# FBDT: <u>F</u>orward and <u>B</u>ackward <u>D</u>ata <u>T</u>ransmission Across RATs for High Quality Mobile 360-Degree Video VR Streaming

## ABSTRACT

The metaverse encompasses many virtual universes and relies on streaming high-quality 360° videos to VR/AR headsets. This type of video transmission requires very high data rates to meet the desired Quality of Experience (QoE) for all clients. Simultaneous data transmission across multiple Radio Access Technologies (RATs) such as WiFi and WiGig is a key solution to meet this required capacity demand. However, existing transport layer multi-RAT traffic aggregation schemes suffer from Head-of-Line (HoL) blocking and sub-optimal traffic splitting across the RATs, particularly when there is a high fluctuation in their channel conditions. As a result, state-of-the-art multi-path TCP (MPTCP) solutions can achieve aggregate transmission data rates that are lower than that of using only a single WiFi RAT in many practical settings, e.g., when the client is mobile. We make two key contributions to enable high quality mobile 360° video VR streaming using multiple RATs. First, we propose the design of FBDT, a novel multi-path transport layer solution that can achieve the *sum* of individual transmission rates across the RATs despite their system dynamics. We implemented FBDT in the Linux kernel and showed substantial improvement in transmission throughput relative to state-of-the-art schemes, e.g, 2.5x gain in a dual-RAT scenario (WiFi and WiGig) when the VR client is mobile. Second, we formulate an optimization problem to maximize a mobile VR client's viewport quality by taking into account statistical models of how clients explore the 360° look-around panorama and the transmission data rate of each RAT. We explore an iterative method to solve this problem and evaluate its performance through measurement-driven simulations leveraging our testbed. We show up to 12 dB increase in viewport quality when our optimization framework is employed.

**ACM Reference Format:**
. 2023. FBDT: <u>F</u>orward and <u>B</u>ackward <u>D</u>ata <u>T</u>ransmission Across RATs for High Quality Mobile 360-Degree Video VR Streaming. In *Proceedings of MMSys Conference (MMSys'23).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

The emerging and highly anticipated concept of the *metaverse* is generally defined as a network of 3D virtual worlds, facilitated by the use of virtual reality (VR) and augmented reality (AR) headsets, and has emphasis on social interaction and diverse future applications [1]. The metaverse and VR/AR applications in general have been recently identified by 3GPP as key emerging application settings for 5G+/6G technologies [2]. 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 [3]. However, the Quality of Experience (QoE) of a mobile VR client can significantly drop in existing wireless networks, as high-quality VR applications require ultra-low latency and ultra-high data rate [4], which is challenging to enable.

In particular, according to several recent reports [5, 6], the required wireless data rates are about 15 Mbps for a 4K 30fps H.264-encoded 360-degree video stream, 800 Mbps for an 8K H.266 encoded stream, and up to a few Gbps for higher resolutions and frame rates [3]. Moreover, MPEG recommends a minimum of 12K high-pixel-quality spatial resolution and 100 frames per second temporal display rate, for a 360° video panorama experienced by a VR user [7]. These requirements would map to a data rate of several Gbps, even after applying state-of-the-art High Efficiency Video Coding (HEVC) compression [8]. Existing RAT technologies cannot consistently provide high downlink data rates. For example, LTE speeds are typically around 10 Mbps [9], which is far lower than the required data rates. Recent sub-6 GHz (11 ac/ax) and 60 GHz (11 ad/ay) WiFi[1] 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. This susceptibility to blockages and mobility is also prevalent in existing cellular millimeter wave (mmWave) solutions (e.g., mmWave 5G), which creates large fluctuations in throughput. Second, the high variability in wireless network bandwidth can cause unexpected re-buffering, which drastically reduces the client QoE [10]. This reduced QoE due to either low quality video frames or re-buffering can also lead to client disorientation or nausea [11].

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 high minimum data rates 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 HoL blocking and sub-optimal traffic splitting across the RATs, particularly in the presence of system dynamics such as when the channel conditions frequently change 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 an 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. First, we introduce <u>FBDT</u>, a <u>F</u>roward and <u>B</u>ackward <u>D</u>ata <u>T</u>ransmission protocol at the transport

[1]60 GHz (mmWave) WiFi is also known as WiGig, 802.11 ad or ay.

*MMSys'23, June 2023, Vancouver, Canada*
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

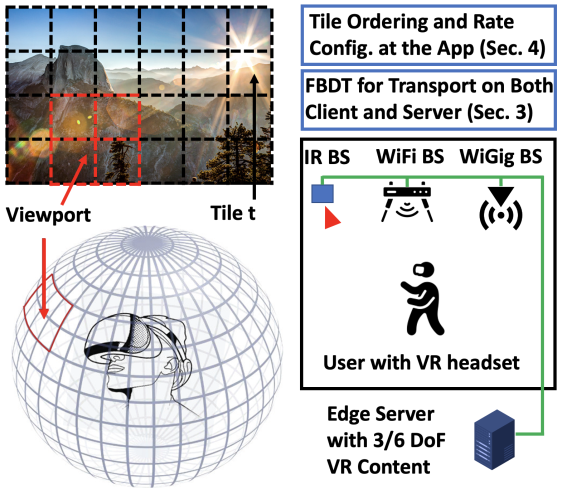
Tile Ordering and Rate Config. at the App (Sec. 4)
FBDT for Transport on Both Client and Server (Sec. 3)
IR BS
WiFi BS
WiGig BS
User with VR headset
Edge Server with 3/6 DoF VR Content
Viewport
Tile t

