Optimum MultiAP Coordination in Wi-Fi With Overlay Time-Sliced Scheduling

Shivam V. Vatsa, Anusha G. P., Vishal Sevani, Purushothaman Saravanan, Rahul Nair, Rishabh Roy, S. V. R. Anand, Joy Kuri, and Anurag Kumar

***Abstract*—We report on the design and implementation of multiAP (Access Point) coordination in a Wi-Fi network, using fine-grained overlay time-slicing, centralized queueing and link activity scheduling. Our approach achieves network wide utility optimal downlink and uplink TCP throughputs, while permitting rate guarantees to designated TCP flows. We also obtain significant improvement in the performance of downlink streaming video, in the presence of cochannel interference from other APs. This work is a substantial extension of the ADWISER (ADvanced WI-fi Service EnhanceR) system reported in Hegde et al. (2013), Sunny et al. (2017), and Sevani et al. (2022). ADWISER is a central scheduler through which all downlink and uplink traffic passes; it overlays periodic time-slices, during which sets of AP-STA links are scheduled. During the time-slices the scheduled AP-STA links use the default IEEE 802.11 PHY/MAC mechanisms; thus ADWISER works without any modifications to the APs and the STAs. Our earlier work required the knowledge of the sets of the AP-STA links that can be scheduled without mutual interference between them, and used time-slices of 100s of milliseconds, which were statically partitioned into scheduling slots to which the known sets of AP-STA links were mapped. The techniques that we now develop and implement perform fine-grained time-slicing, down to 20 ms, thus permitting the handling of delay sensitive Internet applications. ADWISER, now, only needs to know the AP-STA associations. By a process of scheduling and measuring throughputs, using stochastic approximation algorithms, ADWISER dynamically determines the set of AP-STA links to schedule in successive time-slices, thereby achieving global network utility optimal throughputs. We have implemented ADWISER on a Linux OS based server machine, and all performance results we report are from test-beds that use commercial off-the-shelf APs and laptops as STAs.**

***Index Terms*—Wi-Fi QoS controller, utility optimal scheduling in Wi-Fi, overlay scheduling, multi-AP coordination, overlay time slicing, centralized controller, IEEE 802.11, spatial reuse.**

I. Introduction

THE promise of 100s of Mbps access speeds by the recent Wi-Fi standards (11ac, 11ax, and 11be) has led to dense deployments of Wi-Fi access points, in the hope that high Internet throughputs can be ensured no matter where the user is located in the building. For example, Fig. 1 (upper panel) shows a professionally deployed Wi-Fi network over an 8-floor student residence in our academic campus. The building structure comprises steel reinforced cement-concrete, and cement block walls. There are 200 APs, each covering, roughly, four single occupancy rooms, deployed to achieve at least −65 dBm RSS coverage over the entire building.

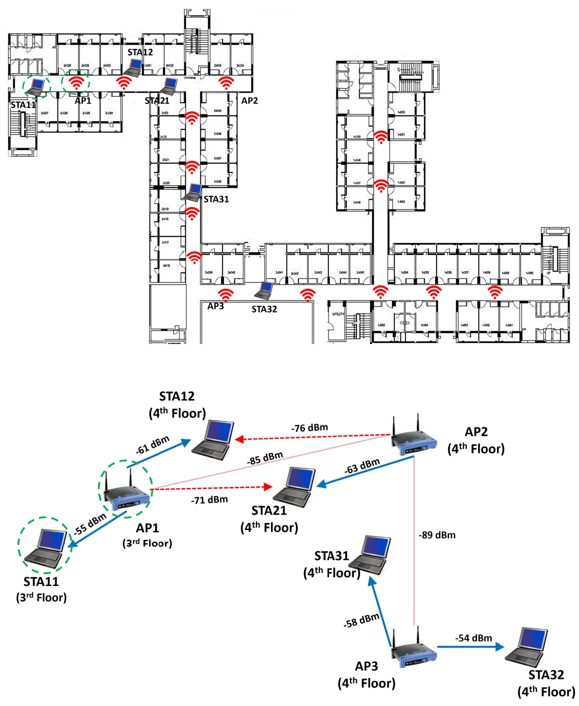

Fig. 1. Building floor plan, with dense deployment of APs (wireless symbols, upper panel). Lower panel: The associations (blue arrows) and RSSs between the APs and the test STAs. Devices enclosed in dashed circles are one floor below.

In this setting, to understand how a multiAP-multiSTA cochannel network would perform, we placed five laptops over two floors, and associated with three APs, as shown in Fig. 1 (lower panel). All these three APs were operating on the same channel, and, for the duration of the experiment, channel switching was disabled.[1] At each laptop the RSS from its AP is > -65 dBm, as per the deployment target.

The device we are reporting about, namely ADWISER (ADvanced Wi-fi Service EnhanceR), was placed between the aggregation switch of this building and the upstream

Received 23 August 2024; revised 26 July 2025; accepted 13 September 2025; approved by IEEE Transactions on Networking Editor H. Hassanieh. Date of publication 28 October 2025; date of current version 5 January 2026. This work was supported in part by the Centre for Networked Intelligence (IISc) and in part by the Arista Networks. *(Corresponding author: Anurag Kumar.)*

The authors are with the ECE Department and the ESE Department, Indian Institute of Science Bangalore (IISc), Bengaluru 560012, India (e-mail: anurag@iisc.ac.in).

This article has supplementary downloadable material available at https://doi.org/10.1109/TON.2025.3613824, provided by the authors.

Digital Object Identifier 10.1109/TON.2025.3613824

[1]The network had an external controller, supplied by the AP vendor, that could switch channels on observing performance anamolies.

TABLE I
TCP DOWNLINK THROUGHPUTS (MBPS) IN THE 3 AP AND 5 STA NETWORK IN FIG. 1

	AP1-STA11	AP1-STA12	AP2-STA21	AP3-STA31	AP3-STA32
Isolated	86.13	49.2	48.33	60.47	90.2
All together	42.28	4.07	0.07	21.24	46.72
With ADWISER	39.23	13.42	11.21	27.29	38.33

campus network switch. For the duration of this experiment, ADWISER was configured *only* to isolate the time periods during which the above 5 laptops could send/receive traffic and when the user devices in the building could send/receive traffic. See Sec. I-A for an explanation of how ADWISER is able to achieve this. Then *iPerf* was used to create long TCP downlink transfers between a server on the wireline campus network, and one or more of the five STAs in the testbed.

Tbl. I shows the individual downlink TCP throughput of each AP-STA link (operating alone), and the downlink TCP throughputs at the five STAs when all the downloads happen together. The isolated throughputs are consistent with the AP-STA RSS values shown in the bottom panel of Fig. 1. Due to sharing cochannel airtime, one would expect the "together" throughputs to be lower but the extremely low throughputs obtained by AP1-STA12 and AP2-STA21 are due to cochannel interference, shown by red dotted lines in Fig. 1. We will analyse this in detail in Sec. III. In short, the default airtime scheduling mechanisms in the network were unable to prevent mutually interfering links from being scheduled together, thereby resulting in severe reduction in throughputs due to the reduction in the PHY rates and aggregation sizes (see Sec. III). The last row of Tbl. I shows the downlink TCP throughputs when ADWISER is put into its queueing and scheduling mode (see Sec. I-A). With the improved airtime scheduling the AP1-STA12 and AP2-STA21 throughputs increase to above 10 Mbps. The final throughputs are the result of sum utility maximizing scheduling (as in cellular networks); see Sec IV-A.

Several techniques have been introduced in IEEE 802.11ax (High Efficiency WLAN (HEW)), such as Spatial Reuse ([4]) where the aim is to dynamically manage the transmission power and the carrier sense threshold when there are overlapping cochannel BSSs [5]. Another feature called Target Wake Time (TWT) [6] is introduced, where an AP negotiates sleep-wake schedules with an associated STA or group of STAs such that these STAs are active only in their respective wake periods and sleep otherwise, thus reducing contention. However, these features cannot be properly utilized to achieve *globally optimal* network performance, without coordination among the APs. For example, for the efficient use of Spatial Reuse, coordination among APs is required where certain information, such as the RSSs at all the STAs from all the coordinating APs are collected and the best AP-STA links are selected for simultaneous transmissions.

IEEE 802.11be (Wi-Fi 7, or Extremely High Throughput (EHT)) [7] proposes multiAP coordination [8] which is expected to enable features such as Coordinated Spatial Reuse (CSR), Coordinated Orthogonal Frequency-Division Multiple Access (C-OFDMA), Coordinated Beam Forming (CBF), and Joint Transmission (JXT), that can alleviate the interference problems in dense AP networks [9], [10]. MultiAP coordination looks promising, but, without a central coordinating node,

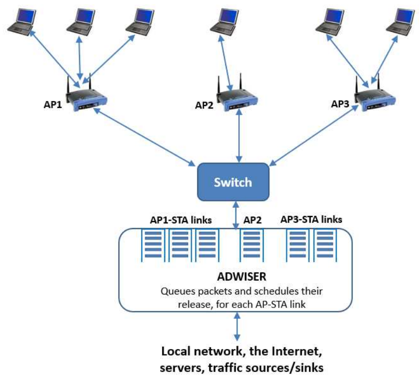
AP1
AP2
AP3
Switch
AP1-STA links
AP2
AP3-STA links
ADWISER
Queues packets and schedules their release, for each AP-STA link
Local network, the Internet, servers, traffic sources/sinks

Fig. 2. ADWISER queues all downlink packets and scheduled their release in time-slices.

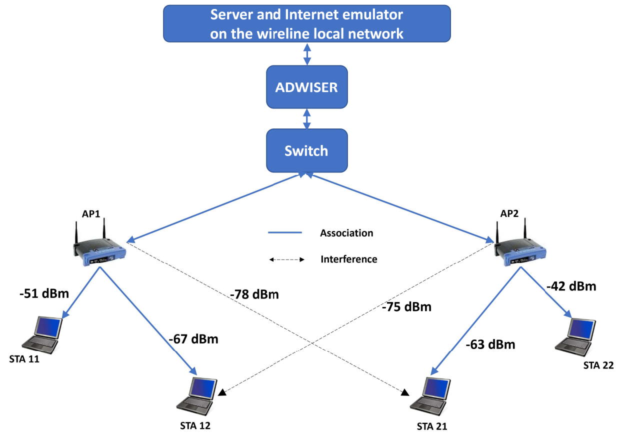

Fig. 3. The 2AP-4STA Testbed: showing associations, signal strengths, and interference powers.

each AP has to exchange data and control information with neighboring APs, giving rise to a lot of signalling overhead and processing complexity [11].

As an example, Nunez et al. [12] propose multiAP coordination by an AP sharing its TxOP with "shared" APs that are known to be able to transmit simultaneously to designated STAs, without interference, thus enhancing spatial reuse. A central controller needs to collect extensive RSS (receive signal strength) measurements and form groups of APs that can share TxOPs. Even if such detailed RSS measurements, between several transmitters and receivers, could be collected it would be difficult to derive from these measurements which sets of AP-STA links can coordinate their transmissions.

In contrast, in our approach (Sec. IV), the central controller (ADWISER) only needs to observe the throughputs when various sets of links are scheduled. "Fair sharing" indices are then computed for various schedulable sets, and the best set is scheduled. These actions, over time, lead to global throughput utility optimization over the network; see, for example, Kushner and Whiting [13, Sec. V-B].

A. The ADWISER Concept

While we will provide details of the ADWISER architecture and algorithms in Sec. IV, the following is a brief overview. Fig. 2 shows several APs and their associated STAs with a device called "ADWISER" (Advanced WiFi Service Enhancer, see also [1], [2], [3],) inserted in the path of all packet flows between the traffic sources/sinks and the STAs.

