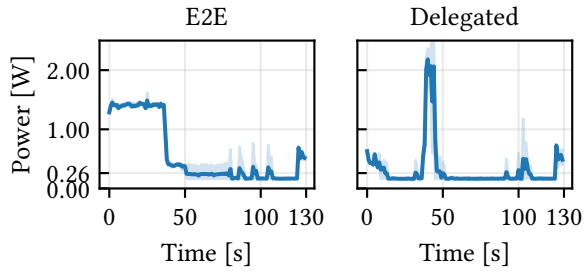
Congestion Control. QUIC endpoints perform congestion control per path [28]. QUIC hosts must reset their congestion control state upon connection migration [10]. Since delegating a connection implies a migration, we do not maintain congestion control state in the portable state. The same applies to the Path MTU discovery information.

QUIC's shared control information is significantly smaller than the maintained information per implementation. First, since delegation occurs after the QUIC handshake, we only need to share the $1-RTT$ handshake keys and the application packet number space. Second, since connection delegation requires the delegate to perform a QUIC migration, all path-dependent information is not shared. The data structures representing the connection state are unique to each QUIC library. As such, the (de)serialization process involves deriving from/to our portable QUIC connection state. Our prototype encodes the QUIC connection state using MessagePack [30], an efficient, compact, and widely supported format. Additionally, MessagePack supports mappings with keys of any data type, aligning well with connection state needs. From empirical measurements, we observe that the size of the connection state when delegating is around 2.5 kB, without connection data.

Summary: Our portable QUIC connection state en-



(a) *Total device energy consumption*



(b) *Power consumption of a* 95 MB *transfer*

Figure 16: *Experimenting with large transfer sizes in Scenario #1 with a sampling window of 130 seconds.*

ables capturing a snapshot of an ongoing QUIC connection at any time after the handshake, from the perspective of a single host. It is lightweight, complete, and fully compliant with QUIC version 1.

C Beyond 95 MB transfer sizes

Figure 16a shows results for transfer sizes beyond 95 MB in Scenario #1 (Table 1), using a 130 s window to ensure completion. We observe energy savings of 36, 39, 55, and 75 J for transfers of 95, 100, 150, and 200 MB, respectively. Due to the maximum heap size limit (`dalvik.vm.heapgrowthlimit`) of 250 MB, we could not measure transfers larger than 200 MB without the binary crashing. For comparison, Figure 10 uses a 50 s sampling window, whereas Figure 16a relies on a 130 s window. This 80 s difference accounts for an additional 21 J of energy consumption, corresponding to the average *Idle* power draw of 0.26 W, as illustrated in Figure 16b.