**Figure 1: Left: Client with a VR headset looking at a 360° video frame. Each frame is composed of tiles. Our goal is to maximize the client viewport quality leveraging empirical data on tile importance and historical data on RAT data rates. Right: An edge server is connected to different Base Stations (BSs) including WiFi, WiGig, and Infrared (IR). IR is used for client localization and mapping. FBDT is implemented at the transport layer on both the client and server. The application leverages rate-distortion optimization to determine the ordering and compression rates for all tiles.**

layer to effectively eliminate HoL and sub-optimal traffic splitting problems in all channel conditions. Second, we consider a setup (Fig. 1 shows our considered setup) with 360° tiling of video frames and IR/IMU tracking equipment installed in the arena. We then pose a rate-distortion based optimization problem formulation to determine the ordering and encoding rate for every tile such that the viewport quality of the client is maximized, while taking into account statistical models on how clients explore the 360° look-around panorama and historical data rates of each RAT. 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 achieves this goal by (i) removing retransmission schemes employed by MPTCP and relying on individual Single-Path TCP (SPTCP) mechanisms for reliable delivery, (ii) optimally placing the pointers to fetch traffic for each RAT by taking into account the historical reliability of each RAT, and (iii) serving 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.
- **Viewport Optimization:** We formulate an optimization problem to maximize the viewport quality (and hence, client QoE), show that the problem is non-convex, and then design an iterative method with controllable computation time to solve it at the application layer.
- **Open-Source Implementation:** We have implemented FBDT in Linux kernel on both client and server side. This allowed us to implement our methods in Commercial-Off-The-Shelf (COTS) hardware (routers, laptops) and evaluate its performance in real-world settings. Prior to conference publication, we will open-source the software and publicly release it on GitHub 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 client QoE. We show that FBDT can achieve summation of individual data rates in all channel conditions including when the client is in LoS, nLoS, or mobile. We also show that compared to the 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 RAT. Finally, we show that our tile ordering and compression rate optimization method provides an average of 12 dB increase in PSNR across a wide variety of 360° videos.

The paper is organized as follows. We discuss the background in Section 2 and the design of FBDT in Section 3. We pose our rate-distortion optimization problem and the method to solve it in Section 4. We discuss the details of our implementation in Section 5 and the results of our evaluations in Section 6. Finally, we discuss the related work in Section 7 and conclude the paper in Section 8.

## 2 BACKGROUND AND MOTIVATION

### 2.1 Preliminaries

**MPTCP Architecture.** MPTCP [12–14] 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 at the meta-level are placed in an out-of-order (OOO) queue. 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 to the application. As a result, packets scheduled on the low-delay path might have to wait

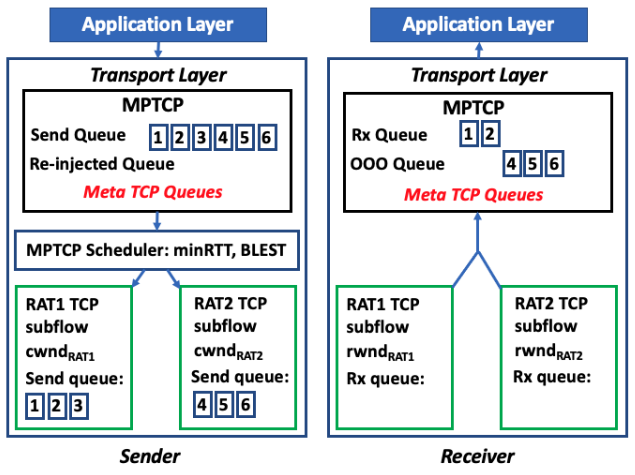
Application Layer
Transport Layer
MPTCP
Send Queue 1 2 3 4 5 6
Re-injected Queue
Meta TCP Queues
MPTCP Scheduler: minRTT, BLEST
RAT1 TCP subflow cwnd_RAT1 Send queue: 1 2 3
RAT2 TCP subflow cwnd_RAT2 Send queue: 4 5 6
Sender
Application Layer
Transport Layer
MPTCP
Rx Queue 1 2
OOO Queue 4 5 6
Meta TCP Queues
RAT1 TCP subflow rwnd_RAT1 Rx queue:
RAT2 TCP subflow rwnd_RAT2 Rx queue:
Receiver

**Figure 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 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 such as susceptibility to blockages [15, 16]. For example, human body alone can block the path between mmWave sender and receiver, suddenly bringing 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., [17–20]) 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 7). 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 [21]. This can result in under-utilization of paths with burstiness in their rate, leading to suboptimal capacity aggregation. BLEST [22] 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 [23] is a recently developed MPTCP scheduler that is designed and evaluated for 802.11 ac+ad dual WiFi (i.e., WiFi+WiGig) 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 [24]. We will additionally show in Section 6 that MuSher can get very low performance when the client frequently switches between LoS and nLoS channels.

## 2.2 Motivation

We conducted preliminary experiments to demonstrate the poor performance of MPTCP in presence of link quality variations and its impact on the delivered 360° video viewport quality. The experimental 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 the 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 a Traffic Controller (TC) between the BSs and the server. The 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 1sec. Each experiment lasts for 5 minutes, and 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: (1 set) when there is no loss/delay introduced by TC (Normal), (2 sets) when TC introduces a 5% packet loss on either WiGig or WiFi links, and (2 sets) when TC increases delay between packets of 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,

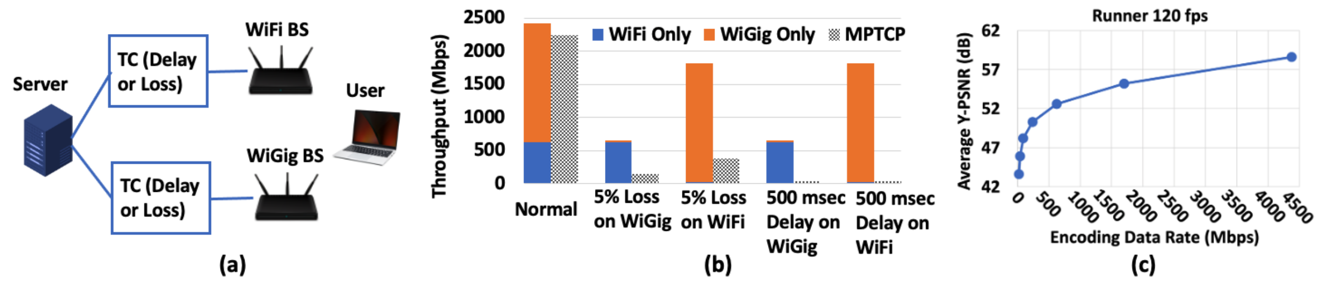

**Figure 3: (a): We conduct experiments leveraging two Netgear Nighthawk X10 routers, a Dell Server and an Acer TravelMate laptop; (b): Throughput comparison of MPTCP's default scheduler (BLEST) and individual SPTCP throughput under ideal, lossy, and delay induced scenarios. Similar drop in performance was observed with minRTT scheduler; (c): Trade-off between delivered 360° video quality and required minimum transmission data rate.**

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.

