FBDT: Sum-Throughput Achieving Transport Layer Solution for Multi-RAT Networks

Suresh Srinivasan, Sam Shippey, Ehsan Aryafar, and Jacob Chakareski

***Abstract*—Emerging mobile applications give rise to new bandwidth-hungry and latency-sensitive traffic classes that challenge existing wireless systems. Addressing them requires innovative approaches such as simultaneous data transmission across multiple Radio Access Technologies (RATs), e.g., WiFi and WiGig. However, existing transport layer multi-RAT traffic aggregation schemes, e.g., multi-path TCP, suffer from Head-of-Line (HoL) blocking and sub-optimal traffic splitting across the RATs that severely penalize their performance. In this paper, we investigate the design of FBDT, a novel multi-path transport layer solution that for the first time can achieve the sum of the throughput rates across the individual RATs network paths, despite their channel conditions' dynamics. We have implemented FBDT in the Linux kernel and show substantial improvement in throughput relative to state-of-the-art schemes, e.g, 2.5x gain in a dual-RAT scenario (WiFi and WiGig) when the client is mobile. Second, we extend FBDT to more than two radios and demonstrate that its throughput performance scales linearly with the number of RATs, in contrast to multi-path TCP, whose performance degrades with an increase in the number of RATs. We evaluate the performance of FBDT on different traffic classes and demonstrate: (i) 2-3 times shorter file download times, (ii) up to 10 times shorter streaming times and 10 dB higher video quality for progressive download video applications, and (iii) up to 9 dB higher viewport quality for interactive mobile VR applications, when our viewport quality maximization framework is employed along with FBDT.**

***Index Terms*—Millimeter-wave, multiple radio access technology (Multi-RAT), single path transport control protocol (SPTCP), multipath transport control protocol (MPTCP), head of the line (HoL), sliding window, re-transmission, transmission window, ACK(s), schedulers, WiFi, WiGig, throughput, line of sight (LoS), non line of sight (nLoS), blockage, mobility, forward and backward data transmission (FBDT), quality of user experience (QoE), mobile traffic class, interactive, background, streaming, conversational, progressive downloads, web-page, file downloads, mobile augmented and virtual reality (AR/VR), 360-degree video streaming.**

Received 8 July 2024; revised 16 April 2025; accepted 2 May 2025. Date of publication 15 May 2025; date of current version 3 September 2025. This work was supported in part by the NSF under Grant CNS-1910517, Grant CNS-1942305, Grant CCF-2031881, Grant ECCS-2032387, Grant CNS-2040088, Grant CNS-2032033, Grant CNS-2106150, and Grant CNS-2346528, in part by the NIH under Grant R01EY030470, and in part by the Panasonic Chair of Sustainability at NJIT. Recommended for acceptance by C. Assi. *(Corresponding author: Ehsan Aryafar.)*

Suresh Srinivasan, Sam Shippey, and Ehsan Aryafar are with the Computer Science Department, Portland State University, Portland, OR 97201 USA (e-mail: earyafar@pdx.edu).

Jacob Chakareski is with the College of Computing, New Jersey Institute of Technology, Newark, NJ 07102 USA.

Digital Object Identifier 10.1109/TMC.2025.3569453

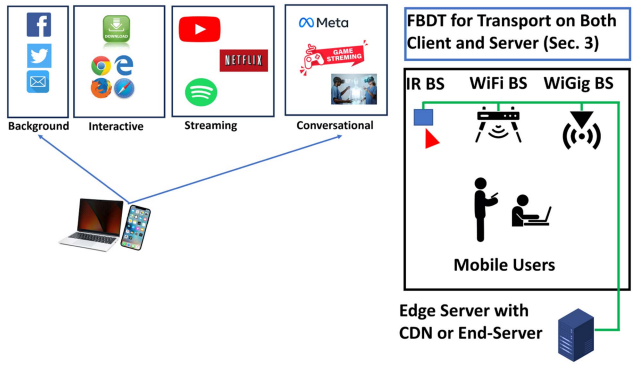

Fig. 1. FBDT - Low Latency and High throughput. Left: Client with Interactive, Background and Conversational class of application Right: An edge server is connected to different Base Stations (BSs) including WiFi, WiGig, and Infrared (IR). IR is used for client localization and orientation mapping. FBDT is implemented at the transport layer on both the client and server.

I. Introduction

THE evolution of social media, video platforms, cloud-based communication, and advanced AR/VR interactive applications has led to a diverse range of mobile devices such as smartphones, VR/AR headsets, and laptops. These devices cater to multifaceted applications that demand varying bandwidth and data rates. Different wireless networks like Wigig, WiFi, 5G, 6G, LTE, each serving distinct purposes – ranging from high bandwidth with limited range to long-range communication with lower bandwidth —- highlight the need for multiple wireless networks to meet specific application Quality of Service (QoS) requirements. Utilizing a single network for these multifunctional devices falls short in delivering the necessary QoS parameters like low/high latency and high data rates. In addressing this challenge, it becomes crucial to categorize these multifunctional applications to ensure the fulfillment of desired QoS. They can be effectively classified into four primary categories: Conversational, Streaming, Interactive, and Background applications, as depicted in Fig. 1. Such a categorization serves as a fundamental step towards ensuring the tailored provisioning of network resources to meet the diverse QoS demands of these multifaceted applications. Conversational and Streaming traffic applications are real-time and highly time-sensitive. Conversational traffic, utilized by applications like the metaverse, AR/VR communication, video conferencing, and gaming, is the most sensitive to delays. Streaming traffic, which includes platforms like YouTube, Netflix, and ESPN, is also real-time but less

sensitive to delays, often involving one-way transport intended for human audiences.

On the other hand, Interactive and Background traffic applications have looser delay requirements and lower bandwidth needs. Interactive traffic is generated by applications like interactive email, file download and web browsing, whereas Background traffic is associated with social media apps and simple mail transfer protocol (SMTP). These distinctions in traffic classes are essential for optimizing network performance and ensuring seamless user experiences in various mobile applications.

The rising need for cloud storage applications, including services for photos, backups, videos, and more, has led to the emergence of a Background class of applications. These applications involve data transfers typically ranging from 10 to 100 gigabytes [1]. This increasing demand highlights the significance of accommodating large-scale data transfers over wireless network.

An exciting and highly anticipated addition to the conversational class of traffic is the metaverse, which is broadly described as a network of 3D virtual worlds facilitated by virtual reality (VR) and augmented reality (AR) headsets. The metaverse places a strong emphasis on social media interaction [2] and holds the promise of diverse future applications [3]. The metaverse and VR/AR applications in general have been recently identified by 3GPP as key emerging application settings for 5G+/6G technologies [4]. Mobile 360° video VR streaming, i.e., observing high-quality 360° video streams on unthetered mobile VR headsets, is anticipated to be one of the key enabling technologies of the Metaverse, and has attracted considerable attention recently [5].

While mobile cloud storage applications are relatively less sensitive to delays, they significantly consume wireless network bandwidth, which can impact the performance of other mobile applications. In contrast, studies indicate that streaming and conversational class applications have demanding requirements, especially in VR/AR contexts. According to several recent studies [6], [7], the required wireless data rates are about 30 Mbps for a 2 K H.264-encoded stream, 800 Mbps for an 8 K H.266 encoded stream, and up to a few Gbps with higher resolution or frame rate. The uplink rates are insignificant, i.e., well below 2 Mbps, as only headset orientation and some other client-generated control need to be transmitted. Existing radio access technologies (RATs) cannot consistently provide high downlink data rates. For example, LTE speeds are typically 10 Mbps [8], which is far lower than the required data rates. Recent sub-6 GHz (11 ac/ax) and 60 GHz (11 ad/ay) WiFi can provide anywhere from 100 Mbps to a few Gbps but sub-6 GHz WiFi can suffer from interference and bad channel conditions (e.g., low rank MIMO) whereas 60 GHz communication is highly susceptible to client mobility and blockages. Each of these conditions can drastically reduce the RAT throughput. Susceptibility to blockages and mobility is also prevalent in existing mmWave 5G solutions, which creates large fluctuations in throughput. Moreover, the high variability in wireless network bandwidth can cause unexpected re-buffering, which drastically reduces the client Quality of Experience (QoE) [9]. The reduced QoE due to either low quality video frames or re-buffering can even lead to client disorientation or nausea [10].

Simultaneous traffic aggregation across multiple RATs such as sub-6 GHz WiFi, mmWave WiFi (WiGig), LTE, and/or 5G can be a key solution to address the aforementioned challenges by boosting the wireless capacity and providing a high minimum data rate across all channel conditions and in presence of client/network dynamics. However, as we will show later in this paper, existing transport layer multi-RAT traffic aggregation schemes can suffer from Head-of-Line (HoL) blocking and sub-optimal traffic splitting across RATs, particularly in presence of system dynamics such as when the channel condition frequently changes from Line-of-Sight (LoS) to non-Line-of-Sight (nLoS) and vice versa. As a result, in many practical situations, the overall throughput of a Multi-Path TCP (MPTCP) protocol can even be less than the throughput that would be achieved by using only a single RAT at all times. This paper makes two key contributions to address the above challenges. We introduce FBDT, a Froward and Backward Data Transmission protocol at the transport layer to effectively eliminate HoL and sub-optimal traffic splitting problems in all channel conditions. Specifically, our contributions can be summarized as follows:

- *FBDT Design:* We propose a new transport layer multi-RAT traffic aggregation scheme that achieves *summation* of individual RAT data rates in all channel conditions. FBDT eliminates HoL blocking and sub-optimal traffic splitting problems by removing retransmission schemes employed by MPTCP and relying on individual Single-Path TCP (SPTCP) mechanisms for reliable delivery. Further, for a given queue of packets to be transmitted, FBDT optimally places the pointers to fetch traffic for each RAT by taking into account the historical reliability of each RAT. Finally, FBDT serves packets from both forward and backward directions in the queue so that each RAT can independently transmit packets without being blocked by the status of other RATs.
- *Open-Source Implementation:* We have extended the implementation of FBDT to multi-RAT communication as a daemon in the user space in Linux on both the client and server sides. This allowed us to deploy the implementation of our protocols and algorithms for multi-RAT communication in Commercial-off-the-shelf (COTS) hardware (routers, laptops) and evaluate its performance in real-world settings. We have open-sourced our software and publicly released it on GitHub [11] so that other researchers in the community can benefit from our work and expand on it.
- *Evaluation:* We have conducted extensive experiments to evaluate the performance of FBDT and the induced Quality of Experience (QoE) for the receiving client. We show that FBDT can achieve an aggregate throughput that equals the sum of the data rates of the individual RAT network paths, for all channel conditions, i.e., when the client is in a stationary LoS, nLoS, or mobile state. We also show that compared to a state-of-the-art MPTCP scheduler, FBDT increases the aggregate throughput by a factor of 2.5x in a dual radio setup with one WiFi and one WiGig (802.11 ad) RAT. This throughput gap increases when using a higher number of RATs. Moreover, compared to the MPTCP scheduler, FBDT provides a throughput gain of up to 30%

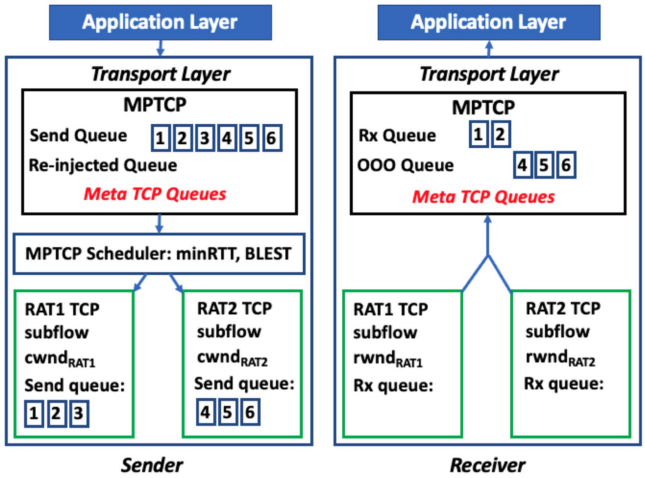

Fig. 2. MPTCP architecture. Packet 3 from RAT1 is lost. Packets 4, 5, and 6 are received at the receiver but moved to the out-of-order buffer. The loss stops subsequent packet deliveries to the Meta Rx queue and the application.

for the Interactive and Background classes of traffic, and up to 18X gain in the case of Streaming traffic. Finally, for the Conversional traffic class, we show that FBDT with FIX - an algorithm which allocates the same transmission rate to each tile in a 360° video segment - provides an average of 1 dB PSNR video gain as compared to MPTCP with FIX. Moreover, we show that by applying an optimal ordering and rate allocation to the 360° video tiles and transmitting them over FBDT can give PSNR gains of 13 dB under mobility conditions [12].

