JUNCTION: Enabling 5G-WiFi Convergence at Scale with Programmable Switches

Anonymous Author(s)
Paper #1570930879

***Abstract*—With the increasing demand for mobile traffic and the proliferation of multi-homed mobile devices, 5G-WiFi convergence has become a trend in mobile networks. To fully leverage this convergence, Access Traffic Steering, Switching, and Splitting (ATSSS) have been introduced to couple 5G and WiFi networks for increased throughput and reliability. However, existing software-based ATSSS proxies suffer from performance overhead and latency issues, making them unsuitable for large-scale deployment. To address this, we propose JUNCTION, an end-to-end system that leverages programmable switches to build a scalable and performant ATSSS solution. JUNCTION consists of JUNCTION-UPF, a hardware-accelerated ATSSS-enabled UPF, and JUNCTION-UE, a userspace client program. Our evaluations demonstrate that JUNCTION can cost 225× less and consume 84.7× less power while keeping the processing latency overhead under 3 µs. We deploy JUNCTION on a real 5G-WiFi testbed and validate its traffic steering functionality.**

I. INTRODUCTION

With the ever-increasing mobile traffic demands alongside the increasing number of mobile users [1], mobile operators are constrained by the available licensed radio spectrum to offer greater bandwidth and to support more use cases and users. The proliferation of multi-homed mobile devices has driven operators to explore access technologies operating in the unlicensed spectrum to complement cellular networks.

In this light, operators are streamlining the management and tightly integrating two radio access technologies (RATs), i.e., 5G and WiFi, at the operator core. This has driven the emerging trend of 5G-WiFi convergence [2]. Under this context, mobile users can now access 5G services with expanded coverage and greater reliability across 5G and the increasingly ubiquitous [3] WiFi networks.

To unleash the potential of 5G-WiFi convergence, Access Traffic Steering, Switching, and Splitting (ATSSS) have been introduced in 3GPP Release 16 [4] to couple together non-3GPP access networks (e.g., WiFi) and 3GPP access networks (e.g., 5G NR) to enable increased throughput, robustness and reliability. We illustrate ATSSS in Fig. 1.

Enabling ATSSS requires support from the 5G user plane function (UPF) and user equipment (UE). At the UPF, an ATSSS component is introduced as the anchor point for the multiple accesses, i.e., 5G NR and WiFi, whereas an ATSSS agent is introduced at the UE. ATSSS masks the multiple accesses from the user and upper-layer applications to enable seamless connectivity and aggregated bandwidth.

As 5G deployments mature [1] and WiFi technologies improve (e.g., WiFi 7 is expected to support up to >40 Gbps peak throughput [5]), it is crucial that ATSSS-enabled UPF can cope with increasing traffic demands to reap the full benefits of 5G-WiFi convergence.

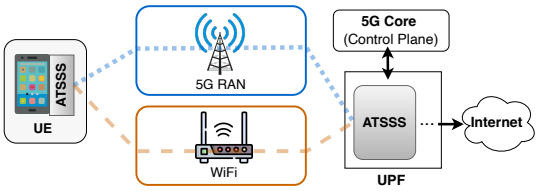


Figure 1: High-level overview of ATSSS.

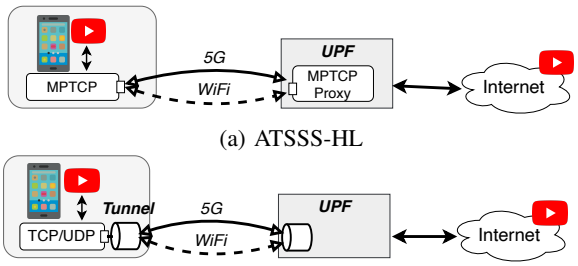
Existing deployments [6]–[8] use software-based ATSSS proxies that use protocols like multipath TCP (MPTCP) [9], to exploit the multiple access networks. However, existing software proxies impose non-negligible overhead on forwarding performance and end-to-end latency is significantly impacted under load, which degrades the overall user quality of experience (QoE).

While one can horizontally scale the ATSSS-enabled UPF (i.e., scale-out) by increasing the number of commodity servers to cope with the enormous traffic loads, such a scale-out approach is unsuitable and unsustainable. The reason is two-fold. First, horizontal scaling increases the CapEx (e.g., hardware and/or rack space acquisition cost) and OpEx (e.g., power, cooling, and engineering time consumed in system troubleshooting). Furthermore, as Moore's law falls behind the Internet traffic growth [10], the single-core CPU performance growth will consistently be outpaced by the increasing traffic growth in the foreseeable future [11].

This necessitates a sustainable and scalable solution for operators to enable the widespread deployment of ATSSS. Programmable switches with up to 12.8 Tbps line rate packet processing capability present a viable and promising platform to realize a performant, and scalable ATSSS solution.

To that end, we propose JUNCTION, an end-to-end system to drive the roll-out of ATSSS at scale. JUNCTION consists of two key components: *(i)* JUNCTION-UPF, a hardware-accelerated ATSSS-enabled UPF build on top of multi-Tbps programmable switches; *(ii)* JUNCTION-UE, a userspace client program to enable ATSSS on the user devices.

Designing JUNCTION-UPF is non-trivial, especially given the limited hardware resources and restricted programming model. Particularly, we need to maintain a large number of user sessions and their corresponding path states to enforce traffic steering decisions. Adopting approaches in existing transport-layer approaches (e.g., MPTCP) results in hugely inefficient use of scarce memory resources. To address this, we



(a) ATSSS-HL

(b) ATSSS-LL

Figure 2: Difference between ATSSS-HL and ATSSS-LL.

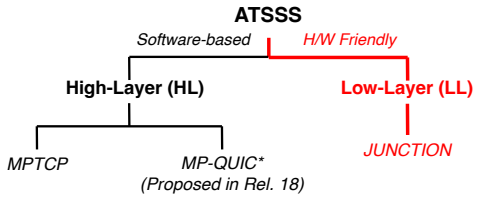


Figure 3: Design space.

design a hierarchical lookup structure to represent the ATSSS sessions and a unified traffic steering structure to maintain path states lest memory wastage on JUNCTION-UPF. Besides, as packets can be delivered in more than one path, i.e., 5G and/or WiFi, packets can arrive out-of-order and need to be reordered on the programmable switch, which is challenging given the restricted memory access patterns, and feed-forward switching pipeline. To address out-of-order packets, we implement a mechanism to reorder packets using pause-able FIFO queues. In this paper, our key contributions are as follows:

1) We show that software-based proxies suffer from performance overhead and latency issues, and the reason is fundamental to the slow CPU performance (§III).
2) We prototype JUNCTION-UPF– the first ATSSS-enabled 5G user plane function (UPF) on an Intel Tofino2 programmable switch (§IV,§V).
3) We show that compared to software-based proxies, JUNCTION can cost 225× less and consume 84.7× less power with <3 µs processing latency overhead, and we validate JUNCTION on an actual 5G-WiFi testbed (§VI).
4) Together with JUNCTION-UPF, we implement JUNCTION-UE, and JUNCTION agent for an end-to-end ATSSS system. JUNCTION's implementation is publicly available at [12].