**Viewport Quality.** We next examine the implications of available transmission bandwidth on the delivered 360° viewport quality. In Fig. 3(c), we examine the expected viewport quality (captured via its PSNR in dB) experienced by a VR client, as a function of the minimum required transmission data rate (or bandwidth). The viewport quality is measured for the popular 8K 120fps 360° video *Runner* [25]. We can observe from Fig. 3(c) that a drop in data rate when MPTCP is used (e.g., when there is loss or delay on either of the RATs) can lead to a major loss in viewport quality as well. Since MPTCP and FBDT both assume the presence of a connection which becomes unreliable under practical scenarios (such as mobility), we want to highlight this fact as it presents a severe challenge to providing the smooth high quality of experience that users demand of next generation virtual reality.

## 3 FBDT DESIGN

In this section, we provide details of FBDT design. First, we describe some of the high level ideas behind the design of FBDT and discuss how it can eliminate HoL blocking and provide a throughput that is close to the summation of throughput values 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 to support N RATs (i.e., more than two) as well as a detailed pseudo-code of FBDT in the Appendix.

### 3.1 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 transmissiom 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 channel. 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

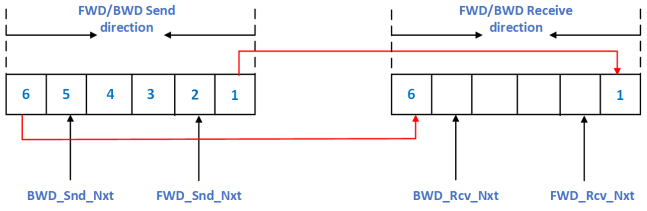

**Figure 4: Left: Transmit window at the sender. Right: Receive window at the receiver. Each window accommodates 6 packets. FBDT sender transmits packets from both forward and backward directions. The pointers showing the next forward and backward packets to be served are also illustrated.**

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:

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

Here $RTT_{fwd}$ and $W_{fwd}$ denote the forward path (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.

## 3.2 Detailed Architecture

We now discuss the details of different components of 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 and fix its size to 1MB in our experiments. 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.

**Sequence Numbers.** To ensure in-order data delivery, FBDT assigns unique Sequence numbers to the data segments to be sent by SPTCPs. These unique sequence numbers are also used by the receiver to discard duplicate packets as they are delivered to the receive meta socket.

**Acknowledgements.** ACK(s) are necessary in FBDT 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 in the presence of unreliability. 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 is 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 wait for at most RTT of the other transmission end, and then would proceed to transmitting unacknowledged packets in the other sliding window. This could be redundant information transmission but is required to ensure a high overall system throughput. As packets are checked based on their sequence numbers at the receiver, they will be dropped if a duplication occurs at the receiver. 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 an $RTT_{bwd}$ before transmitting packets 4 and 5 over its SPTCP. After transmitting this redundant data, the forward or backward will not wait for the ACK(s) any longer since the data should be reached either by forward or backward, which ensures in-order delivery as well. As a result, the data in the transmission window is updated with the next set of ten packets to be transmitted from the meta-socket buffer[2].

**RAT to Direction Mapping.** The mapping of RATs to forward or backward directions impacts the performance. In FBDT, the forward direction is mapped to the more reliable RAT (e.g., WiFi) and the backward direction is mapped to the less reliable RAT (e.g., WiGig). Suppose the reverse mapping and packets 1 to 6 in the transmit window size. If WiGig is assigned to the forward direction but is blocked, packet 1 would take a long time until it's reached at the receiver. This also blocks delivery of packets 6, 5, and 4 (which are sent on WiFi) to the receive meta socket and application (since packets need to be delivered in-order to the higher layer). Assigning the more reliable RAT to the forward direction removes this type of blocking, which can be particularly important if initial packets have a higher priority (e.g., importance) than later packets in the transmission window. FBDT can also dynamically adapt this assignment based on historical RATs' performance.

**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[3] and Snd_Nxt pointers for both forward and backward sliding windows (see Fig. 5). FBDT schedules packets to SPTCPs depending on 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 (FWD and BWD) congestion control algorithms (CCAs) and lets each CCA to decide on its transmission based on the congestion and reliability of its underlying network.

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.

[2]It is possible that both SPTCPs fail to deliver packets 4 and 5. In this case, FBDT will inform the application layer, which would result in retransmission or another mitigation scheme by the application.

[3]Una stands for Un-acknowledged.

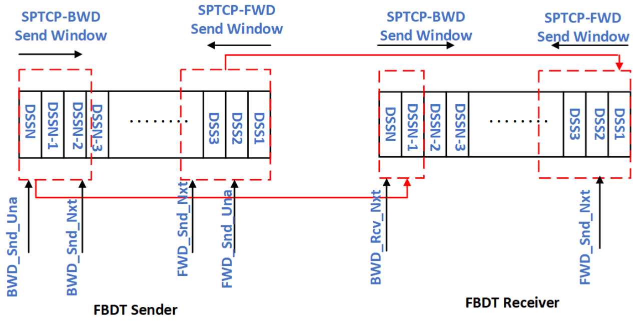

**Figure 5: FBDT Scheduler**

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:

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

Here, $\gamma$ 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, $\gamma$ is very close to one.

**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 [14, 27, 28].

IETF study on MPTCP suggests three goals for a practical coupled CCA [14]: (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[4].

**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

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

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 $\alpha$ and $\beta$ to scale up the forward and backward send window sizes, respectively. By controlling $\alpha$ and $\beta$ 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, ensuring a balanced congestion.

## 3.3 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.

## 4 360° VIDEO RATE-DISTORTION PACKET SCHEDULING AND RATE ALLOCATION

In this section, we propose a rate-distortion (R-D) optimization framework to maximize the client viewport quality. We consider the setup depicted in Fig. 1, in which an edge server is connected to different BSs, which may use different access technologies. The 360° video is streamed from the server and the application at the server has statistical models on how clients explore different 360° videos. The video is streamed to the client leveraging a multi-RAT transport layer protocol such as MPTCP or FBDT.

Videos 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. For example, the 360° video dataset that we will later use in our performance evaluation (Section 6) uses 30 frames per GoP universally. In 360°videos, each video frame 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.

## 4.1 Maximizing the Viewport Quality

We aim to maximize the viewport quality of the delivered content across all clients, given statistical models on how clients explore the 360° video look-around panorama as well as historical data on the average throughput of each RAT. For ease of presentation, we assume only one client.

We use $C$ to denote the set of $K$ RATs and $C_k$ as a random variable representing the data rate of the $k^{th}$ RAT. For a given GoP, let $T$ denote the set of tiles $t$, $N$ the total number of tiles, $\pi$ an ordering of the tiles, and $R(t)$ as the compression parameter of tile $t$ . We also use $B_{R(t)}$ to denote the size of the compressed tile $t$ in bits.