The rest of the paper is organized as follows. First, we discuss background and motivation information for our work in Sections II and III. Section IV describes the design of FBDT. We discuss the details of our hardware implementation in Section V and present the results of our comprehensive evaluations in Section VI. Finally, we discuss related work in Section VII and conclude the paper in Section VIII.

II. Background

MPTCP Architecture: MPTCP [13], [14], [15] increases the performance for the application by pooling resources from multiple RATs. Fig. 2 depicts the main components of MPTCP. At the sender, a single TCP socket is exposed to the application and outgoing application segments are copied to a send queue at the meta-level. A separate re-injected queue is used at the meta-level for segments that needs to be retransmitted. Below this MPTCP interface, TCP subflows are created for each RAT (path). The pooling of the subflow's resources is achieved by multiplexing individual segments across the different subflows. When multiplexing individual segments, MPTCP uses a *scheduler* to decide on which subflow to schedule each segment. The scheduler has access to the state of each TCP subflow, including congestion window ($cwnd$) and round-trip time (RTT). The rate at which segments are sent out on each subflow is determined by $cwnd$. On the receiver side, the segments arrive first at the receive queue on each subflow and then delivered in-order at the subflow level to the common receive queue at the meta-level. Segments arriving out-of-order are placed on an out-of-order (OOO) queue at the meta-level. Note that the receive and OOO queues at the meta-level are shared among all subflows. The size of the required buffer is critical to allow high MPTCP throughput and the amount of the buffer left in the shared buffer is advertised by the receiver to the sender as the receive window ($rwnd$).

Head-of-Line (HoL) Blocking: MPTCP leverages multiple paths with different delay and loss profiles. As packets are multiplexed across the different subflows, the paths' delay and loss differences might cause OOO delivery at the receiver. Note that MPTCP ensures in-order packet delivery. As a result, packets scheduled on the low-delay path might have to wait for the high delay path packets in the OOO queue. This phenomenon is known as the HoL blocking. Fig. 2 depicts an example in which packets 4, 5 and 6 have to wait at the OOO queue as packet 3 from RAT1 is lost. The severity of HoL blocking depends on the delay, throughput, stability of the links, and the size of the buffers (e.g., switch buffers) along the different paths. Wireless networks generally suffer from more unstable links (as compared to wired networks) due to substantial variations in link quality due to mobility, interference, and changes in the multi-path environment. Moreover, emerging wireless technologies such as 5G or 802.11 ad/ay operate in the higher frequency mmWave bands. These bands introduce additional challenges, e.g., mmWave bands are known to suffer from blockages [16], [17], which can suddenly bring down the rate of a few Gbps link to zero. Additionally, these technologies use highly directional beams for communication, which can cause significant performance degradation with mobility. Even if future MAC and PHY improvements (e.g., [18], [19], [20], [21]) result in faster beam steering and link recovery, many realistic wireless setups would contain a very high number of disconnection events, which can significantly degrade the MPTCP and application level performance.

Schedulers: A wrong scheduling decision might result in HoL blocking, OOO packet arrivals, or receive buffer bloats. Accurately scheduling data across multiple paths while trying to avoid these issues is challenging, especially if different paths have very different delay/loss/rate profiles, which is the case in today's heterogeneous wireless networks. To minimize these issues, several schedulers have been proposed in the literature (for full discussion refer to Section VII). The most important schedulers that are used by Linux are minRTT and BLEST. minRTT starts by filling the congestion windows of the subflow with the lowest RTT before advancing to other subflows with higher RTTs. When one of these subflows blocks the connection, the scheduler retransmits the segments blocking the connection on the lowest delay path and penalizes the paths that caused the issue [22]. This can result in under-utilization of paths with burstiness in their rate, leading to suboptimal capacity aggregation. BLEST [23] is designed to increase MPTCP's performance over heterogeneous paths. BLEST monitors the MPTCP send window to reduce the time where the faster subflow cannot send a packet due to insufficient space. We have observed in our experiments that BLEST has a superior performance to minRTT. Thus, unless otherwise specified, we use BLEST as the default MPTCP scheduler. MuSher [24] is a recently developed MPTCP

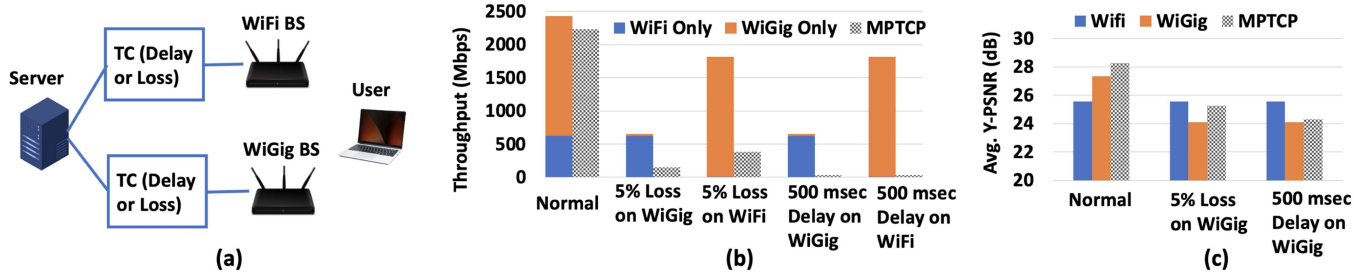

Fig. 3. (a): We conduct experiments leveraging two Netgear Nighthawk X10 routers, a Dell Server and an Acer TravelMate laptop; (b): Throughput comparison of MPTCP's default scheduler (BLEST) and individual SPTCP throughput under ideal, lossy, and delay induced scenarios. Similar drop in performance was observed with minRTT scheduler; (c): Multi-RAT video quality measured as PSNR across all video frames.

scheduler that is designed and evaluated for 802.11ac/802.11ad dual WiFi setups. Their key finding is that the optimal MPTCP performance is achieved if the packet assignment ratio matches the throughput ratio of the two subflows. MuSher is recently shown to achieve low throughout with small queues [25]. We will additionally show in Section VI that MuSher can get very low performance when the client frequently switches between LoS and nLoS channels.

III. Motivation

We conduct preliminary experiments to demonstrate the poor performance of MPTCP in presence of link quality variations and motivate the design of FBDT. The experiment setup is composed of two Netgear Nighthawk X10 routers, which are connected to a server (Fig. 3(a)). The router supports both sub-6 GHz WiFi and 802.11 ad (mmWave WiFi or WiGig) technologies. We set one router as a WiFi BS and the other router as a WiGig BS. The two BSs are connected to the server using a 10G LAN SFP+ interface and at the other end the server is equipped with a 10G high speed Ethernet card. We use an Acer Travelmate laptop as a client to connect to both the WiFi and WiGig BSs. In all our experiments, the client remains static. Both Server and client run Ubuntu-22.04 with kernel version-5.18.0-rc7+. MPTCP-V1 is part of the upstream kernel and it is enabled on both the server and client. To study the performance of MPTCP in a controlled environment with configurable loss and delay profiles, we setup Traffic Controller (TC) between the BSs and the server. TC can be configured to induce controlled delay or loss on the traffic between the server and BSs. We use iPerf3 over MPTCP to generate the TCP traffic and log the throughput results for every 1 sec. Each experiment lasts for 5 minutes, is repeated two times, and we take the average of the measurements to derive the average throughput values.

Throughput: Fig. 3(b) depicts the impact of packet loss or delay on the performance of each individual RAT as well as when the two RATs are used simultaneously by MPTCP. We conduct five sets of experiments: when there is no loss/delay introduced by TC (Normal), when TC introduces 5% packet loss on either WiGig or WiFi links, and when TC increases the delay between packets by 500 msec on either WiGig or WiFi links. When no delay/loss is introduced (i.e., Normal), the standalone TCP rates of WiFi and WiGig RATs are about 630 and 1800 Mbps, respectively. We also observe that MPTCP achieves about 2200 Mbps, which is about 10% less than the summation of throughput across the individual RATs. Introducing packet loss or delay drastically reduces the throughout of the affected RAT in a standalone manner as well as when the two RATs are used by MPTCP. For example, introducing a 5% packet loss to the WiGig RAT reduces its standalone TCP rate to about 30 Mbps and the MPTCP rate to 147 Mbps. Similarly, introducing a 500 msec delay to the WiGig RAT reduces the individual TCP rate and even further reduces MPTCP throughput to 30 Mbps. Note that in either of these two scenarios, a standalone WiFi RAT achieves about 630 Mbps throughput, which shows there is a lot of room for improvement for Multi-RAT transport layer protocol design.

Multi-RAT Video Streaming Quality: In Fig. 3(c), As part of Streaming and Conversational class of traffic, we examine the quality expressed in terms of average PSNR in dB (measured across all video frames). We stream a video of 250 samples per second for different client configurations - WiFi only, WiGig only and MPTCP, and observe that with no delay or packet loss introduced by TC (i.e., Normal), MPTCP achieves a 28.5 dB PSNR. We also observe that packet loss and/or delay significantly reduce the PSNR value for the affected RAT as well as the MPTCP protocol. Note that the MPTCP PSNR (when loss/delay is introduced to the WiGig RAT) is even less than the PSNR that would be achieved by only using the WiFi RAT. This is because as we showed in Fig. 3(b), MPTCP throughput in this case is substantially lower than only using the WiFi RAT. In Fig. 3(c) we show the results when only the WiGig RAT is affected by loss or delay. Similar performance results are observed when loss/delay is introduced to the WiFi RAT.

IV. Design

In this section, we provide details of FBDT design. First, we describe the high level idea of our design and discuss how it can eliminate HoL blocking and provide a throughput that is close to the summation of throughput across individual RATs. Next, we discuss different components of the design, including scheduler, ACKs, sequence numbers, and congestion control, among others. Finally, we summarize how FBDT replaces different MPTCP components. For ease of discussion, we focus on two paths (RATs). We provide details on how FBDT extends

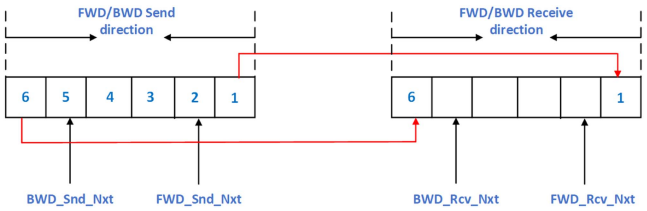

Fig. 4. Here, transmission window size is six. FBDT serves packets from both forward and backward directions. The pointers showing the next forward and backward packets to be served are also illustrated.

to support N paths (RATs) as well as a detailed pseudo-code of FBDT in the Appendix, available online.

A. High Level Description

We propose Forward and Backward Data Transmission (FBDT) to eliminate HoL blocking and Out-Of-Order packet arrival issues when streaming data over multiple SPTCP(s). FBDT supports a *common meta-socket, send and receive buffer* to all the applications to send data across multiple SPTCPs. FBDT assigns sequence numbers to the data segments to be transmitted and starts transmitting data in small in-ordered batches - referred as *transmission window*. For ease of discussion, we assume sequence numbers of packets transmitted from the transmission window are in ascending order (from right to left).