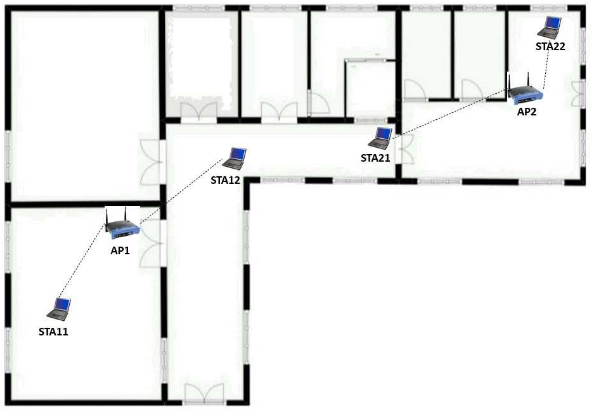

Fig. 4. Placement of devices for the network shown in Fig. 3.

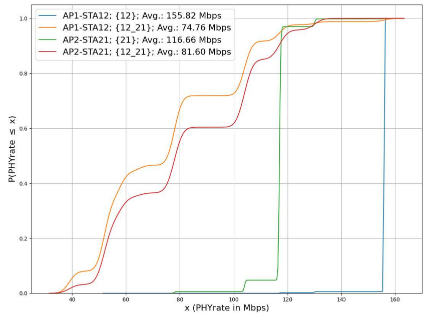

Fig. 5. CDFs of PHY rates for isolated and together (denoted by "12-21") TCP flows to cell-edge STAs, STA12 and STA21.

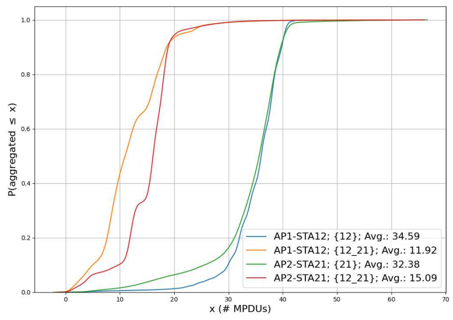

Fig. 6. CDFs AMPDU sizes for isolated and together (denoted by "12-21") TCP flows to cell-edge STAs, STA12 and STA21.

ADWISER is a Wi-Fi controller through which all packets between the wireline network and the Wi-Fi network pass (see Fig. 2). It queues all downlink packets, including ACK packets for uplink TCP DATA packets. This permits the control of uplink and downlink TCP flows, and downlink UDP flows. Uplink UDP flows are *not controlled*.[2] ADWISER "slices" the wireless medium into periodic time-slices, thus, creating *a framework for overlay time-division scheduling of the wireless medium* (see Fig. 8). To each time-slice a single AP-STA link or a set of AP-STA links is *dynamically* assigned. These scheduling decisions are made using the same type of learning algorithms that have been used in cellular systems; see ([14, Chapters 5 and 6] and [13]). When managing *downlink* (AP to STA) (resp., *uplink* (STA to AP)) TCP controlled transfers,

[2]In ongoing research, we time-synchronize uplink UDP devices (sensors, robots) and assign these devices to mini-slices within the ADWISER time slices. We demonstrate tight control of delays to and from these real-time devices, as compared to running all the applications directly on Wi-Fi.

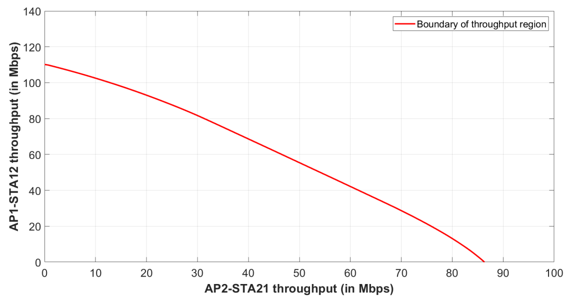

Fig. 7. The rate region for downlink data transfer in the AP1-STA12 and AP2-STA21 subnetwork in Fig. 3.

ADWISER queues the TCP data packets (resp., TCP ACK packets) destined for the STAs, and releases these to the corresponding APs only when the corresponding AP-STA link is scheduled in a time-slice. In each time-slice, the packets/ACKs released are allowed to drain out of the system, so that within a time-slice only those AP-STA links are active to which packets/ACKs are released. This isolates in time the activity in various parts of the WiFi network that interfere under the default operation (recall Table. I). No changes are made to the APs and STAs; indeed, within a time-slice nodes use default CSMA/CA for channel access.

To ensure that a scheduled set of AP-STA links complete their activity within a time-slice, ADWISER tracks (passing through it back to the wireline network) the flows of uplink TCP ACKs (in the case of downlink TCP flows), and uplink TCP data packets (in the case of uplink TCP flows), to determine the end of the link activities due to the released packets. This is called the "drain" time. ADWISER releases the packets in "bursts" (to promote AMPDU aggregation at the APs), and adaptively calculates the release burst sizes with the aim to align the end of the drain time with the end of the time-slice in which the data is released. At the end of each time-slice, ADWISER updates the measurements and determines the next set of AP-STA links to schedule. ADWISER, thus, schedules a sequence of sets of AP-STA links so as to achieve an overall network performance objective.

Such time-sliced operation, however, introduces *access delay* if a user initiates a file transfer or HTTP request when their STA is not scheduled, or when real-time traffic (such as interactive video) is being handled, thus, motivating the need for *fine-grained* time-slicing.

In the example in Fig. 1, the slices were of 20 ms, and ADWISER discourages the scheduling of AP1-STA12 and AP2-STA21 in the same time-slice, while encouraging the scheduling of noninterfering AP-STA links.

ADWISER performs multiAP coordination by overlay scheduling *only in time.* ADWISER is unable to use the *additional degrees of freedom* provided by MU-MIMO-OFDMA, or configure TWT so that STAs are awake only when ADWISER releases data for them, since the firmware in commercially available APs does not provide hooks using which we can create such a control plane.

B. Related Literature

Our prior work based on overlay queueing, time-slicing, and scheduling appeared in [1], where we aimed resolve several performance anomalies in single AP IEEE 802.11g

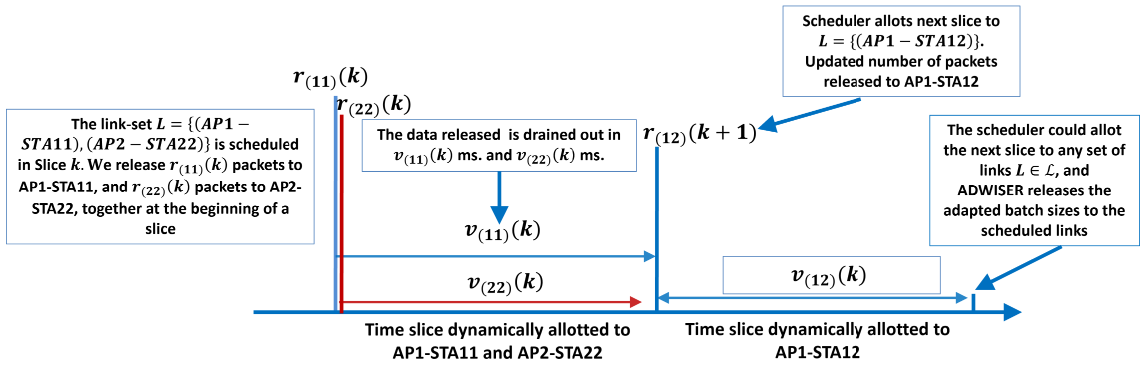
Scheduler allots next slice to $L = \{(AP1 - STA12)\}$. Updated number of packets released to AP1-STA12
$r_{(11)}(k)$
$r_{(22)}(k)$
The link-set $L = \{(AP1 - STA11), (AP2 - STA22)\}$ is scheduled in Slice k. We release $r_{(11)}(k)$ packets to AP1-STA11, and $r_{(22)}(k)$ packets to AP2-STA22, together at the beginning of a slice
The data released is drained out in $v_{(11)}(k)$ ms. and $v_{(22)}(k)$ ms.
$r_{(12)}(k+1)$
The scheduler could allot the next slice to any set of links $L \in \mathcal{L}$, and ADWISER releases the adapted batch sizes to the scheduled links
$v_{(11)}(k)$
$v_{(22)}(k)$
$v_{(12)}(k)$
Time slice dynamically allotted to AP1-STA11 and AP2-STA22
Time slice dynamically allotted to AP1-STA12

Fig. 8. Time slicing with adaptive batch release. Example of link set scheduling, burst releases, and drain times in two successive time slices for the 2 AP and 4 STA example. The two bursts, $r_{(11)}(k)$ and $r_{(22)}(k)$, are released together, at the start of slot k, but are shown separately for clear illustration.

TABLE II
TCP Downlink Throughputs for "isolated" and"together" Operation

Setting	θ in Mbps $\pm$ 95% CI			
	AP1-STA11	AP1-STA12	AP2-STA21	AP2-STA22
Isolated	108.63±1.05	94.16±0.93	97.39±1.28	126.48±0.98
Together (default Wi-Fi)	61.47±1.28	2.91±2.04	7.45±1.42	69.72±0.97
Together (ADWISER)	49.78±2.05	21.90±1.76	21.91±2.57	56.01±1.69

Wi-Fi networks. The technique was further developed in [2], for managing TCP controlled file transfers in multiAP IEEE 802.11g networks. A preliminary version of the current work has appeared in the conference paper [3], where, with prior knowledge of the interference pattern between the Wi-Fi devices, the time-slices created by ADWISER are grouped into successive periodic *frames* comprising time slices. To each each time-slice in a frame *a link-set* (a set of AP-STA pairs) is *premapped*. No such prior knowledge is assumed in our current paper.

In our current paper, we report on using this approach for "high throughput" (HT) and "very high throughput" (VHT) IEEE 802.11 networks, which use higher order modulation schemes and packet aggregation, both of which adapt *downwards* under high packet loss (see Sec. III; also see [15]). Further, due to the high overheads of RTS-CTS, commercial devices make insufficient use of this interference protective mechanism in HT and VHT system implementations, so that interference unaware scheduling leads to very poor system performance, as illustrated in Tbl. II.

We focus on *fine-grained allocation of airtime* (20ms-80ms) (to sets of AP-STA links), whereas Hegde et al. [1], and Sunny etal. [2] dealt with IEEE 802.11g networks, and time-slices of 100s of ms. The substantial reduction in the time-slice permits us to handle delay sensitive traffic, such as real-time video (see Sec. IV-H), but poses significant new implementation challenges which we have discussed in Sec. IV and Sec. V.

In [2], *the controller is assumed to know* which AP-STA links can be scheduled together. In our current work, which links should be scheduled together (to promote spatial reuse), and which must not be scheduled together, is *learnt* by the process of scheduling the links, and making throughput measurements, in much the same way as cellular system schedulers work ([14, Chapters 5 and 6]). No prior RSS measurements are made to determine interference relationships to facilitate multiAP coordination, as, for example, in [9].