Maximizing the viewport quality is equivalent to minimizing the respective viewport distortion (or reconstruction loss), since there is a one-to-one mapping between the two objectives [3, 29]. Let $L_\pi(t)$ denote the observation-weighted reconstruction loss of tile $t$. $L_\pi(t)$ is the distortion of the compressed tile weighted by the likelihood that the tile will be observed and the chance that it will miss its deadline under ordering $\pi$. We formally capture $L_\pi(t)$ as:

$$L_\pi(t) = \epsilon_\pi(t, d_t)P(t)D(R(t)) \tag{3}$$

Here, $P(t)$ is the probability that a tile $t$ will be observed by the client (which can be easily derived since we have statistical models, e.g., probability density functions, on how clients view 360° videos), $d_t$ the deadline of tile $t$ in seconds (a tile received after its deadline would not be shown to the client), $D(R(t))$ is the distortion for tile $t$ at compression rate $R(t)$, and $\epsilon_\pi(t, d_t)$ is the probability that a tile under ordering $\pi$ will miss its deadline $d_t$. We can easily derive $\epsilon_\pi(t, d_t)$ using historical throughput statistics for each RAT.

The viewport maximization (distortion minimization) problem can then be posed as minimization of the reconstruction loss across all tiles subject to a capacity constraint:

$$\begin{aligned} \mathcal{P}_1: \quad & \min_{\pi,R} \frac{1}{N} \sum_{t\in T} L_\pi(t) \\ s.t. \quad & \sum_{t\in T} B_{R(t)} \le C_{\text{agg}} \end{aligned}$$

The output of the optimization problem are the ordering of all tiles $\pi$ and the compression rate $R(t)$ for each tile $t$. Note that we assume a 1 second GoP length, and hence the sum of the size of all tiles should be less than the aggregate throughput, which is posed as a constraint in problem $\mathcal{P}_1$. The above optimization problem is non-convex as it represents a mixed-integer programming, depending on discrete and continuous variables at the same time. In the next section, we propose an approximation algorithm to solve the problem.

## 4.2 Observation Weighted Sorting (OWS)

We pursue a solution strategy called *Observation Weighted Sorting (OWS)*, which orders the tiles such that if parts of a frame are dropped (e.g. due to a connection changing unexpectedly in the middle of sending a GoP), the impacted frames can still be assembled without losing too much quality.

To address the non-convexity of the problem, we adopt a decomposition approach that divides the problem into two sub-problems in a manner that allows the two to reinforce each other. For a given ordering of tiles in a GoP, we first compresses the tiles based on their likelihoods of being observed as part of the client viewport and the expected aggregate transmission rate. We then find an ordering of the tiles by sorting them according to a loss function, which incorporates both the likelihood that the client will observe the tile as well as the likelihood that the rate will be too low to send the tile before the deadline. By iterating over the two sub-problems for a given maximum number of iterations, we arrive at an ordering of the tiles and a compression rate for each tile.

**Algorithm 1** Observation Weighted Sorting (OWS)

```
1: Inputs: Tiles $T$, Rate data $C$, $M$, Observation
           likelihood for tile $t$ $P(t)$
2: Output: $\pi$ and $R(t)$, $\forall t \in T$
3: Initialization: Default($t$)
4: for m = 1 to $M$ do
5:     Rates $\leftarrow \min_R \sum_{t \in T} P(t)D(R(t))$
                 s.t. $B_{R(t)} \leq C_{agg}$
6:     Losses $\leftarrow \epsilon(t, d_t)P(t)D(R(t))$ for $t \in T$
7:     $\pi \leftarrow \texttt{sort}(\texttt{zip}(T, Losses), key = Losses)$
8: end for
9: return $\pi$, $R(t)$
```

Algorithm 1 summarizes the key steps in OWS. The algorithm initially orders the tiles based on their default placements in the frames (Line 3). Next, we derive the compression rates for all tiles (Line 5). Note that the objective function here does not have the $\epsilon_{\pi}(t, d_t)$ parameter from the definition of loss function in Eq. (3). This turns the problem into a convex problem, which can be solved optimally and in a fast manner through conventional solvers such as CVX. Next, we compute the loss $L_{\pi}(t)$ of each tile (Line 6) and sort the entire tiles in the GoP in ascending order by their loss (Line 7). The algorithm iterates over the two solutions for a fixed given number of iterations $M$ and the best overall ordering and compression is ultimately selected as the output of OWS.

## 5 IMPLEMENTATION

We implemented FBDT as a daemon in the user space in Linux on 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. 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 more reliable RAT - WiFi- as forward transmission and the less reliable RAT - WiGig- as backward transmission. FBDT transmit window on the server maintains FWD pointer and BWD pointer for both forward and backward transmissions. Both 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 backward ACKs. When both FWD and BWD pointers meet at some point within the transmission window, the transmission window is reloaded with a new set of data to be transmitted. FBDT receive window on the client also maintains a FWD and BWD pointer, and they move inward as they receive packets from the server.

**Migrating FBDT to the Linux Kernel.** We have migrated FBDT 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 plan to publicly release our software on GitHub prior to paper publication so that other researchers in the community can reproduce our results and expand on our software.

## 6 PERFORMANCE EVALUATION

In this section, we evaluate the performance of our system through a mixture of experiments and measurement-driven simulations. We first conduct over-the-air experiments to quantify the throughput that can be achieved with different MPTCP protocols and under different channel and client mobility conditions. We consider a dual WiFi+WiGig setup with only a single client. Next, we conduct simulations to study the viewport quality under different MPTCP protocols and tile ordering and coding schemes. We leave performance evaluation in multi-client scenarios as part of our future work.

### 6.1 Experimental Setup

**Network Deployment.** Fig. 6(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 2. Our client device is an Acer TravelMate laptop that has both WiFi and WiGig wireless 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. 6(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 6ft by 2ft, as depicted in Fig. 6(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 clinet frequently switches between LoS and nLoS channel conditions.