The paper is organized as follows: We present the background and related work in §II. Then, we highlight the motivation for JUNCTION in §III followed by the technical challenges in §IV. We present the design and implementation of JUNCTION in §V. Finally, we present the evaluations in §VI before concluding the paper in §VII.

II. Background and Related Work

In this section, we provide the required background and introduce the related work.

A. ATSSS

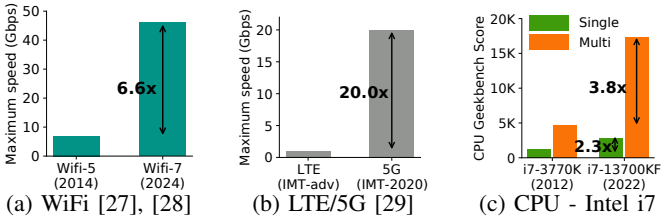
3GPP Release 16 enables 5G-WiFi convergence by introducing a standardized function called ATSSS[1] – Access Traffic Steering, Switching, and Splitting [4]. The goal of ATSSS is to deliver seamless connectivity and aggregated throughput by leveraging multiple access networks. This is achieved through the following *traffic steering modes* defined in the standards [4], [13]: *(i) active-standby*: traffic is switched over to the backup access when the primary fails; *(ii) load balance*: traffic is split across both networks based on a static/dynamic ratio; *(iii) smallest delay*: traffic is steered over the link with the lowest RTT; *(iv) priority-based*: traffic is sent over priority access, and overflows to the backup access, if congested.

To activate ATSSS, an ATSSS session, technically called the MAPDU (Multi-Access Packet Data Unit) session, is established between the user equipment (UE) and the user-plane function (UPF) of the 5G core network (see Fig. 1). The ATSSS sessions are established on top of the underlying PDU sessions for 5G and/or WiFi. During the session establishment process, the 5G core networks' control plane (i.e., SMF and PCF) conveys both the UE and the UPF with information such as the supported steering modes, (virtual) IP address, measurement assistance information, etc. Between a UE and the UPF, there can be multiple ATSSS sessions in parallel to cater to the different steering modes required to support different applications. For example, on-demand video applications (e.g., YouTube, NetFlix) could require *load balancing* while interactive online games could prefer *smallest delay* as the steering mode. Note that ATSSS sessions are specific to an application, i.e., YouTube and NetFlix together require two ATSSS sessions even if they use the same steering mode.

Network operators can use two *approaches* to realize the ATSSS (see Fig. 2) – at the higher-layer (ATSSS-HL) or at the lower-layer (ATSSS-LL)[2]. ATSSS-HL implements steering at the transport layer through a proxy e.g. an MPTCP [9] proxy as shown in Fig. 2a. Being at the transport layer itself, the ATSSS-HL includes mechanisms for reliable delivery, congestion control, and multi-path packet ordering. On the other hand, ATSSS-LL implements steering at the lower layers of the network protocol stack (Ethernet to IP) through "tunneling" (see Fig. 2b). ATSSS-LL only needs to ensure multi-path ordering within a limit permitted by the transport layer [15] as it uses higher-layer transport (TCP or QUIC [16]) to support reliable delivery and congestion control. The key difference between ATSSS-HL and ATSSS-LL is that the latter is only responsible for multi-path packet ordering, as it offloads reliable delivery and congestion delivery to other transport layer protocols such as TCP or QUIC. In addition, both approaches need to include mechanisms to measure network characteristics such as the RTT in order to facilitate steering modes such as *smallest delay*. This makes ATSSS-LL easier to implement and independent of the transport protocols

[1] ATSSS is more general and can enable convergence between one 3GPP and one non-3GPP access network. We focus on 5G and WiFi access networks.

[2] ATSSS-HL and ATSSS-LL can be configured to work standalone or co-exist to complement each other [14].



(a) WiFi [27], [28] (b) LTE/5G [29] (c) CPU - Intel i7

Figure 4: WiFi/LTE/5G peak throughput over the past decade and the growth of single/multi-core CPU Geekbench6 scores.

used by an application. Other than being simpler, the primary advantage of ATSSS-LL over ATSSS-HL is that it is not tied to a specific transport protocol.

B. Related Work

Protocols for ATSSS: As of 3GPP Release 17 [13], [14], MPTCP [9] has been selected as the *de-facto* protocol for the ATSSS-HL approach. In addition to MPTCP, there have been other proposals that are under study [17]. For the QUIC [16] transport, MP-QUIC [18] is being considered in 3GPP Release 18 [19] while MP-DCCP [20] is an ATSSS proposal for UDP in general. In practice, these ATSSS-HL protocols are implemented using proxy software such as ShadowSocks [21] that run as a part of the UPF in the Core Network (see Fig. 2a). This applies to even commercial ATSSS-HL offerings such as by Tessares [7], [22], [23]. However, as we show in §III, the problem with such a software-based approach is that they incur a high latency overhead and they do not scale efficiently with regards to power and cost. While the 0-RTT-Convert protocol [23] reduces the initial setup latency, it does not reduce the per-packet processing latency of an ATSSS-HL software-based proxy. In contrast, JUNCTION uses commodity programmable switches to support ATSSS through a hardware-driven design that not only keeps the latency overhead low but also scales more efficiently. In order to do so, JUNCTION takes the ATSSS-LL approach and thus represents an unexplored[3] point in the ATSSS design space (see Fig. 3).

Cellular and WiFi: The idea of augmenting cellular networks with WiFi to offload traffic has been around since 3G [24] and has also been standardized in LTE [25]. In particular, Voice over WiFi or VoWiFi [26] has been widely adopted by operators to deliver voice/ IMS services over WiFi to their subscribers. These older approaches work mainly for voice and simply send traffic over WiFi (when available) without any sophisticated traffic steering. ATSSS [14] on the other hand, supports sophisticated steering modes that integrate WiFi and cellular access networks together to deliver services beyond just voice. Thus, ATSSS needs to handle orders of magnitude more traffic as compared to existing VoWiFi deployments.

III. MOTIVATION: LIMITATION OF SOFTWARE PROXIES

Over the last decade, the peak throughput supported by WiFi and LTE/5G networks has increased 6.6x and 20x, respectively (see Fig. 4). This has been accompanied by ∼41% increase in the density (per 100 people) of mobile devices [30]. As a result, given the same geographical region, the expected total traffic volume that needs to be handled by the core network infrastructure for converged networks has significantly increased as compared to 3G/LTE networks a decade ago [31]. In addition to the high traffic volume, ATSSS in today's 5G networks needs to keep its latency overhead low in order to support low-latency and high bandwidth applications such as online gaming and AR/VR. Given the aforementioned requirements, in this section, we evaluate how existing ATSSS solutions (§II-B) scale with increasing traffic volume and if they are able to keep the latency overhead low.