The approach of creating virtual queues in an AP's host software (above the firmware), and the scheduled release of packets to the MAC-PHY of the AP, so as to control the performance of flows, has also been exploited by Zehl et al. in [16], and by Bhattacharya et al. in [17], albeit for a single AP and its associated STAs. In [16], time is virtually sliced into 10 ms slots, and air-time guarantees and traffic isolation are provided to some STAs associated with an AP by appropriately releasing packets to the AP's MAC-PHY in slots designated for the corresponding STAs. In [17], specifically for downlink video flows through a single AP to several STAs, the authors install OpenWRT (a stripped-down version of Linux for wireless networking devices) as the OS on a Wi-Fi AP and create virtual queues using the *tc* utility in the OS kernel. These queues feed the single video queue in the Wi-Fi MAC layer. By segregating the flows via these OS level queues, utilizing video performance feedback from the video clients on the STAs, and employing a reinforcement learning based controller to assign flows to queues and to determine how many packets to release, the authors succeed in providing significantly better video performance.

In relation to the above two papers, the work reported in this paper concerns overlay queueing and scheduling *for a multiAP network.* The key concern is the scheduling of AP-STA links across the multiple APs, so as to eliminate interference while promoting spatial reuse, all being done by an online learning approach that aims to achieve global throughput utility maximization. Similar to [17], creating multiple queues for the same STA in the ADWISER controller permits the differentiation of service between flows, e.g., a bulk data transfer flow and a video flow for the same STA, even when there are separate best-effort and video MAC queues in the AP (see Sec. IV-H).

C. Outline of the Paper

In Sec. II, we describe the testbed on which ADWISER experiments reported in this paper are peformed. In Sec. III, after reporting the results from the default Wi-Fi performance with multiple interfering APs and STAs, we provide detailed experimental results on the distribution of PHY rates and AMPDU aggregation sizes to explain the observations of the poor performance. Sec. IV provides the details of the objectives of scheduling in ADWISER, and the various algorithms that achieve optimum overlay time-sliced scheduling.

This section also includes details of TCP ACK tracking to determine the ends of drain times, so as to align them with the time-slices. We also explain how downlink UDP based real-time traffic is handled.

While Sec. IV contains the details of the algorithms that were fine-tuned based on the many challenges faced during implementation, Sec. V further discusses some of the challenges and suggests them as topics for further research, and suggests directions. Sec. VI then provides the results of extensive experiments on the testbed described in Sec. II, and discusses the observations made. The experiments include proportional fairness for downlink TCP transfers, and for uplink TCP transfers, downlink TCP transfers with a rate guarantee for one STA, and a real-time video flow together with TCP flows.

II. A Lab Testbed

In order to conduct repeated experiments, as we developed ADWISER, we set up, in our own academic department, the 2 AP and 4 STA Wi-Fi testbed shown in Fig. 3. The co-channel APs, AP1 and AP2, are about 30 meters apart, as shown in the building layout in Fig. 4, and are not in carrier-sense (CS) range. AP-STA distances for STA12 and STA21 were $\approx$ 10 meters, and for STA11 and STA22 were $\approx$ 6 meters. STA12 and STA21 are placed between the APs, so that each can also receive substantial signal power from the other AP.

The measured RSSs (received signal strengths) between the APs and the STAs are also shown in Fig. 3. Notice that STA11 has 16 dB higher RSS from AP1 than STA12, and STA22 has 21 dB higher RSS from AP2 than STA21. Thus, adopting cellular systems terminology, we refer to STA12 and STA21 as "cell edge" STAs, and STA11 and STA22 as "cell interior" STAs. Each of the cell-interior STAs, i.e., STA 11 and STA 22, has negligible interference from the other AP.

The physical layout, the relationships between the devices, and the RSS values are similar to those in the experiment on the dense AP network discussed in Sec. I (see Fig. 1).

As shown in Fig. 3, the two APs are connected to a common switch, on the other side of which is connected the ADWISER machine, followed by a desktop computer with the Linux OS. This computer acts as an Internet emulator and as a server on the Internet. The server, the switch, and ADWISER are placed in the same room as AP2 (see Fig 4). The Linux *netem* utility, on the server, is used to emulate propagation delays in the Internet. The Linux *iPerf* utility is used to generate unlimited size TCP "file" transfers from the server to the STAs.

Equipment details: ADWISER: a multicore high performance machine running the Linux OS. APs: IEEE 802.11ax, dual-band, Tx-Rx 2×2; STAs: Intel i5/8GB RAM laptops with Ubuntu 20.04. "*Sniffers*" were laptops in the monitor mode.

Wi-Fi configuration: As there were no active APs in the 5 GHz band on Channel 153, in the building, the APs in the testbed were configured for the IEEE 802.11ac mode on Channel 153 in the 5 GHz band; the channel bandwidth was 20 MHz. The transmit powers were: APs 30 dBm, and STAs 22 dBm. Two spatial streams were used throughout, always to the same STA; downlink MU-MIMO (multiuser MIMO) was never used by the AP, due to lack of support from the STAs.

Table. II[3] shows the measured TCP file transfer throughputs when the AP-STA links are *isolated*, i.e., each AP-STA link operates by itself, or *together* (i.e., all the links operate simultaneously using the default IEEE 802.11 mechanisms). The last row in Table. II shows the TCP throughputs that are obtained with ADWISER. With the default operation, the cell-edge STAs (STA12 and STA21) get a mere 2.91 Mbps and 7.45 Mbps. The reasons for this poor performance will be examined extensively in Sec. III.

We will evaluate network performance in terms of the *utility* of the throughputs provided to the STAs, with the utility being evaluated as the sum of the natural logs of the throughputs (as in cellular systems [14, Chapters 5 and 6]); see Sec. IV-A. With the default operation, the utility of the average throughputs for "together" (or default) operation can be seen to be $\ln 61.47 + \ln 2.91 + \ln 7.45 + \ln 69.72 = 11.4394$. With ADWISER, the utility is $\ln 49.78 + \ln 21.90 + \ln 21.91 + \ln 56.01 = 14.1066$, which, due to the use of the log utility function, indicates a more equitable distribution of the throughputs. Further, the total throughput is 141.55 Mbps with the default operation, and 149.60 Mbps with ADWISER. Thus, in this example, both the total system throughput and fairness have been improved by the centralized queueing and overlay scheduling in ADWISER.

An extensive experimental study of this network scenario is provided in Sec. VI.

III. Understanding the Impact of Interference

In this section we analyze the poor default performance shown in Table II for the 2AP-4STA testbed in Fig. 3.

Pairwise RSS measurements in the 2AP-4STA lab setup are shown in Table III.[4] The RSSs from APs to STAs were measured using beacon packets received at the STAs, while the RSSs from STAs to APs and from STAs to STAs were measured using a laptop-based sniffer tool (with approximate adjustments being made for the difference in the antenna gains between the APs and the laptop-based sniffers).

It is evident from Table III that, as in Fig. 3, STA12 and STA21 can be viewed as cell-edge stations, and each receives interference power from the other AP, whereas STA11 and STA22 are close to their associated APs and do not receive interference from the other APs. For example, when AP1 and AP2 transmit simultaneously, the SINR at STA12 is $\approx$ 12.7 dB, and at STA21 $\approx$ 14.69 dB. Without interference (only noise at the noise floor) the RSS is sufficient, at both STA12 and STA21, so that an MCS of 256-QAM, $\frac{3}{4}$ coding, could be sustained, yielding (for 800 ns channel delay spread) a PHY bit rate of 156 Mbps, since the required SNR of 29 dB is exceeded; see the tables at [18]. The same tables can be used to see that, with interference, the SINRs shown above will yield bit rates that are much lower: 52 Mbps or 78 Mbps.

In the remaining part of this section, we will study, in detail, the impact of interference on STA12 and STA21 only. For

[3]In this and later tables, the confidence intervals (CI) are obtained by collecting throughput measurements over 250 sec, in steps of 5 sec, thus yielding 50 samples of throughput from which the CIs are obtained.

[4]The experiments we report were conducted over several days, with the equipment locations and orientations being slightly different from day to day, resulting in variations in the RSS and throughput measurements.

TABLE III

RSS Measurements (in dBm $\pm$ 95% CI) for the Setup in Fig. 4; the Cell at Row i and Column j Shows the RSS at Device i from Device j. A * in a Cell Indicates That the Measured RSS Value Was Less Than -85 dBm, the CS Threshold

	AP1	AP2	STA11	STA12	STA21	STA22
AP1	-	$*$	-58.1$\pm$0.73	-69.49$\pm$0.94	-82.6$\pm$1.28	$*$
AP2	$*$	-	$*$	-78.9$\pm$1.92	-71.6$\pm$2.13	-52.6$\pm$1.23
STA11	-44.6$\pm$0.88	$*$	-	$*$	$*$	$*$
STA12	-56.6$\pm$0.56	-69.3$\pm$2.27	$*$	-	-72.3$\pm$1.71	$*$
STA21	-73.83$\pm$1.85	-59.14$\pm$1.03	$*$	-72.48$\pm$2.17	-	$*$
STA22	$*$	-39.3$\pm$0.48	$*$	$*$	$*$	-

these AP-STA links, packet loss, due to interference from the other AP, is the trigger to adapt to the lower PHY rates.

In addition to the higher modulation and coding schemes that were introduced in IEEE 802.11n, and the PHY/MAC standards thereafter, a major advance was the introduction of *MPDU aggregation*. Aggregation permits a Wi-Fi transmitter to send several packets (MPDUs) in a batch when it gets a transmission opportunity, thereby amortising the contention and transmission overheads over many packets. The devices we experimented with used AMPDU aggregation, which has higher overheads (than the alternative), but which, in case of a burst of errors hitting the AMPDU, permits some of the MPDUs to be recovered. When packet losses take place, however, due to the Block ACK Window based retransmission control between the transmitter and the receiver, the aggregation window can decrease substantially; see [15]. Indeed, due to corruption of the physical layer headers that precede the AMPDU, an entire AMPDU transmitted by an AP will be lost, if this transmission starts when the other AP is already transmitting an AMPDU.

Thus, we see that the interference from another AP not only reduces the PHY rates, but also reduces aggregation, an essential aspect of High Throughput Wi-Fi. This observation is illustrated by the experimental results in Fig. 5 and Fig. 6.

The cdfs (cumulative distribution function) of PHY rates for together and isolated operation, for both AP1-STA12 and AP2-STA21, are shown in Fig 5. For isolated operation, the mean PHY rates are 155.82 Mbps for AP1-STA12, and 116.66 Mbps for AP2-STA21, the difference being due to the higher RSS for AP1-STA12. But for together operation, the mean PHY rates drop, respectively, to 74.76 Mbps and 81.60 Mbps, the lower together PHY rate for AP1-STA12 being due to the lower SINR of 12.7 dB vs. the SINR of 14.7 dB for AP2-STA21.

Fig. 6 shows that, for isolated operation, the mean AMPDU size is 34.59 MPDUs for AP1-STA12, and 32.38 MPDUs for AP2-STA21. With together operation, these values drop to 11.92 MPDUs for AP1-STA12 and 15.09 MPDUs for AP2-STA21, with the AP1-STA12 transfer experiencing higher MPDU error rate due its lower SINR. The combined effects of the steep drops in the PHY rates and the number of MPDUs in an AMPDU, when the APs transmit to their associated cell-edge STAs, are the reasons for the poor together TCP throughputs shown in Table II.

In the setup of Fig 4, RTS-CTS would help to eliminate simultaneous transmissions by the two APs. In our experiments the RTS-CTS mechanism was enabled. Tbl III shows that the APs would have been unable to decode RTS packets from each other. On the other hand, if on winning a contention, AP1 sends an RTS to STA12, the CTS sent by STA12 would be decoded by STA21, blocking it from responding to an RTS from AP2, thus preventing a simultaneous AP2-STA21 transmission. We found, however, that the RTS-CTS mechanism is not used liberally enough, perhaps due to the "false blocking" it can lead to; see [19].

