BURST: Proxy-Assisted Traffic Batching for Energy-Efficient Wi-Fi Networking

VANY V. INGENZI*, UCLouvain, Belgium
LOUIS NAVARRE, UCLouvain, Belgium
FLORENTIN ROCHET, UNamur, Belgium
JULIEN MONTAVONT, University of Strasbourg, France
OLIVIER BONAVENTURE, UCLouvain & WELRI, Belgium

Optimizing battery life remains a challenge for IoT and mobile devices. For Wi-Fi interfaces, power consumption is largely dictated by the absolute time the chip spends in an active state, rather than its bandwidth usage. In scenarios where the Local Area Network (LAN) offers significantly higher capacity than the Wide-area Access Networks (WANs), this throughput asymmetry creates an opportunity. By temporarily buffering traffic and delivering it in dense, high-speed bursts, a client device can increase the time its Wi-Fi chip spends in idle/low-power mode and thus reduce its battery usage for long transfers.

We propose BURST, a TCP-level buffering proxy deployed on the local network. For downloads, BURST continuously receives data from the Internet at the WAN rate while the mobile client remains in low-power mode. Then it regularly flushes this buffered data over the high-capacity LAN, which keeps the client's WNIC active for short durations. Unlike previous proxy-based solutions, BURST is application-agnostic and preserves the end-to-end encrypted TLS session. We implement BURST and evaluate with real smartphones. Our results show that BURST achieves up to 45 % energy savings during bulk transfers, with an increase in Flow Completion Time (FCT) of less than 1.06 s in 95 % of our measurements for a single connection.

1 Introduction

Smartphones and IoT devices have become widely adopted tools due to their increasing computational power, enabling applications to run with higher throughput and lower latency for ever more complex tasks. However, this increase in capabilities also increases the device's energy consumption during transmission. For example, in cellular networks, 5G-enabled devices consume more power during transmission than 4G/LTE devices [1], and the same holds for recent Wi-Fi standards [2]. To alleviate this, devices are becoming more energy-efficient, i.e., consuming less energy for the same task. However, the efficiency regime of recent wireless technologies is only available when devices communicate at high throughput [1, 2], which is not always achievable across the Internet [3].

Indeed, Cloudflare's measurements of per-connection throughput under average utilization (such as web browsing) show that the actual throughput can be orders of magnitude lower than what recent Wi-Fi standards offer [3] (Discussed in Section 2.2). This can be caused by server-side shaping [4], internet congestion [5], etc. In case the Local Area Network (LAN) is not the bottleneck, one can exploit this asymmetry in throughput to allow the device to consume less energy per transfer. Deploying the proxy in the *un-bottlenecked* network segment can allow to accumulate slow incoming traffic from the WAN before flushing it rapidly over the high-speed LAN.

Several prior works have explored proxy-based approaches to reduce network energy consumption [6–11]; They are discussed in details in Section 7. To our knowledge, all prior works are either application specific [9–11], or need access to decrypted traffic or application state [6–8]. We address the need for a generic TCP-level, application-agnostic mechanism for wireless clients to enable

*Vany V. Ingenzi is a FRIA grantee of the Fonds de la Recherche Scientifique - FNRS.

Authors' Contact Information: Vany V. Ingenzi, UCLouvain, Belgium, vany.ingenzi@uclouvain.be; Louis Navarre, UCLouvain, Belgium, louis.navarre@uclouvain.be; Florentin Rochet, UNamur, Belgium, florentin.rochet@unamur.be; Julien Montavont, University of Strasbourg, France, montavont@unistra.fr; Olivier Bonaventure, UCLouvain & WELRI, Belgium, olivier.bonaventure@uclouvain.be.

energy savings. Our approach preserves the end-to-end security of TLS connections, requires no server-side modifications, and still enables significant energy savings.

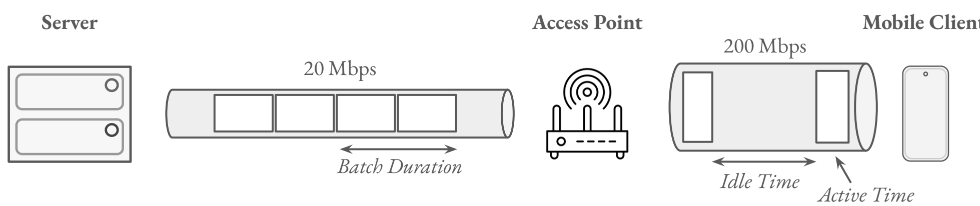


Fig. 1. High level overview of BURST under throughput asymmetry.

In this paper, we show that by proxying a TCP connection, we can decrease the Wi-Fi chip active time for bulk transfers. As a result, the mobile device consumes less energy. We focus on bulk transfers, i.e transfers that can tolerate packet delays, e.g, file transfers. Our solution, BURST, leverages the asymetry between LAN and WAN throughput to allow the client to have idle/low-power periods during long transfers. BURST is a system composed of a proxy that provides *buffering* services to TCP clients. The clients and proxy collaborate to minimize the impact on the transfer's completion time.

Figure 1 shows a typical deployment of BURST. The Access Point batches data received from the server at a lower bandwidth, accumulating it over a certain *batch duration*. This allows the mobile client to enter power-save mode. Then it flushes the buffered data to the client in short bursts at a higher throughput, thus decreasing the overall time the client's Wi-Fi NIC is awake.

The contributions made by this paper are:

- **TLS-friendly buffering mechanism.** We design BURST, a TCP proxying mechanism that exploits throughput asymmetry between the WAN and LAN to reduce mobile energy consumption by batching data. By organizing incoming traffic into periodic temporal bursts, the system increases client idle opportunities without requiring visibility into encrypted application-layer data
- **Experimental evaluation on real devices.** We implement and evaluate BURST in our lab with three different smartphones. Our experiments show that in the presence of Wi-Fi Power Save Mode (PSM), BURST can save up to 45 % of the total device energy consumption during transfers, while increasing the average flow completion time of long transfers by less than 2 s and a median of 0.53 s.

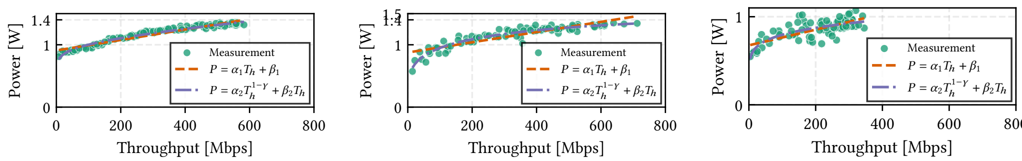
We focus on the system design and network-level performance of BURST, targeting bulk transfers such as cloud synchronization, software updates, and large file downloads. By evaluating the system in controlled, benchmark-oriented scenarios, we isolate and demonstrate its core energy-saving capabilities at the transport layer.

In the remainder of this paper, we start by motivating our problem (§2). Then we present the detailed design of BURST (§3) and its implementation (§4). Furthermore, we evaluate through small-scale experiments the potential benefits of BURST (§5), followed by our extension to the 0-RTT protocol (§6). Finally, we compare our solution to prior work (§7), discuss the limitations and further work (§8).

Ethics. This work does not raise any ethical issues. **AI Usage Disclosure.** The use of AI was limited to assist with the copy-editing.

2 Motivation

Several factors influence smartphone energy consumption during Wi-Fi data transfers (Section 2.1). Prior work indicates that *the faster the data transfer, the better the energy efficiency* for mobile



(a) Pixel 7A (b) Pixel 10 Pro (c) Fairphone 5

Fig. 2. Power consumption of the device during HTTPS downloads over Wi-Fi at various throughputs. We display the linear model [2, 14] (■) and the generic model proposed by [15] (■).

devices; we validate this in our testbed. We then show that Internet throughput rarely matches the achievable throughput of modern Wi-Fi networks (Section 2.2). Finally, we show that existing Wi-Fi buffering mechanisms designed for energy savings can be further improved for bulk transfers (Section 2.3).

2.1 Energy consumption of smartphones while exchanging data

Power consumption of smartphone communications has been extensively studied [2, 12–15]. Gupta et al. [12] report that during a 720p YouTube streaming, the Wi-Fi chip alone accounts for 42-65 % of the total device power consumption, depending on the device. Friedman et al. [14] analyzes 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 show that the power consumption of the Wi-Fi chip of a smartphone heavily depends on the time it is active, and only slightly on the throughput.

To validate this hypothesis in our environment, we measure the energy consumption of three recent smartphones: Pixel 7A, Pixel 10 Pro, and Fairphone 5. The devices perform HTTPS downloads from a server on our campus network via an 802.11ac Wi-Fi Turris Omnia access point. We vary the available network bandwidth by applying shaping at the server's egress with `tc`. We perform 100 runs with distinct throughputs per device, and adapt the transfer size to ensure the runs lasts 30 seconds. We keep the default Wi-Fi PSM configuration on the smartphones.