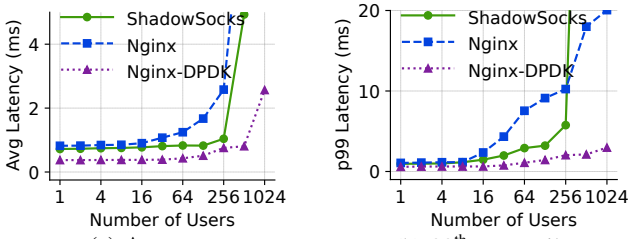
Setup: As outlined in §II-A, existing ATSSS deployments are ATSSS-HL only which are based on software-based MPTCP proxies. We compare three open-source production-grade, high-performance multi-threaded TCP proxy server implementations, namely the Rust-based ShadowSocks [21], Nginx [32], and Nginx with DPDK-based F-Stack [33]. We run the software proxies on a commodity server running Ubuntu 22.04 with the low-latency Linux kernel v5.15, utilizing a dedicated 16-core Intel Xeon Gold 6326 CPU set to performance mode. We use an NVIDIA CX-6 Dx 100G NIC [34] that is connected directly to the CPU via PCIe.

We evaluate the processing latency of each software proxy as the number of connections/user sessions increases. To do so, we use an iPerf server/client running on two additional servers to generate TCP traffic that passes through the proxy. We vary the load on the proxy by increasing the number of parallel iPerf connections with each connection representing a bandwidth-limited (100 Mbps) access network user. Using a fourth server, we measure (sample) the software proxy's processing latency by sending 100K ping-pong TCP packets through the proxy for each load configuration. We show the results in Fig. 5.

We can see that among the three proxies, Nginx-DPDK performs better, and yet at 1,024 users, it incurs an average and 99p latency overhead of ∼2.6 ms and ∼3 ms, respectively. To put this overhead in perspective, the target end-to-end latency as outlined by the ITU [29] is just 1 ms! The results from Fig. 5 also show that for a relaxed latency overhead SLO of 3 ms, Nginx-DPDK requires a 16-core CPU to support 1,024 users. Considering that a converged network core would require to support users on the order of hundreds of thousands, we would require 100's of CPUs to support ATSSS using a software-based proxy such as Nginx-DPDK (more in §VI-A).

While the above results may seem specific to our testbed setup and the specific software proxies being tested, the reason for poor scalability is more fundamental – over the past decade, the increase in network speeds (WiFi/LTE/5G) has far outpaced the increase in CPU performance (see Fig. 4). Over the past decade, the single-core and multi-core CPU performance has increased by only 2.3x and 3.8x, respectively. which is much lower than the increase in the peak throughput for WiFi or LTE/5G. As observed by Pan et al. [11], CPU-based packet processing has therefore become a bottleneck

[3]While the standards propose the ATSSS-LL approach, they do not specify any specific algorithm/design for ATSSS-LL.



(a) Average (b) 99th-percentile

Figure 5: Processing latency of ATSSS-HL proxies.

for per-flow throughput due to the stagnated growth of single-core performance. Our findings using the Nginx-DPDK high-performance TCP stack further corroborates this.

Key takeaway: Given the requirements of handling high traffic volume while keeping latency overhead low, the deployment of ATSSS-HL based on software proxies presents a significant challenge to the CapEx (cost) and OpEx (power) required for 5G-WiFi convergence.

Case for programmable switches: Given the fundamental limitation of using software-based proxies, we turn towards data-plane programmable switches as an alternative hardware platform to realize scalable and low-latency ATSSS. Today's programmable switches can support multi-Tbps line-rate packet processing e.g., 12.8 Tbps on the Intel Tofino2 [35]. Any data plane program that can be compiled and loaded on such switches can run at line rate with deterministic low latency (few µs) [36]. Furthermore, since the programmable switching hardware is optimized for packet processing, a network function running on a programmable switch scales more efficiently w.r.t. power and cost compared to its software implementation on general-purpose CPUs [37]. What this means is that **if** we can realize an ATSSS solution using a programmable switch, then by the natural property of the hardware, we can scale more efficiently to a higher traffic volume while keeping the latency overhead low. Therefore, the key question that we address in the rest of the paper is – *how can we realize ATSSS using a programmable switch?*

IV. Challenges

Realizing ATSSS on programmable switches is not straightforward because of their restricted programming model. Furthermore, given the feed-forward architecture of the packet processing pipeline, a memory location can be accessed at most once during the entire processing of a packet on the switch [38]. The ATSSS-HL approach requires supporting congestion control and reliability (§II-A) which involves complex state machines and memory access patterns that do not fit the programming model and architecture of programmable switches. As a result, in JUNCTION, we opt for the ATSSS-LL approach and realize ATSSS through tunneling at the network layer. However, even for ATSSS-LL, we have the following three key technical challenges:

C-1. Supporting a large number of ATSSS sessions: Since each user is expected to have multiple ATSSS sessions (§II-A), the ATSSS function running at the UPF requires significant

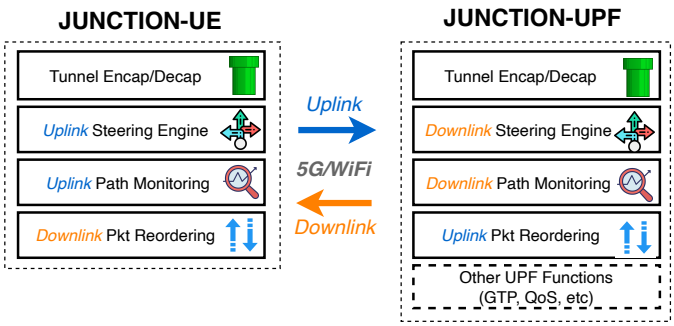


Figure 6: JUNCTION overview.

state. With the ATSSS-HL approach implemented as software-based proxies, this state requirement is manageable due to the abundant memory (100's of GB DRAM) available on a server. However, since programmable switches have limited memory (10's of MB SRAM), a naive ATSSS-LL implementation could significantly restrict the number of users that can be supported.

C-2. Additional state for distinct path metrics.: A user can have ATSSS sessions with different steering modes. Since each steering mode maintains distinct path metrics (e.g., RTT) to make steering decisions, additional state is required for the path metrics per-user per-steering mode.

C-3. Putting out-of-order packets back in order: In the uplink, as packets can arrive from either 5G and WiFi, they may arrive out-of-order which is detrimental to application performance [19]. Therefore, we need to reorder the received packets before forwarding them. Reordering packets on programmable switches is non-trivial as programmable switches do not naturally support selective buffering and sorting out-of-order packets in the data plane [39].

V. JUNCTION: Design and Implementation

A. Overview of JUNCTION

JUNCTION enables ATSSS between the user-plane function in the 5G core and user device through multi-path tunneling. The tunneling mechanism is similar to VPN implementations. There are three components in JUNCTION: (i) the JUNCTION-UE client that runs on the user device, (ii) the JUNCTION-UPF that is at the 5G core, and (iii) the JUNCTION agent. The JUNCTION-UE communicates with the JUNCTION agent in the 5G core to establish ATSSS sessions. Application traffic that is configured to use the ATSSS sessions is tunneled between JUNCTION-UE and JUNCTION-UPF while enforcing the configured steering mode. As the network condition varies, the steering mode steers the traffic across the two access networks.