FBDT sends two sets of ordered packets from transmission window - one with ascending and another with descending sequence numbers. Packets with ascending sequence numbers are sent as forward data transmission over one SPTCP (and the associated RAT) and packets with descending order are sent as backward data transmission over another SPTCP. For example, consider six packets to be transmitted over two SPTCPs with heterogeneous delays as shown in Fig. 4. FBDT starts by sending packet1 and moving towards the end of the transmission window over SPTCP1 (*forward data transmission*). In parallel, FBDT starts sending packet6 and moves towards the beginning of the transmission window over SPTCP2 (*backward data transmission*). FBDT relies on the fact that SPTCP delivers reliable and in-order packets to the receiver.

Eliminating HoL Blocking: FBDT eliminates HoL and Out-Of-Order packet arrivals by delivering an in-ordered small set of packets - referred as transmission window- to the application through meta sockets. For example, consider the setup depicted in Fig. 4, in which there are 6 packets to be transmitted from the transmission window. FBDT starts by sending packet1, packet2,... from the forward data transmission over SPTCP1 and packet6, packet5,... from the backward data transmission over SPTCP2. Suppose that packet2 is delayed or lost over SPTCP1 due to the uncertainty of the wireless network. The backward data transmission would continue serving packets from backward including packet2 over SPTCP2 since SPTCP1 was unable to deliver packet2. As a result, FBDT will never encounter HoL blocking or Out-Of-Order packets at the meta-level since it will receive the packets either from the forward or backward directions.

Eliminating Re-Transmission and Re-Transmission Timeout: Unacknowledged packets in the FBDT transmission window are transmitted by either of the two SPTCPs or both in case of a delayed SPTCP transmission. FBDT will discard duplicate packets, which can happen due to delayed SPTCP transmissions. For example, consider there are six packets in the FBDT transmission window as show in Fig. 4. Suppose that packet2's transmission is being handled by SPTCP1 but it is delayed. Meanwhile if backward transmission successfully transmits packet2 over SPTCP2, then the receiver would receive packet2. Now, suppose packet2 is also finally delivered to the receiver by SPTCP1- forming a duplicate packet. When this happens, the receiver will discard the duplicate packet based on the packets' sequence numbers. This eliminates the need for re-transmission and re-transmission timeout since the best effort solution adopted by FBDT ensures packet transmission on other SPTCP(s).

FBDT always Achieves the Optimal Rate: One of the key issues faced by traditional MPTCP architectures is to decide on how to optimally split the traffic across SPTCPs? Additionally, we strive for an architecture that can get the *summation* of individual RAT throughput values (in isolation) across all channel conditions. FBDT ensures optimal traffic splitting across RATs and as we will show later through experimental evaluation, achieves the desired *additive* throughput aggregation in all channel conditions. Recall that FBDT transmits data from both forward and backward directions and the two pointers move inwards (after successful ACK reception) at their own rates, meeting at a certain point. The rate at which the pointers move towards each other depends on the rate at which each of the individual SPTCPs is able to complete their transmission successfully. As a result, this mechanism automatically splits the traffic optimally between SPTCP(s) based on their individual throughput values. Using the TCP throughput equation in [26], we can theoretically derive the number of packets served from the forward direction as:

$$N_{fwd} = N\left(\frac{W_{fwd}RTT_{bwd}}{W_{fwd}RTT_{bwd} + W_{bwd}RTT_{fwd}}\right) \quad (1)$$

Here RTT_{fwd} and W_{fwd} denote the forward RAT RTT and forward path transmission window size, respectively. N denotes the total number of packets to be served. The number of packets to be served from the backward direction can be derived by swapping fwd subscripts with bwd and vice versa.

B. Detailed Architecture

We now discuss the details of different components of FBDT as shown in Fig. 5, and Algorithm 1 also provides a pseudo-code of FBDT.

FBDT Transmission Window: To transmit in both forward and backward directions, FBDT buffers data in a transmit buffer before transmitting across SPTCP(s). We refer to this buffer as FBDT transmission window. The size of this buffer can be as low as $(1 + \lceil \max(\frac{R_{bwd}}{R_{fwd}}, \frac{R_{fwd}}{R_{bwd}}) \rceil)$ bytes. For example, suppose the average forward and backward throughput values (R_{fwd} and R_{bwd}) are 2 Gbps and 400Mbps, respectively. Then the

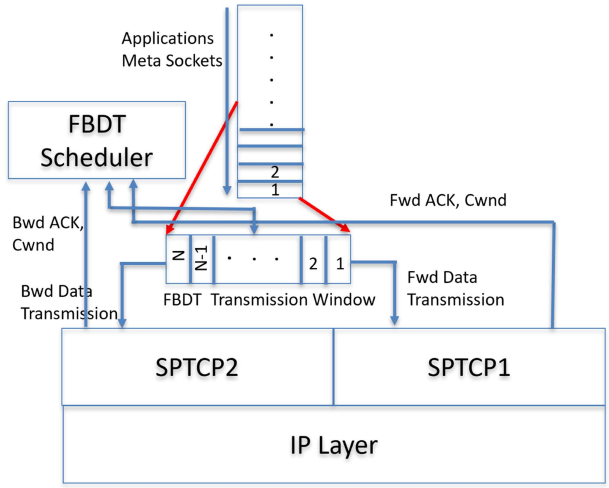

Fig. 5. FBDT architecture with transmission window, FBDT scheduler and interface to FWD and BWD SPCTP.

minimum size of transmit window should be 6 bytes. In our experiments, we fix the size of this window to 64KB. FBDT sender and receiver will agree on the transmission window size before transmission and can periodically adapt that to accommodate any dynamics in RATs' data rates.

FBDT Sequence Numbers: To ensure data in-order data delivery, FBDT assigns unique sequence numbers to the data segments sent by SPTCPs. These sequence numbers help the receiver identify and discard duplicate packets before they reach the application. Since high-speed data communication can cause sequence numbers to wrap back to 0, FBDT uses a random starting sequence number agreed upon by both the sender and receiver. It uses 32-bit unsigned integers for sequence numbers, and when these wrap back to 0, the receiver's expected next sequence numbers in both forward and backward direction are also reset to 0. This mechanism ensures proper handling of sequence number wrapping issue in the FBDT protocol.

FBDT Acknowledgments: ACK(s) at the FBDT level are necessary for both forward and backward transmission pointers to advance inwards in the FBDT transmission window. In SPTCP, when an ACK is sent by the receiver, it also includes information about the next set of packets that are expected to be sent by the sender. MPTCP also uses ACKs at its level. In particular, MPTCP piggy backs its ACKs on the option fields of the SPTCP ACKs. The information includes the expected next set of packets (Data Segment Sequence numbers or DSSs) MPTCP expects to receive. We use the same idea of piggybacking information on SPTCP ACKs. However, as FDBT leverages two (forward and backward) data transmission paths, our piggyback data specifies the next set of packets (sequence numbers) the receiver expects to receive on both the forward and backward directions. This idea can significantly boost performance when there is system dynamics. For example, consider a dual radio WiFi + WiGig setup. Suppose there is an outage on the uplink transmission of WiGig, which blocks its TCP ACKs. In this case, downlink WiGig data packets would still be acknowledged through FBDT ACKs transmitted by WiFi. As a result, there would be no need for redundant transmission of WiGig data packets over WiFi.

In-Order Delivery Across FBDT Transmission Window: Forward and backward send windows move inward as they receive FBDT ACKs. The two send windows will meet at a certain point based on the throughput of the two directions. As they reach this point, it is possible that one of the SPTCP send windows has completed successfully and ready to move on to next set of data, while the other sliding window is still waiting for ACKs of the data transmitted.

When this situation happens, the early completed sliding send window would proceed with transmitting unacknowledged packets in the other sliding window, which results in redundant packet transmission. To provide a balance between delay and redundant packet transmissions, in FBDT the transmitter waits for min(RTT, RTO) before transmitting unacknowledged packets. Here, RTO is the redundant transmission timeout and RTT is the Round-Trip Time of the other transmission end. After this period, the early completed sliding window can proceed with transmitting unacknowledged packets. Selecting an appropriate RTO is critical, as it significantly impacts throughput. At the receiver, duplicate packets will be dropped as packets are checked based on their sequence numbers.

For example, assume a transmission window size of ten with ten packets for transmission ordered from one to ten. Forward and backward transmission start by sending from one and ten, respectively, and proceed to moving inwards. Let us assume forward sliding window has successfully completed sending packet three and backward has transmitted packets five and four and is waiting for their ACK(s). The forward sliding window will wait for a min(RTT_{bwd}, RTO) - where RTO can be 0- before transmitting packets 4 and 5 over its SPTCP. In the case of redundant data transmission, neither the forward nor the backward direction will wait for the ACK(s) any longer, as the data is expected to reach its destination through either direction. This ensures in-order delivery. Consequently, the transmission window is updated with the next set of ten packets to be sent from the meta-socket buffer.

RAT to Direction Mapping: FBDT uses the most reliable RAT in the forward direction as some applications may be able to receive and process orderly received segments without having to wait for the entire receive window to be filled. By assigning the more reliable RAT to the forward direction, we seek to reduce delays and blockages that may happen to these types of applications. Further, the reliability of a RAT is determined by its historical throughput and packet delivery ratio data, and therefore is adjusted over time in order to accommodate dynamics in RAT reliability.

FBDT Scheduler: FBDT schedules segments from its transmit window to the SPTCPs from forward and backward directions by maintaining Send _ Window$_{fwd}$ (with W_{fwd} size) and Send _ Window$_{bwd}$ (with W_{bwd} size) sliding windows, respectively. For each of the two (forward and backward) directions, the size of these send windows are determined similar to how SPTCP calculates them, i.e., $send\_window = \text{Min}(cwnd, rwnd)$. Additionally, FBDT maintains separate Snd _ Una[1] and Snd _ Nxt pointers for both forward and backward sliding windows (see Fig. 6). FBDT schedules packets to SPTCPs depending on

[1]Una stands for Un-acknowledged.

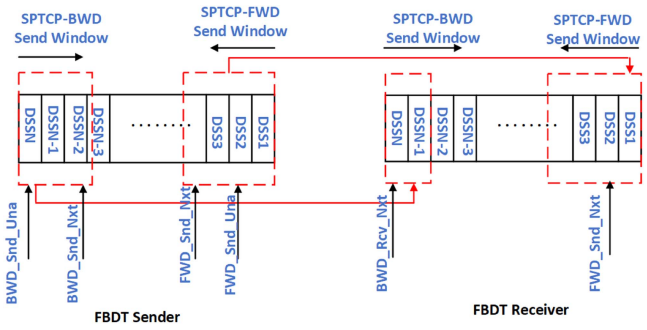

Fig. 6. FBDT scheduler.

the available space in $cwnd$ and moves Snd _ Una and Snd _ Nxt appropriately in their FBDT transmission windows. FBDT sender will move the Snd _ Una pointers of both forward and backward when it receives the corresponding acknowledgments at FBDT level. Additionally, FBDT receiver will acknowledge both forward and backward with the expected Fwd _ Snd_Nxt and Bwd _ Snd_Nxt, respectively. Finally, FBDT moves Snd _ Nxt pointer of forward and backward as it schedules packet to SPTCP for transmission. The two sliding windows will meet at a point depending on the throughput of forward and backward SPTCPs. FBDT decouples the two congestion control algorithms (CCAs) and lets each CCA to decide on its transmission based on the congestion and reliability of its underlying network.

Transmission Window Update as Pointers Cross: FBDT completes serving its transmission window when Fwd _ Send_Nxt and Bwd _ Send_Nxt are equal or cross over each other. At this point, FBDT will wait for either of the Snd _ Una to meet their Send _ Nxt before redundant packet transmission begins. Suppose backward transmission was able to successfully complete earlier and both Fwd _ Send_Nxt and Bwd _ Send_Nxt has crossed over each other. Then backward transmission of FBDT will wait for RTT$_{fwd}$ and start transmitting redundant unacknowledged packets in the forward sliding window. FBDT will not wait for the ACKs of redundant packets since the packets are scheduled in both of the SPTCPs, which will likely deliver them in-order to the receiver. FBDT will move on to load the transmission window with the next set of data and proceeds with their transmission.