Energy Efficiency. Figure 2 shows the power consumption as a function of the TCP throughput across the three devices. The results show similar power consumption across devices. Due to space limitation, we focus on the Pixel 7a(Figure 2a). We observe that power consumption *slightly* increases with throughput, consistent with previous models [2, 14, 15]. Indeed, the increase is relatively limited: while throughput increases by a factor of 276 (from 2.1 to 580 Mbps), power consumption increases by only 62.76 % (from 0.84 W to 1.38 W), suggesting improved energy efficiency at higher bandwidths. We compute the energy efficiency (Mbit/J) as the overall data transferred divided by the total device energy consumption during each 30-second experiment. The energy efficiency improves significantly at higher throughput. At 580 Mbps, the smartphone transmits 46× more bits per Joule (420.3 MbpJ) compared to 46.1 Mbps (9.11 MbpJ). This shows the potential power savings that can result in shortening the transfer duration[1].

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

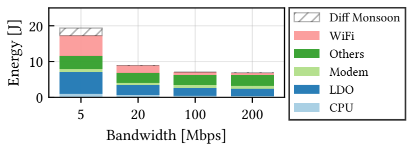


Fig. 3. Energy consumption profile for a 12.5 MB download over a 30-second measurement window on Pixel 7a.

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, we downloaded a 12.5 MB file[2] on our Google Pixel 7a at {5, 20, 100, 200} Mbps, while monitoring power consumption over a 30 s window. Figure 3 reports the total energy consumed by the device, as measured by a Monsoon High Voltage Power Monitor (HVPM), together with per-component energy consumption obtained from the On-Device Power Monitor (ODPM) rails [16]. Since the ODPM rails do not account for all power consumption and exhibit conversion inefficiencies at high power levels [17], we show the difference between the Monsoon and the sum of all rails (*Diff Monsoon*).

As expected, for a fixed download size, higher throughput leads to lower energy consumption. From 19.3 J at 5 Mbps to 6.8 J at 200 Mbps. Figure 3 also highlights that most energy savings come from the Wi-Fi chip (*WiFi*). Since faster transfers complete more quickly, the device can *potentially*[3] enter a low-power state sooner. Additionally, Low-Dropout (LDO) regulator energy *consumption* decreases because low bandwidth draws more continuous current, which increases the load on the LDOs. All together, the results from Figures 2 and 3 confirm our first hypothesis: higher throughput leads to a greater energy efficiency since the device (notably its Wi-Fi chip) transmits more information for the same quantity of energy and, for a fixed file size. This means that it must stay active for a shorter time.

2.2 Achievable server throughput

From an energy-efficiency perspective, the previous section shows that a smartphone should transfer data at the highest possible throughput. However, the Internet is a shared infrastructure with limited bandwidth. Servers often limit their throughput for various reasons [4, 5, 18], thereby decreasing the potential energy-saving benefits. To illustrate this, we analyze Cloudflare's Internet Quality Index (IQI) [3], which represents the measured throughput of real connections, such as web browsing. Figure 16 (Appendix A) 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 [19]), thus confirming our second hypothesis.

2.3 Further extending Power Save Modes for Bulk transfers

The 802.11 Wi-Fi standards incorporate Power Save Modes (PSMs) to minimize Wi-Fi Network Interface Controller (WNIC) uptime during periods of inactivity, consequently reducing power consumption. While our approach generalizes across Wi-Fi standards, we focus on 802.11ac (Wi-Fi 5), which supports Unscheduled Automatic Power Save Delivery (U-APSD).

Wi-Fi PSM Can Be Ineffective For A Continuous Stream of Data. When using U-APSD, the Access Point (AP) buffers packets for a client (STA) that is in a low-power state. The client periodically wakes up to listen for Beacon frames, which indicate whether the AP holds buffered

[2]We chose this size to limit the 5 Mbps experiment to a 30 s transfer.
[3]Whether the device enters a low-power mode also depends on the operating system.

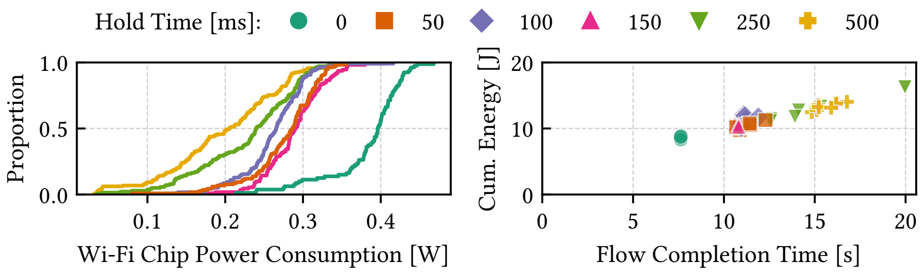


Fig. 5. Effect of AP-side IP packets buffering (for x ms) during a 20 MB transfer. The left plot shows the CDF of the WNIC power consumption. The right plot shows the cumulative total device energy consumption along the Flow Completion Time (FCT). The experiment is repeated 10 times for each hold time.

data for the STA . Though effective for traffic with significant inter-packet arrival times [7, 20, 21], PSM modes do not provide benefits for continuous data streams. During a bulk transfer, new data continuously fills the client's buffers before they can be fully drained, preventing the WNIC from entering low-power mode.

Figure 4 shows this limitation: during a continuous transfer, the wireless link remains constantly busy, keeping the WNIC active. At the MAC layer, this behavior is expected by design as it cannot make assumptions about upper-layer requirements. However, for applications that are not latency-sensitive and can tolerate inter-packet delays, intentionally buffering at the AP becomes beneficial.

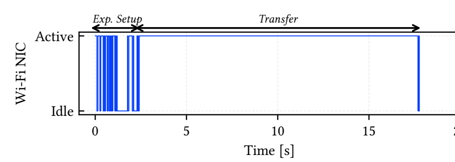


Fig. 4. Pixel7a WNIC activity during a 40 MB transfer at 22 Mbps. The first stage is the *experiment setup*, configuring the client via wireless ADB.

Why not buffering IP packets at the AP? Our proposal in this paper is to proxy a TCP connection and buffer TCP byte-stream, why not simply buffer IP packets at the Access Point (AP)? To investigate this, we perform an experiment where the AP periodically pauses transmission for a short duration and then flushes the queued data to the client, roughly approximating static PSM. We use `tc` to configure the wireless interface of the AP to periodically buffer IP packets for x milliseconds and then flush the queues for 1 s. While there is no standardized maximum buffer size for an AP handling an STA in a power-save state, the Linux 802.11 implementation defaults to 128 frames.[4] To accurately emulate this, we capped the `tc` queue capacity at 128 frames, dropping any subsequent incoming packets. Since `tc netem` buffers packets before they reach the AP's Wi-Fi driver, the client's WNIC is successfully allowed to enter a low-power mode.

Figure 5 illustrates the limitation of this approach. Increasing the hold delay at the AP successfully decreases the client's WNIC power consumption (left graph). However, the right graph demonstrates that as the hold delay increases, the total energy consumption of the device actually rises. This occurs because AP-side buffering severely inflates the end-to-end Round Trip Time (RTT) and induces potential packet drops when the 128-frame limit is reached. The remote server's TCP congestion control *misinterprets* these signals as network congestion, drastically reducing throughput. Consequently, the transfer takes significantly longer to complete, and this prolonged active time entirely negates the WNIC energy savings seen in the left graph of Figure 5.

Takeaway. The transport layer provides a unique point to infer application needs, estimate end-to-end available bandwidth, and query the MAC layer for wireless channel conditions. We

[4]OpenWRT uses Linux's upstream mac80211 since 2007. https://codebrowser.dev/linux/linux/net/mac80211/ieee80211_i.h.html

leverage the transport layer operating point to buffer data on behalf of the client without incurring buffering, jitter, or packet loss, thereby allowing longer WNIC idle/low-power durations. Later in this paper, we demonstrate that this approach yields significant energy savings with minimal impact on Flow Completion Time. It is important to note that since BURST operates entirely at the transport layer, we achieve these gains without modifying the device's PSM.

3 BURST: Design and Architecture

In this section, we present the architectural design of BURST and detail its operation throughout the lifecycle of a TCP connection. Our system relies on the two core insights presented previously: firstly, Wi-Fi interface power consumption is mainly dictated by its active residency time rather than its instantaneous bandwidth utilization. Secondly, per-connection local area network (LAN) capacity is significantly higher than that of the wide-area network (WAN).