**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 values sampled over one second intervals).

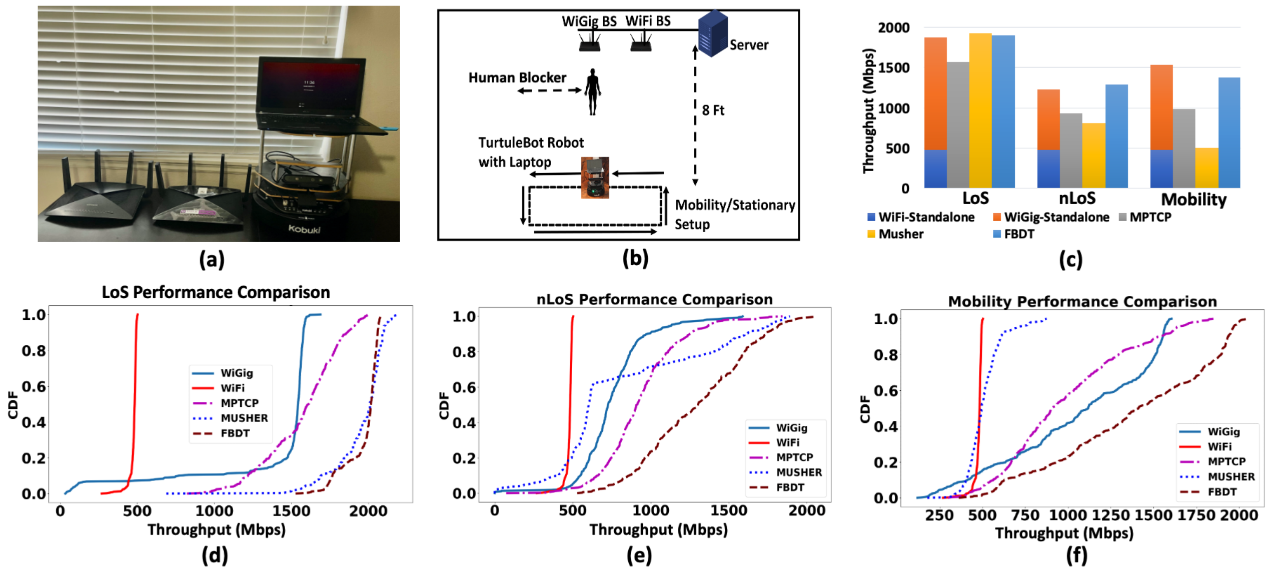

**Figure 6: (a): Our hardware setup. TravelMate laptop is placed on top of TurtleBot robot for controlled mobility,(b): Network deployment layout. A human stands in front of the WiGig BS to create blockage, (c): Average throughput results across stanalone WiFi, standalone WiGig, MPTCP (BLEST), MuSher, and FBDT in LoS, nLoS, and mobility, (d): LoS CDFs, (e): nLoS CDFs, (f): Mobility CDFs.**

**Implemented Solutions.** We experimented with the following protocols: (i) FBDT: our proposed architecture, (ii) MuSher [23], which is the state-of-the-art MPTCP scheduler designed for dual radio WiGig+WiFi (i.e., 802.11 ad+ac) 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 2, 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).

## 6.2 Throughput Statistics

**Throughput in Dual Radio WiFi+WiGig Setup.** Fig. 6(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 470 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 1920 Mbps. This is 20% higher than MPTCP. The CDF of the throughput variations in the LoS condition is plotted in Fig. 6(d). The throughput of MPTCP scheduler varies from 1000 Mbps to 1800 Mpbs, whereas both FBDT and MuSher achieve a minimum of 1700 Mbps 85% of the time.

The second group of bars in Fig. 6(c) show the throughput values 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 930Mbps, which is lower than its LoS throughput and is due to drop in WiGig performance. MuSher gets 814 Mbps throughput, which is even lower than baseline MPTCP. On the other hand, FBDT provides additive throughput gains with an average of 1300 Mbps. The corresponding nLoS CDF curves are plotted in Fig. 6(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. 6(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. 6(f)) and a higher average throughput (Fig. 6(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

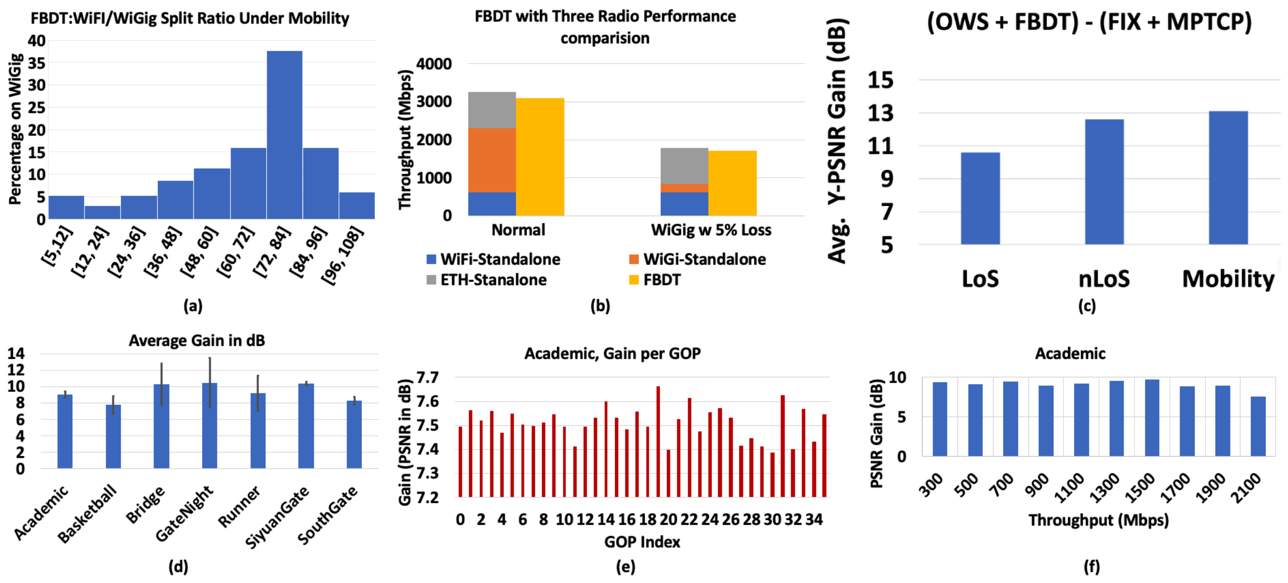

**Figure 7: (a): Histogram of traffic share passed through WiGig under mobility and when using FBDT. The results show that FBDT uses WiGig to pass majority of the traffic,(b): FBDT average throughput when using three RATs: WiFi, WiGig, and Ethernet (ETH). FBDT gets very close to summation of throughput across individual RATs even when a 5loss is introduced to WiGig packets, (c): OWS+FBDT viewport quality gain vs. FIX+MPTCP measured across all videos and in LoS/nLoS/Mobility channel conditions, (d): OWS average gain over FIX averaged over all GoPs for a selection of videos, (e): OWS gain over FIX per GoP for one video, (f): OWS gain over FIX across a variety of different achieved rates.**

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 [23]. Second, our mobility pattern is 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 conditions as faced in real-world environments[5]. 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.