The rate at which Send _ Window$_{fwd}$ and Send _ Window$_{bwd}$ move towards each other depends on the rate (throughput) of each SPTCP. As FBDT adds up the throughput of each individual SPTCP, we can approximate its total throughput leveraging SPTCP throughput formula [26] as:

$$Throughput_{FBDT} = \gamma \times \left(\frac{W_{fwd}}{RTT_{fwd}) } + \frac{W_{bwd}}{RTT_{bwd})} \right) \quad (2)$$

Here, γ is a scalar factor that accounts for overhead. In practice, we have observed that FBDT throughput is very close to the summation of individual RAT data rates, irrespective of the channel conditions. In other words, γ is very close to one.

C. FBDT Design for N RAT(s).

FBDT design for two SPTCPs can be extended to N number of SPTCPs. Suppose we have N number of SPTCPs. Let $FBDT_{SPTCP} = \{SPTCP_1, SPTCP_2, \ldots, SPTCP_N\}$, where $FBDT_{SPTCP}$ is the set of SPTCPs - sorted based on reliability. The most reliable SPTCP will serve as forward transmission SPTCP and the other SPTCPs will serve as backward transmission as shown in Fig. 7. FBDT considers lower sequence numbers in the transmission window as high priority and expects it to be delivered to the receiver through the more reliable SPTCP network. FBDT uses backward SPTCP network to transmit lower priority packets (higher sequence numbers). Suppose that the backward SPTCP network can't deliver the packets due to its network unreliability. Then the forward network will eventually transmit the backward data. Though reliable networks are comparatively slower, using most reliable network as forward transmission guarantees reliable and faster packet delivery (for a detailed example of this phenomenon, please refer to "RAT to Direction Mapping" paragraph in Section IV). The overall throughput of FBDT can be approximated as:

$$Throughput_{FBDT} = \gamma \times \left(\frac{W_{fwd}}{RTT_{fwd}} + \sum_{i=1}^{i=N} \frac{W_{bwd_i}}{RTT_{bwd_i}} \right) \quad (3)$$

To avoid under-utilizing backward SPTCPs, we place them from the forward SPTCP such that there is a sufficient gap between SPTCPs. Each SPTCP uses $send\_window = \text{Min}(cwnd, rwnd)$ to send data and $cwnd$ will grow/shrink based on the underlying network congestion. In FBDT, the backward and forward transmission send windows move towards each other. To avoid crossing over each other quickly, the backward SPTCPs has to be placed far enough so that they have sufficient packets to fill the $cwnd$ of the SPTCP. To achieve the highest possible FBDT throughput, backward transmission segment size and offset are vital. Each backward SPTCP is placed at $FBDT_{offset}$ such that it has sufficient packets to fill its SPTCP. The best way to estimate the segment size and offset of backward SPTCP is based on estimated throughput. To begin with, FBDT considers no packet loss and shadows RTT values based on the corresponding SPTCP RTT values. Based on the throughput estimation, the segment size and offset can then be set by:

$$SegSize_{j,j-1} = N \left(\frac{Throughput_j}{\sum_{k=1}^{k=N-1} Throughput_k} \right) \quad (4)$$

$$SegOffset_{j,j-1} = \sum_{K=1}^{K=j-1} SegSize_{K,K-1} \quad (5)$$

Here, j and $j-1$ denote two adjacent backward SPTCPs and N is the total number of bytes in the transmission window.

Though FBDT places backward SPTCP segments at the appropriate offset but due to network throughput variations, some of the SPTCPs will complete their assigned segments earlier than other SPTCPs and tend to cross over into other SPTCP segments. To handle this, when an SPTCP completes its assigned segments then FBDT will recompute its offset and segment sizes

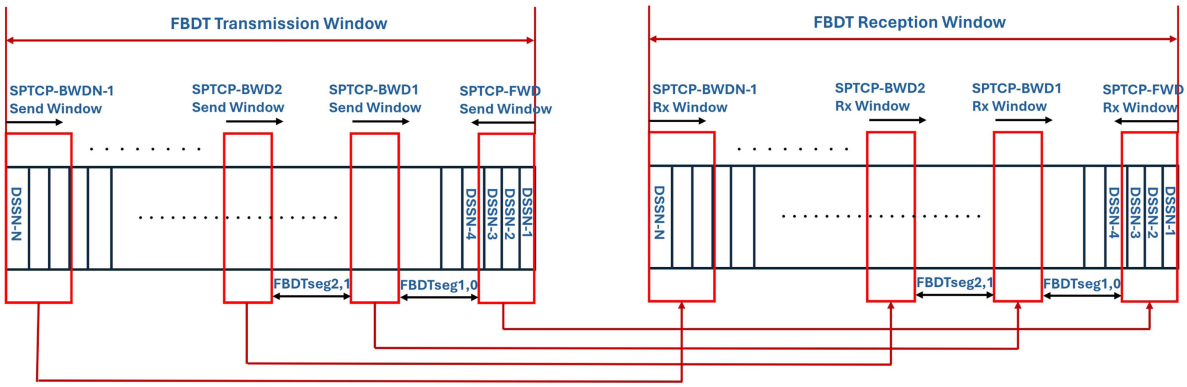

Fig. 7. FBDT scheduler with N number of SPTCPs. There is one SPTCP-FWD and $N-1$ SPTCP-BWD.

in the transmission window. Once the new SPTCP's alignment in the transmission window is determined, FBDT sender sends a message to the receiver with the new alignment before resuming transmission.

Algorithm 1: FBDT Pseudo-Code.

$SPTCP \in SPTCP1, SPTCP2, ...SPTCPn$
$TotalBytes \leftarrow 0$
function FBDTSendData(SPTCP)
  if $TotalBytes = TransWindowSize$ **then**
    return
  end if
  //Check if all the seg have data to transmit
  if CheckAlignment==False **then**
    AlignFBDTSendWin($FwdSPTCP$)
  end if
  $BytesSent \leftarrow SendSPTCP(SPTCP, W_{SPTCP})$
  $TotalBytes \leftarrow TotalBytes + BytesSent$
  $SPTCP \leftarrow NextSPTCP$
  FBDTSendData(SPTCP)
end function
function AlignFBDTSendWin($SPTCP$)
  if SPTCP=NULL **then**
    return
  end if
  if SPTCP=FwdSPTCP **then**
  //Arrange UnAck pkt contiguously
    RealignTransWinBufTransWin
    $C \leftarrow \frac{N}{TotalThroughput}$
    $SegSize[SPTCP] \leftarrow C \times \frac{W_{fwd}}{RTT_{fwd}}$
    $SegOffset[SPTCP] \leftarrow 0$
  else
    $SegSize[SPTCP] \leftarrow C \times \frac{W_{bwd}}{RTT_{bwd}}$
    $SegOffset[SPTCP] \leftarrow PrevSegOffset$
  end if
  $PrevSegOffset \leftarrow SegSize[SPTCP] + PrevSegOffset$
  AlignFBDTSendWin$NextSPTCP$
end function

D. FBDT Congestion Control

FBDT supports both coupled and decoupled Congestion Control Algorithms (CCAs). Out of the box design supports decoupled CCA where it lets each of the two SPTCPs' congestion control algorithms work to the fullest extent without FBDT's interference. However, FBDT can be easily extended to coupled congestion control by controlling the amount of packets scheduled for each of the two SPTCP(s). Coupled CCA is sometimes preferred over decoupled CCA since it is shown to better maintain fairness over bottleneck links [15], [27], [28].

IETF study on MPTCP suggests three goals for a practical coupled CCA [15]: (i) Improve Throughput, i.e., higher performance than a single path TCP, (ii) Do no Harm, i.e., not take up more capacity from any of the resources shared by its different paths than if it were a single flow using only one of these paths, and (iii) Balanced Congestion: move as much traffic as possible off its most congested paths, subject to meeting the first two goals. We next show that we can achieve these goals by optimizing the scheduler and therefore, by regulating the amount of packet scheduled to the individual SPTCP(s) without the need for SPTCP modifications.[2]

Goal 1 (Improve Throughput): Since FBDT uses forward and backward data transmission it gives equal chance to both the SPTCPs to transmit to the fullest extent possible. If any of the two SPTCPs performs poorly, the other SPTCP will transmit the data without the need to wait for the other SPTCP to complete. For example, consider FBDT has assigned SPTCP1 and SPTCP2 to transmit from forward and backward, respectively. If SPTCP1 is unable to transmit due to the underlying network uncertainty, SPTCP2 will transmit all the packets from the backward, compensating for SPTCP1's poor performance. In the worst case, if one SPTCP is completely blocked, FBDT will achieve a throughput that is equal to the other SPTCP's throughput. As a result, FBDT will *always provide additive throughput gains.*

Goal 2 (Do no Harm): FBDT can be extended to ensure fairness towards single path clients without modifying the SPTCP protocol. To achieve fairness, we let the $cwnd$ of the SPTCP(s) untouched, and introduce scaling factors α and β to scale up the forward and backward send window sizes, respectively.

[2]We leave a comprehensive design of *fair* coupled CCA for FBDT as part of our future work.

By controlling α and β we can regulate the amount of traffic scheduled for each SPTCP.

Goal 3 (Balanced Congestion): FBDT by design optimally divides the traffic across SPTCPs. For example, if Send _ Window$_{fwd}$ is congested, by design FBDT will keep on transmitting from the backward direction. Thus, this aspect of CCA is naturally handled by FBDT.

E. FBDT: Beyond MPTCP

MPTCP-IETF standards propose different components for the protocol such as schedulers, CCAs, re-transmission and Out-Of-Order queues at MPTCP level on top of similar components at the SPTCP level. FBDT breaks this design by relying on strategic scheduling and the conventional SPTCP for reliable transmission, congestion control and retransmission. To summarize, FBDT eliminates retransmission protocols by sending redundant packet over the other subflows, eliminates additional congestion control by controlling the amount of packets scheduled to each SPTCP and eliminates re-arranging OOO packets through a forward and backward data transmission scheme. We believe that MPTCP has overlooked conventional SPTCP's congestion control and reliability mechanisms, and is lavish in proposing additional retransmission, congestion control and re-arranging OOO packets at the MPTCP level. This comes with an additional cost, adds complexity, and reduces the overall system throughput.

V. Implementation

We extended the implementation of FBDT for multi-RAT communication, configuring it as a daemon in the user space in Linux for both the client and server. Client connects to the server using two different BSD SPTCP sockets - one for WiFi and another for WiGig. FBDT maintains a transmission window using char array - whose size is configurable based on the class of application. The FBDT daemon on the server sends packets through FBDT transmission window. Every segment in the transmission window -set to Maximum Segment Size (MSS)- is numbered with an FBDT sequence number. As mentioned in the design section, we use the most reliable RAT - WiFi- as forward transmission and the less reliable RAT(s) - WiGig, Ethernet- as backward transmission. Based on the throughput (5), FBDT transmit window on the server maintains FWD pointer and BWD1, BWD2, BWD3 etc., pointers for forward and all the backward transmissions. Forward and all backward pointers move inward as cumulative ACKs are received from the client. In our implementation, the client sends a cumulative ACK on each transmission from the user space with both forward and all the backward ACKs. FBDT receive window on the client also maintains a FWD and one pointer for each BWD transmission, and they move inward as they receive packets from the server. When any of the two pointers meet at some point within the transmission window, then the FWD and BWD pointers are re-estimated as per (5). FBDT resumes its transmission with the new pointers moving inwards. Once all pointers (FWD and BWD) meet at a point inwards, the transmission window is reloaded with a new set of data to be transmitted.