The following is a brief summary of JUNCTION's components with Fig. 6 showing the sub-components of JUNCTION-UE and JUNCTION-UPF:

1) The JUNCTION-UE manages and establishes ATSSS sessions with the 5G core. The steering engine performs uplink traffic steering according to the ATSSS session configurations and reorders packets on the downlink. It

encapsulates the packets leaving the JUNCTION-UE with the JUNCTION header and decapsulates the packets in the reverse direction. It also performs continuous path measurements by generating probes and reporting link failures.

2) The JUNCTION-UPF includes general UPF processing (e.g., GTP, QoS, etc). In addition to performing encap/decap of JUNCTION headers, it reorders the out-of-order packets in the uplink direction and performs traffic steering in the downlink direction as per the mode configured in the ATSSS session. It also participates in the path measurements process by responding to RTT probes from the JUNCTION-UE, keeps track of the path metrics for traffic steering, and forwards a copy of the measurements to the JUNCTION agent.
3) The JUNCTION agent interfaces with the 5G core's control plane in managing the UPF functionalities on the JUNCTION-UPF. In addition to managing the ATSSS sessions, it monitors and makes necessary configurations based on the user path metrics.

Our work focuses on the design of the JUNCTION-UPF. As basic UPF functions have already been shown to be implementable on current programmable switches (e.g. P4UPF [40]), our work focuses on JUNCTION-specific designs and omit details of supporting other UPF-functionalities.

B. JUNCTION-UE

ATSSS being an emerging feature, current user devices do not have the required support for it (especially ATSSS-LL). We therefore provide JUNCTION-UE which is a userspace client application that functions akin to a VPN tunnel client and allows applications to interact with the JUNCTION-UPF. The JUNCTION-UE takes on the following responsibilities:

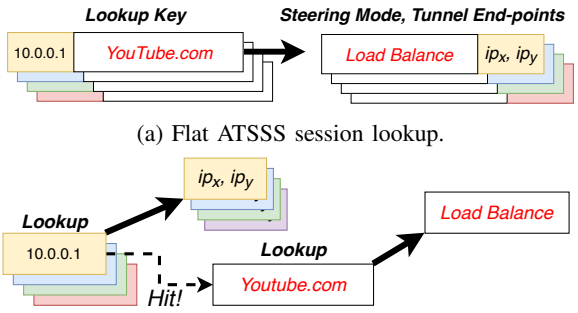
1) ATSSS session establishment and management: The JUNCTION-UE first communicates with the 5G core control plane to exchange connection parameters including the per-session traffic steering modes, and then configures tunnels for multiple ATSSS sessions required by a user device.

2) Uplink/downlink packet processing: JUNCTION-UE encaps the uplink traffic and decaps the downlink traffic with the JUNCTION header. It steers the uplink traffic as per the steering mode and also marks the uplink packets to aid the JUNCTION-UPF in uplink packet reordering (§V-D1). It implements a reordering buffer to restore packet order for downlink traffic.

3) Continuous path measurements and reporting: JUNCTION-UE actively probes the path states/metrics together with the JUNCTION-UPF (§V-E) and notifies the JUNCTION-UPF in case of link failures.

C. JUNCTION-UPF: *Downlink Packet Processing*

In the downlink, as packets are received by the JUNCTION-UPF, it first performs an ATSSS session lookup with the steering engine. If an ATSSS session is found, it makes further lookups and applies steering decisions followed by standard UPF functionalities (e.g., GTP and QoS). Otherwise, the packet proceeds directly to the UPF functionalities.



(a) Flat ATSSS session lookup.

(b) Hierarchical ATSSS session lookup.

Figure 7: Flat versus Hierarchical ATSSS session lookup

Recall that each user can have multiple ATSSS sessions, with each session dedicated to a specific application (e.g., YouTube) and traffic steering mode (§II-A). At the JUNCTION-UE, each ATSSS session is mapped to a distinct tunnel, and assigned a unique virtual tunnel IP address, serving as the ATSSS session ID.

1) Hierarchical ATSSS session lookup: An ATSSS session comprises a "key" and a "value". The key consists of the ATSSS session ID and the corresponding application, while the value contains the applied steering mode and the tunneling endpoint IP addresses for 5G and WiFi. The existing structure, as shown in Fig. 7a, results in $N \times M$ entries to store the session, and application part of the key, respectively, for N sessions and M applications, which is memory-inefficient, particularly for programmable switches (**C-1**).

Key insight: As operators often use a common steering mode for specific application traffic, such as *load-balance* for YouTube, we can summarize ATSSS sessions with similar traffic steering modes.

To achieve this, we implement a hierarchical lookup for the ATSSS session "key". First, we look up the ATSSS session ID. If a match is found, it returns the tunneling endpoint IP addresses for 5G and WiFi as the "value". Otherwise, the packet is not part of an ATSSS session, and we proceed with the regular UPF processing. Then, we retrieve the application to find the corresponding traffic mode to apply. This approach reduces the number of application entries to M (Fig. 7b).

Handling exception cases: To handle ATSSS sessions that cannot be summarized, we perform an additional lookup on the entire ATSSS session key, similar to that shown in Fig. 7a, after the initial lookup in Fig. 7b. For this supplementary lookup table, we provision only a small number of entries, for instance, 10% of the total number of users. This approach efficiently addresses exceptions while reducing overall memory utilization.

2) Unified traffic steering structure: To avoid redundancy and optimize memory usage (**C-2**), we propose a single unified data structure to represent per-user path metrics, accommodating multiple ATSSS sessions with different traffic steering modes dependent on distinct path metrics. With this approach, we eliminate the need to maintain separate path states for each individual ATSSS session of the same user.

By employing the unified structure, we can efficiently support the various required steering modes while streamlining the implementation surrounding this single representation.

In dissecting the requirements of the traffic steering modes (§II-A), we find that *active standby* and *load balance* necessitate binary information (0 or 1) to represent the availability of an access network. Conversely, *smallest delay* and *priority-based* require round-trip time (RTT) values, represented as non-zero integers (e.g., 10 ms), to determine the less-congested access network for traffic forwarding. As the RTT measurements presume the availability of the access network (non-zero RTT indicates availability), maintaining one set of path states for each user is sufficient, irrespective of the number of sessions or steering modes used.

To represent the unified structure, we use two arrays to maintain the path states: one for 5G and another for WiFi. Each user is assigned with a unique index, which is determined by the JUNCTION agent during the setup of the first ATSSS session. When accessing a user's path state, we retrieve the index during the ATSSS session lookup in §V-C1, along with the tunnel endpoint IPs for 5G and WiFi. Subsequently, we read the user's path state from the unified structure using the obtained index. This efficient approach allows us to access and manage the path states for each user effectively, regardless of the number of ATSSS sessions or the chosen steering modes.

3) Implementing different traffic steering modes: We discuss how the different traffic steering modes are realized in JUNCTION-UPF using the path states in §V-C2.

First, for *active standby*, a primary path and a backup path are defined. If the retrieved RTT value for the primary path is non-zero, it indicates that the access is active, and traffic can continue to be forwarded on it. Otherwise, traffic is directed to the backup path if available. In *smallest delay*, the path with the smallest, non-zero RTT is selected for traffic forwarding.