3.1 Overview

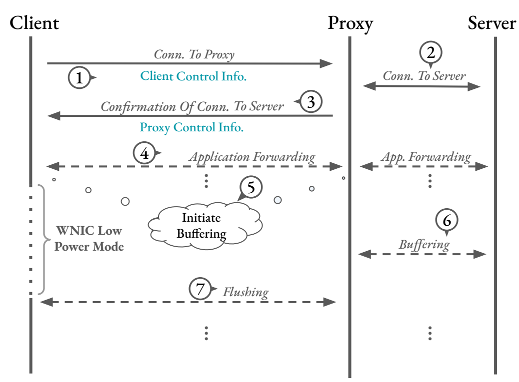


Fig. 6. Example diagram of BURST.

Figure 6 illustrates BURST in action. (①) The client first establishes a TCP connection to the proxy, providing the target server's information alongside BURST's control parameters (detailed in Section 3.2). (②) After validating this request, the proxy creates a secondary TCP connection to the target server and (③) the proxy confirms the setup by returning its own control parameters to the client. With the setup complete, application traffic (④) is routed transparently, just as it would be through a standard TCP proxy (e.g. TLS handshake, etc.). In this example, the client is downloading data via HTTPS. (⑤) Throughout the connection, the proxy detects a bandwidth asymmetry: its LAN link to the client has a much higher achievable throughput than the WAN link to the server. Recognizing that the WAN proxy-server link is the bottleneck, the proxy calculates a buffering duration based on the client's maximum buffering duration and starts the buffering phase (detailed in §3.3). During the buffering phase (⑥), the proxy buffers the server's responses. The client's WNIC observes no incoming traffic, transitions to Power Save Mode (PSM) via standard 802.11 mechanisms. When the buffering duration expires (⑦), the proxy drains the queued data downstream to the client in a high-throughput burst. BURST dynamically repeats these buffer-and-burst batches throughout the transfer whenever it detects *sufficient* bandwidth asymmetry, otherwise it proxies the connection like a standard TCP proxy [22].

3.2 Connection Establishment

BURST's proxy is an explicit proxy that the client must explicitly initiate a connection to. In this subsection we focus on the BURST connection setup.

Control Information Exchange. BURST assumes the proxy resides in the LAN. Hence, we use the initial handshake to exchange control information between the client and the proxy. Currently, the control messages for both the client and the proxy fit within a single Maximum Transmission Unit (MTU) packet. Consequently, BURST introduces exactly one RTT of connection-establishment overhead compared to a direct connection. However, because this RTT occurs over the LAN link to the proxy, the added latency is negligible and easily amortized over the long-lived connections that

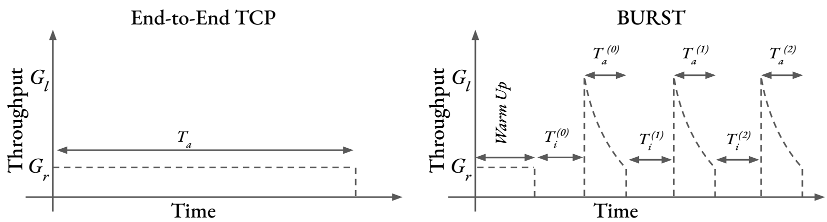


Fig. 7. Simplified time diagram comparing an end-to-end connection to a BURST proxied connection, along with the operating throughput towards the client. The superscript denotes the batch number.

Symbol	Meaning	Source
G_r	Observed WAN Throughput	Continuously measured
G_l	Observed LAN Throughput	Measured during active phase
$\widehat{G_l}$	Estimated *achievable* LAN Throughout	Calculated Eq. (3)
L	Available LAN link bandwidth	Measured from Wi-Fi chip Eq. (1)
T_i	Client WNIC's idle/low-power duration during a batch	Calculated Eq. (8)
T_a	Client WNIC's active duration during a batch	Calculated Eq. (7)
B	Maximum Batch duration throughout a connection	Provided by the client
β	Buffering-admission threshold (0.9)	Constant

Table 1. BURST parameters and variables with their definitions.

BURST primarily targets. An alternative to in-band signaling within the initial TCP byte stream would be to establish a dedicated, out-of-band control connection. We explicitly opted against this design as synchronizing state across two separate TCP connections would significantly complicate the proxy's architecture and hinder its scaling.

Content of Control Information. The control information serves three purposes: authenticating the client, specifying the target server, and defining the maximum batch duration. To ensure that the client is communicating with a legitimate proxy, this payload is secured using previously exchanged cryptographic material (Section 6). Upon successful client authentication, the proxy attempts to open a TCP connection to the specified target server. If successful, the proxy replies to the client with its own control information; if it fails, it returns an error message and terminates the session. The proxy's control response includes a legitimacy proof (Section 6) and the maximum buffer size allocated for the ongoing flow.

3.3 Buffering Mechanism

Table 1 shows the parameters and variables that we use in the BURST control logic. We use the superscript notation to indicate the batch number, e.g., $G_r^{(n)}$ depicts the measured WAN throughput during batch n.

Operating Principle. Figure 7 illustrates BURST's behavior to an end-to-end baseline, from a temporal and throughput perspective at the client. The standard end-to-end connection operates continuously at the WAN throughput (G_r). On the other side, once the connection is established, BURST segments the transfer into a sequence of batches indexed from 0 to N, each lasting a duration B. BURST calculates the durations of *Active* (T_a) and *Idle* (T_i) phases at the start of each batch. As a result it alternates the operating throughput between the high-speed LAN link (G_l), and the slow WAN link (G_r). In Figure 7, the client-perceived throughput decreases during the active period, because BURST empties buffers at higher rate till matching the arrival rate of data.

Per-batch control loop. We first describe the control loop during a batch n and then look at the initial warm-up phase.

(1) *Plan the batch.* The first step for a batch is to calculate $T_a^{(n)}$ and $T_i^{(n)}$ using Eq. (7) and Eq. (8). It is based on the LAN throughput observed $G_l^{(n-1)}$ during batch $n-1$, and the latest continuously-measured remote throughput $G_r^{(n)}$, and available link capacity $L^{(n)}$.

(2) *Idle Phase.* If the throughput asymmetry is *sufficient* (Eq. (5)) and the buffer from the previous batch $n-1$ is emptied, Then the proxy pauses downstream transmission for the calculated $T_i^{(n)}$, while it buffers incoming data, allowing the client's WNIC to sleep.

(3) *Active Phase.* The proxy first drains any data buffered during $T_i^{(n)}$, then forwards incoming data until $T_a^{(n)}$ expires. Throughout, it also measures the LAN throughout $G_l^{(n)}$.

Warm-up phase. BURST starts with a warm-up phase that allows connection establishment and allows short connections to complete without a penalty. This warm-up phases lasts 2 s.

Estimating the achievable LAN throughput. At beginning of batch n, BURST computes an *a posteriori* estimate of the LAN throughput achievable for the active phase. This checks if whether is worth going into the *idle* phase. We discuss the robustness of this approach in the next subsection Section 3.4. This estimation is performed in two steps.

First, the proxy derives the medium available link capacity (L) by passively monitoring the wireless medium's busy/idle ratio; a metric previously shown reliable [23]:

$$L^{(n)} = (1 - \text{Medium\_Busy\_Fraction}) \times \text{Link\_Capacity} \tag{1}$$

Secondly, comes a confidence factor, $\alpha \in [0,1]$. α accommodates for other factors that may limit the local throughput but are not reflected in L (e.g shaping). It is inferred from the continuously measured WAN throughput and the previously measured LAN throughput:

$$\alpha^{(n)} = \min\left(1, \frac{G_r^{(n)}}{G_l^{(n-1)}}\right) \tag{2}$$

Finally, the estimated $\widehat{G}_l^{(n)}$ is an weighted sum:

$$\widehat{G}_l^{(n)} = \alpha^{(n)} G_l^{(n-1)} + (1-\alpha^{(n)}) L^{(n)} \tag{3}$$

Whenever asymmetry is present ($G_l^{(n-1)} > G_r^{(n)}$) we have $\alpha^{(n)} = G_r^{(n)}/G_l^{(n-1)}$, so the first term collapses to $\alpha^{(n)} G_l^{(n-1)} = G_r^{(n)}$ and Eq. (3) simplifies to an interpretable form:

$$\widehat{G}_l^{(n)} = G_r^{(n)} + \left(1 - \tfrac{G_r^{(n)}}{G_l^{(n-1)}}\right) L^{(n)}. \tag{4}$$

We can see that the estimate is the WAN rate plus a fraction of the spare medium capacity, where the fraction grows with the observed asymmetry. During the warm-up there is no buffered data, so the measured G_l equals G_r and reveals no headroom. We therefore set the first estimate to the available link capacity, $G_l = L$.