TABLE IV

RTS-CTS Use in a 2 AP and 2 STA Testbed

Experiment	Link	Throughput (Mbps)	% AMPDU with RTS-CTS
Isolated	AP1-STA12	106.1 $\pm$ 1.62	5.30
	AP2-STA21	88.23 $\pm$ 1.04	7.57
Together (without ADWISER)	AP1-STA12	4.66 $\pm$ 0.83	49.79
	AP2-STA21	8.18 $\pm$ 2.14	66.59
Together (with ADWISER)	AP1-STA12	48.60 $\pm$ 1.12	11.65
	AP2-STA21	39.83 $\pm$ 0.89	15.29

To understand the frequency of RTS-CTS use, we conducted an experiment on the AP1-STA12 and AP2-STA21 subnetwork in Fig. 3. The results are shown in Tbl. IV. When the AP1-STA12 and AP2-STA21 links are operated separately, AMPDU losses are only due to simultaneous attempt collisions, and RTS-CTS use is low. When the two links are operated together, without ADWISER, as already discussed in this section, there is mutual interference, and frequent MPDU losses. We observe that RTS-CTS is used before 49.79% of the AMPDUs on one link, and 66.59% in the other. In together operation with ADWISER, the RTS-CTS use is much less than without ADWISER, but more than with isolated operation; this is because there is transient interference when switching between time-slices. This experiment makes it clear RTS-CTS is used when losses are observed, rather than proactively.

IV. Overlay Time Slicing & Dynamic Scheduling

A high-level description of the ADWISER concept has already been given in Sec. I-A. In this section we provide details of the scheduling objectives, the algorithms, and the implementation techniques for scheduling TCP flows.

Recalling Sec. I-A: ADWISER "overlays" on the Wi-Fi network a sequence of equal duration time-slots, that we refer to as *time-slices*. We have chosen this term to avoid confusion with the back-off slots used in IEEE 802.11 CSMA/CA, and also to capture the concept that the slices are much longer than the time-scales of the various PHY and MAC activities, which are of the order of a few microseconds. Indeed, in each time-slice several CSMA/CA contention cycles take place. The main objective of ADWISER is to schedule a set of AP-STA links in each time-slice, and calculate the bursts of packets to be released to each link (in the downlink direction) so that the individual, and overall performance of the network is improved compared with the default operation without ADWISER.

A. Network Rate Region and Utility Optimization

In this section we will describe the performance objective for TCP controlled bulk elastic flows. Downlink real-time flows will be discussed in Sec. IV-H.

For downlink and uplink TCP flows, the scheduling in ADWISER aims for utility optimization over the network rate region. We briefly outline these ideas, well-known in cellular wireless access technology; see [14, Chapters 5 and 6].

A multiAP network is a collection of single-hop AP-STA links, carrying several user flows (say, N), some downlink and some uplink. Given the PHY technologies, the channel variations, the interference between the transmitters and the receivers, etc., the set of achievable vectors of average data rates for all the flows *together* is defined as the *rate region*.

For the restriction of the network in Fig. 3 to AP1-STA12 and AP2-STA21, the experimentally estimated rate region is shown in Fig. 7, with the red curve being the *boundary*.

For M STAs, and, for simplicity, considering one downlink flow to each STA, the important properties of a rate region are: 1. The rate region is a *convex set* $\overline{\mathcal{R}} \subset \mathbb{R}_+^M$. 2. The rate region is *coordinate convex*; i.e., if $(r_1, r_2, \cdots, r_M) \in \overline{\mathcal{R}}$, then so does $(r'_1, r'_2, \cdots, r'_M) \in \mathbb{R}_+^M$ such that $r'_i \leq r_i, 1 \leq i \leq M$.

Given the rate region for a wireless network, at which vector of average throughputs should a set of bulk elastic flows operate? One approach is to evaluate the "utility" of the vector $(r_1, r_2, \cdots, r_M) \in \overline{\mathcal{R}}$ by a function

$$U(r_1, r_2, \cdots, r_M) = \sum_{i=1}^{M} U_i(r_i) \tag{1}$$

where each $U_i(\cdot)$ is an increasing and strictly concave function ([20, Chap. 8], [14, Chapters 5 and 6]). Then the desired operating point is chosen as

$$\arg \max_{(r_1, r_2, \cdots, r_M) \in \overline{\mathcal{R}}} U(r_1, r_2, \cdots, r_M) \tag{2}$$

We will proceed with this approach, further taking $U_i(r_i) = \ln(r_i), 1 \leq i \leq M$ (see [13]). Due to the strict concavity of the utility function and the convexity of the rate region, there is a unique utility optimizing point, on the boundary of the rate region, that we seek in the algorithms described next.

B. Downlink TCP Flows: Scheduling and Data Release

In Fig 8, we illustrate the idea of overlay time-slicing, by showing the details of the activities in two consecutive time-slices in the 2 AP and 4 STA network, in Fig. 3. Time-slices are time intervals over which activity is permitted only over a set of AP-STA links, such that the set contains at most one AP-STA link from each AP. We use the following notation

S:	The nominal time-slice duration, in milliseconds (e.g., 20 ms, 40 ms, etc.)
$\mathcal{L}$:	The collection of sets of AP-STA links such that each set contains at most one AP-STA from each AP
L:	$\in \mathcal{L}$; a valid link-set

The reason for not scheduling more than one AP-STA link out of an AP is the following. If data is released for multiple AP-STA links from one AP, the actual release of data to the STAs will depend on scheduling mechanisms *within* the AP. These mechanisms are proprietary to the manufacturer, and their behavior could jeopardize the global optimality objective in the scheduling algorithms in ADWISER. In the APs that we have used, we have observed that the AP scheduler gives preference to the STAs that are associated with higher RSSs.

For the 2 AP and 4 STA network shown in Fig. 3, the number of sets in $\mathcal{L}$ is given by $|\mathcal{L}| = (3 \times 3) - 1 = 8$; for either AP (AP1 or AP2) we can schedule either of the two AP-STA links or no link (3 possibilities), but we remove the case of no link being scheduled. Evidently, for many cochannel APs and many STAs for each AP, $|\mathcal{L}|$ can be a very large number. We will discuss these *scalability* issues in Sec. V-B.

As shown in Fig. 8, at the beginning of time-slice k, ADWISER decides to schedule the link set $L = \{AP1 - STA11, AP2 - STA22\}$. Some of the packets queued in ADWISER are released to the scheduled AP-STA links in a burst at the beginning of each time-slice. For a certain number of packets to be released in a time-slice, burst release of packets was found to be better than smooth release, as burst release promotes MPDU aggregation at the MAC level, thereby providing a much higher throughput during the slice.

These bursts of packets are released by ADWISER at high speed to the respective APs. The two APs then use the IEEE 802.11 MAC and PHY mechanisms to transmit AMPDUs of sets of these packets to the respective STAs, and receive back the TCP ACK packets from the STAs. In the network activity over a time-slice, any contention and interference affects the service rates of the released packets, and, hence, the "drain times" until the released packets and their TCP ACKs are drained from the network; see Fig. 8.

The following additional notation will be used (see Fig. 8).

$L(k)$:	The link-set (in $\mathcal{L}$) scheduled in Slice k; the evolution of this random process depends on the history of measurements and the scheduling algorithm
$r_i^{(L(k))}(k)$:	$i \in L(k)$, the *burst sizes* released to the AP-STA links in the set $L(k)$ (in TCP data packets; each with a 1448 bytes payload)
$v_i^{(L(k))}(k)$:	$i \in L(k)$, the *drain times*, in ms, at the AP-STA links to which the packet bursts were released
$\xi_i^{(L)}(k)$:	$L \in \mathcal{L}, i \in L$, the estimated throughputs, in Mbps, during slot k, over each of the link-sets $L \in \mathcal{L}$, and links $i \in L$
$l^{(L)}(k)$:	The index of the slice in which the set $L \in \mathcal{L}$ was scheduled last, before the current slice k
$k_i^{(L)}$:	for $i = 1, 2, \cdots$, the subsequence of slices in which the link-set L is scheduled
$\bar{\theta}_i(k)$:	The exponentially weighted moving average (EWMA) throughput, in Mbps, achieved by Flow i up to the end of Slice k.

We will now discuss several algorithms that have been implemented in ADWISER.

C. Estimating Drain Times

When releasing a burst of packets to the AP of an AP-STA link, ADWISER remembers the sequence number of the first byte of the first fresh packet sent in the current slice, and of the last byte of the last fresh packet sent in the slice. This provides the total number of bytes sent in the slice.

TCP ACK processing: ADWISER monitors the progress of an ongoing transmission of a burst of packets to an AP-STA link, by keeping track of the acknowledgement numbers present in the TCP ACK packets sent by an STA back to the TCP server. If SACK (selective ACKs) is enabled over a TCP connection, when multiple packets are lost, the information provided by the Selective ACK (SACK [21]) feature (included in TCP ACK packets) must be processed carefully. When ADWISER receives an ACK packet, it parses the TCP header and reads the acknowledgement number, and for each SACK block in the ACK packet, the left edge and right edge of the range of received bytes in-between the lost bytes. After reading these fields, the number of bytes acknowledged is counted while ensuring that each such acknowledged byte has sequence number not less than the first sequence number sent in the slice. This ensures that the packets sent in the previous slice are not considered in the current slice. Another important situation to be considered is that the left and right sequence numbers in a SACK block are greater than the acknowledgement number carried by the TCP ACK packet. This condition ensures that a D-SACK (RFC 2883 [22]) block is not considered while computing the bytes acknowledged. A D-SACK block is added by the receiver to inform the sender about reception of multiple copies of a packet. Hence, considering a D-SACK block would lead to over-counting of the bytes acknowledged.

If the ACK corresponding to the last byte transmitted in the burst is received within the time-slice boundary, then the drain time is obtained accurately (e.g., in Fig. 8, $v_{(22)}(k)$ is less than the time-slice duration). On the other hand, if the last TCP ACK is not obtained within the time-slice boundary, ADWISER *estimates* the corresponding drain time as follows:

$$v_i^{(L(k))}(k) \leftarrow \begin{cases} v_i^{(L(k))}(k) & if\ v_i^{(L(k))}(k) \le S \\ \dfrac{r_i^{(L(k))}(k) \times 1448}{(r_i^{(L(k))}(k) \times 1448) - \epsilon_i^{(L(k))}(k)} S & otherwise \end{cases} \tag{3}$$

where $\epsilon_i^{(L(k))}(k)$ is the "unacked" bytes out of the batch $r_i^{(L(k))}(k)$ at the end of the time-slice (i.e., when S ms have elapsed), with $0 \le \epsilon_i^{(L(k))}(k) \le r_i^{(L(k))}(k)$. The multiplication by 1448 converts the packet counts into payload bytes. Having taken care of the straggling TCP ACKs in this manner, in the next time-slice, we start tracking the TCP ACKs for the bytes sent in this slice, ignoring the ACKs from the previous slice.

This approach of waiting only up to the end of the scheduled slice duration balances two trade-offs. If we wait until the end of the latest drain time over all the bursts released in a slice, then those AP-STA links that finished early, are forced to idle. On the other hand, waiting only up to the end of the slice time, leaves some data from the current slice in transit at the beginning of the next slice, thus resulting in a short duration of possibly poor performance for the link set scheduled in the next slice. We have found the approach in Eqn. 3 is the better, in being able to achieve the desired steady-state performance.