In *load balance*, packets are divided based on the rightmost digit of their sequence numbers. For instance, to achieve a 30:70 split ratio, we route packets with the rightmost digits 0 to 2 over the first path and those with digits 3 to 9 over the other path. Before forwarding, we verify that the next selected path has a non-zero RTT value. Finally, in *priority-based*, we designate a high-priority path. We compare the retrieved RTT value for the high-priority path with a pre-defined threshold. If it surpasses the threshold, the packet overflows to the low-priority path for forwarding.

Design limitation: For *load balance* and *priority-based* modes that split packets across two paths, sequencing becomes necessary for packet reordering at the JUNCTION-UE. Thus, a separate sequencing counter must be maintained for ATSSS sessions using these steering modes. This separate counter cannot be summarized as in §V-C1, making ATSSS sessions requiring *load balance* and *priority-based* to be treated as exception cases in the supplementary lookup table.

D. JUNCTION-UPF*: Uplink packet processing*

In the uplink, a packet undergoes general UPF processing, including GTP and QoS. Next, we check if the incoming packet is a JUNCTION packet and, if so, JUNCTION-UPF proceeds with packet decapsulation. Due to packets arriving from both 5G and WiFi, they may arrive out-of-order. If unaddressed, this issue could negatively impact upper-layer transport/applications (§VI-C). Therefore, JUNCTION-UPF is responsible for restoring the order of out-of-order packets.

1) Restoring packet order with FIFO queue pausing: We devise a packet reordering engine to restore packet order, leveraging pause-able FIFO queues on modern programmable switches to avoid the necessity of packet recirculation (**C-3**). Recent research [39] has demonstrated its viability, particularly when incoming packets exhibit predictable patterns in data center environments. Instead of reordering individual packets, we reorder a "stream" of packets based on the corresponding stream's relative order. This eradicates the need for maintaining sequence numbers as we only need to differentiate the two "streams", i.e., packets that are in the EX-PATH and CURRENT-PATH after a new steering decision is made.

For example, if packets are directed to the CURRENT-PATH and arrive ahead of those in the EX-PATH (resulting in an out-of-order scenario), we employ a paused FIFO queue to buffer the CURRENT-PATH packets until all EX-PATH packets have arrived. Subsequently, the paused FIFO queue releases the CURRENT-PATH packets. To distinguish between EX-PATH and CURRENT-PATH, the JUNCTION-UE marks the packets accordingly during traffic steering between paths (§V-B).

While the mechanism proposed in [39] proves effective in environments with controlled path latency, such as data centers, it faces challenges in 5G-WiFi scenarios due to the presence of time-varying wireless channels and a larger number of user sessions. To address these issues within the data plane's limited resources (up to 128 queues per egress port [41] on recent programmable switches), we propose two adaptations: *(i)* Admission control - Prior to initiating a reroute on the JUNCTION-UE, it will always verify the availability of a queue for reordering through the latest RTT probe. *(ii)* Reordering timeout - The JUNCTION-UE utilizes RTT measurements to determine the reordering timeout on the programmable switch, removing the need for JUNCTION-UPF to maintain such records as in [39]. Statistical multiplexing ensures that the required number of queues does not scale linearly with the number of active user sessions.

Design limitation: With the reordering capabilities on JUNCTION-UPF, out-of-order packets caused by user mobility using steering modes that utilize at most one path at a time, such as *active standby* and *smallest delay*, can be effectively handled. However, for other steering modes that utilize both paths concurrently, like *priority-based* and *load balance*, they should not be used at the JUNCTION-UE for applications sensitive to uplink out-of-order packets. Fortunately, in the uplink scenario, steering modes that use both paths are generally less necessary due to the uplink's lower bandwidth demands.

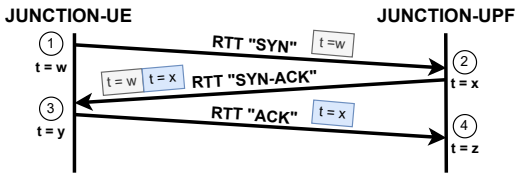


Figure 8: Round-trip time measurements between JUNCTION-UE and JUNCTION-UPF.

E. Continuous path measurements

To assess the access network's availability and measure round-trip times (RTT), we entrust the primary responsibility to the JUNCTION-UE, which initiates RTT measurements and path failure notifications (§V-B).

The RTT measurement flow is depicted in Fig. 8, resembling the TCP three-way handshake. The JUNCTION-UE calculates the RTT as $y - w$ at ③, while the JUNCTION-UPF computes the RTT as $z - x$ at ④. By default, RTT measurements are done periodically for every second. However, if the JUNCTION-UE is active, the probing rate is increased w.r.t. to the packet rate.

Fast in-network updates: To swiftly adapt to changes in the access network conditions, we update the computed RTT directly to the unified traffic steering structure (§V-C2) entirely in the data plane once the RTT "ACK" is received ④ from either 5G or WiFi. This enables subsequent packets to be steered based on the latest network information. Concurrently, the RTT "ACK" is forwarded to the JUNCTION agent. We follow the same process for path failure notifications received from the JUNCTION-UE. This approach facilitates real-time adjustments and ensures the network operates efficiently in response to varying conditions.

Preventing path state "bouncing": Since the measurements/notifications are directly updated in the JUNCTION-UPF data plane, situations may arise where the measured RTT of the links experiences "flip-flopping," leading to frequent path switching. To address this issue, the JUNCTION agent continuously monitors the received measurement/notification packets. If such occurrences are detected, the control plane takes action by suspending direct updates in the data plane for that specific user until network conditions stabilize. This measure helps to mitigate the adverse effects of unstable RTT measurements and ensures a more reliable path selection.

F. JUNCTION *agent*

The JUNCTION agent serves as the control plane for JUNCTION-UPF and fulfills three essential roles. Firstly, it acts as a PFCP [42] agent, facilitating the conversion of 5G-specific messages to manage UPF functionalities through P4Runtime, interacting with the 5G core. Since the current 5G cores lack support for ATSSS, the JUNCTION agent handles ATSSS user session management outside of the 5G core. Consequently, it plays a crucial role in ATSSS session establishment and management, as well as ATSSS session summarization for hierarchical lookups (§V-C1). Lastly, the JUNCTION agent continuously monitors time-series changes in user path statuses through the forwarded RTT "ACK"s. If frequent path switching due to path instability occurs, the JUNCTION agent intervenes by suspending direct updates in the data plane until the network conditions stabilize (§V-E).

Table I: JUNCTION-UPF's hardware resource consumption on the Intel Tofino2. The current configuration supports 256K users with 2 ATSSS sessions each. We build on top of P4UPF [40], [44].

Resource	P4UPF [40], [44]	JUNCTION-UPF
SRAM	38.4%	52.8%
TCAM	10.4%	10.4%
Stateful ALUs	21.3%	26.3%
Hash Bits	11.5%	15.6%
Hash Dist Unit	15.0%	15.0%
VLIW Ins.	11.7%	12.2%

G. JUNCTION*: Implementation*

JUNCTION's implementation is publicly available at [12].