Sufficient Throughput Asymmetry. To prevent buffering when the asymmetry is not significant enough, we only initiate an idle phase when:

$$\frac{G_r^{(n)}}{\widehat{G}_l^{(n)}} \le \beta < 1 \tag{5}$$

The threshold β serves as a sensitivity control for the system. A high β value (closer to 1) encourages aggressive buffering, which increases the probability of Flow Completion Time (FCT) inflation in case of capacity estimation errors. Conversely, a low β value forces a highly conservative behavior, causing the system to miss valid energy-saving opportunities on moderately asymmetric links. In our experiments, we set it arbitrarily to 0.9.

Calculating phase durations. T_a represents the WNIC active time during a given batch. To calculate this duration, we leverage the previously calculated $\widehat{G}_l$ and the measured G_r. T_a and T_i are coupled through a FIFO queue of the proxy's buffer. During an idle phase the proxy accumulates $Q = G_r\, T_i$ bytes. In the subsequent active phase it must flush this backlog *and* the data that continues to arrive at G_r, all at the achievable LAN rate $\widehat{G_l}$. This means:

$$\widehat{G_l}\, T_a = Q + G_r T_a = G_r T_i + G_r T_a, \tag{6}$$

Since a complete batch duration is bounded by $B = T_i + T_a$, we can substitute $T_i = B - T_a$ directly into Eq. (6) to isolate T_a:

$$T_a + \frac{\widehat{G_l} - G_r}{G_r}\, T_a = B \Longrightarrow \frac{\widehat{G_l}}{G_r}\, T_a = B \Longrightarrow T_a = \frac{G_r}{\widehat{G_l}}\, B, \tag{7}$$

Therefore, T_i represents the WNIC idle time during a batch, throughout which the proxy buffers incoming data from the server. Given B and T_a, T_i becomes:

$$T_i = B - T_a = \frac{\widehat{G_l} - G_r}{\widehat{G_l}}\, B. \tag{8}$$

3.4 Robustness and bounds

Channel Volatility during the Idle Phase. BURST assumes short-term channel stability. Specifically, we assume that during batch n, the WAN throughput remains relatively constant throughout the idle phase ($T_i^{(n)}$), and that the LAN throughput observed during the subsequent active phase ($T_a^{(n)}$) matches our prior estimate ($\widehat{G}_l^{(n)} \approx G_l^{(n)}$). If the assumption holds, the data queued at the proxy during $T_i^{(n)}$ is completely drained during $T_a^{(n)}$. However, if channel volatility violates this assumption (e.g., the LAN link unexpectedly degrades), the proxy's buffer will not fully empty. To safeguard against channel volatility, BURST enforces a strict recovery mechanism: the system will not initiate a new idle phase (T_i^{n+1}) if residual data remains in the proxy's queue at the end of the active phase ($T_a^{(n)}$). The client's WNIC remains active, skipping the idle phase until the queue is completely flushed and the bandwidth asymmetry is observed in the future batch. Note that the maximum capacity of this queue is dictated by the proxy's available memory limits.

Signaling Buffering. Since the proxy controls buffering decisions, we opt for implicit signaling, in which the proxy pauses downstream data transmission for up to B seconds. However, this implicit approach must carefully account for OS-level background network behaviors. For instance, the Android operating system periodically transmits system-wide TCP keep-alive packets (empty TCP segments) to prevent intermediate firewalls or NATs from terminating seemingly idle connections. If the proxy's buffering duration exceeds this interval, the OS will awaken the WNIC for a few milliseconds solely to transmit a keep-alive probe, diminishing the energy savings of the idle phase. To prevent these spurious wake-ups, the client must ensure that B remains less than or equal to the OS-configured keep-alive interval (e.g., on Android devices used in our evaluation the default is 75 seconds).

4 BURST's Prototype

This section details the software architecture of BURST at the client and the proxy. Both codebases are written entirely in Rust, and the source code will be made publicly available upon acceptance. The code contains 3697 source lines of Rust code.

4.1 Client Implementation

The client-side modifications required to support BURST are strictly limited to the control exchange during connection establishment (Section 3.2). Once the initial setup with the BURST proxy is complete, standard upper-layer protocols seamlessly take over.

To conduct our evaluations, we built a custom HTTPS client leveraging three widely adopted open-source Rust crates: *Tokio* for the asynchronous runtime and TCP socket management, *Tokio Native TLS* for cryptographic operations, and *Hyper* for the HTTP/1.1 protocol. The sole deviation from a standard HTTPS client built on this stack occurs during the setup phase: before handing the raw TCP socket to the TLS connection manager, our client transmits the BURST control payload and waits for the proxy's acknowledgment. Appendix C explains what an Android client application can do to integrate with BURST exposed as a library.

4.2 Proxy Implementation

The BURST proxy is a user-space process built atop Tokio's asynchronous runtime. The system comprises three logical entities: the *Orchestrator*, the *Coordinator*, and the *Forwarding* tasks. These components and their interactions are depicted in Figure 8. We opt for an event-driven architecture over a poll-based design. The primary motivation is to minimize CPU usage of BURST on resource-constrained APs.

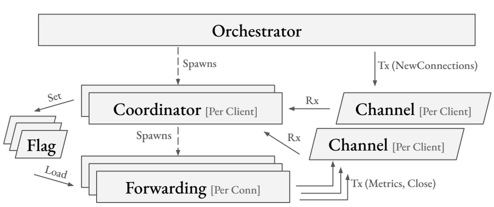


Fig. 8. TCP Proxy High-level Architecture.

Orchestrator. The *Orchestrator* is a singleton task per proxy instance. It is responsible for listening for incoming connections, verifying client credentials, and returning the proxy's control parameters. If the proxy is already handling an active flow from a given client (identified by its configured *Key ID*), the *Orchestrator* routes the new connection to the client's existing *Coordinator* via a message-passing channel. Otherwise, it spawns a new, dedicated *Coordinator* task for that client.

Coordinator. A dedicated *Coordinator* task is maintained per client, based on the configured *Key ID*. It manages scheduling and calculation of the active and idle phases per batch (Section 3.3) for all concurrent flows belonging to that client. Indeed, this grouping enables BURST to coordinate buffering across a client's concurrent flows, which is essential for energy savings when multiple flows compete on the same WNIC. Additionally, the *Coordinator* is responsible for instantiating individual *Forwarding* tasks as new flows from the client arrive.

Forwarding. A *Forwarding* task is instantiated for every active flow. It manages all data path operations: buffering incoming server traffic, flushing the queued downstream to the client, and continuously reporting flow metrics to the *Coordinator*. These metrics include the throughput observed during the active phases and current buffer occupancy.

To store the incoming TCP byte stream from the server, we use a ring buffer with a strict maximum capacity. If the buffer size reaches this limit, the proxy starts draining the buffer towards the client, regardless of the current phase, active or idle phase. Simultaneously, the proxy pauses receiving data from the server whenever the ring buffer runs out of headroom. The maximum size of this is defined by the *Orchestrator* during the initial connection setup, and it is configurable at runtime to allow custom memory budget.

5 Evaluation

In our evaluation we address the following questions: (i) What are the benefits of BURST, and under which operating conditions do they appear? (ii) What overhead does BURST introduce at the proxy?

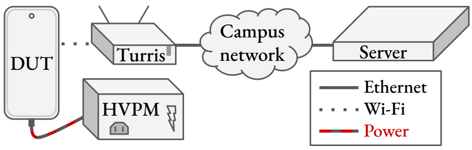


Fig. 9. Testbed.

Device	Platform	HVPM Power Channel	Android Version
Pixel 7A	Google Tensor G2	USB	14
Fairphone 5	Qualcomm QCM6490	Main Channel	14
Pixel 10 Pro	Google Tensor G5	USB	16

Table 2. The smartphones used across our measurements.

5.1 Methodology

Environment. Figure 9 illustrates the environment we use across all our evaluations. Table 2 presents the smartphones used in the evaluation. The two Pixel devices have integrated batteries that are powered over USB, while the Fairphone 5 uses the HVPM main channel to bypass the battery. 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 client smartphones, we removed all pre-installed applications and services, so no network traffic is outside our control. The server is in our university's data center, and the campus network is shared, which explains the potential variance in some measurements. For the BURST's proxy, we use a Turris Omnia Wi-Fi[5] Access Point (AP).

Power Measurement. We measure the device's absolute power consumption using a Monsoon High Voltage Power Monitor [24]. To gain fine-grained insights into component-level energy draws, we also use the On-Device Power Monitor (ODPM) rails [16], only present on Pixel devices. We combine both tools because prior work [12, 17] demonstrates that, under heavy workloads, aggregate ODPM rail measurements can diverge significantly from battery counters and total device power measurements. We sample the rails each 250 ms using perfetto[6]. The increased precision afforded by these tools justifies our focus on Pixel devices in our analysis. All energy and power comparisons throughout our evaluation report the *total* device energy consumption, unless explicitly stated otherwise. For network transfers, we measure this total consumption from flow initiation to completion.