Migrating FBDT to the Linux Kernel: We have migrated FBDT with multi-RAT communication to the latest Linux Kernel-5.18-rc7+ and replaced the existing MPTCP protocol. This includes replacing the default MPTCP scheduler as well as the protocol with FBDT. Additionally, we modified the (i) protocol.c file of mptcp in the kernel, (ii) mptcp _ sendmsg with FBDT transmission algorithm, and (iii) mptcp _ send_ack and subflow ack with FBDT ACK(s). These are piggy bagged on TCP ACK(s). As per FBDT design, we eliminated MPTCP retransmission logic, Out-Of-Order buffer logic, and retransmission timeout from the MPTCP code, since it is no longer needed by FBDT. We have open-sourced our software and publicly released it on GitHub [11] so that other researchers in the community can benefit from our work and expand on it.

VI. Performance Evaluation

In this section, we evaluate the performance of FBDT through experiments. We first conduct experiments to quantify the throughput that can be achieved with different MPTCP protocols and under different channel and client mobility conditions. We mostly consider a dual WiFi+WiGig setup but we also study FBDT performance with three RATs. Finally, we study the viewport quality under different MPTCP protocols and tile ordering and coding schemes. In all of our experiments, we only consider a single client. We leave performance evaluation in multi-client or multi-BS (e.g., multiple WiGig and WiFi BSs) scenarios as part of our future work.

A. Experimental Setup

Network Deployment: Fig. 8(a) depicts some of our hardware. Our network is composed of two Netgear Nighthawk X10 routers. These routers support both WiFi and WiGig. We set one router as WiFi only and the other router as WiGig only to emulate a non-colocated BS scenario. The routers are connected to a Dell server with connections discussed in Section II. Our client device is an Acer TravelMate laptop that has both WiFi and WiGig cards. This laptop is placed on top of a TurtleBot robot. The robot can be programmed to stay stationary or mobile with a configurable speed and mobility pattern. These devices are deployed in an indoor office environment depicted in Fig. 8(b).

Channel Conditions and Mobility Patterns: We consider three different scenarios: (i) LoS: In this setup, the client (TravelMate laptop) is placed at a fixed location about 8 feet from the WiGig BS. The client has a LoS channel to both BSs and remains fixed at its location throughout the experiment, (ii) nLoS: In this setup, the client remains fixed at the location similar to the LoS experiment but a human blocker stands about 3 feet in front of the WiGig BS throughout the experiment, (iii) Mobility: In this setup, the client device (on top of robot) moves in a rectangular pattern of 6 ft by 2 ft, as depicted in Fig. 8(b). There is no human blocker between the robot and the BSs. In this setup, the client initially faces directly the BSs in a LoS channel condition. But then the robot turns 90° followed by two other 90° turns. In these positions, the client channel becomes nLoS because the body of the laptop blocks the LoS path. As a result, in this setup the client frequently switches between LoS and nLoS channel conditions.

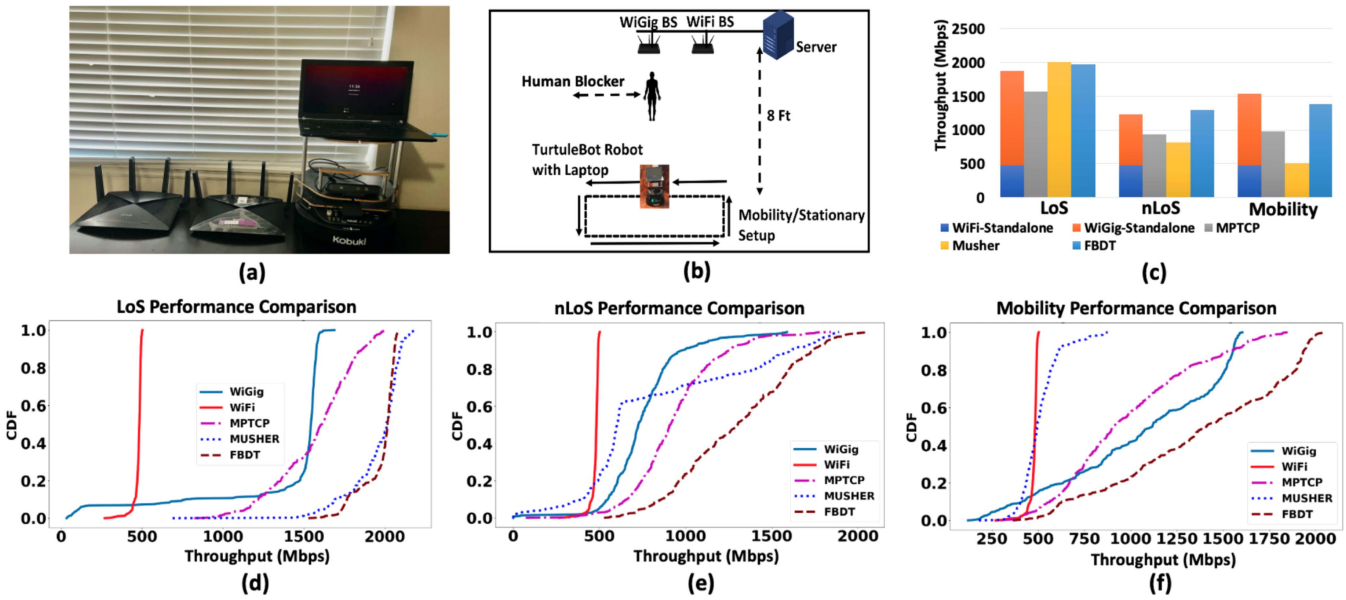

Fig. 8. (a): Our hardware setup. TravelMate laptop is placed on top of TurtleBot robot for controlled mobility, (b): Network deployment layout. A human stands in front of the WiGig BS to create blockage, (c): Average throughput results across standalone WiFi, standalone WiGig, MPTCP (BLEST), MuSher, and FBDT in LoS, nLoS, and mobility, (d): LoS CDFs, (e): nLoS CDFs, and (f): Mobility CDFs. The spread in measured throughput across all schemes (except for standalone WiFi) increases as channel condition changes from LoS to nLoS/mobility. MuSher has a low throughput spread under mobility, which is because the scheme primarily resorts to using only WiFi for communication.

Traffic Generation: We use iPerf to generate downlink TCP traffic from the network to the client. This communication lasts for 5 minutes. We repeat each experiment two times and plot the average and CDF curves (with throughput sampled across one second intervals).

Implemented Solutions: We experimented with the following protocols: (i) FBDT: our proposed architecture, (ii) MuSher [24], which is the state-of-the-art MPTCP scheduler designed for dual radio WiGig+WiFi setups. We used the software code that was publicly released by MuSher authors to implement it in our hardware. We have been able to successfully replicate their results; (iii) MPTCP: As we mentioned in Section II, minRTT and BLEST are two of the default MPTCP schedulers implemented in Linux Kernel. We present only the results achieved under BLEST as in all of our experiments BLEST outperformed minRTT. Finally, in addition to the above MPTCP protocols, we also conduct SPTCP experiments when only one RAT is used for communication (e.g., when only WiFi or WiGig BS is turned on).

B. Throughput Statistics

Throughput in Dual Radio WiFi+WiGig Setup: Fig. 8(c) depicts the average throughput results across all protocols, channels, and mobility conditions. Recall that in LoS/nLoS experiments, the client remains stationary. Under standalone LoS configuration (leftmost bar), WiFi and WiGig achieve an average throughput of 450 and 1400 Mbps, respectively. In standalone mode, we use SPTCP and only a single RAT to measure throughput. MPTCP (with BLEST) aims to use the fastest subflow (i.e., WiGig) most and directs an estimated amount of bytes on the slower subflow with a combined throughput of 1550 Mbps. FBDT and MuSher utilize both subflows to the fullest extent achieving a combined throughput of 1970 Mbps. This is 20% higher than MPTCP, and 40% and 4.5x better than standalone WiGig and WiFi, respectively. The CDF of the throughput variations in the LoS condition is plotted in Fig. 8(d). The throughput of MPTCP scheduler varies from 1000 Mbps to 1800 Mpbs, whereas FBDT gives a throughput of 1970 Mbps 80% of the time.

The second group of bars in Fig. 8(c) show the throughput value in stationary nLoS condition. First, we observe no change in WiFi standalone throughput, whereas WiGig drops by almost 50%. This is because sub-6 GHz communication is mostly immune to human blockage, whereas WiGig beam switching to a reflective nLoS path significantly drops the signal SNR and throughput. We also observe that MPTCP gets a combined average throughput of 930 Mbps, which is lower than its LoS throughput and is due to drop in WiGig performance. MuSher gets 814Mbps throughput, which is even lower than baseline MPTCP. On the other hand, FBDT provides additive throughput gains with an average of 1300Mbps. In fact, FBDT throughput is slightly higher than simple summation of average WiFi and WiGig standalone rates. The slight difference (higher or lower) than pure summation of individual RAT data rates can be attributed to the following: (i) each time we run an experiment, we naturally gets some variation in throughput values. We try to minimize this by running two experiments for each measurement but one would need to repeat experiments more to minimize this impact; (ii) FBDT has several mechanisms such as piggybacked ACKs, duplicate transmissions, etc. built in to the protocol, which can add or remove from the overall system overhead

depending on the experiment. The corresponding nLoS CDF curves are plotted in Fig. 8(e). We observe a much higher increase in throughput spread across all schemes except for standalone WiFi. This is because the WiGig RAT routinely switches its beam in nLoS channels creating data rate fluctuations.

The last throughput bars in Fig. 8(c) show throughput values under client mobility. We observe no major change in WiFi throughput, whereas WiGig throughout is higher than its rate in stationary nLoS condition. This is because under mobility the client switches between LoS and nLoS channels. In LoS channels, WiGig benefits from much higher throughput values, which results in a higher spread as shown in the corresponding CDF throughput curve (Fig. 8(f)) and a higher average throughput (Fig. 8(c)). On the other hand, MPTCP is not able to benefit effectively from increase in the average WiGig rate as its throughput remains close to its nLoS throughput. This is because the BLEST scheduler is too conservative in its estimation of WiGig blockage, which stops MPTCP from fully benefiting from WiGig when mobile client switches to LoS. We also observe that MuSher gets a performance that is close to half of baseline MPTCP, while FBDT gets a throughput that is slightly less than pure summation of standalone WiFi and WiGig data rates. There are several reasons for MuSher's poor performance. First, our baseline MPTCP is the newest implementation of MPTCP in the Linux kernel and is more up to the date than the baseline compared against in MuSher [24]. Second, our mobility pattern is much more complex than the mobility pattern studied in MuSher, in which the client remains in LoS and moves towards, away or left-right in front of the BS. Our mobility pattern involves frequent switching between LoS and nLoS as expected in real-world environments. Third, we have observed that MuSher is unable to timely estimate the bandwidth of different RATs, which makes the protocol sub-optimally split the traffic between RATs with much more emphasis on WiFi to a degree that it achieves even a lower performance than MPTCP.

We use the Traffic Controller (TC) setup that we discussed in Section II to perform the experiments. Fig. 3(b) shows the standalone RAT data rates as well as when they are simultaneously used by FBDT. We show the results under normal (static, no blockage) and when we introduce 5% loss to WiGig. We observe that FBDT is very close to the summation of standalone throughput values. We have conducted other experiments with delay/loss introduced to Ethernet or WiFi. In all, we have seen the same performance.

C. FBDT: Evaluating Mobile Traffic Class QoE

FBDT offers optimal performance across various classes of applications, regardless of their traffic characteristics, including short and bursty, large file downloads, and delay-sensitive data. For Interactive and Background classes of traffic, FBDT dynamically adjusts the transmission window size to ensure optimal performance. Despite the lower bandwidth and looser delay requirements of these classes, FBDT efficiently adapts its transmission window size for smaller and bursty traffic. This is done by adjusting the window size based on the data to be transmitted and utilizing both radios additively, while minimizing transmission of redundant data. For larger data transfers, FBDT divides the data into fixed transmission sizes and optimally transfers data using both radios.