JUNCTION tunnel: JUNCTION tunnels packets between JUNCTION-UE and JUNCTION-UPF over UDP with the JUNCTION header, resulting in a total overhead of 32 bytes. The JUNCTION header is an 8-byte header consisting 8-bit flag to indicate the packet type, 8-bit protocol field, 16-bit length, and 32-bit sequence number.

JUNCTION-UE: We implement a multi-threaded JUNCTION-UE in ∼800 lines of C++. We use the Linux TUN/TAP [43] to create the tunnels for the individual sessions. The JUNCTION-UE manages the tunnels and injects the tunnel routes into the routing table.

JUNCTION-UPF: We build on top of P4UPF [40], [44] and implement the JUNCTION-UPF in ∼1300 lines of P4 [45] code. We use the Intel P4 studio v9.11.1 to compile and load JUNCTION-UPF into the Intel Tofino2 switch data plane.

In the downlink, we use match-action tables to implement the hierarchical lookup table (§V-C1) and two 8-bit register arrays to realize the unified traffic steering structure (§V-C2). We update the computed path RTTs directly to the register arrays (§V-E).

In the uplink, we leverage the queue pause/resume feature on the Intel Tofino2 to manage out-of-order packets effectively. We reserve $N - 1$ queues out of the N available queues from an upstream port (e.g., 127 out of 128 queues for a 400 Gbps port). To restore packet order, an available queue is assigned to a user, and we use two-way associative hash tables, realized through two register arrays, to track the available queues. When a queue is flushed, we generate a packet to update this structure, freeing the queue for subsequent users.

To address cases where the last EX-PATH packet (that resumes the paused queue) is lost, we implement a timer using a "timer" packet that is continuously recirculated once the first out-of-order packet arrives. Upon queue resumption or expiry, the "timer" packet is discarded.

Table I presents a comparison of hardware resource utilization between the JUNCTION-UPF and the P4UPF [40] implementation on the Intel Tofino2 [35]. This configuration supports 512K GTP sessions, which translates to 256K users

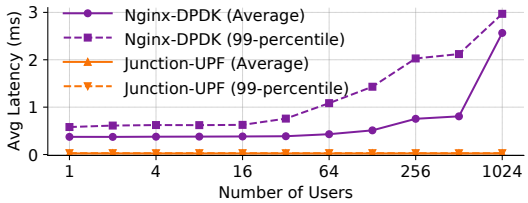


Figure 9: Processing latency of Nginx-DPDK and JUNCTION.

Table II: JUNCTION's power and cost analysis against its CPU counterpart.

per user throughput	# users per TF2	# servers required	cost ratio	power ratio
100 Mbps	128,000	125	48.6	29.5
200 Mbps	64,000	226	229	84.7

connected over 5G and WiFi concurrently. For the 256K users, each can have up to 2 ATSSS sessions. 16K entries are dedicated to the supplementary part of the hierarchical lookup table (§V-C1). From Table I, there remain ample hardware resources remaining to increase the support for the number of users. We leave further optimizations as future work.

JUNCTION agent: At present, we have not extended the existing PFCP agent [46] that accompanies P4UPF [44] due to the lack of ATSSS support on the 5G core [47] we are utilizing. As a result, we have implemented a separate Python-based control plane application to manage the JUNCTION-specific components in the JUNCTION-UPF. Additionally, this control plane application communicates separately with the JUNCTION-UE, efficiently handling ATSSS session management and other JUNCTION functionalities.

VI. EVALUATION

A. Scalability

To initiate our performance evaluation, we compare the latency performance of JUNCTION with Nginx-DPDK in Fig. 9, using the same methodology as discussed in §III. The results clearly indicate that JUNCTION-UPF consistently maintains deterministic latencies at 28μs, even as the number of users progressively increases from 1024 to 128K. This demonstrates JUNCTION-UPF's capability to efficiently handle additional users while maintaining low packet forwarding latencies. The outcomes underscore the scalability and reliability of JUNCTION-UPF, making it suitable for accommodating a higher number of users with consistently low latencies.

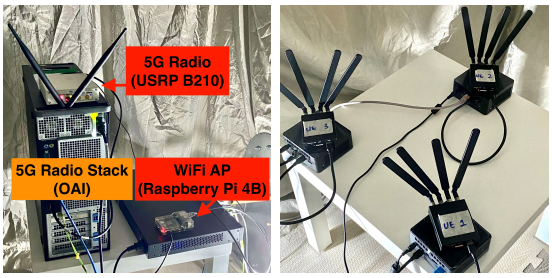
Then, we conduct a power-cost analysis in Table II based on publicly available information regarding the cost and power of the Intel Tofino2 [48], [49] utilized in JUNCTION and its CPU counterpart, the Intel Xeon Gold 6326 [50].

With a capacity of 12.8 Tbps, the Tofino2 can handle up to 128K users at 100 Mbps, ensuring a 3 ms 99th-percentile processing latency guarantee. In contrast, the equivalent number of servers required to achieve the same performance would be 125. This leads to an exorbitant cost of over 48.6× the hardware acquisition cost for the Tofino2 switch alone, not to mention the astounding 29.5× increase in power consumption. Furthermore, if the per-user throughput is relaxed to 200 Mbps, the cost and power would skyrocket even further, reaching an astonishing 229× the cost of a Tofino2 switch and an 84.7× higher electricity bill.

The power-cost analysis shows the tremendous advantages of JUNCTION's highly efficient design and the substantial cost and power savings it offers in comparison to using traditional CPU-based servers to accommodate the same number of users.

Table III: Testbed hardware and software configuration.

5G Basestation	Intel Xeon W-2155; 64GB RAM; USRP B210 [51]; 20MHz @ 3.5GHz (n78)
WiFi Access Point	Raspberry Pi 4B; WiFi5 20MHz @ 5GHz
UEs (3x)	Intel NUC with Quectel RM500Q-GL 5G Module [52] and Intel AX211 [53] WiFi Modem
Servers (2x)	Intel Xeon Gold 6326;256GB RAM
Ethernet NICs (2x)	NVIDIA ConnectX-6 DX 100GbE NIC [34]
Ethernet Switch	Intel Tofino2 [35] switch
OS	Ubuntu Server 22.04; Linux Kernel v5.15
5G Radio Stack	OpenAirInterface5G [54] (tag 2023.w12) [55]
5G Core	OpenAirInterface 5G Standalone Core (v1.5) [47]



(a) 5G gNB and WiFi AP. (b) Multi-homed UEs.

Figure 10: Our multi-access testbed in an EMI/RFI-shielded tent [56] with three 5G-WiFi multi-homed commercial UEs. The UEs are 1 meter away from the 5G gNB and WiFi AP.

B. Steering Engine Validation

Next, we show JUNCTION not only delivers significant performance gains in §VI-A it does so without compromising on the ATSSS steering modes' functionalities.

We evaluate JUNCTION on our 5G-WiFi testbed (see Table III) placed within an EMI/RFI-shielded Faraday tent [56] as depicted in Fig. 10 and validate the proposed unified structure discussed in §V-C2 on the realization of different traffic steering modes.