Wi-Fi Power Save Mode. Our objective is to extend the utility of native Wi-Fi PSMs by allowing the client to enter longer low-power states during bulk transfers. Natively, the continuous arrival of packets at the WNIC strictly prohibits sleep states in these scenarios (Section 2.3). We leave the default Wi-Fi Power Save Mode configurations unchanged on both the client and the Access Point. Looking at the network traffic, we observe that the clients was using Unscheduled Automatic Power Save Delivery (U-APSD) [21].

Evaluation Workload and Methodology. For each set of parameters, we run 10 repetitions and report the 95 % confidence interval in our figures as vertical bars. Our evaluation primarily focuses on bulk HTTPS downloads. Additionally, all data transfers are performed entirely in memory to isolate network performance and eliminate disk bottlenecks.

Network Bandwidth Configuration. By default, we configure the testbed with an unconstrained LAN link while artificially limiting the WAN bandwidth to 22 Mbps per flow. We select this specific WAN limit to mirror the median per-connection internet throughput reported by Cloudflare (Section 2.2). Later in this section, we evaluate how varying these baseline bandwidth constraints impacts our system and its benefits.

5.2 Validating BURST Buffering Mechanism

[5]OS version 7.1
[6]https://perfetto.dev/docs/

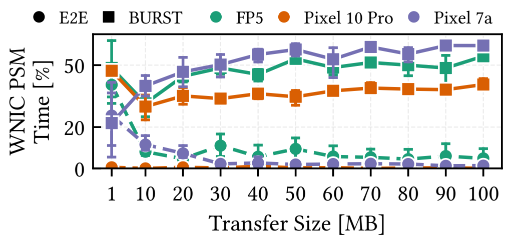
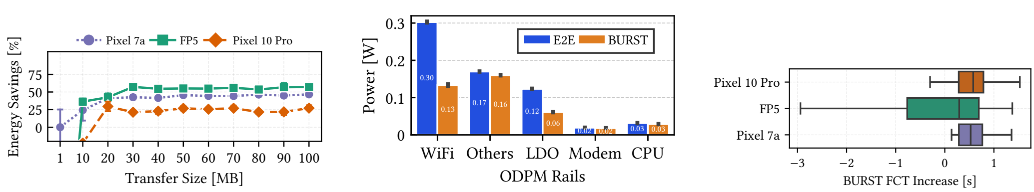


Fig. 10. Percentage of time the client's WNIC spends in PSM per transfer.

BURST enables WNIC low-power. We determine the WNIC's total sleep duration by tracking the Power Management (`PwrMgnt`) flag in the QoS MAC frames sent from the client to the AP. By setting this flag, the client signals the AP to temporarily buffer incoming frames, allowing the WNIC to safely transition into a low-power state. Figure 10 illustrates that under throughput asymmetry, BURST allows the client WNIC to be in low-power 38-51% of the transfer, an improvement over the to 1-20% achieved by end-to-end baseline. The WNIC idle residency depends on the smartphone. For small 1 MB flows, we observe no significant difference; the transfer completes almost instantaneously (in under one second), finishing entirely within the initial active phase. Beyond 1 MB, for the baseline, continuous packet arrivals natively force WNIC to remain awake (see Section 2.3), restricting its average PSM time to 4 %. On the contrary, BURST actively paces the flow, enabling the WNIC to spend an average of 48 % of the transfer time in PSM. The FP5 is more energy conservative with an average Wi-Fi PSM time of 8.19 %. Additionally, Figure 18 illustrates that BURST adheres to the 1 s batch constraint, as the 99th percentile of sleep periods is at 0.80 s (see Appendix D.1 for more details).

5.3 BURST's Energy Savings and FCT Impact



(a) Mean device energy savings of BURST vs. the end-to-end baseline. (b) WNIC power consumption reported by ODPM rails on Pixel 7a. (c) BURST impact on the Flow Completion Time (FCT).

Fig. 11. Single-flow evaluation with a batch duration of 1 s.

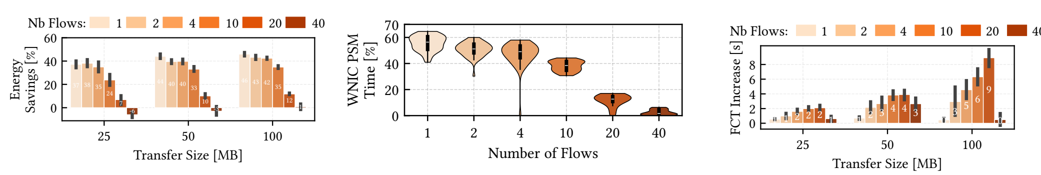
BURST reduces WNIC and system power consumption. Figure 11b illustrates the per-second power consumption across hardware components of the Google Pixel 7a. Compared to the baseline, BURST reduces WNIC power consumption during the transfer by ∼56 % (from 0.3 W to 0.13 W). Alongside these Wi-Fi energy savings, we observe a reduction in the power drawn by the Low Dropout (LDO) regulators. The baseline end-to-end transfer necessitates a higher sustained current draw, thereby increasing the regulatory load on the LDOs. Furthermore, we observe that BURST achieves these energy savings without additional power consumption elsewhere, as CPU and other component power profiles remain unaffected at the client.

Total Device Energy Savings Per Transfer Size. Figure 11a illustrates how total device energy savings evolve alongside transfer size. BURST's relative energy savings increases proportionally with transfer size until plateauing at approximately 24, 53, and 45 % for the Pixel 10 Pro, Fairphone 5 and Pixel 7a respectively. For short 1 MB transfers, we observe high variance and a net energy penalty (averaging −4.9 %). This is an expected caveat of the system's design: BURST imposes an initial one LAN-RTT setup penalty, which outweighs the benefits for transfers completing in under a second. However, for payloads between 10 MB and 30 MB, relative savings increase to reach the maximum energy savings for all transfers up to 100 MB. Measurements up to 1 GB show 48% energy savings at similar throughput asymmetry on a Pixel 7a (see Appendix D.3). These energy savings are driven by two interacting hardware effects: direct reductions in WNIC activity and indirect relief on the LDO voltage regulators. The data shows that the majority of the overall energy savings originates directly from the WNIC, where power consumption drops by more than 55 %. The absolute energy savings expressed in Joules are depicted in the appendix (Figure 20).

Impact of BURST on Flow Completion Time. Figure 11c plots the impact of BURST on Flow Completion Time (FCT) for payloads ranging from 1 MB to 100 MB. Across all evaluated flows, BURST introduces a median FCT penalty of 0.4 s, and a 95th percentile of 1.06 s. For small flows under one second, this translates to a 40 % relative increase in transfer time. However, for transfers taking several seconds or longer, this overhead decreases to a negligible fraction of the total FCT: under 8 % from 10 MB and above. This confirms that while BURST is unsuitable for micro-transfers, its overhead is acceptable for bulk downloads, for which it is designed. On the Fairphone 5, its aggressive Wi-Fi PSM to impact the baseline FCT, that is why with BURST can actually improve the FCT.

5.4 Impact of Traffic Load and WAN Throughput

BURST Maintains Energy Savings Across Concurrent Flows. Mobile applications multiplex traffic across multiple concurrent network flows. To evaluate this scenario, we configure a client to simultaneously download identical-sized files over parallel HTTPS connections. We measure the total energy consumption from the initiation of the first transfer until the completion of the final flow. For BURST's evaluation, all concurrent flows are multiplexed through a single proxy instance. The WAN bandwidth is set to the number of flows n, times the 22 Mbps, e.g for 10 flows we set the WAN bandwith at 220 Mbps. However, the Pixel7a can only attain 600 Mbps of LAN bandwidth.



(a) Mean device energy savings of BURST vs the end-to-end baseline.
(b) Client's WNIC total PSM duration with BURST transfers.
(c) Flow Completion Time (FCT) variation relative to the baseline.

Fig. 12. Multiple flows (n) evaluation on a Google Pixel 7a, with a batch duration of 1 s. For this experiment the WAN bandwidth is set to $n \times 22$ Mbps

Figure 12a illustrates that when the local wireless medium operates under a bandwidth ceiling, BURST's energy gains diminish as flow concurrency increases. This behavior is expected: while the WAN bottleneck maintains a stable 22 Mbps allocation per flow, the finite aggregate LAN bandwidth must be divided among an increasing number of concurrent clients. Consequently, as the flow count grows, the fair-share local throughput available to each individual flow decreases, making LAN and WAN throughput converge. Since BURST's batching loop relies entirely on throughput asymmetry between these two networks to clear buffers rapidly and put the radio to sleep, a shrinking asymmetry ratio inevitably diminishes the client's low-power sleep opportunities (formal calculation in Section 3.3). For medium and large transfers (≥ 25 MB), the average energy savings decreases from 46% down to 25% for 1 to 10 concurrent flows, respectively. At 20 flows, savings decrease to 12%, and by 40 concurrent flows, the throughput asymmetry vanishes entirely, eliminating any observable energy benefits.