**Traffic Split Ratio Under Mobility**. Fig. 7(a) denotes the histogram of FBDT traffic split across WiFi and WiGig. 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 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.

[5]Note that even with a stationary client and BS, mobile blockers such as humans can frequently cause a change in channel condition between the client and BS from LoS to nLoS and vice versa.

**Evaluation with Three RATs.** Unlike many other protocols (e.g., MuSher, which only supports dual RAT setups), FBDT supports any number of RATs. Details of FBDT operation with N RATs are provided in the Appendix. We now investigate FBDT performance in a three RAT scenario. We choose wired Ethernet as our third RAT 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 2 to perform the experiments. Fig. 7(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.

## 6.3 Viewport Quality

**Setup.** We evaluated our tile ordering and transmission rate selection algorithm (OWS) through measurement-driven simulations using communication data rates achieved by our dual radio WiFi+WiGig setup discussed in the previous section. Leveraging the throughput rate traces on each RAT for every one second, we developed a simulator to test out different 360° video tile ordering and rate selection methods. We considered all the 15 videos from the publicly available 8K UHD video dataset [25] as well as the 3DoF head navigation data [30] to choose quality points (QPs) for each of the tiles of the video based on how likely they are to be observed, as part of the user's viewport. We assume that RAT data rates are available to the packet scheduling and rate allocation algorithm. In addition to OWS (detailed in Algorithm 1), we consider FIX, an alternative algorithm which performs the same transmission rate selection, but does not reorder the tiles at all. We have also conducted experiments using other, alternative tile-orderings (such as sorting them by how likely a viewer is to see them) but this did not result in any improvements and therefore have not included their results here.

**Viewport Quality.** Fig. 7(c) depicts the mean PSNR gain across all 360° videos for three simulated channel conditions: LoS, nLoS, and mobility. The PSNR gain is calculated as the difference in PSNR across two schemes: OWS on top of FBDT and FIX on top of MPTCP. We observe that even in the LoS scenario where the throughput gap between MPTCP and FBDT is low, there is still more than 10 dB gain in PSNR due to the superior rate and tile ordering selection of OWS compared to FIX. This relative gain further increases to 13 dB in nLoS/Mobility scenarios where there are larger throughput gaps between the two transport layer solutions.

Fig. 7(d) shows the spread of results over several videos in the dataset. The margins for each entry represent the variance over the different GoPs over that video. These further show that gain is consistent, but dependent on individual properties of the video. We observe that as average gain grows, so does variance. Fig. 7(e) shows a more detailed view of the gain over FIX over each GoP of one video. Finally, Fig. 7(f) shows the average gain over all GoPs over segmented data rates. We encoded every video as discussed above under 10 different achieved rates ranging from 300Mbps to 2100Mbps (the range of achievable rates for FBDT under observed channel conditions) and measured the gain over FIX. This performance curves drops off at the end because at higher rates, the fixed ordering scheme catches up to OWS. OWS does not change the actual encoding of the video, only the order it is sent in, and as channel conditions improve (especially in reliability) the relative advantage to this strategy decreases. However, even at these very high rates, OWS still maintains a considerable advantage over FIX, and this gain is never negative.

## 7 RELATED WORK

**MPTCP Evaluation.** Several works [31–35] have studied MPTCP performance but these works consider scenarios that consist of wired paths or wireless setups with only sub-6 GHz RATs. Other works have studied MPTCP performance in networks that use mmWave RATs, e.g., [36, 37] studied dual WLANs with 802.11 ac (sub-6 GHz)+ad (60 GHz) and show that MPTCP can get a lower performance than using only WiGig, [38] explores MPTCP in 5G+LTE through simulations, and MuSher [23] explores dual 802.11 ac+ad through implementation. FBDT is implemented in Linux, 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 2, several schedulers have been proposed, including schedulers that try to: (ii) address the challenges associated with heterogeneous paths [22, 39–41], (ii) leverage the differences in subflow RTTs [22, 39, 41–44], (iii) improve MPTCP performance for special use cases [45–47], and (iv) require modifications to the application [48]. FBDT supports multi-RAT scenarios with vastly different characteristics across RATs, has superior performance in mobile, LoS, or nLoS scenarios, and does not require explicit information from the lower layers or modifications to the application.

**360° Video.** Despite the popularity of 360° video, only low quality 360° videos are widely available. Published 360° video navigation data sets includes the works by [30, 49–51]. We used the public data set by [30] to generate the user look-around panorama. Other works have studied the R-D characteristics of 360° videos, including: (i) the quality-rate dependency of two-layer scalable encoding [52], (ii) R-D dependency of compressed 360° videos under diverse sphere-to-planar projection methods [53], and (iii) tradeoffs of tiled 360° video for end-to-end streaming [54]. Our work considers the joint operation of R-D with the underlying multi-RAT transport protocol and substantially improves the viewport quality.

**Multi-RAT VR.** Several recent works have studied the potential 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 [55], (ii) optimizing tile rate selection while leveraging MPTCP [56, 57], (iii) leveraging visible light communication (VLC) in addition to WiFi and optimizing the placement of VLC BSs [29], and (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 [58, 59]. In contrast, we (i) replace MPTCP with FBDT and (ii) build a real prototype, which proves high system performance in LoS, nLoS, and mobility conditions.

## 8 CONCLUSION

In this paper, we introduced a new multi-RAT transport layer protocol named "FBDT" to address the underlying causes of MPTCP's poor performance when used on top of high frequency wireless radios. We also developed an optimization framework (deployed on top of the transport layer) to maximize the client viewport quality by taking into account statistical models on how clients explore the 360° look-around panorama and data rates of each RAT. We implemented our protocols on COTS hardware and conducted numerous experiments to evaluate the system performance in practice. We showed that FBDT provides a 2.5x gain against state-of-the-art MPTCP protocol when a mobile client routinely switches between LoS and nLoS conditions. We also showed that our viewport optimization method provides an average of 12 dB increase in PSNR across a variety of UHD videos and client channel conditions.