D. Burst Size Adaptation

Burst sizes (in terms of the number of full size TCP packets), $r_i^{(L(k))}, i \in L$, are determined based on the "experience" when the link-set L was last scheduled. For each $i \in L(k)$, we determine the burst size to be used as follows.

$$r_i^{(L(k))}(k) = (r_i^{(L(k))}(l^{(L(k))}(k)) + \alpha(S - v_i^{(L(k))}(l^{(L(k))}(k))))^+ \tag{4}$$

where, as usual, the notation $(x)^+ = \max\{0, x\}$. This is a Robbins-Monro stochastic approximation algorithm; see [23]. Evidently, the burst-sizes to be used in Slice k will be adapted up or down (resp.), by the amount by which the drain times were less or more (resp.) as compared to the nominal slice duration, S, multiplied by the *learning rate* α.

Effect of α: The following theory has been developed in the Supplementary Material, Sec. VII. We consider the assumption that $\mathbb{E}(v_i^{(L)}|r_i^{(L)}) = \frac{r_i^{(L)}}{\beta_i^{(L)}}$ and $\mathbb{V}ar(v_i^{(L)}|r_i^{(L)}) = (\sigma_i^{(L)})^2 r_i^{(L)}$, where $\beta_i^{(L)}$ is the average TCP throughput over Link i when the link-set L ($i \in L$) is scheduled in a slice, and $\sigma_i^{(L)}$ is a variability parameter. Letting $\{k_j^{(L)}, j = 1, 2, \cdots\}$ be the sequence of time-slices in which the link-set L is scheduled, we can show that, for $|1 - \frac{\alpha}{\beta_i^{(L)}}| < 1$, $\lim_{j\to\infty} \mathbb{E}(r_i^{(L)}(k_j^{(L)})) = \beta_i^{(L)} S$ and $\lim_{j\to\infty} \mathbb{V}ar(r_i^{(L)}(k_j^{(L)})) = \frac{(\sigma_i^{(L)})^2 (\beta_i^{(L)})^2 S}{\frac{2}{\alpha} - \frac{1}{\beta_i^{(L)}}}$. □

Choice of α: For a TCP throughput of, say, 100 Mbps, the MPDU service rate (generically written as β) is about 8.256 packet per millisecond (for 1514 byte TCP packet payload with the MPDU header). By the condition above, we need $0 < \alpha < 2\beta = 16.512$. In the experimental results we report, we have used $\alpha = 1$. We note, from the above analysis, that a small α will lead to a small variance in the steady-state value of the burst that the burst-size adaptation algorithm yields. □

Remark: By the above analysis, over the time-slices in which link-set $L \in \mathcal{L}$ is scheduled, the algorithm in Eqn. (4) adapts the burst sizes on the AP-STA link $i \in L$ such that TCP throughput is approx. the achievable throughput over that link when the links in the set L are active together. This underlies the proportional fair scheduling in Sec. IV-E. □

E. Downlink multiAP Proportional Fair Scheduling

EWMA throughputs: For the purpose of the scheduling algorithm to be described next, we need to calculate the EWMA throughputs achieved by each flow. The throughput during slice k is given by $\frac{r_i^{(L(k))}(k)}{v_i^{(L(k))}(k)}$ if $i \in L(k)$, and zero otherwise. Then, for $i \in L(k)$

$$\bar{\theta}_i(k) = \bar{\theta}_i(k-1) + a\left(\frac{r_i^{(L(k))}(k) \times 1448 \times 8}{v_i^{(L(k))}(k) \times 1000} - \bar{\theta}_i(k-1)\right) \tag{5}$$

and, for $i \notin L(k)$,

$$\bar{\theta}_i(k) = \bar{\theta}_i(k-1) + a\left(0 - \bar{\theta}_i(k-1)\right) \tag{6}$$

where $0 < a < 1$ is the "step size" parameter. The change of units in Eqn (5) yields EWMA throughputs in Mbps. EWMA has a stochastic approximation form, which facilitates a convergence analysis of the scheduling algorithm [13].

Proportional fair scheduling of link-sets: We adopt log-utility,

$$U(r_1, r_2, \cdots, r_M) = \sum_{i=1}^{M} \ln(r_i),$$

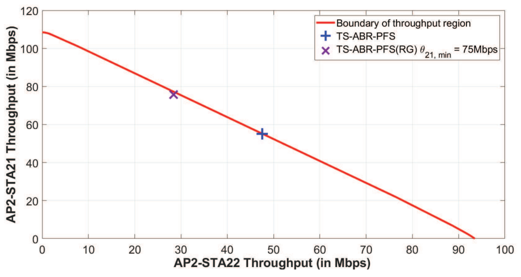

Fig. 9. Rate region: AP2-STA21 and AP2-STA22 subnetwork in Fig. 3, when STA21 is at Location 2 (see Fig. 16).

popular in wireless cellular network scheduling [14, Chapters 5 and 6]. For the utility optimization problem in Eqn. 2, we use the gradient algorithm; see [24]. Given the achievable rates $\xi_i^{(L)}(k), L \in \mathcal{L}, i \in L$, the gradient algorithm schedules

$$\arg\max_{L\in\mathcal{L}} \nabla U(\bar{\theta}_1(k-1), \bar{\theta}_2(k-1), \cdots, \bar{\theta}_M(k-1)) \cdot (\boldsymbol{\xi}^{(L)}(k)) \tag{7}$$

where $\boldsymbol{\xi}^{(L)}(k) = (\xi_1^{(L)}(k), \xi_2^{(L)}(k), \cdots, \xi_M^{(L)}(k))$, with $\xi_i^{(L)}(k) = 0$ when $i \notin L$. These $\xi_i^{(L)}(k), L \in \mathcal{L}, i \in L$ are estimated as follows:

$$\xi_i^{(L)}(k) = \frac{r_i^{(L)}(l^{(L)}(k)) \times 1448 \times 8}{v_i^{(L)}(l^{(L)}(k)) \times 1000} \tag{8}$$

It can be shown (see [24] and [25])) that, for small a, with this algorithm, the EWMA throughputs will eventually fluctuate around the solution of Eqn. 2, the convergence rate and the steady-state fluctuations being governed by a.

For log-utility, at the start of time-slice k, the algorithm in Eqn.7 calculates the following "index" for the link-set L,

$$\Gamma^{(L)}(k) = \sum_{i\in L} \frac{\xi_i^{(L)}(k)}{\bar{\theta}_i(k-1)} \tag{9}$$

and schedules the L that has the largest index.

We note that the index calculation for all the link-sets in $\mathcal{L}$ can only be done if every $L \in \mathcal{L}$ was scheduled in the recent past (i.e., in a slot $l^{(L)}(k) < k$). Since scheduling is stochastic, and, over time, the scheduler will schedule certain link-sets with higher probability, we perform *forced scheduling* so that every link-set is scheduled at least once every few seconds (in the 2 AP and 4 STA network, with 8 link-sets, with 20 ms slices, it was ensured that a link-set was scheduled at least once every 400 time-slices). This causes the actual fractions of slots in which link-sets are scheduled to be a little different from the optimal values, and also this simple approach is not scalable as the network size increases; see Sec. V-B.

We refer to the above overall algorithm (involving scheduling a link-set in each time-slice, and determining the release batch sizes) as TS-ABR-PF (Time Sliced, Adaptive Batch Release, Proportional Fair), where the "proportional fair" terminology comes from the choice of the log-utility function. The scheduling scheme is similar to that used in 4G and 5G cellular network schedulers. Hence, the ADWISER approach can be viewed as *cellularising a Wi-Fi network*.

While the underlying fair scheduling concept is similar to that in 4G and 5G cellular systems, the mechanisms for scheduling by an overlay approach over a *multiAP Wi-Fi network* are the main engineering contributions of our work, along with extensive evaluation by experimentation with a variety of combinations of flows. In the absence of "hooks" in the APs, such as for "muting" a dynamically selected set of links, or for signalling when a burst of data released to an AP has drained out, we need the algorithms in ADWISER for calculation of burst sizes, burst release, and estimation of

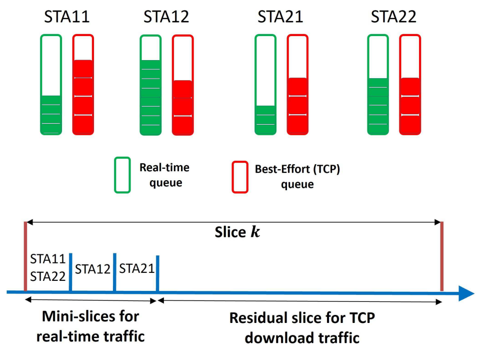

Fig. 10. Handling real-time flows in ADWISER: TCP queues (red) and real-time queues (green) (top); time-slice, mini-slices, and scheduling pattern within the mini-slices (bottom).

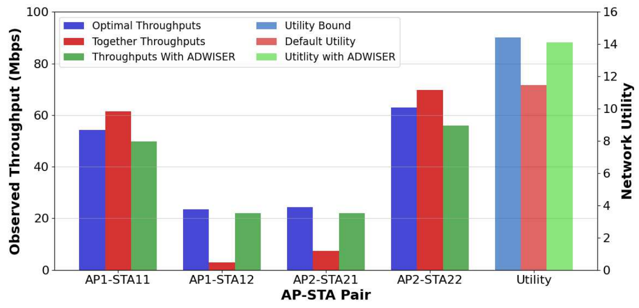

Fig. 11. 2AP 4STA Test-Bed: comparison of TCP throughputs with default scheduling and ADWISER controlled scheduling. The first 4 sets of bars: left y-axis; last set: right y-axis.

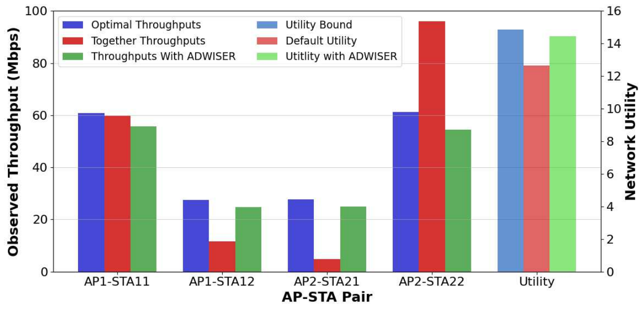

Fig. 12. 2AP 4STA Test-Bed; 150 ms round-trip Internet propagation delay: Downlink TCP transfers; comparison of default scheduling and ADWISER. Left y-axis: first 4 sets of bars; right y-axis: last set of bars.