This degradation is further explained in Figure 12b, which plots the duration the client's WNIC spends in Power Save Mode (PSM). The results confirm that as per-flow LAN goodput shrinks, WNIC sleep opportunities decrease. For 10 or fewer flows, the median WNIC PSM duration accounts for 39% to 50% of the total transfer time. However, at 40 flows, this median drops to just 3%, explaining the absence in energy savings observed in Figure 12a.

Finally, Figure 12c evaluates the impact of concurrency on Flow Completion Time (FCT). As expected, increasing the number of concurrent flows increases the absolute FCT overhead. Large flows are the most heavily impacted, exhibiting an average penalty of up to 9 s when comparing 20 concurrent flows pairwise against the baseline. However, examining the FCT distribution (Figure 17) reveals a synchronization artifact. Because BURST buffers and flushes all concurrent flows simultaneously, it enforces nearly identical FCTs across the entire batch. The low variation in BURST's flow completion times, contrasts with the baseline's uncoordinated flows FCTs. Consequently, this synchronization inflates the pairwise FCT differences in Figure 12c, while the completion time of the last flow shows BURST adding under 2 s relative to the baseline.

Interestingly, the absolute FCT overhead at 40 concurrent flows is lower than at 20 flows, even dropping to zero for 100 MB transfers. This occurs because, at 40 multiplexed flows, the available per-flow LAN throughput is the bottleneck (observed LAN limit at 600 Mbps). BURST detects it and acts as a normal proxy, resulting on less impact on the FCT.

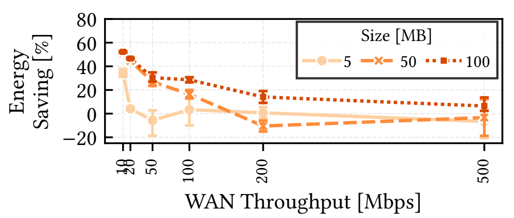


Fig. 13. Impact of WAN throughput on single-flow energy savings on a Pixel 7a. The unconstrained LAN link achieves up to 600 Mbps.

Impact of the WAN Throughput. Cloudflare measures the per-connection median throughput within ranges of 7 Mbps and 27 Mbps [3] (Illustrated in Figure 16). We evaluate BURST's performance across a broader spectrum of WAN throughputs (from 10 Mbps to 500 Mbps). Figure 13 plots the resulting device energy savings.

The figure illustrates that, as WAN throughput increases, the throughput asymmetry shrinks, naturally diminishing the proxy's ability to generate WNIC sleep opportunities. Furthermore, these savings are heavily dependent on the transfer duration. Under asymmetric conditions, longer transfers allow the per-batch energy savings to compound significantly. Short transfers (under 2 s) simply finish before this compounding can take effect. For medium-to-large bulk downloads on WAN links below 200 Mbps, BURST reliably delivers 20 % to 50 % energy savings. For the same bandwidth, the gains depend transfer size: the larger the transfer size, the longer the transfer duration, the more the energy savings.

5.5 BURST Impact at the AP

Wi-Fi Gateway CPU Overhead. Figure 14 plots the CPU usage of the dual-core Access Point, which runs the BURST proxy. We limit our evaluation to 60 concurrent flows due to an observed limitation of the Pixel 7a Wi-Fi driver[7].

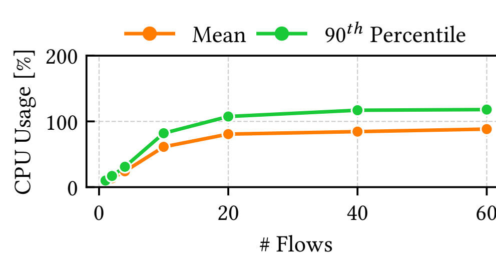


Fig. 14. Mean and 90^{th} percentile CPU utilization of the BURST process executing on the Wi-Fi Access Point.

The proxy can handle 60 multiplexed flows, with an average CPU utilization of 89 % and a 90^{th} percentile of 120 %. In our measurements, we observe peak CPU usage occurs during connection establishment and teardown phases, primarily due to the control-plane operations such as memory allocation and asynchronous task spawning. At 40 concurrent flows or beyond, the WAN-LAN asymmetry disappears entirely (see Section 5.4), meaning the proxy defaults to standard data forwarding. Despite the absence of

[7]Exceeding 60 concurrent flows triggers a driver-level overflow on the Pixel 7a, causing TCP connections to fail.

batching operations, CPU consumption remains high, revealing that user-space application data forwarding is the true system bottleneck. Porting BURST's proxy to kernel space represents a promising path forward; previous work on a kernel-space proxy show they can efficiently [25].

6 Authenticated Converters

To implement the client-proxy connection, we extend the 0-RTT TCP Convert Protocol [26]. This experimental IETF RFC was designed to ease the deployment of TCP extensions like Multipath TCP [27]. It introduces converters that provide conversion service for one or more TCP extensions. To minimize its latency impact, it uses TCP Fast Open [28]. The Converter Protocol [26] exchanges control information using Type-Length-Value (TLV) records at the head of the TCP byte stream. The `Connect` TLV carries the IP address, TCP Port of the target server, and requested TCP extensions.

The Converter Protocol was designed to be deployed in a trusted environment such as enterprise and access networks [29]. However, for BURST to be deployed over cellular networks (e.g Mobile Virtual Network Operators deployment), the proxy needs to verify the credentials of a client. In this section, we propose an authenticated extension to the Converter Protocol. This extension allows (i) the proxy to *authenticate* a client, and (ii) the client to *verify* the legitimacy of the proxy.

6.1 0-RTT Protocol Extension

Our extension adds two new TLVs: **`AConnect`**, and **`AConfirm`**. Figure 15 depicts when these TLVs are used. **`AConnect`** hides destination address and port, along with a random challenge. The proxy decrypts the latter, and responds with a clear-text **`AConfirm`** TLV, containing the initial challenge to prove legitimacy. The details about the content of the new TLVs are detailed in Appendix E.

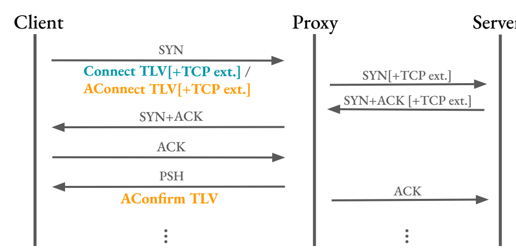


Fig. 15. The setup of 0-RTT Converter Protocol [26]. In teal (■) is the 0-RTT Convert Protocol and in orange (■) our authenticated Converter Protocol.

6.2 The use of AEAD

To guarantee the confidentiality and authenticity of the **`AConnect`** payload, our design leverages the Authenticated Encryption with Associated Data (AEAD) framework [30]. An AEAD provides both a CPA-Secure encryption and a secure MAC, as long as the Initialization Vector/Nonce usage is unique for a given key [30]. This combination offers non-malleable authenticated encryption, meaning that if the decryption succeeds, the recovered plaintext is guaranteed to be authentic and produced by the client owning the same encryption key. This framework requires both the client and the proxy to share the cryptographic state, we detail the information in Appendix E.

To distribute this shared cryptographic material, we rely on an out-of-band control plane (e.g., a centralized configuration server managed by the service provider) and is orthogonal to the protocol design.

7 Related Work

The principle of offloading host network functions to a peer for energy efficiency has been studied prior to our work [6–10]. In this section we compare, BURST, to the state-of-the-art.

CatNap. Dogar et al. [7] introduced Catnap, which to our knowledge is the closest work to BURST. Catnap successfully exploits WAN-LAN throughput asymmetry to split TCP connections and save client energy for delay-tolerant applications. However, its scheduling logic relies on explicit, clear-text application server hints (e.g., total transfer size). Because it was proposed prior to the adoption of TLS, this requirement makes it incompatible with modern secured traffic. Furthermore, Catnap requires server-side modifications which hinders its adoption, as most servers lack clear incentives to optimize client battery life.

In contrast, BURST achieves energy savings without inspecting application payloads or requiring server cooperation. By restricting all coordination strictly between the client and the proxy, our solution seamlessly supports modern encrypted protocols over TCP such as TLS. Preserving the end-to-end encryption introduces a performance trade-off. Because Catnap knows the exact transfer lengths, it can schedule long WNIC sleep phases, keeping the device in PSM for up to 80% of the connection duration. Operating without access to encrypted metadata, BURST must dynamically schedule traffic into shorter, periodic batches (Section 3.3). This necessitates more frequent WNIC wake-ups, yielding the client's WNIC PSM fraction to roughly 50%. A direct, empirical head-to-head comparison was not feasible due to the unavailability of Catnap's source code. However, we conduct measurements with a client-side oracle that has access to transfer size and we obtain similar results to Catnap in Appendix D.2.