## REFERENCES

[1] Metaverse. https://en.wikipedia.org/wiki/Metaverse.
[2] 3GPP. Study on localized mobile metaverse services. In *5G Release 19*, 2022.
[3] J. Chakareski, M. Khan, and M. Yuksel. Towards enabling next generation societal virtual reality applications for virtual human teleportation. *IEEE Signal Processing Magazine*, 39(5):22–41, September 2022.
[4] Bo Begole. Why the Internet pipes will burst when virtual reality takes off. Forbes Magazine, Feb. 2016.
[5] Gsma. https://www.gsma.com/futurenetworks/wiki/cloud-ar-vr-whitepaper/.
[6] Speed recommendations. https://help.netflix.com/en/node/306.
[7] M. Champel, T. Stockhammer, T. Fautier, E. Thomas, and R. Koenen. Quality requirements for VR. In *Proc. 116th MPEG Meeting of ISO/IEC/JTC1/SC29/WG11*, number MPEG 116/m39532, Chengdu, China, October 17–21, 2016.
[8] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circuits and Systems for Video Technology*, 22(12):1649–1668, December 2012.

[9] Verizon 4G LTE speeds vs. your home network. https://www.verizon.com/articles/4g-lte-speeds-vs-your-home-network/.

[10] 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 *Proceedings of ACM SIGCOMM*, 2014.

[11] A. Seam, A. Poll, R. Wright, J. Mueller, and F. Hoodbhoy. Enabling mobile augmented and virtual reality with 5G networks. In *AT&T white paper*, 2017.

[12] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, and C. Paasch. TCP extensions for multipath operation with multiple addresses. In *Internet Engineering Task Force (IETF)*, 2020.

[13] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural guidelines for multipath TCP development. In *Internet Engineering Task Force (IETF)*, 2011.

[14] C. Raiciu, M. Handley, and D. Wischik. Coupled congestion control for multipath transport protocols. In *Internet Engineering Task Force (IETF)*, 2011.

[15] M. Gapeyenko, A. Samuylov, M. Gerasimenko, D. Moltchanov, S. Singh, M. R. Akdeniz, E. Aryafar, S. Andreev, N. Himayat, and Y. Koucheryavy. Spatially-consistent human body blockage modeling: a state generation procedure. In *IEEE Transactions on Mobile Computing*, 2020.

[16] C. G. Ruiz, A. Pascual-Iserte, and O. Munoz. Analysis of blocking in mmwave cellular systems: Application to relay positioning. In *IEEE Transactions on Communications*, 2021.

[17] M. K. Haider and E. W. Knightly. Mobility resilience and overhead constrained adaptation in directional 60 GHz WLANs: Protocol design and system implementation. In *Proceedings of ACM MOBIHOC*, 2016.

[18] T. Nitsche, A. B. Flores, E. W. Knightly, and J. Widmer. Steering with eyes closed: mm-wave beam steering without in-band measurement. In *Proceedings of IEEE INFOCOM*, 2015.

[19] S. Sur, X. Zhang, P. Ramanathan, and R. Chandra. BeamSpy: Enabling robust 60 GHz links under blockage. In *Proceedings of USENIX NSDI*, 2016.

[20] A. Zhou, X. Zhang, and H. Ma. Beam-forecast: Facilitating mobile 60 GHz networks via model-driven beam steering. In *Proceedings of IEEE INFOCOM*, 2017.

[21] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of USENIX NSDI*, 2012.

[22] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli. BLEST: blocking estimation-based MPTCP scheduler for heterogeneous networks. In *Proceedings of IEEE 2016 IFIP Networking Conference (IFIP Networking)*, 2016.

[23] 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. In *IEEE Transactions on Networking*, 2022.

[24] H. Cech. Analyzing and realizing multipath TCP schedulers in linux. In *Technical Report, Technical University of Munich*, 2020.

[25] Xu Liu, Yongcheng Huang, Li Song, Rong Xie, and Xiaokang Yang. The SJTU UHD 360-Degree Immersive Video Sequence Dataset. In *2017 International Conference on Virtual Reality and Visualization (ICVRV)*, pages 400–401, October 2017. ISSN: 2375-141X.

[26] J. Kurose and K. Ross. Computer networking: A top-down approach. In *Pearson; 7th edition*, 2016.

[27] R. Khalili, N. Gast, M. Popovic, and J. Le Boudec. MPTCP is not pareto-optimal: performance issues and a possible solution. In *IEEE/ACM Transactions on networking*, 2013.

[28] Q. Peng, A. Walid, J. Hwang, and S. H. Low. Multipath TCP: Design, analsysis and implementation. In *IEEE/ACM Transactions on networking*, 2016.

[29] J. Chakareski and M. Khan. WiFi-VLC dual connecitivty streaming system for 6DOF multi-user virtual reality. In *Proceedings of ACM NOSSDAV*, 2021.

[30] J. Chakareski, R. Aksu, V. Swaminathan, and M. Zink. Full UHD 360-degree video dataset and modeling of rate-distortion characteristics and head movement navigation. In *Proceedings of ACM MMSYS*, 2021.

[31] Y. Chen, R. J. Gibbens Y. Lim, E. M. Nahum, R. Khalili, and D. Towsley. A measurement-based study of multipath TCP performance over wireless networks. In *Proceedings of ACM IMC*, 2013.

[32] Q. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure. A first analysis of multipath TCP on smartphones. In *Proceedings of Passive and Active Measurement Conference*, 2016.

[33] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan. WiFi, LTE, or both? measuring multi-homed wireless internet performance. In *Proceedings of ACM IMC*, 2016.

[34] 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 *Proceedings of ACM MOBICOM*, 2016.

[35] 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 *Proceedings of IEEE INFOCOM*, 2014.

[36] K. Nguyen, M. Kibria, K. Ishiz, and F. Kojima. Feasibility study of providing backward compatibility with MPTCP to WiGig/IEEE 802.11ad. In *Proceedings of IEEE VTC*, 2017.

[37] S. Sur, I. Pefkianakis, X. Zhang, and K. Kim. Wifi-assisted 60 GHz wireless networks. In *Proceedings of ACM MOBICOM*, 2017.

[38] M. Polese, R. Jana, and M. Zorzi. TCP in 5G mmwavenetworks: Link level retransmissions and MP-TCP. In *Proceedings of IEEE Workshop on 5G New Radio Technologies*, 2017.

[39] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli. DAPS: intelligent delayaware packet scheduling for multipath transport. In *Proceedings of IEEE ICC*, 2014.

[40] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju. QAware: a cross-layer approach to MPTCP scheduling. In *Proceedings of IFIP Networking*, 2018.

[41] Y. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens. ECF: an MPTCP path scheduler to manage heterogeneous paths. In *Proceedings of ACM SIGMETRICS*, 2017.