TABLE V
ONE AP AND ONE STA: TIME-SLICED TCP THROUGHPUTS ANDMEAN AGGREGATION FOR TWO AP MODELS

Experimental Setup	AP Model A STA Throughput(Mbps) / Mean Aggregation(Counts)	AP Model B STA Throughput(Mbps) / Mean Aggregation(Counts)
Without ADWISER	124.32 ± 0.38 / 35.02	143.15 ± 2.18 / 39.95
With ADWISER $S = 20ms$	101.31 ± 1.45 / 28.19	133.55 ± 1.87 / 38.00
With ADWISER $S = 80ms$	117.56 ± 1.93 / 33.82	140.35 ± 2.53 / 40.12

TABLE VI

2AP 4STA Test-Bed: Downlink TCP Transfers Over All fourAP-STA Links; Optimal Vs. Achieved Scheduling Fractions

Schedulable Sets (L)	{11}	{12}	{21}	{22}	{11, 21}	{11, 22}	{12, 21}	{12, 22}
Optimum scheduling fraction	0.0	0.25	0.25	0.0	0.0	0.50	0.0	0.0
With ADWISER	0.01	0.25	0.24	0.0	0.0	0.48	0.0	0.01
With ADWISER (150ms RTT WAN Delay)	0.01	0.26	0.22	0.0	0.02	0.46	0.0	0.02

TABLE VII

2AP 4STA Testbed: "isolated" Downlink and Uplink TCPthroughputs

Setting	Mbps±95% CI			
	AP1-STA11 θ_1	AP1-STA12 θ_2	AP2-STA21 θ_3	AP2-STA22 θ_4
Isolated-Downlink	125.03± 1.24	92.7±1.6	95.5±1.17	126.66±1.07
Isolated-Uplink	65.0 ± 0.41	43.2± 0.46	42.07 ± 0.68	98.6± 1.92

TABLE VIII

2AP 4STA Testbed: Uplink TCP Throughputs for Some Combina-tionsof Active STAs

Setting	Mbps±95% CI			
	AP1-STA11 θ_1	AP1-STA12 θ_2	AP2-STA21 θ_3	AP2-STA22 θ_4
Isolated Uplink	65.0 ± 0.41	43.2± 0.46	42.07 ± 0.68	98.6± 1.92
STA21 & STA22 Active	-	-	0.32±0.0041	94.03±2.95
STA11, STA12 & STA21 Active	64.65±0.95	0.057±0.0003	39.91±1.01	-
STA11, STA12, STA21 & STA22 Active	61.28±1.33	0.045±0.0002	0.037±0.0009	78.0±3.62

link throughputs in scheduled sets, so as to enable the overlay scheduling for interference management.

Bounding the optimum utility for PF scheduling: In Fig. 11 and Fig. 12 we compare the network utility of the TCP throughputs obtained from the experiments against a utility bound. This bound is obtained by solving the utility optimization problem in Eqn. (2) over the following inner bound for the rate region $\overline{\mathcal{R}}$ (writing p.m.f. for probability mass function).

$$\underline{\mathcal{R}} := Co\Big\{\mathbf{0}, \Big\{(r_1, r_2, \cdots, r_M) : r_i = \sum_{L \in \mathcal{L}} \Pi_L \beta_i^{(L)}, \text{for some p.m.f. } \Pi(\cdot) \text{on } \mathcal{L}\Big\}\Big\} \tag{10}$$

where $\beta_i^{(L)}$ is the *long term* time average TCP throughput over the AP-STA link i when this link is activated along with the links in the set L, and $Co\{\cdots\}$ denotes convex hull (see Tbl. VII and Tbl. VIII for examples of such measurements). The rate region $\overline{\mathcal{R}}$ is a convex set and contains the vectors $(r_1, r_2, \cdots, r_M) : r_i = \beta_i^{(L)}$ if $i \in L$, and 0 otherwise, the rate vector obtained by scheduling the link set L in every time-slice. Hence, $\underline{\mathcal{R}}$, which is the convex hull of all such rate vectors is a subset of $\overline{\mathcal{R}}$. Thus, the problem in Eqn. (2) solved over $\underline{\mathcal{R}}$ will provide a lower bound to the optimum utility.

Indeed, $\overline{\mathcal{R}} \supset \underline{\mathcal{R}}$ since $\overline{\mathcal{R}}$ captures the possibility of scheduling based on variations in the link-set throughputs from time-slice to time-slice (see [26]). However, since $\overline{\mathcal{R}}$ is difficult to obtain, it is the above bound that we report along with our experimental results in Fig. 11 and Fig. 12. Looking at the rate regions shown in Fig. 7, and in Fig. 9, we observe that these regions are close to the inner bound displayed above. This is because, averaging over a 20 ms time-slice reduces the variability of the TCP throughput from slice to slice. Hence, we expect that the optimum utility lower bound is close to the optimum utility (over the entire rate region). □

F. Uplink multiAP Proportional Fair Scheduling

As described in Sec. I-A, when handling TCP flows, ADWISER queues all downlink data packets and downlink TCP ACK packets. Thus, we control uplink TCP transfers by controlling the release of TCP ACK packets from the corresponding ADWISER queue. In the experiments that we will describe, each AP-STA link has either a downlink TCP flow or an uplink TCP flow. Hence, it suffices that in ADWISER there is just one queue for controlling TCP flows over each AP-STA link.

For uplink TCP flows (just as for downlink TCP flows; see Sec. IV-B) once it is decided to schedule an AP-STA link in a time-slice, TCP ACKs destined to that STA, are released, in a burst, at the beginning of the slice. As the STA receives the ACK packets, its TCP window "opens", and it can send additional TCP data packets. Each TCP ACK released by ADWISER acknowledges a certain number of bytes. The number of bytes acknowledged by the burst of ACKs released is adapted, so that, accounting for the number of packets the STA will send during the time-slice, the activity just fills up the time-slice.

The release burst sizes, the estimate of activity times, and throughput estimates at the beginning of a time-slice, and the calculation of the proportional fair scheduling indices are all done in a manner identical to the way they are done in the downlink; Eqns. (4), (8), and (9). The set of STAs with highest index is scheduled in the time-slice.

An experimental study of uplink TCP transfers in the testbed of Fig. 3 is provided in Section VI.

G. Proportional Fair Scheduling With Rate Guarantees

The need for a throughput guarantee to a TCP flow (e.g., to a specific STA) could arise in several situations, e.g., a doctor at a patient's bedside needing to rapidly access high-quality diagnostic images. ADWISER can be configured to provide such throughput guarantees to some of the TCP flows, provided the required guarantees are *feasible*. Consider, for example, the rate region in Fig. 9 for a particular placement of AP2 and two associated STAs, STA21 and STA22. Notice that STA21 can obtain a downlink TCP throughput up to about 105 Mbps. Thus, a downlink throughput guarantee of 75 Mbps to STA21 is feasible, whereas 120 Mbps is not feasible. Of course, when STA21 gets 75 Mbps, STA22 will only receive a throughput of a little below 30 Mbps.

This leaves open questions such as admission control of rate-guaranteed flows, and the provision of a handover mechanism, for continuation of the rate guarantee, e.g., if the doctor (in the example above) moves into the coverage of another AP. These issues are topics of our ongoing research.

Rate guarantees are provided to specific TCP flows as an extension of the PF algorithm discussed in Sec IV, called TS-ABR-PFS(RG) (TS-ABR-PFS(Rate Guarantee))

The TS-ABR-PFS(RG) algorithm: Suppose, in a multiAP-multiSTA network, a particular AP-STA link (say, Link i, for a single i, $1 \leq i \leq M$) has to be provided a downlink TCP throughput guarantee denoted by $\theta_{i,\min}$. Then, the optimization problem in Eqn. (2) has a single additional constraint

$$r_i \geq \theta_{i,\min} \tag{11}$$

We have a nonlinear optimization problem with a strictly convex objective function, and we still have a convex constraint set (the intersection of the rate region $\overline{\mathcal{R}}$, and the above minimum rate constraint). Hence, provided that the minimum rate constraint is feasible, there is a unique optimum, which we call the proportional fair sharing with rate guarantee (PFS(RG)) operating point. In addition, there is no duality gap, and there is a Lagrange multiplier $\nu_i^* \geq 0$ corresponding to the PFS(RG) operating point (see, for example, [27]).

In Fig. 9 we notice that the PFS point provides a little over 50 Mbps to STA21, hence, a positive Lagrange multiplier is required to achieve the PFS(RG) point, at which the rate pair is about 28 Mbps and 75 Mbps to STA22 and STA21.

a. Calculation of the PFS(RG) index: We recall the TS-ABR-PFS calculation from Eqn. (9). To provide a rate guarantee to Link i, the index calculations are modified as follows. For Link i we maintain a nonnegative *index bias*, $\nu_i(k)$. When taking a scheduling action for time-slice k, the index for the set of AP-STA links L is calculated as follows:

$$\Gamma^L(k) = \sum_{j \in L} \xi_j^L(k) \left(\frac{1}{\overline{\theta}_j(k-1)} + \nu_i(k) \mathbb{1}_{\{j=i\}} \right) \tag{12}$$

i.e., the index bias is added to the term corresponding to Link i in every set L that contains Link i. ADWISER schedules, in time-slice k, the link-set L that has the highest PFS(RG) index.

b. Updating the index-bias $\nu_i(k)$: At the end of time-slice k, the index-bias for Link i is updated:

$$\nu_i(k+1) = \left(\nu_i(k) + b(\theta_{i,\min} - \overline{\theta}_i(k))\right)^+ \tag{13}$$

where b is a scalar constant, the learning rate for the index-bias update. The index-bias converges to the Lagrange multiplier. The value of b is taken to be much smaller than a, the learning rate in Eqn. (5). The other steps are the same as in the TS-ABR-PFS algorithm; see Sec. IV-E

This algorithm can be compared with the one studied by Mandelli et al. in [28]. They introduce the idea of a "token counter," which is similar to our *index-bias,* with two important differences: 1. Our index-bias is updated based on the EWMA throughput (not the instantaneous slice throughput), and 2. By making $b \ll a$, we use a two time-scale algorithm (see [23]) (as compared to a single time-scale algorithm in [28]). We have performed an extensive simulation and experimental study of these algorithms, and have found that our approach provides a better tracking of the Lagrange multiplier, and, therefore, more accurate tracking of the correct PFS(RG) throughputs; see [29]. Experimental results from our Wi-Fi testbed are provided in Sec. VI-D.

H. Downlink Real-Time Flows

We have found that delay sensitive flows (e.g., video conferencing applications, such as Zoom, and Microsoft Teams), by default, do not mark the DSCP (Differentiated Services Code Point) field in the IP header. Thus, even though Wi-Fi APs can provide voice and video packets preferential treatment during contention (by mapping these packets into AC-VO and AC-VI queues), these packets are mapped to the AC-BE (best-effort) queue, where they are "served" along with TCP flows, without any preferential treatment. Since the Wi-Fi network will, typically, be the bottleneck on the path of the TCP flows, a large fraction of the TCP window will occupy the AC-BE buffer. As we will show in Sec. VI-C, this results in very poor performance for the delay sensitive, lower bit rate, video flows.

ADWISER addresses this problem, in the downlink direction, by maintaining two queues per STA, one for TCP traffic and another for real-time traffic; see Fig. 10. ADWISER uses configured filtering rules to segregate the incoming packets into each of the two queues.[5] With the delay sensitive packets segregated, at the beginning of each time-slice, these packets are released to the APs before any TCP packets are released. Assuming that the aggregate bit-rate of the real-time flows is small, it may be expected that the number of real-time packets queued at the beginning of a time-slice is not large, and all can be released and served by the Wi-Fi network within a fraction of the time-slice. For example, HD video has an average bit rate of 1.6 Mbps, whereas during a slice, a TCP transfer would typically have the throughput of 100 Mbps.

The details of real-time packet release are as follows.

1. Packets from subsets of real-time flows are released together in successive mini-slices at the beginning of each time-slice (Fig. 10). These sets are just the link-sets defined in Sec. IV-B. At the beginning of each time-slice, the link-sets are rank ordered according to the PFS-indices obtained in Eqn. (9). From each link-set, the AP-STA links with empty real-time queues are removed, yielding rank ordered residual link-sets. If in pairs of neigboring residual link-sets (after the rank ordering), one is contained in the other, the smaller one is dropped. Then the real-time packets of the highest ranked residual link-set are released at the start of the first mini-slice, the time taken to drain out these packets from the APs to the STAs (the mini-slice duration) is estimated (see 2. below), and, then, the batch of packets from a next set of real-time queues is released; this is done successively until all the nonempty real-time queues are emptied.