Network Connection Proxy. Jimeno et al. [6] proposed a Network Connection Proxy (NCP), that enables a client to enter low-power mode while maintaining network connectivity. When the client wants to enter idle mode, it signals its intent and updates the application state with the NCP. Then the NCP maintains the full network presence by generating packets and/or responding to incoming packets. Once custom events occur the NCP wakes up the client and moves the application logic back to the client. Requiring access to the application state, unlike BURST.

Somniloquy. In efforts to limit power consumption within enterprise network PCs, Agarwal et al. [8] introduced Somniloquy. Their work proposes an external low-power processor within the PC's network interface to maintain network presence while the whole PC is idle. Similarly to NCP[6], this requires cross stack integration with the OS, the application and their daemon. Therefore this is not as flexible as BURST, at least for mobile devices.

8 Discussion and Future Work

In this section, we address the current limitations of BURST and outline directions for future research.

QUIC Compatibility. QUIC [31] is a modern transport that unlike TCP, encrypts its transport-layer control information, such as packet numbers, shielding it from the network. Consequently, an access point cannot *transparently* buffer application data and acknowledge packets on behalf of the client, without having packet protection information. However, recent literature, such as Sidekick [32], has demonstrated the feasibility of exploiting QUIC's unencrypted network-level characteristics to construct non-intrusive performance proxies. Future work could leverage these side-channel techniques to extend BURST's energy-saving architecture to support QUIC traffic.

Small transfer sizes. For short transfer sizes (< 5 MB), BURST does not improve the energy consumption. This is due to the transfer completing in sub-second duration, which is not enough for BURST's loop to operate (at least 2 s needed). Future work could also exploit Round Trip Time (RTT) differences between the WAN and LAN to start buffering after initial application data. However, this would require strong timing constraint to ensure minimum impact on the FCT.

System-Wide Energy Analysis. Prior literature demonstrates that for network equipment such as routers [33] and Customer Premises Equipment (CPE) [34], power consumption increases marginally with operational traffic load. While this paper primarily focuses on maximizing client-side energy savings under this premise, we acknowledge that BURST's CPU overhead at the access point could introduce additional power draw on the AP. Future work will look at the system wide energy consumption, across diverse hardware configurations and network workloads

Cellular Networks. With the Authenticated 0-RTT extension introduced in Section 6, BURST can expand its deployment scope beyond Wi-Fi environments. BURST's batching architecture is possible with modern cellular power-saving states. Specifically, the 5G New Radio *RRC Inactive* state allows for rapid connection resumption with minimal signaling overhead.

9 Conclusion

With this work, we aim to revive the discussion surrounding energy-aware proxying for mobile devices. We propose and implement BURST, an application-agnostic and TLS-friendly proxy that exploits WAN-LAN throughput asymmetry to increase the client sleep opportunities during transfer. By batching traffic into high-speed LAN bursts, BURST directly extends the duration of native Wi-Fi Power Save Modes. Our evaluation on three different smartphones shows that BURST allows the Wireless NIC to spend 38-51% of the transfer duration in low-power mode. While total energy savings are inherently hardware-dependent, BURST achieves total device energy savings between 25 % and 45 %, with an overall Flow Completion Time (FCT) overhead under 2 s.

Artifacts. For reproducibility, we will publish the BURST's source code along with the measurements scripts, upon acceptance.

References

[1] 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.

[2] 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.

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

[4] 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.

[5] Europe's Hidden Mobile Performance Gap: Peak-Hour Congestion and Seasonality | Ookla® — ookla.com. https://www.ookla.com/articles/europe-mobile-peak-hour-congestion-2026. [Accessed 20-05-2026].

[6] 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.

[7] Fahad R Dogar, Peter Steenkiste, and Konstantina Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 107–122, 2010.

[8] 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.

[9] Ning Ding, Abhinav Pathak, Dimitrios Koutsonikolas, Clay Shepard, Y Charlie Hu, and Lin Zhong. Realizing the full potential of psm using proxying. In *2012 Proceedings IEEE INFOCOM*, pages 2821–2825. IEEE, 2012.

[10] Nan Jiang, Mehmet Can Vuran, Sheng Wei, and Lisong Xu. Qos-aware network energy optimization for danmu video streaming in wifi networks. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pages 1–10. IEEE, 2021.

[11] Aaron Yi Ding, Bo Han, Yu Xiao, Pan Hui, Aravind Srinivasan, Markku Kojo, and Sasu Tarkoma. Enabling energy-aware collaborative mobile data offloading for smartphones. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 487–495, 2013. doi: 10.1109/SAHCN.2013.6645020.

[12] 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.

[13] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 119–130, 2002.

[21] Adriano Vinhas, Vitor Bernardo, Marilia Curado, and Torsten Braun. Performance analysis and comparison between legacy-psm and u-apsd. In *Proc. 13th Portuguese Conf. Comput. Netw.*, pages 7–12, 2013.

[22] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135 (Informational), June 2001. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3135.txt.

[23] Dhruv Gupta, Daniel Wu, Prasant Mohapatra, and C-N Chuah. Experimental comparison of bandwidth estimation tools for wireless mesh networks. In *IEEE INFOCOM 2009*, pages 2891–2895. IEEE, 2009.

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

[25] Nicolas Keukeleire, Benjamin Hesmans, and Olivier Bonaventure. Increasing broadband reach with hybrid access networks. *IEEE Communications Standards Magazine*, 4(1):43–49, 2020.

[26] O. Bonaventure (Ed.), M. Boucadair (Ed.), S. Gundavelli, S. Seo, and B. Hesmans. 0-RTT TCP Convert Protocol. RFC 8803 (Experimental), July 2020. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc8803.txt.

[27] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684 (Proposed Standard), March 2020. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc8684.txt.

[28] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413 (Experimental), December 2014. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7413.txt.

[29] Matthieu Baerts, Nicolas Keukeleire, and Olivier Bonaventure. Leveraging the 0-rtt convert protocol to improve wi-fi/cellular convergence. In *Proceedings of the 2021 Applied Networking Research Workshop*, pages 46–48, 2021.

[30] D. McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116 (Proposed Standard), January 2008. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc5116.txt.

[31] 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.

[32] Gina Yuan, Matthew Sotoudeh, David K Zhang, Michael Welzl, David Mazières, and Keith Winstein. Sidekick:In-Network assistance for secure End-to-End transport protocols. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1813–1830, 2024.

[33] Romain Jacob, Lukas Röllin, Jackie Lim, Jonathan Chung, Maurice Béhanzin, Weiran Wang, Andreas Hunziker, Theodor Moroianu, Seyedali Tabaeiaghdaei, Adrian Perrig, and Laurent Vanbever. Fantastic Joules and Where to Find Them. Modeling and Optimizing Router Energy Demand. In *IMC '25: Proceedings of the 2025 ACM Internet Measurement Conference*, Madison, WI, USA, October 2025. Association for Computing Machinery. doi: 10.1145/3730567.3732920. URL https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/728960/3730567.3732920.pdf.

[34] Robin Dethienne, Jérôme Louveaux, and David Bol. A power consumption model for customer premise equipment: Methodology and application. In *2025 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 01–06. IEEE, 2025.

[35] W. Eddy (Ed.). Transmission Control Protocol (TCP). RFC 9293 (Internet Standard), August 2022. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc9293.txt.

[36] X. Zhang and T. Tsou. IPsec Anti-Replay Algorithm without Bit Shifting. RFC 6479 (Informational), January 2012. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc6479.txt.

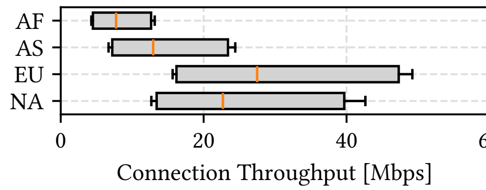


Fig. 16. Cloudflare IQI Bandwidth from January to July of 2025.