In the case of Interactive and Background classes, FBDT achieves an optimal transmission rate, as detailed in Section IV. For Streaming and Conversational classes, FBDT provides applications with the flexibility to prioritize packets for more reliable forward transmission or less reliable backward transmission. Furthermore, applications can duplicate packets within the FBDT transmission window to achieve the desired low-latency reliability in wireless networks.

On-demand streaming applications have the content available in advance at the sender (server) and transmit video streams to a client upon request. These applications are generally less delay-sensitive. FBDT continuously transmits streaming data in fixed chunks known as transmission windows. The rate at which the client receives the data streams depends on the throughput of the underlying wireless network.

Conversational traffic, which is highly delay-sensitive and involves limited data within a given time, benefits from FBDT's optimal performance based on the organization of data within the transmission window. More details about Conversational traffic are provided in [12].

Evaluating QoS for Interactive and Background Traffic Classes under Mobility.

Setup: We assessed FBDT performance with Interactive and Background class traffic using web pages and file downloads. For file downloads, we developed server-side scripts to transmit files of known sizes (ranging from 100 KB to 1 GB) to the client. On the client side, we automated a test script to download these files and recorded the time taken under various configurations. For web pages, we obtained objects from well-known sites and hosted a local web server, as detailed in Section III. We selected four highly popular sites for evaluation based on their diverse number of page objects and sizes. Using automated client-side scripts, we repeatedly downloaded these web page objects with one-second intervals between downloads. These scripts logged the time taken to download objects from the webpages under different configurations: WiFi only, WiGig only, MPTCP, and FBDT.

Evaluating File-download QoS under Mobility: Fig. 9(a) shows file download time for WiFi only, WiGig only, MPTCP and FBDT different configuration. Though WiFi performs better than WiGig for smaller file sizes up to 100 K but under performs for larger file sizes ($>$ 100K) by 2x as compared to WiGig or MPTCP. This disparity arises from WiGig waking up from sleep mode due to the large volume of data transfers. FBDT performs better than up to 30% higher than WiGig or MPTCP and up to 3x better than WiFi. FBDT Utilizes WiGig and WiFi additively for better performance while minimizing transmitting redundant data.

Traffic Split Ratio Under Mobility: Fig. 9(b) denotes the histogram of FBDT traffic split across WiFi and WiGig during file download. The x-axis shows the percentage of traffic over WiGig in brackets of 12 percent intervals. The y-axis shows the percentage of the event happening. We observe that 38% of the times about 72-84% of traffic passes through WiGig (the highest bar). We also observe that WiGig is used quite effectively by FBDT under mobility as majority of the traffic

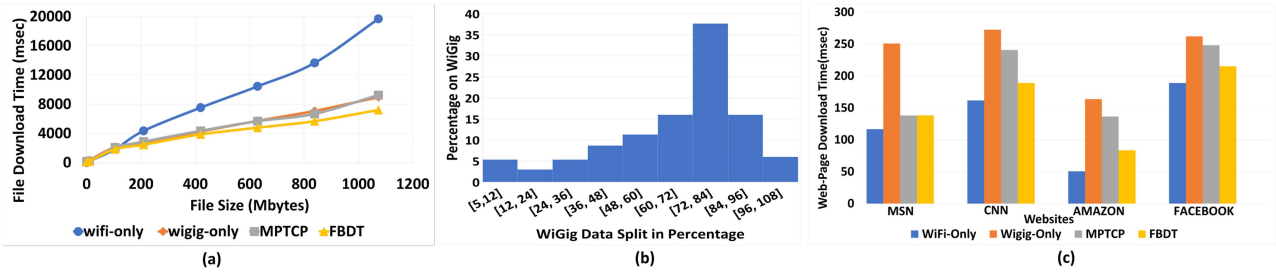

Fig. 9. (a): File download time comparison between FBDT, MPTCP and standalone RAT(s) (b):Histogram of traffic share passed through WiGig under mobility and when using FBDT. The results show that FBDT uses WiGig to pass majority of the traffic,. (c) Web-page Download time comparison between FBDT, MPTCP and standalone RAT(s).

TABLE I
WEBSITE OBJECT STATISTICS

Website	No of Objects	Total size	Average ± SD
msn	33	5.8M	179K ± 389K
cnn	277	21M	75K ± 231K
amazon	325	8.9M	27K ± 121K
facebook	162	20M	122K ± 249K

is passed through WiGig. This is because FBDT by design always optimally splits the traffic across RATs. It is therefore very quick at leveraging WiGig when channel condition turns LoS and WiGig rates drastically increase.

Evaluation of Web-page download QoE under Mobility: In Fig. 9(a), the download times of web pages from various websites are presented under different mobility scenarios: WiFi only, WiGig only, MPTCP, and FBDT configurations. The page download time encompasses the duration required to download all the objects associated with the respective webpage. A typical webpage consists of multiple object requests/downloads from the server, such as images, scripts, and fonts, each varying in number and size, as detailed in Table I. Webpage downloads often exhibit intermittent bursts of file downloads. Despite WiGig's higher throughput compared to WiFi, WiFi outperforms WiGig in terms of webpage downloads.

To optimize power consumption, mobile clients configure wireless radios to transition from active to sleep or idle states. In our setup, the wakeup time for WiGig is longer than that for WiFi. The intermittent and bursty nature of webpage downloads prompts wireless radios to enter sleep or idle states, enhancing WiFi's webpage download performance compared to WiGig. FBDT demonstrates a 1.2 to 2x improvement over WiGig and MPTCP but lags 15% behind the WiFi performance. Although FBDT predominantly schedules data over WiFi, it also attempts to utilize WiGig when feasible. However, if the data scheduled over WiFi is successfully acknowledged within the transmission window while the scheduled WiGig data is pending acknowledgment, FBDT waits for the WiGig data transmission to avoid redundant data transmission over WiFi. This approach, while reducing redundancy, results in slightly lower performance compared to WiFi.

Evaluating Conversational and Streaming class of Traffic QoE.

Setup: We examined Conversational and Streaming traffic categories through Progressive Downloads and Multi-RAT Live Video Streaming. For progressive video download assessment, we streamed a 5-minute segment of a compressed 8 K UHD video with varying bit rate over time, from the server to the client, as outlined in Section III. The client received these video streams via either a single RAT or dual RAT (WiFi and WiGig) and stored the data in a circular buffer. At one-second intervals, the client checked the cumulative bytes received and recorded pauses if the received data fell short of the required amount for that one second compressed interval (known as Group of Pictures, or GOP) of the video content. This process was automated for 110 trials. We repeated this experiment with various recorded UHD video streams from the server, using the configuration depicted in Fig. 3(a). The client received these video streams using WiFi, WiGig, MPTCP, and FBDT configurations.

For Conversational Multi-RAT video streaming, we consider Videos that are encoded as Groups of Pictures (GoPs) representing one second worth of video data. Each compressed GOP comprises multiple video frames captured at a given temporal rate. A video set uses 30 frames per GoP universally. In Multi-RAT video streaming, each video is spatially broken up into small sectors or tiles that can be independently compressed across the duration of a GOP. In our study, the tiles are encoded at a given quantization parameter (QP) independently, so a frame can include tiles with many different QPs. The tiles at the same spatial location are jointly encoded across a GoP. We considered all the 15 videos from the publicly available 8 K UHD video dataset [25] to choose quality points (QPs), i.e., transmission rates, for each of the tiles of the video based on how likely they are to be observed. we consider FIX, an algorithm which performs the same transmission rate selection i.e., the video tiles are given equal priority without any tile ordering and transmitted using FBDT and MPTCP transport layer.

Evaluation of Progressive Download under Mobility: In Fig. 10(a), we present the pauses encountered by the client (Travelmate) across various video and connectivity configurations. For 8 K video at 30 fps and 425 Mbps, no pauses occurred in any configuration. However, when the video quality increased to 8 K at 60 fps and 900 Mbps, WiFi alone led to about 33% of pauses with varying dispersion levels measured using standard deviation. In contrast, WiGig-only, MPTCP, and FBDT configurations experienced no pauses. When the server streamed 8K video at 90 fps and 1.3 Gbps, WiFi-only and WiGig-only setups led to pauses for the client, with pause percentages at 55% and 7% respectively, each with different dispersion levels.

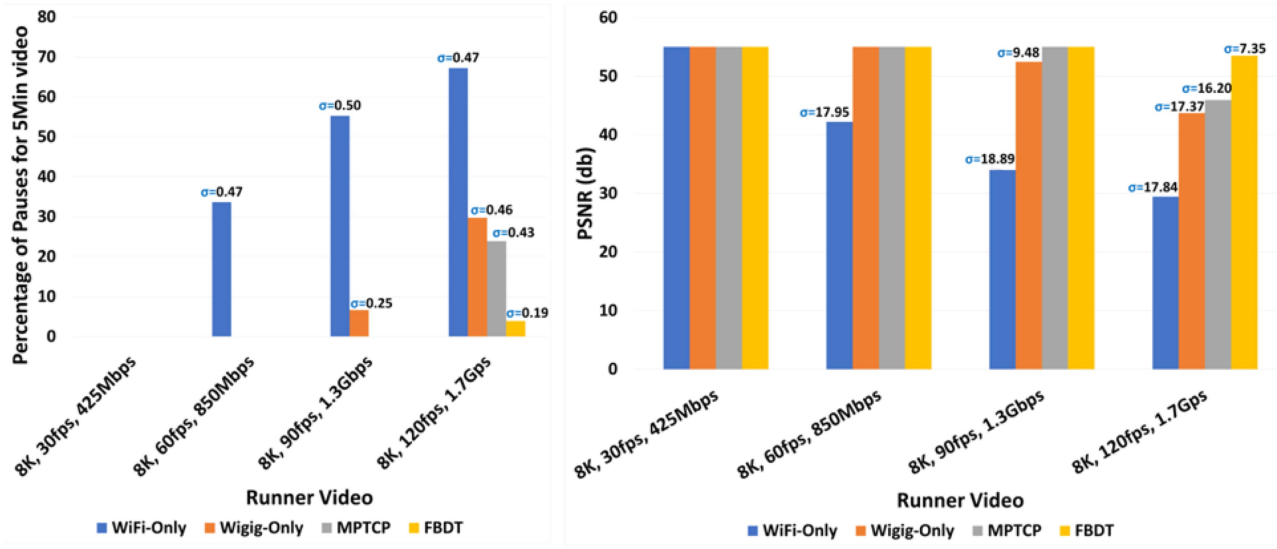

Fig. 10. (a): Bars represent the percentage of pauses during progressive video downloads, while the values on top indicate the standard deviation of pauses over a 5-minute interval. (b): Bars represent the Quality of user Experience (QoE) measured using Average PSNR, while the values on top of each bar indicate the PSNR standard deviation.

Notably, FBDT and MPTCP showed no pauses. For 8K video at 120 fps and 1.7 Gbps, WiFi-only, WiGig-only, and MPTCP setups caused pauses for the client, with pause percentages at 67%, 30%, and 24% respectively, all with high dispersion. FBDT, however, resulted in only 4% of pauses with low dispersion.

Furthermore, in Fig. 10(b), we illustrate the Quality of Experience (QoE) measured as the average PSNR for the client (Travelmate) under different video and connectivity configurations. For 8 K video at 30 fps and 425 Mbps, there was no QoE degradation with average PSNR of 55 dB. When the video quality increased to 8 K at 60 fps and 900 Mbps, WiFi alone achieved an average PSNR of 42 dB with varying dispersion levels. In contrast, WiGig-only, MPTCP, and FBDT configurations experienced no QoE degradation. When the server streamed 8 K video at 90 fps and 1.3 Gbps, WiFi-only and WiGig-only setups led to QoE degradation for the client, with average PSNR values of 34 dB and 52 dB respectively, each with different dispersion levels. Interestingly, FBDT and MPTCP configurations resulted in no QoE degradation. For 8 K video at 120 fps and 1.7 Gbps, WiFi-only, WiGig-only, and MPTCP setups caused QoE degradation for the client, with average PSNR values of 30 dB, 44 dB, and 46 dB respectively, all with high dispersion. In contrast, FBDT resulted in an average PSNR of 53.5 dB with low dispersion.