2. The duration of a mini-slice for the link-set L, at the start of time-slice k, is estimated as follows.

$$s^{(L)}(k) = \max_{j \in L} \left(\frac{b^{(L)}_{(udp,j)}(k)}{\xi_j^{(L)}(k)} \right)$$

where, $\xi_j^{(L)}(k)$ is the TCP downlink rate estimated over the Link j, if link-set L is scheduled in slice k (see Eqn. (8)), and $b^{(L)}_{(udp,j)}(k)$ denotes the number UDP payload bits to be released over Link j in time-slice k. The TCP downlink rate would be an underestimate of the UDP throughput, since there are no uplink ACKs for UDP packets. This would lead to an overestimate of the mini-slice duration, and, hence, some waste of air-time. Optimization of this aspect will be a topic of our future work.

[5]The problem of detecting the types of flows and automatic generation of the filtering rules is out of scope of our project; see, for example, Advanced Traffic Classification at sandvine.com

Thus, in the example in Fig 10, the real-time queues of all the STAs are non-empty, hence the link-sets {STA11, STA22}, {STA12} & {STA21} are scheduled, if this was the order of the indices computed in Eqn. (9)

In Sec. VI-C, we provide results from our experiments on our lab testbed with a real-time video flow and two TCP flows, one of which interferes with the video flow.

V. Implementation Challenges

The implementation techniques in Sec. IV have been arrived at after several rounds of revisions and experimentation. Due to limited space, we cannot describe the details of all these iterations, but, in this section, we will explain the basic reasons for why these challenges arose, and some directions that further research can take.

A. Small Time-Slices

In particular, reducing the time-slice durations to 20 ms, from the 100 ms that we started with, proved to be a significant challenge. To explain this, we report the results from a subset of our testbed, comprising just one AP and one STA. A single *iPerf* TCP transfer is set up between the "Internet" server and the STA. We studied two APs, of similar capabilities, from different manufacturers. In Tbl. V we report, for each AP model (A and B), the TCP throughputs, without and with ADWISER, two time-slice durations, 20 ms and 80 ms.

We observe that the time-sliced throughputs are lower than the throughputs without ADWISER. For the Model A AP (resp. Model B), the throughput drops by about 5% (resp. 2%) with an 80 ms time-slice, and by about 18.5% (resp. 7%) with the 20 ms time-slice. We elucidate the following reasons:

1. As seen from Tbl. V, the aggregation counts drop with time-slicing. A detailed study (refer Supplementary Material, Sec. VII-B) of the number of MPDUs in the AMPDUs released by the AP (for downlink TCP flows) revealed that, after the MAC queues empty out, the APs only gradually build up the aggregation levels, some implementations being more aggressive than others. Thus, average aggregation over a time-slice is less for small time-slices. We observe that the Model B AP builds up the aggregation faster than the Model A AP.

2. As explained in Sec. IV-C, ADWISER has no direct knowledge of when the data released in a time-slice has drained out of the system. There is variation in the time taken to drain out a given burst, due to variation in the wireless channel, and the randomness in the CSMA/CA access mechanism. As a result, so that the drain time is close to the slice duration, the burst size that is released is, on the average, smaller than that which will provide the non-time-sliced throughput. The analysis of Eqn. (4) in Sec. IV-D makes idealized assumptions; in particular, that the packet transmission rate is the isolated rate, and every burst is allowed to drain out completely.

3. We have found that laptops enter the Power Save Mode (PSM) at time instants that are not synchronized with the time-slices in which they are scheduled. As a consequence, an STA can be in PSM when it is scheduled by ADWISER, leading to loss of airtime for that STA even though it is scheduled. We have found, however, that the STAs were adapting their entry into PSM to the pattern of traffic that was scheduled for them, leading to lower overhead than might be expected; see [30]. Ideally, when overlay time-slicing is employed, there could be a signalling mechanism between the APs and the STAs so that STAs sleep only when they are not scheduled.

Together, these three phenomena reduce the efficiency of time-slicing. We say that we have an *overhead* due to time-slicing, which can be as much as 20% for 20 ms time-slices. The overhead results in the rate region with ADWISER being smaller than the actual rate region, as defined in Sec. IV-A.

Since we wish to demonstrate the performance with delay-sensitive flows, all the experiments we report in Sec. VI have been performed with 20 ms time-slices. If, in an application, only TCP bulk transfers have to be managed, then larger time-slices, such as 80 ms can be used.

B. Scalability With Increasing APs and STAs

a. Large dimensionality of scheduling data and actions: A glimpse of this issue has already been provided in Sec. IV-B when introducing the concept of link-set scheduling. With 2 APs, and 2 STAs associated with each AP, there are 8 link-sets. However, as an example, if there are 20 cochannel APs and 4 STAs associated with each AP, then there are $|\mathcal{L}| = 5^{20} - 1$ link-sets. It is clear from Eqn 9 that $\xi_i^{(L)}(k)$ will need to be available for each $i \in L, L \in \mathcal{L}$. However, since the optimal schedule will rarely schedule most of the link-sets, many unnecessary measurements have to be made online, by stealing time from the optimal scheduler, and all this has to be done within a coherence time.

This difficulty with exact scheduling in wireless networks is well known, as, in a network of interacting wireless links, a large number of possibilities need to be considered for optimal scheduling in each slot; see, e.g., [31] and [32].

The following are two approaches that can be explored.

1. In the 3 AP and 5 STA experiment in Sec. I, there were 17 link-sets. By studying the network topology, we restricted the collection of link-sets to 12, which yielded the results shown.

2. A scheduler maps certain measurements, and system state, to a link-set $L \in \mathcal{L}$. We could train a DNN (Deep Neural Network) to learn the mapping from lower dimensional data (such as the beacon powers received at the STAs from the APs) to a link-set in $\mathcal{L}$; see [32]. Such an approach will require beacon powers measured at the STAs; feasible in a private network situation, such as in a networked smart factory. Our recent experiments show that, in a 6 AP and 24 STA setup, which has $5^6 - 1 = 15,624$ link-sets, DNN-based scheduling can be done in about 0.1 ms per time-slice (and yields $90 - 95\%$ of the optimum utility), whereas the search for an optimal link-set has to be done over all link-sets, and takes 10s of milliseconds.

b. TCP ACK tracking in ADWISER: As the number of STAs grows, ADWISER can get overloaded due to needing to track the packets from the STAs back to the Internet for a large number of connections. We have demonstrated the working of a prototype implementation in Sec.VI (some ADWISER host performance measurements are provided in the next paragraph). Going forward, a natural approach would be to place the data handling part of ADWISER (the queues, the burst release, the tracking of packets from the STAs) in the APs, and leave only the scheduler functionality in the central controller. This could also help to directly determine when the AP queues drain out, thereby reducing some of the overhead of small time-slices (see Sec. V-A). This is, also, a topic of our current research.

c. Scalability of ADWISER as a central controller: ADWISER has been implemented on a high performance server (Intel Xeon Gold 6212U, 48 Cores, 128 GB RAM, Intel (X540-AT2) 10Gb dual port Ethernet controller, running Linux 6.8.0-62-generic). For our experiments, three cores are utilized: one handling packets exiting towards the Internet, one towards Wi-Fi, and one performing the scheduling. To study the scalability of this implementation, we have emulated downlink TCP flows on a wireline testbed comprising a server, ADWISER, and a machine to emulate the APs and STAs. We emulated the equivalent of a 5 AP and 20 STA network using the Linux *tc* utility, each STA being able to sink data at 100 Mbps. With ADWISER set up to behave exactly as if it was controlling a 5 AP and 20 STA network, the maximum utilization was of the core handling packets towards Wi-Fi, about 11%. The maximum RAM usage was 221 MB.

VI. Experimental Results on the Testbed in Fig. 3

In this section we demonstrate the performance of the ADWISER approach via detailed experimental results for the network topology shown in Fig. 3, physically deployed in our academic building as shown in Fig. 4. Whereas the setup in Fig 1 serves to demonstrate that the performance issues that concern us in this paper can occur in actual deployments, our laboratory testbed permits us to conduct repeated experiments day-after-day, and make measurements that are hard to make in an operational network.

The RSSs would be similar to those shown in Tbl. III. However, since the devices were placed at the lab-table level, their precise location could vary from day to day, yielding small variations in the results, even of the same experiment.

We note that all our TCP experiments reported here are with long lived TCP flows (generated using *iPerf*). In [3], we have reported on an experiment with HTTP traffic. In HTTP, when a burst of HTTP data to an STA ends, the corresponding queue in ADWISER empties out. The scheduling in ADWISER then takes place only over the non-empty TCP queues, thereby improving the throughput of the bulk transfer flows. The HTTP response time is improved by interference management during the HTTP transfers.

A. Downlink TCP Flows: All Four Links Active

In this experiment the objective is to study the performance with a single downlink *iPerf* TCP transfer from the server on the "Internet" to each of the four STAs. We report three sets of observations. All throughput measurements are taken for 250 seconds, during which 50 measurements were done over successive 5 seconds windows, to get confidence intervals around the average throughputs.

1) As explained in Sec. IV-B, there are 8 link-sets $L \in \mathcal{L}$. For each such L, we initiate downlink *iPerf* TCP transfers over each link in the link-set. This provides $\beta_j^{(L)}, j \in L$, the long-term TCP throughputs obtained over each of the AP-STA links in L, when the set of links in L are activated together. We can use these measurements to obtain an estimate of the fraction of time that each link-set should be scheduled to maximize network utility, and also to estimate a lower bound to the optimum utility (see the end of Sec. IV-E).

2) All four *iPerf* TCP transfers are run together, with the network operating in the default mode (without being controlled by ADWISER).

3) ADWISER is enabled, with 20 ms time-slices, and all four *iPerf* TCP transfers are initiated. In all our experiments, we chose the following learning rates in the algorithms in Eqn. (4), Eqn. (5): $\alpha = 1$, $a = 0.1$. The choice of α has been discussed in Sec. IV-D.

The results from the experiment are displayed in Fig 11. The first four sets of bars are throughputs for each of the AP-STA pairs, and the last set of bars show the network utilities. The blue bars were obtained from the first set of experiments, above, and are an estimate of the throughputs of the AP-STA pair, and network utility, with utility optimum operation of the network. The red bars correspond to the default operation, without ADWISER. The green bars show the performance with ADWISER.

Fig 11 corrobrates the detailed results shown in Tbl. II. Fig 11 also shows the substantial drop in network utility with the default scheduling, and that the utility with ADWISER is close to the optimum. We make the following observations:

The improvement in the performance with ADWISER is due to proper scheduling of STAs. The scheduling fractions derived by solving the *sum log utility maximization problem* over the inner rate region (se Sec. IV-E) is to schedule the edge stations i.e., STA 12 and STA 21, alone, each for 25% of the time-slices, and STA 11 and STA 22, together for 50% of the time-slices. With ADWISER, the link-sets are scheduled dynamically, driven only by the current measurements (Eqn. 9), without knowledge of the ideal fractions. The scheduling fractions obtained, over 250 sec experiments are shown in Tbl. VI. The optimum values have been obtained as explained earlier in this section; the errors are partly due to the forced scheduling of link-sets, as explained in Sec. IV-E.

The network utility is improved from 11.44 (default) to 14.10 (with ADWISER), which is close to the utility bound of 14.42. The dip in the network utility with ADWISER from the utility bound is due to overheads of time-slicing (Sec. V).