Fast handovers: First, we demonstrate that JUNCTION emulates the behavior of its ATSSS-HL counterpart, MPTCP, in quickly handing over data traffic between 5G and WiFi networks using the *active-standby* traffic steering modes when the JUNCTION-UE loses access to the WiFi connection at $t = 20$. To simulate real-time high-definition video streaming scenarios, we employ iPerf3 to generate constant bitrate traffic at 25 Mbps. The recorded throughput of the application is plotted in Fig. 11. The results show that JUNCTION exhibits similar responsiveness to changes in link characteristics as MPTCP, effectively adapting to network conditions while minimizing disruptions to upper-layer applications.

Link aggregation: We showcase JUNCTION's capability to effectively leverage both 5G and WiFi networks simultane-

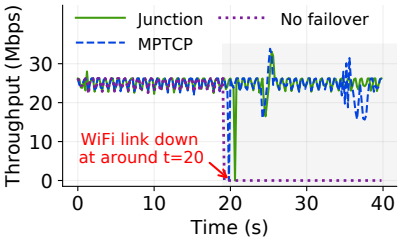


Figure 11: Seamless handover through *active standby* steering mode.

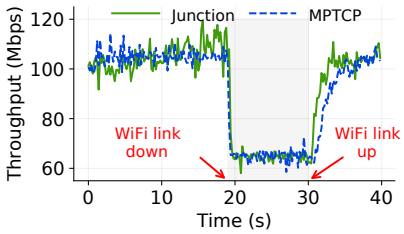


Figure 12: Link aggregation through *load balance* steering mode.

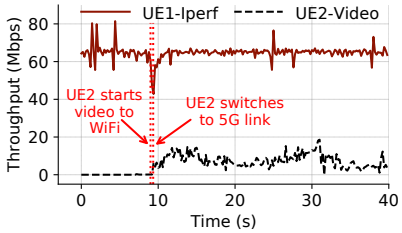


Figure 13: Low-latency path selection through *smallest delay* steering mode.

Table IV: The number of TCP retransmissions w/ and w/o the uplink packet reordering engine (§V-D1) at JUNCTION-UPF.

	average	**99th-%ile**
w/o Packet Reordering Engine	186.8	219.0
w/ Packet Reordering Engine	1.2	3.0

ously, resulting in higher overall bandwidth through *load balance*. In our testbed, both the 5G and WiFi networks can be saturated at approximately 60 Mbps in the downlink, respectively. To simulate bandwidth-intensive workloads, such as large-file downloads, we utilize iPerf3 to generate traffic.

As shown in Fig. 12, the reported aggregated throughput for JUNCTION closely approaches the sum of the individual throughputs of 5G and WiFi networks when both are available. This indicates that JUNCTION efficiently combines the bandwidth of both accesses, resulting in higher overall throughput, comparable to that of MPTCP.

Low-latency path selection: We illustrate JUNCTION's capability in reacting to the underlying path conditions under competition. In Fig. 13, UE1 runs an iPerf3 session over WiFi with its steering mode set as *active standby*.w Then, we introduce UE2 at $t = 9$ to stream a high-bitrate *live* video at 15 Mbps using WiFi – competing with UE1. UE2 uses the *smallest delay* steering mode.

Given UE1's background traffic, the path latency of WiFi is measured to be higher than that of the 5G path. Thus, JUNCTION-UPF's steering engine moves UE2 over the 5G path due to the lower path latency. Once UE2 leaves the WiFi network, UE1's throughput immediately recovers too. Here, we did not evaluate MPTCP on UE2 given the lack of support for *smallest delay* scheduler in the upstream kernel's MPTCP implementation [57]. As *priority-based* shares similar identical results with *smallest-delay*, we omit it for brevity.

C. Effectiveness of the Uplink Packet Reordering Engine

We evaluate the effectiveness of JUNCTION's uplink packet reordering mechanism (§V-D1). We emulate a scenario where a user moves across 5G and WiFi with different path latencies using our 5G-WiFi testbed. In our testbed, the end-to-end latency varies between 2-5ms and 7-15ms for the 5G and WiFi accesses, respectively. Uplink traffic is generated by uploading a 7 MB short video. We trigger the JUNCTION-UE to switch between the links for every 2000 packets. The experiment is repeated 100 times.

Table V: Increase in supported users using the proposed techniques – hierarchical table lookups in §V-C1 ("HTL") and unified traffic steering structure in §V-C2 ("UTSS").

ATSSS sessions/user	Naive	HTL only	UTSS only	JUNCTION
1	3,022,848	3,702,784	3,022,848	3,702,784
2	1,511,424	1,851,392	1,705,984	3,702,784
4	755,712	925,696	913,408	1,175,552
8	377,856	462,848	473,088	616,448

We measure TCP retransmissions because when reordering is severe enough, the out-of-order packets are considered lost. Thus, TCP reduces its sending rate and impacts application performance. Our evaluations (see Table IV) indicate that TCP retransmissions are reduced by 155.7×, on average, with the uplink packet reordering engine in JUNCTION-UPF.

D. JUNCTION *Micro-benchmarks*

We quantify how the hierarchical lookup table (§V-C1) and unified steering structure (§V-C2) impact the total number of users that can be supported. To achieve this, we use a simplified data plane program and try to maximize the number of users that that can be supported while saturating the available memory. From Table V, we observe that JUNCTION can accommodate 22% more users than the naive approach when there is one ATSSS session per user. The scalability of the design becomes more apparent when the number of ATSSS sessions per user increases to eight, with JUNCTION supporting up to 1.63× more users than the naive design. This showcases the efficiency and scalability of our approach, allowing for a significant increase in the number of supported users while optimizing memory utilization.

VII. CONCLUDING REMARKS

We present JUNCTION, an end-to-end system that enables the deployment of ATSSS at scale with programmable switches. We show that JUNCTION-UPF is highly scalable and at a given traffic volume, it can provide up to 225× lower CapEx and 84.7× less OpEx for operators when compared to software-based proxies. While JUNCTION-UPF's design is efficient, it does pose certain limitations regarding the usage of certain traffic steering modes, e.g., limited support for uplink reordering. However, considering the scale that JUNCTION enables for 5G-WiFi convergence, it is a worthwhile trade-off. We believe JUNCTION can be key to enabling 5G-WiFi convergence at scale.

REFERENCES

[1] Ericsson, "Ericsson Mobility Report June 2023," Telefonaktiebolaget LM Ericsson, Tech. Rep., June 2023.

[2] WBA and NGMN Alliance, "RAN Convergence Paper," 2019, https://shorturl.at/FGKQ5.

[3] Cisco, "Cisco Annual Internet Report (2018–2023)," Cisco Systems Inc., Tech. Rep., 9 March 2020.

[4] 3GPP, "TS 23.501 v16.17.0: System architecture for the 5G System (5GS)," 2023.

[5] C. Deng, X. Fang, X. Han, X. Wang, L. Yan, R. He, Y. Long, and Y. Guo, "IEEE 802.11 be Wi-Fi 7: New challenges and opportunities," *IEEE Comm. Surveys & Tutorials*, 2020.