[42] S. Baidya and R. Prakash. Improving the performance of multipath TCP over heterogeneous paths using slow path adaptation. In *Proceedings of IEEE ICC*, 2014.

[43] J. Hwang and J. Yoo. Packet scheduling for multipath TCP. In *Proceedings of IEEE ICUFN*, 2015.

[44] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu. OCPS: offset compensation based packet scheduling mechanism for multipath TCP. In *Proceedings of IEEE ICC*, 2015.

[45] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon. Cross-layer scheduler for video streaming over MPTCP. In *Proceedings of ACM MMSYS*, 2017.

[46] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. MPDASH: adaptive video streaming over preference-aware multipath. In *Proceedings of ACM CONEXT*, 2016.

[47] A. Nikravesh, Y. Guo, X. Zhu, F. Qian, and Z. Mao. MP-H2: a client-only multipath solution for HTTP/2. In *Proceedings of ACM MOBICOM*, 2019.

[48] Y. Guo, A. Nikravesh, Z. Mao, F. Qian, and S. Sen. Accelerating multipath transport through balanced subflow completion. In *Proceedings of ACM MOBICOM*, 2017.

[49] X. Corbillon, F. Simone, and G. Simon. 360-degree video head movement dataset. In *Proceedings of ACM MMSYS*, 2017.

[50] E. David, J. Gutiérrez, A. Coutrot, M. Silva, and P. Callet. A dataset of head and eye movements for 360 videos. In *Proceedings of ACM MMSYS*, 2018.

[51] S. Fremerey, A. Singla, K. Meseberg, and A. Raake. AVtrack360: an open dataset and software recording people's head rotations watching 360 videos on an HMD. In *Proceedings of ACM MMSYS*, 2018.

[52] L. Sun, F. Duanmu, Y. Liu, Y. Wang, Y. Ye, H. Shi, and D. Dai. Multi-path multi-tier 360-degree video streaming in 5G networks. In *Proceedings of ACM MMSYS*, 2018.

[53] M. Yu, H. Lakshman, and B. Girod. A framework to evaluate omnidirectional video coding schemes. In *Proceedings of IEEE ISMAR*, 2015.

[54] J. Chakareski, R. Aksu, X. Corbillon, G. Simon, and V. Swaminathan. Viewport-driven rate-distortion optimized 360-degree video streaming. In *Proceedings of IEEE ICC*, 2018.

[55] J. Wu, B. Cheng, and M. Wang. Improving multipath video transmission with raptor codes in heterogeneous wireless networks. In *IEEE Transactions on Multimedia*, 2018.

[56] 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 *Proceedings of IEEE ICC*, 2021.

[57] A. Ravichandran, I. Jain, R. Hegazy, T. Wei, and D. Bharadia. Poster: Facilitating low latency and reliable vr over heterogeneous wireless networks. In *Proceedings of ACM MOBICOM*, 2021.

[58] 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 Transactions on Multimedia Computing Communications and Applications*, 2022.

[59] S. Gupta, J. Chakareski, and P. Popovski. mmWave networking and edge computing for scalable 360-degree video multi-user virtual reality. *IEEE Trans. Image Processing*, 32:377–391, 2023.

## 9 APPENDIX: EXTENDING FBDT TO N RATS

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.8. 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

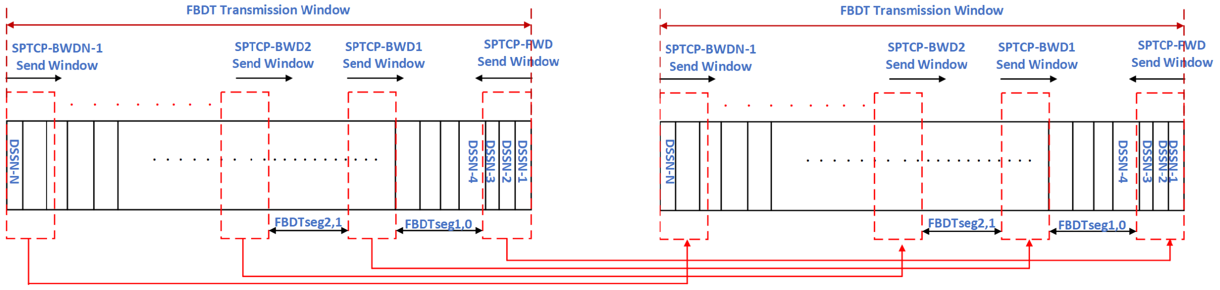

**Figure 8: FBDT scheduler with N number of SPTCPs. There is one SPTCP-FWD and $N-1$ SPTCP-BWD.**

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 3). The overall throughput of FBDT can be approximated as:

$$Throughput_{FBDT} = \gamma \times (\frac{W_{fwd}}{RTT_{fwd}} + \sum_{i=1}^{i=N} \frac{W_{bwd_i}}{RTT_{bwd_i}}) \tag{4}$$

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 $send\_window$ = 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:

$$SegSize_{j,j-1} = N(\frac{Throughput_j}{\sum_{k=1}^{k=N-1} Throughput_k}) \tag{5}$$

$$SegOffset_{j,j-1} = \sum_{K=1}^{K=j-1} SegSize_{K,K-1} \tag{6}$$

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 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 2** FBDT Pseudo-Code

```
1:  $SPTCP \in SPTCP1, SPTCP2, ....SPTCPn$
2:  $TotalBytes \leftarrow 0$
3:  function FBDTSendData(SPTCP)
4:      if $TotalBytes = TransWindowSize$ then
5:          return
6:      end if
7:      if CheckAlignment==False then
8:          AlignFBDTSendWin($a$)
9:      end if
10:     $SPTCP \leftarrow NextSPTCP$
11:     $BytesSent \leftarrow$ SendSPTCP($SPTCP, WMemSize$)
12:     $TotalBytes \leftarrow TotalBytes + BytesSent$
13:     FBDTSendData(SPTCP)
14: end function

15: function AlignFBDTSendWin($b$)
16:     if SPTCP=LastSPTCP then
17:         return
18:     end if
19:     if SPTCP=FwdSPTCP then
20:         $SegSize \leftarrow C \times \frac{W_{fwd}}{RTT_{fwd}}$
21:         $SegOffset \leftarrow 0$
22:         AlignFBDTSendWin($a$)
23:     else
24:         $SegSize \leftarrow C \times \frac{W_{bwd}}{RTT_{bwd}}$
25:         $SegOffset \leftarrow SegSize + PrevSegOffset$
26:         AlignFBDTSendWin($a$)
27:     end if
28: end function
```