Evaluation of Conversational Multi-RAT video under Mobility: Fig. 11 depicts the mean PSNR gain across all videos frames for three channel conditions: LoS, nLoS, and mobility. The PSNR gain is calculated as the difference in PSNR across two schemes: FIX on top of FBDT and MPTCP. We observe that even in the LoS scenario where the throughput gap between MPTCP and FBDT is low, there is still more than 0.8 dB gain in PSNR. Whereas under nLoS and mobility, we observe a gain of 0.5 dB gain in PSNR. Further, we show that by reordering video tiles and transmitting over FBDT can give PSNR gains of up to 13 db under mobility conditions [12].

Evaluation with Four RATs: Unlike many other protocols (e.g., MuSher, which only supports dual RAT setups), FBDT

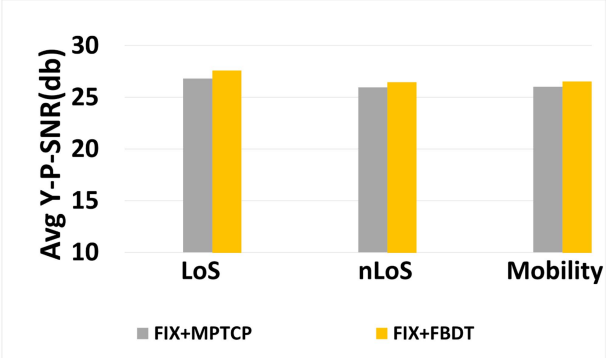

Fig. 11. Viewport quality measured as PSNR across all video frames.

supports any number of RATs. FBDT operation with N RATs are provided in Section IV. We now investigate FBDT's performance in a four RAT scenario. We choose two more Ethernet connections as our third and fourth radios instead of adding an additional WiFi or wireless RAT because we have the capability to fully control the wired medium (delay, loss) unlike the wireless channel. We use the Traffic Controller (TC) setup that we discussed in Section III to perform these experiments. Fig. 12(a) shows the performance comparison of FBDT against MPTCP as the default scheduler. This evaluation was carried out with four RATs operating under normal conditions (static, no blockage). The throughput of FBDT grows linearly with the number of RATs, and gives up to 2x better performance as compared to MPTCP. Further, we introduce 5% loss to WiGig, and observe that FBDT is very close to the summation of standalone throughput values as shown in Fig. 12(b). We have conducted other experiments by introducing 5% loss on one of the Ethernet. Fig. 12(c) shows the performance of FBDT growing linearly with the available BW of the underlying wireless network, while MPTCP collapses as we add more loss to the underlying wireless network. In all the three scenarios with four radio variations as

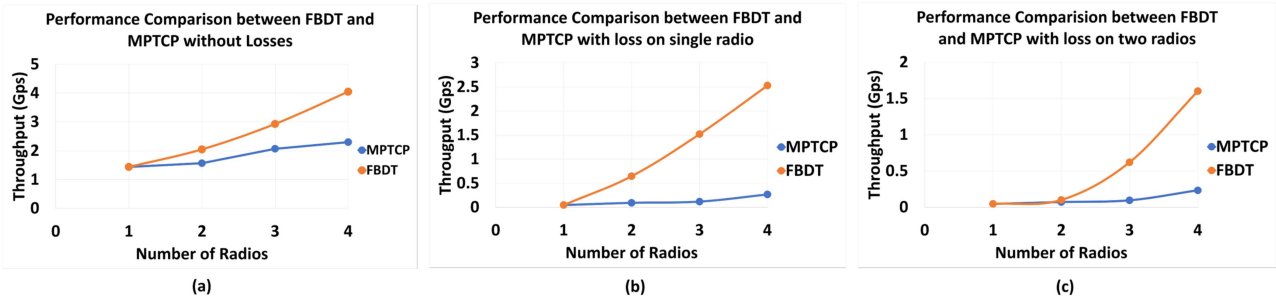

Fig. 12. Multi-RAT throughput comparison between FBDT and MPTCP (a): Without Loss (b): TC introduced 5% loss on single Radio (WiGig) (c): TC introduced 5% loss on two radios (WiGig and Ethernet).

shown in Fig. 12, we have seen the same performance, where MPTCP starts to collapse as we add radios and introduce losses. The MPTCP scheduler (minRTT/BLEST) is unable to schedule packets while eliminating HoL issues.

VII. Related Work

MPTCP Evaluation: Several papers [29], [30], [31], [32], [33] have studied MPTCP performance but they consider scenarios that consist of Internet paths or wireless setups with only sub-6 GHz RATs. Other works have studied MPTCP performance in networks that use mmWave RATs, e.g., [34], [35] studied dual WLANs with 802.11 ac+ad and show that MPTCP can get a lower performance than using WiGig only, [36] explores MPTCP in 5G+LTE through simulations, and MuSher [24] explores dual WiFi 802.11 ac+ad through implementation. FBDT is implemented in Linux kernel, supports any number of RATs, and significantly outperforms MuSher when client frequently switches between LoS and nLoS channels.

MPTCP Schedulers: In addition to the schedulers discussed in Section II, several schedulers have been proposed in the community, including schedulers that try to: (ii) address the challenges associated with heterogeneous paths [23], [37], [38], [39], (ii) leverage the differences in subflow RTTs [23], [37], [39], [40], [41], [42], (iii) improve MPTCP performance for special use cases [43], [44], [45], and (iv) require modifications to the application [46]. FBDT can address multi-RAT scenarios with vastly different characteristics across RATs, has superior performance in static/mobile and LoS/nLoS scenarios, and does not require explicit information from the lower layers or modifications to the application.

Application class of traffic: Many studies [47], [48] have explored the performance of Interactive, Background, Conversational and streaming classes of traffic over WiFi and LTE. Other research work has proposed improving the performance of Interactive and Background classes of traffic classes over MPTCP schedulers (DAPS, ECF, OTIAS, LRF) as compared to SPTCP [49], [50], [51]. There is an extensive survey literature on the limitations of wireless streaming over MPTCP [52]. To overcome them, there have been proposed application oriented solutions for Streaming class of traffic (progressive downloads) over multipath TCP [53], [54], [55], [56]. FBDT proposes a common transport layer solution to overcome limitations posed by MPTCP for application class of traffic.

Multi-RAT VR: Several recent studies have explored the benefits of using multiple RATs to enhance mobile virtual reality systems and 360° video streaming performance, e.g., by: (i) using Raptor codes and developing Raptor coding adaptation [57], (ii) optimizing tile rate selection while leveraging MPTCP [58], [59], (iii) leveraging visible light communication (VLC) in addition to WiFi and optimizing the placement of VLC BSs [60], (iv) integrating millimeter wave and WiFi access points and optimizing the allocation of communication and computation resources in a mobile multi-user VR arena system [61], [62], and (v) integrating LTE and 5G access points and optimizing the allocation of communication and computation resources in a mobile edge multi-user VR streaming system [63]. In contrast to these studies, we (i) replace MPTCP with FBDT and (ii) build a real prototype and comprehensively evaluate the system's performance in LoS, nLoS, and mobility conditions.

Millimeter wave mobile VR: Most recently, a few studies have started exploring the feasibility of using exclusively millimeter wave networks for delivering high quality 360° video content to mobile VR headsets, in potential conjunction with edge computing and machine learning techniques [64], [65].

Our paper has been motivated by our preliminary work that appeared in [12] and focused on the scenario of two RATs only. Here, we go beyond this conference paper by: (1) Extending our framework to more than two RATs and showing that its throughput performance scales linearly with the number of RATs, in contrast to multi-path TCP, whose performance degrades with an increase in the number of RATs; (2) Evaluating the performance of FBDT on different application traffic classes and demonstrating: (i) 2-3 times shorter file download times, (ii) up to 10 times shorter streaming times and 10 dB higher video quality for progressive download video applications, and (iii) up to 9 dB higher viewport quality for interactive mobile VR applications, when our viewport maximization framework is employed along with FBDT.

VIII. Conclusion and Future Work

We introduced a new multi-RAT transport layer protocol named "FBDT" to address the underlying causes of MPTCP's poor performance and advance the state-of-the-art. We implemented our protocols on COTS hardware and conducted numerous experiments to evaluate their system performance in practice. We showed that FBDT provides a 2.5x throughput

gain against state-of-the-art MPTCP protocol when a mobile client routinely switches between LoS and nLoS conditions. We also showed that as compared to MPTCP, FBDT effectively aggregates traffic with more than two RATs and gets very close to the summation of the individual RAT data rates, irrespective of the channel conditions (client mobility conditions). Moreover, we demonstrated that FBDT provides 3x, 10x and 9 dB throughput and video quality gains for background, streaming and conversational traffic class, respectively.

As part of our future work, we plan to extend this work in several ways. First, in our existing evaluation we only use a single client device (with multiple RATs). This limitation is due to our access to only a few hardware devices with WiGig radios. In the multi-client and/or multi-BS scenarios, we expect the existing MAC protocols to handle medium access and interference in between the devices. Thus, FBDT should be able to provide similar performance gains against other solutions. We leave performance evaluation in these scenarios as part of our future work. Second, we plan to explore and compare the performance of FBDT against that of the QUIC protocol for different classes of application traffic. finally, we plan to complete the kernel implementation of FBDT and will prepare and submit respective Request For Comments (RFCs) technical contributions for review by the IETF standardization committee.

REFERENCES

[1] R. Siddavaatam, I. Woungang, G. H. S. Carvalho, and A. Anpalagan, "Mobile cloud storage over 5G: A mechanism design approach," *IEEE Syst. J.*, vol. 13, no. 4, pp. 4060–4071, Dec. 2019.

[2] J. Chakareski and P. Frossard, "Context-adaptive information flow allocation and media delivery in online social networks," *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 4, pp. 732–745, Aug. 2010.

[3] Metaverse, May 2025. [Online]. Available: https://en.wikipedia.org/wiki/Metaverse

[4] E. Guttman, "Study on localized mobile metaverse services," 3GPP 5G Tech. Rep. 22.856, Dec. 2023.

[5] J. Chakareski, M. Khan, and M. Yuksel, "Towards enabling next generation societal virtual reality applications for virtual human teleportation," *IEEE Signal Process. Mag.*, vol. 39, no. 5, pp. 22–41, Sep. 2022.

[6] GSMA white paper on cloud AR/VR, 2019. [Online]. Available: https://www.gsma.com/futurenetworks/wiki/cloud-ar-vr-whitepaper/

[7] Internet connection speed recommendations, May 2025. [Online]. Available: https://help.netflix.com/en/node/306

[8] Verizon 4G LTE speeds vs. your home network, 2023. [Online]. Available: https://www.verizon.com/articles/4g-lte-speeds-vs-your-home-network/

[9] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 187–198.

[10] A. Seam, A. Poll, R. Wright, J. Mueller, and F. Hoodbhoy, "Enabling mobile augmented and virtual reality with 5G networks," *AT&T White Paper*, 2017.

[11] 2025. [Online]. Available: https://github.com/earyafar/FBDT/tree/main/TMC

[12] S. Srinivasan, S. Shippey, E. Aryafar, and J. Chakareski, "FBDT: Forward and backward data transmission across RATs for high quality mobile 360-degree video VR streaming," in *Proc. 14th ACM Conf. Multimedia Syst.*, 2023, pp. 130–141.

[13] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch, "TCP extensions for multipath operation with multiple addresses," *Internet Eng. Task Force*, Mar. 2020.

[14] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development," *Internet Eng. Task Force*, Mar. 2011.

[15] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," *Internet Eng. Task Force*, Oct. 2011.