[6] "Tessares 5G Hybrid Access achieves cost-efficient fiber-like speed on Proximus' live network," https://shorturl.at/gwL06 [Accessed: Mar. 2023].

[7] Tessares, "5G ATSSS," https://shorturl.at/cqsA0 [Accessed: March 2023].

[8] Deutsche Telekom AG, "Deutsche Telekom demonstrates Multipath for Fixed Mobile Convergence on Campus," https://shorturl.at/lzFW2 [Accessed: July 2023], May 2021.

[9] A. Ford *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020.

[10] L. Roberts, "Beyond moore's law: Internet growth trends," *Computer*, 2000.

[11] T. Pan *et al.*, "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches," in *Proceedings of ACM SIGCOMM*, 2021.

[12] "JUNCTION GitHub repository," The link is hidden for double-blind review, 2023.

[13] 3GPP, "TS 23.501 v17.9.0: System architecture for the 5G System (5GS)," 2023.

[14] 3GPP, "TS 24.193 v17.3.0: 5G System;Access Traffic Steering, Switching and Splitting (ATSSS); Stage 3," 2022.

[15] E. Blanton *et al.*, "A conservative loss recovery algorithm based on selective acknowledgment (SACK) for TCP," *RFC 6675*, 2012.

[16] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021.

[17] 3GPP, "TR 23.700-93 v17.0.0: Study on access traffic steering, switch and splitting support in the 5G System (5GS) architecture; Phase 2," 2021.

[18] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *ACM CoNEXT*, 2017.

[19] A. K. Salkintzis, M. Kühlewind, S. Rommer, and R. Liebhart, "Multipath quic for access traffic steering switching and splitting in 5g advanced," *IEEE Communications Standards Magazine*, 2023.

[20] M. Amend *et al.*, "A multipath framework for udp traffic over heterogeneous access networks," 2019.

[21] shadowsocks.org, "shadowsocks-rust," GitHub, https://github.com/shadowsocks/shadowsocks-rust, [Accessed: Jan 2023], commit: e5f0ab9.

[22] CommsUpdate, "KT, Tessares claim first '5G low latency multi-radio access technology' test on live network," https://shorturl.at/buQWY [Accessed: July 2023], 2019.

[23] M. Baerts *et al.*, "Leveraging the 0-rtt convert protocol to improve wi-fi/cellular convergence," in *Proceedings of the Applied Networking Research Workshop*, 2021.

[24] A. Balasubramanian *et al.*, "Augmenting mobile 3g using wifi," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010.

[25] D. Laselva *et al.*, "3gpp lte-wlan aggregation technologies: Functionalities and performance comparison," *IEEE Communications Magazine*, 2018.

[26] 3GPP, "TS 23.234 v13.1.0: 3GPP system to Wireless Local Area Network (WLAN) interworking; System description," 2017.

[27] Aruba Networks, "White paper - 802.11ac in-depth," https://shorturl.at/hnDIJ [Accessed: Mar. 2023], 2014.

[28] Intel, "Next generation wifi: Wifi 7 and beyond," https://shorturl.at/BLNQZ [Accessed: Mar. 2023], 2020.

[29] ITU, "Itu-r faq on international telecommunications (imt)," https://www.itu.int/en/ITU-R/Documents/ITU-R-FAQ-IMT.pdf [Accessed: Mar. 2023].

[30] T. W. Bank, "Mobile cellular subscriptions (per 100 people)," https://data.worldbank.org/indicator/IT.CEL.SETS.P2, [Accessed: July 2023].

[31] D. Kumar *et al.*, "Scaling telecom core network functions in public cloud infrastructure," in *IEEE CloudCom*. IEEE, 2020.

[32] F5, Inc, "Nginx," https://www.nginx.com/ [Accessed: July 2023], 2023.

[33] Tencent Cloud, "F-stack," https://github.com/F-Stack/f-stack [tag: v1.22], 2022.

[34] NVIDIA, "NVIDIA Connect X-6," https://www.nvidia.com/en-sg/networking/ethernet/connectx-6 [Accessed: March 2023].

[35] "Intel Tofino 2," https://shorturl.at/IUYZ5 [Accessed: July 2023].

[36] X. Jin *et al.*, "Netcache: Balancing key-value stores with fast in-network caching," in *SOSP*, 2017.

[37] R. Miao *et al.*, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of ACM SIGCOMM*, 2017.

[38] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *SIGCOMM CCR*, 2013.

[39] C. Song *et al.*, "Network Load Balancing with In-network Reordering Support for RDMA," in *Proceedings of ACM SIGCOMM*, 2023.

[40] R. MacDavid *et al.*, "A P4-Based 5G User Plane Function," in *Proceedings of the Symposium on SDN Research*, 2021.

[41] J. Lee, "Advanced Congestion & Flow Control with Programmable Switches," 2020, https://shorturl.at/cFM16.

[42] 3GPP, "TS 29.244 v17.9.0: Interface between the Control Plane and the User Plane nodes," 2023.

[43] M. Krasnyansky, "Universal tun/tap device driver," https://docs.kernel.org/networking/tuntap.html.

[44] Open Networking Foundation, "Fabric-TNA," https://github.com/stratum/fabric-tna.git [Commit: fd3c3f0].

[45] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, 2014.

[46] Open Networking Foundation, "Upf," https://github.com/omec-project/upf [commit: 35f11a7].

[47] EURECOM, "Openair-cn-5g: An implementation of the 5g core network by the openairinterface community." https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed [tag: v1.5.0], 2023.

[48] Intel Corporation, "Intel® tofino 2 12.8 tbps, 20 stage, 4 pipelines," 2018, https://shorturl.at/gquy5 [Accessed: March 2023].

[49] "Intel Tofino2 – A 12.9 Tbps P4-Programmable Ethernet Switch," https://shorturl.at/prvCW [Accessed: Mar. 2023], 2020.

[50] Intel Corporation, "Intel® xeon® gold 6326 processor," 2021, https://shorturl.at/sDH39 [Accessed: July 2023].

[51] E. Research, "Usrp b210 usb software defined radio (sdr)," https://www.ettus.com/all-products/ub210-kit/ [Accessed: June 2023].

[52] Quectel, "5g rm50xq series," https://www.quectel.com/product/5g-rm50xq-series [Accessed: June 2023].

[53] Intel, "Intel® Wi-Fi 6E AX211," https://shorturl.at/hjsX8 [Accessed: June 2023], 2021.

[54] F. Kaltenberger *et al.*, "Openairinterface: Democratizing innovation in the 5g era," *Computer Networks*, 2020.

[55] EURECOM, "Openairinterface," https://gitlab.eurecom.fr/oai/openairinterface5g [tag: 2023.w12], 2023.

[56] Holland Shielding Systems BV, "EMI/RFI-shielded Faraday tent," https://shorturl.at/zHSUY [Accessed: July 2023].

[57] "Linux mptcp upstream project," https://github.com/multipath-tcp/mptcp_net-next/wiki/#changelog [Accessed: July 2023].