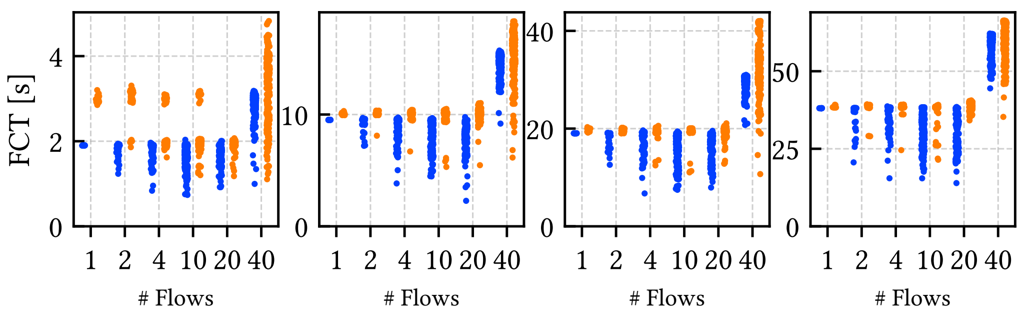


Fig. 17. Individual FCT as we increase the number of clients.

```
// 1. Connect to proxy
Socket sock = new Socket(proxyIp, proxyPort);
OutputStream out = sock.getOutputStream();
InputStream  in  = sock.getInputStream();
// 2. Send AConnect TLV (built by Rust)
byte[] tlv = BURST.buildAConnectTlv(serverIp, serverPort, ...);
out.write(tlv);
byte[] resp = in.readNBytes(PMTU);
long[] decoded = BURST.decodeResponse(resp);
// 3. Wrap in TLS - Android SSLSocket over the existing connection
SSLSocket tls = (SSLSocket) SSLSocketFactory.getDefault()
      .createSocket(sock, serverIp, serverPort, true);
tls.startHandshake();
// Send HTTPS request and receive body ...
```

Listing 1. Example of Java HTTPS client integrating with BURST.

A Cloudflare IQI

Figure 16 shows the 25^{th}, 50^{th} and 75^{th} percentile of the per-connection throughput under typical usage as measured by Cloudflare.

B FCT with different number of flows

Figure 17 shows the FCT as we increase the number of flows. The individual points are the individual flows.

C Client's Listing

Listing 1 illustrates a Java pseudo-code of HTTPS client integrating with BURST . The only deflection to a standard client using the same stack is at the connection establishment. When integrating with android application, we provide a Java Native Interface API to our Rust code that generates and parses the BURST control messages. Letting the android app send and receive control messages. Once the BURST setup is done the android app can parse the TCP socket to Android's SSL socket to initiate the TLS handshake.

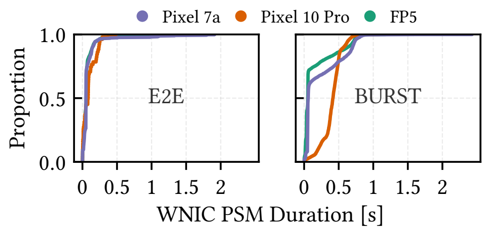


Fig. 18. Cumulative Distribution Function of the client's WNIC low-power phase durations during BURST transfers.

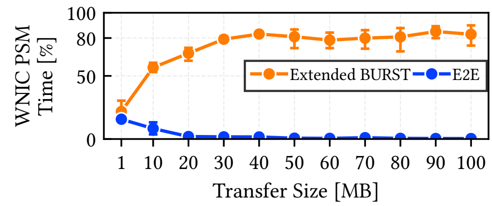


Fig. 19. WNIC PSM duration fraction for single transfer on a Pixel7a. With an extended version of BURST where the client is responsible for buffering.

D Other Measurements Results

D.1 WNIC Idle Duration

Figure 18 illustrates the CDF of WNIC sleep durations (idle phases, T_i) compiled from 4,841 samples across 110 transfers. The data verifies that BURST adheres to the 1 s batch constraint, we observe an 99$^{\text{th}}$ percentile of sleep periods at 0.8 s. Additionally, the distribution of these durations shows the proxy's capacity to dynamically adapt its active/idle scheduling to fluctuating network conditions.

D.2 Emulating Catnap's Heuristic

To facilitate a fair quantitative comparison in the absence of Catnap's [7] source code, we implement an emulated version of its core heuristic using BURST. The advantage of Catnap over BURST comes from its access to clear-text payloads or server-side metadata, which provides the system with the total transfer size. To replicate this environment, we extend our client application to leverage the HTTP headers to get the total transfer length and manage the scheduling loop. As a result this reduces the proxy to a passive network-side buffer that continues to receive data but does not flush it to the client unless the client indicates it. We leverage the TCP client window size to force the proxy to buffer by indicating a buffer size of zero [35].

Figure 19 shows that this variant allows the client WNIC to remain in a low-power state for 80 % of the total transfer duration. This outcome closely replicates the empirical results reported by Dogar et al. [7]. Ultimately, this validation confirms that Catnap's marginal residency advantage over BURST is derived exclusively from its visibility into the transfer size: additional information that requires either un-encrypted clear-text traffic or explicit server-side cooperation.

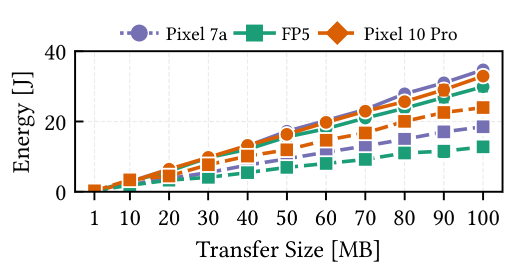
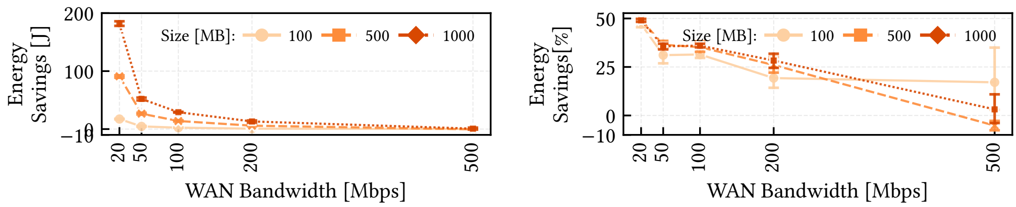


Fig. 20. The energy savings brought by BURST expressed in Joules.



(a) Absolute energy savings of BURST compared to the baseline.

(b) Relative energy savings of BURST compared to the baseline.

Fig. 21. Evaluating Energy Savings beyond 100 MB on the Pixel 7a. Additionally we vary the WAN bandwidth.

D.3 Large Transfer Sizes

Figure 21 shows the absolute and relative energy savings up to 1 GB, on the Pixel 7a. We observe that the relative energy savings remain at ~48 % similar to transfers under 100 MB. The new finding is that volume cannot compensate for weak asymmetry: at a 500 Mbps WAN the savings vanish despite transfers well past the 2 s warm-up. This is because the throughput asymmetry $G_r/\widehat{G_l} = 0.83$ (Eq. (5)) is close to β of 0.9 (our default), so BURST almost never enters an idle phase and the WNIC stays awake throughout.

E 0-RTT Extension Details

E.1 Content of TLVs

AConnect (Authenticated Connect)

The baseline **`Connect`** TLV contains information about the server's IP address, server's TCP port and requested TCP extensions. Our extended authenticated version, **`AConnect`**, adds: Cipher Length, Sequence Number (*Seq*), Key Identifier (*Kid*), TCP Options and a Ciphertext block. The **`AConnect`**'s Ciphertext contains the following fields:

- **Destination IP address and TCP Port.** These indicate the target server to the proxy.
- **Challenge.** This 64-bit random value is generated by the client and subsequently returned by the proxy. It acts as a proof-of-decryption, cryptographically confirming the proxy's authenticity to the client before data transfer begins.
- **Auxiliary data.** An opaque field for extensible control signaling. For example BURST uses this field to encode the requested client's *batch duration*.

AConfirm (Authenticated Confirm)

The **`AConfirm`** TLV acts as the proxy's binding acknowledgment that it will handle the client's connection. It contains:

- **Challenge.** The exact 64-bit value extracted from the **`AConnect`** ciphertext, validating the proxy's identity to the client.
- **Auxiliary data.** As for the auxiliary data in the **`AConnect`**, it allows extensive control signaling. However, the auxiliary data in the confirmation are not encrypted. The proxy can use this field to advertise deployment-specific information. BURST uses this field to encode the available buffer size for ongoing flow.

E.2 The use of AEAD

The AEAD framework requires both the client and the proxy to share the following cryptographic state:

- An AEAD encryption algorithm and its Initialization Vector (IV). The IV can be global to all clients as long as keys are unique.
- An encryption key (*Ke*) and its corresponding Key Identifier (*Kid*). The length of the (*Ke*) depends on the chosen suite, however for the *Kid* we chose a 64-bit value.

The nonce used within the AEAD for encryption and decryption must be unique and we must be able to prevent the adversary from replaying messages. To cover the uniqueness, the nonce would be built from XORing the lower 64 bits of the IV with the sequence number. The initial sequence number can start at 0 but must monotonically increase. The proxy remembers a window around the highest sequence number that lead to a valid decryption and verify that any new connection attempt fill a empty hole within this window, or reject otherwise [36].