[16] M. Gapeyenko et al., "Spatially-consistent human body blockage modeling: A state generation procedure," *IEEE Trans. Mobile Comput.*, vol. 19, no. 9, pp. 2221–2233, Sep. 2020.

[17] C. G. Ruiz, A. Pascual-Iserte, and O. Munoz, "Analysis of blocking in mmWave cellular systems: Application to relay positioning," *IEEE Trans. Commun.*, vol. 69, no. 2, pp. 1329–1342, Feb. 2021.

[18] M. K. Haider and E. W. Knightly, "Mobility resilience and overhead constrained adaptation in directional 60 GHz WLANs: Protocol design and system implementation," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 61–70.

[19] T. Nitsche, A. B. Flores, E. W. Knightly, and J. Widmer, "Steering with eyes closed: Mm-wave beam steering without in-band measurement," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 2416–2424.

[20] S. Sur, X. Zhang, P. Ramanathan, and R. Chandra, "BeamSpy: Enabling robust 60 GHz links under blockage," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 93–206.

[21] A. Zhou, X. Zhang, and H. Ma, "Beam-forecast: Facilitating mobile 60 GHz networks via model-driven beam steering," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[22] C. Raiciu et al., "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2012, Art. no. 29.

[23] S. Ferlin, Ã. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *Proc. IEEE IFIP Netw. Conf.*, 2016, pp. 431–439.

[24] S. K. Saha, S. Aggarwal, R. Pathak, D. Koutsonikolas, and J. Widmer, "MuSher: An agile multipath-TCP scheduler for DualBand 802.11 ad/ac wireless LANs," *IEEE/ACM Trans. Netw.*, vol. 30, no. 4, pp. 1879–1894, Aug. 2022.

[25] H. Cech, "Analyzing and realizing multipath TCP schedulers in Linux," Tech. Univ. Munich, Aug. 2020.

[26] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. London, U.K.: Pearson, 2016.

[27] R. Khalili, N. Gast, M. Popovic, and J. L. Boudec, "MPTCP is not pareto-optimal: Performance issues and a possible solution," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1651–1665, Oct. 2013.

[28] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Design, analsysis and implementation," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 596–609, Feb. 2016.

[29] Y. Chen, R. J. G. Y. Lim, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of MultiPath TCP performance over wireless networks," in *Proc. Conf. Internet Meas. Conf.*, 2013, pp. 455–468.

[30] Q. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A first analysis of multipath TCP on smartphones," in *Proc. Passive Act. Meas. Conf.*, 2016, pp. 57–69.

[31] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "WiFi, LTE, or both? Measuring multi-homed wireless internet performance," in *Proc. Conf. Internet Meas. Conf.*, 2016, pp. 181–194.

[32] A. Nikravesh, Y. Guo, F. Qian, Z. Mao, and S. Sen, "An in-depth understanding of multipath TCP on mobile devices: Measurement and system design," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 189–201.

[33] Y. Lim, Y. Chen, E. M. Nahum, D. Towsley, and K. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 1815–1823.

[34] K. Nguyen, M. Kibria, K. Ishiz, and F. Kojima, "Feasibility study of providing backward compatibility with MPTCP to WiGig/IEEE 802.11ad," in *Proc. IEEE 86th Veh. Technol. Conf.*, 2017, pp. 1–5.

[35] S. Sur, I. Pefkianakis, X. Zhang, and K. Kim, "WiFi-assisted 60 GHz wireless networks," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 28–41.

[36] M. Polese, R. Jana, and M. Zorzi, "TCP in 5G mmWave networks: Link level retransmissions and MP-TCP," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2017, pp. 343–348.

[37] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay aware packet scheduling for multipath transport," in *Proc. Int. Conf. Commun.*, 2014, pp. 1222–1227.

[38] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju, "QAware: A cross-layer approach to MPTCP scheduling," in *Proc. IFIP Netw. Conf. Workshop*, 2018, pp. 1–9.

[39] Y. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol.*, 2017, pp. 147–159.

[40] S. Baidya and R. Prakash, "Improving the performance of multipath TCP over heterogeneous paths using slow path adaptation," in *Proc. IEEE Int. Conf. Commun.*, 2014, pp. 3222–3227.
[41] J. Hwang and J. Yoo, "Packet scheduling for multipath TCP," in *Proc. 7th Int. Conf. Ubiquitous Future Netw.*, 2015, pp. 177–179.
[42] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu, "OCPS: Offset compensation based packet scheduling mechanism for multipath TCP," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 6187–6192.
[43] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer scheduler for video streaming over MPTCP," in *Proc. 7th Int. Conf. Multimedia Syst.*, 2017, Art. no. 7.
[44] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "MP-DASH: Adaptive video streaming over preference-aware multipath," in *Proc. 12th Int. Conf. Emerg. Netw. Experiments Technol.*, 2016, pp. 129–143.
[45] A. Nikravesh, Y. Guo, X. Zhu, F. Qian, and Z. Mao, "MP-H2: A client-only multipath solution for HTTP/2," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2019, Art. no. 10.
[46] Y. Guo, A. Nikravesh, Z. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 141–153.
[47] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "Observing real smartphone applications over multipath TCP," *IEEE Commun. Mag.*, vol. 54, no. 3, pp. 88–93, Mar. 2016.
[48] V. Adarsh, P. Schmitt, and E. Belding, "MPTCP performance over heterogenous subpaths," in *Proc. 28th Int. Conf. Comput. Commun. Netw.*, 2019, pp. 1–9.
[49] Y.-S. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Improving energy efficiency of MPTCP for mobile devices," 2014, *arXiv:1406.4463*.
[50] A. Nikravesh, Y. Guo, F. Qian, Z. M. Mao, and S. Sen, "An in-depth understanding of multipath TCP on mobile devices: Measurement and system design," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 189–201.
[51] P. Hurtig, K.-J. Grinnemo, A. Brunstrom, S. Ferlin, Ö. Alay, and N. Kuhn, "Low-latency scheduling in MPTCP," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 302–315, Feb. 2019.
[52] S. Afzal, V. Testoni, C. E. Rothenberg, P. Kolan, and I. Bouazizi, "A holistic survey of multipath wireless video streaming," *J. Netw. Comput. Appl.*, vol. 212, 2023, Art. no. 103581.
[53] J. Wu, C. Yuen, B. Cheng, Y. Yang, M. Wang, and J. Chen, "Bandwidth-efficient multipath transport protocol for quality-guaranteed real-time video over heterogeneous wireless networks," *IEEE Trans. Commun.*, vol. 64, no. 6, pp. 2477–2493, Jun. 2016.
[54] T. Hiraoka and J. Funasaka, "A progressive download method using multiple TCP flows on multiple paths," in *Proc. 10th Int. Conf. Broadband Wireless Comput. Commun. Appl.*, 2015, pp. 318–324.
[55] A. Elgabli and V. Aggarwal, "SmartStreamer: Preference-aware multipath video streaming over MPTCP," *IEEE Trans. Veh. Technol*, vol. 68, no. 7, pp. 6975–6984, Jul. 2019.
[56] J. Funasaka, "Evaluation on progressive download methods based on timer-driven requesting schemes on multiple paths with shared links," in *Proc. 8th Int. Symp. Comput. Netw. Workshops*, 2020, pp. 14–20.
[57] J. Wu, B. Cheng, and M. Wang, "Improving multipath video transmission with raptor codes in heterogeneous wireless networks," *IEEE Trans. Multimedia*, vol. 20, no. 2, pp. 457–472, Feb. 2018.
[58] W. Wei, J. Han, Y. Xing, K. Xue, J. Liu, and R. Zhuang, "MP-VR: An MPTCP-based adaptive streaming framework for 360-degree virtual reality videos," in *Proc. IEEE Int. Conf. Commun.*, 2021, pp. 1–6.
[59] A. Ravichandran, I. Jain, R. Hegazy, T. Wei, and D. Bharadia, "Poster: Facilitating low latency and reliable VR over heterogeneous wireless networks," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 723–725.
[60] J. Chakareski and M. Khan, "WiFi-VLC dual connecitivty streaming system for 6DOF multi-user virtual reality," in *Proc. 31st ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, 2021, pp. 106–113.
[61] J. Chakareski, M. Khan, T. Ropitault, and S. Blandino, "Millimeter wave and free-space-optics for future dual-connectivity 6DOF mobile multi-user VR streaming," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 19, 2022, Art. no. 57.
[62] S. Gupta, J. Chakareski, and P. Popovski, "mmWave networking and edge computing for scalable 360-degree video multi-user virtual reality," *IEEE Trans. Image Process.*, vol. 32, pp. 377–391, 2023.
[63] J. Chakareski and M. Khan, "Live 360° Video Streaming to Heterogeneous Clients in 5G Networks," *IEEE Trans. Multimedia*, vol. 26, pp. 8860–8873, 2024.
[64] S. Shippey, S. Srinivasan, H. P. Dang, E. Aryafar, and J. Chakareski, "An experimental evaluation of 360-degree ABR video streaming over mmWave wireless links," in *Proc. IEEE Int. Conf. Metaverse Comput. Netw. Appl.*, 2024, pp. 113–120.
[65] B. Badnava, J. Chakareski, and M. Hashemi, "Joint communication and computation resource allocation for emerging mmWave multi-user 3D video streaming systems," in *Proc. IEEE Glob. Commun. Conf.*, 2024, pp. 1821–1826.

Suresh Srinivasan received the BEng degree from the Department of Electronics and communication Engineering, Bangalore University, India, in 1998, and the MS degree from the Department of Computer Science, Southern Methodist University, Dallas, Texas, in 2006, and the PhD degree from the Department of Computer Science, Portland State University, Portland, Oregon, in 2024. He has more than 20 years of work experience in wireless and embedded systems and he currently works with Intel Corporation. His research interests are include advanced 5G/6G wireless communication systems, edge computing, and wireless/distributed machine learning. He has more than 14 patents issued and pending in the area of mobile wireless and systems.

Sam Shippey completed the BSc degree in computer science from Portland State University, in 2020, and is currently working toward the PhD degree. His research interests include next generation wireless networks, video streaming for virtual reality, and network measurement and performance modeling.

Ehsan Aryafar received the BS degree in electrical engineering from the Sharif University of Technology, Iran, in 2005, and the MS and PhD degrees in electrical and computer engineering from Rice University, Houston, Texas, in 2007 and 2011, respectively. He is the David E. Wedge Vision associate professor of computer science with Portland State University (PSU). Prior to his employment at PSU and from 2013 to 2017, he was a research scientist with Intel Labs in Santa Clara, CA. From 2011 to 2013, he was a post-doctoral research associate with the Department of Electrical Engineering, Princeton University. His research interests include the areas of wireless networks and networked systems, and span both algorithm design as well as system prototyping. He is a recipient of the 2020 NSF CAREER award and served as an associate editor of *IEEE Transaction on Mobile Computing* from 2019 to 2024.

Jacob Chakareski received the PhD degree in electrical and computer engineering from Rice and Stanford. He is an associate professor with the College of Computing, New Jersey Institute for Technology (NJIT), where he holds the Panasonic chair of Sustainability and directs the Laboratory for Future AI-Enabled Wireless XR Network Systems and Integrated Societal Applications. His research interests span next generation virtual and augmented reality systems, UAV IoT sensing and networking, fast reinforcement learning, 5 G wireless edge computing and caching, ubiquitous immersive communication, and societal applications. He received the Adobe Data Science Faculty Research Award in 2017 and 2018, the Swiss NSF Career Award Ambizione (2009), the AFOSR Faculty Fellowship in 2016 and 2017, and Best Paper Awards at ICC 2017 and MMSys 2021. He held research appointments with Microsoft, HP Labs, EPFL, and Vidyo, Inc, and served on the advisory board of Frame, Inc. His research has been supported by the NSF, NIH, AFOSR, Adobe, Tencent Research, NVIDIA, and Microsoft.