With wide area Internet propagation delay: In our earlier work, reported in [2], since large time-slices were used, the periods during which a TCP flow was not served were long enough to result in TCP sender time-outs, retransmissions, and, therefore, loss of throughput. The solution we had proposed was to place a "proxy" server between the enterprise LAN and the wide-area network, so that the time-slicing only affects the local TCP connections between the proxy and the client device. It is, therefore, important to ask whether, with the much shorter, 20 ms, time-slices the TCP sender time-out problem still exists. In Fig. 12, we show the results that can be compared with those of Fig 11. Allowing for the variation due to the experiments having been conducted on different days (with the equipment having to be placed afresh from day to day), with the 20 ms time slicing we saw no significant effects of not serving a TCP connection for a few time-slices.

B. Uplink TCP Flows

We now report on an experiment in which all four STAs, in the 2 AP 4 STA testbed (Fig. 3), are performing TCP transfers *to* the server on the "Internet."

1) Isolated Throughputs: We begin by measuring throughputs when each STA, in isolation, performs an *iPerf* TCP

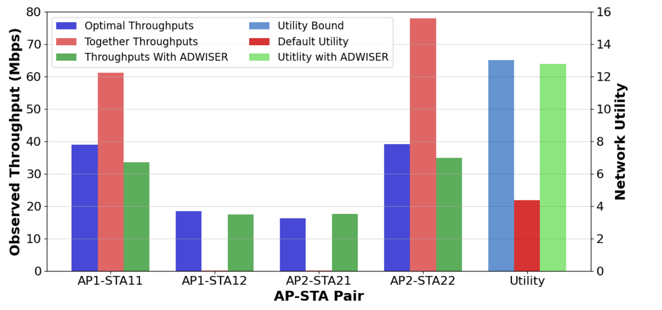

Fig. 13. 2AP 4STA testbed: uplink TCP flows; left y-axis: first four sets of bars; right y-axis: last set of bars.

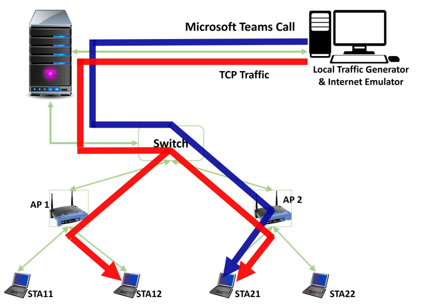

Fig. 14. AP1-STA12, AP2-STA21 testbed: Downlink real-time video and one TCP flow over each AP-STA link.

transfer to the server. The results are shown in Tbl. VII, where we compare the downlink, isolated TCP throughput on each AP-STA link, with the uplink throughput. We notice that the uplink isolated throughputs are smaller than the downlink isolated throughputs shown in Tbl II. There is no contention, nor any interference in this experiment, and this difference is due to the difference in the transmit powers between the STAs (22 dBm) and APs (30 dBm). Indeed, we notice from Tbl. III that the RSS from an STA to its AP is about 13 dB lower than that from the AP to the STA. A part of this difference (8 dB) is explained by the difference in transmit powers. The remaining difference could be ascribed to the STA antennas being directional, with nonuniform directionality; further, there will be nonreciprocity in the channel due to the placement of the STAs in relation to the APs. For such a difference in RSS, the PHY rate drops considerably. For example, from [18] we see that, for 20 Mhz channel bandwidth, and 2 spatial streams (the situation we had in our testbed), when reducing the RSS from -59 dBm to -70 dBm, the PHY bit rate drops from 156 Mbps to 78 Mbps (where, since there is only noise, the SNR is adequate; see [18]). These are PHY bit rates; the TCP throughputs are smaller due to the overheads of TCP ACKs, and contention mode access.

2) Together Throughputs: Default Operation: Tbl. VIII shows the results from several uplink TCP transfer experiments. The first row of numbers repeats the isolated link experiments, for quick comparison with the other rows. The next three rows are for "together" uplink TCP transfers for the STA sets {STA21, STA22}, {STA11, STA12, STA21}, and {STA11, STA12, STA21, STA22}. When {STA21, STA22}, both associated with AP2, are active together, the throughput of STA21 drops to about 320 Kbps and the throughput of STA22 is close to its isolated value. When {STA11, STA12, STA21} are active together, STA12 performs extremely poorly, getting a throughput of about 57 Kbps, whereas the performance of {STA11, STA21} is almost as in the isolated situation. Notice that, in this situation, STA21 gets good uplink throughput compared to STA12 despite both being cell-edge STAs. When all the four links are active together, the throughputs of both the cell-edge STAs, STA12 and STA21, drop close to 40 Kbps.

To understand the performance in Tbl. VIII, let us first consider {STA21, STA22}, both associated with AP2, simultaneously engaged in long, uplink TCP flows. From Tbl. III we observe that these two STAs, though associated with the same AP, are outside the CCA threshold of each other. Hence, simultaneous uplink transmissions take place from these two STAs. As seen from Tbl. III, the STA21 to AP2 RSS is -71.6 dBm, whereas the STA22 to AP2 RSS is -52.6 dBm, resulting in an SINR of -19 dB for the STA21 to AP2 transmission, when both transmissions occur simultaneouly. As explained in Sec. III, this results in severe packet losses during STA21 to AP2 transmissions, a huge reduction in the PHY rate, and very poor aggregation. STA21 starves continuously when STA22 is transmitting data and keeps re-attempting with longer backoffs and smaller aggregation sizes. STA22, thus, gets almost all the air-time, as compared to STA21. Therefore, the uplink TCP throughput from STA22 is close to that in the isolated case, while the throughput of STA21 drops enormously.

In the other two cases shown in Table VIII, in addition to the above explanation holding during uplink transmissions, there is interference between downlink TCP ACK transmissions from AP1 to STA12 and from AP2 to STA21. Since TCP ACKs are cumulative, the loss of a TCP ACK is not as significant as that of a TCP data packet, thus, downlink mutual interference between APs has a smaller effect on the performance (than in the case of downlink TCP flows).

3) Together Throughputs: With ADWISER: In Fig. 13, for all four STAs performing long, uplink, *iPerf* TCP transfers, we show the expected throughputs and utility from an analytical model (blue bars), the throughputs and utility with default operation of the network (red bars), and those with ADWISER (green bars). With ADWISER, the utility improves drastically from 4.38 to 12.79, while the analytical estimate is 13.03. The optimum fraction of slots in which {STA11, STA21} should be scheduled is 45%, 15% for {STA11, STA22}, and the remaining slots for {STA12, STA22}. ADWISER improves performance by not scheduling together the STAs associated with the same AP, achieves spatial reuse by scheduling together the independent links {STA11, STA22}, and the weakly dependent links {STA11, STA21} and {STA12, STA22}.

C. Downlink Streaming Video: The 2AP-2STA Network

On the subnetwork comprising AP1-STA12 and AP2-STA21 (see Fig. 14), we set up a Microsoft Teams call between the server on the "Internet" and STA21. On this Teams call, the "user" at the server plays a video clip to the user at STA21. At the same time we set up downlink TCP transfers from the server to each STA.

The Microsoft Teams application is built on top of the WebRTC environment [33]. Some of the video performance plots we show have been captured from WebRTC. WebRTC carries the realtime video over UDP. During the experiment,

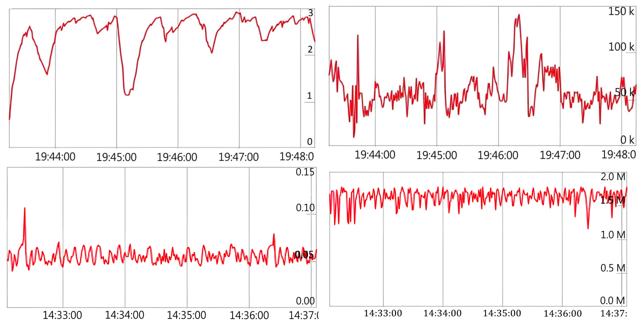

Fig. 15. Video jitter (sec; left panels) and rate (Kbps or Mbps; right panels) vs. time, without ADWISER (top panels), with ADWISER (bottom panels).

TABLE IX

THROUGHPUTS OF THE TCP FLOWS IN THE EXPERIMENTS ON THE TESTBED IN FIG. 14

	STA12 (Throughput in Mbps)	STA21 (Throughput in Mbps)
Isolated	95.6± 1.14	91.5±1.03
Together (Without Teams & without ADWISER)	23.8±2.48	24.3±2.67
Together (With Teams & with ADWISER)	38.7±2.81	36.9±3.07

we observed that, with the default settings, the video packets are not marked with any DSCP values by the application. This was also the observation we made at the student residence hall; Sec. I. Thus, due to the absence of DSCP values, the packets are served as best effort traffic at the AP.

In this experiment, we demonstrate that ADWISER can provide scheduling priority between a UDP flow and a TCP flow being handled by the best-effort service in the AP.

a. Only downlink video over Teams: We begin with video transfer over the MS Teams call to STA 21 without any TCP traffic, and without ADWISER. This experiment exhibits the default performance of the video, when treated as best-effort traffic by AP2. We observed a mean jitter of around 50 ms, and an average video rate of about 1.6 Mbps. As defined in [34], the mean jitter is the EWMA of the absolute deviation of the sender-to-receiver delay of successive packets.

b. Downlink video and TCP traffic, without ADWISER: In this experiment, the video transfer over the Teams call happens while there is a downlink *iPerf* TCP flow over both the links, i.e., AP1-STA12 and AP2-STA21, without ADWISER. We observe spikes in video packet delay jitter values going upto 3 seconds, and the video rate falling below 150 Kbps; see Fig. 15, top two plots. The poor performance of the video is due to interference from TCP packets from AP1 to STA12. The video playout keeps stalling, and is not useful from the user's point of view. A video of this demonstration is available at https://cni.iisc.ac.in/projects/adwiser/. The throughputs of the two TCP flows are shown in the middle row of data in Table IX.

c. Downlink video and TCP traffic, with ADWISER: Next, with the video clip playing over the Microsoft Teams call, and the two TCP flows, as above, ADWISER controls (as explained in Sec. IV-H) were enabled. As shown in the two bottom plots in Fig. 15, the video jitter reduces to an average of about 50 ms and the average rate goes up to 1.6 Mbps, as they were without the two TCP flows. In Table IX, the third row of data, shows that, not only did the video performance become what it was without the TCP flows, the throughputs of the TCP flows improved by 50% to 60%.

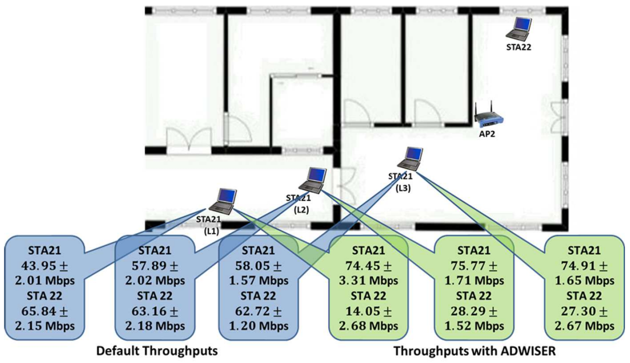
STA22
AP2
STA21 (L1)
STA21 (L2)
STA21 (L3)
STA21 43.95 ± 2.01 Mbps STA 22 65.84 ± 2.15 Mbps
STA21 57.89 ± 2.02 Mbps STA 22 63.16 ± 2.18 Mbps
STA21 58.05 ± 1.57 Mbps STA 22 62.72 ± 1.20 Mbps
STA21 74.45 ± 3.31 Mbps STA 22 14.05 ± 2.68 Mbps
STA21 75.77 ± 1.71 Mbps STA 22 28.29 ± 1.52 Mbps
STA21 74.91 ± 1.65 Mbps STA 22 27.30 ± 2.67 Mbps
Default Throughputs
Throughputs with ADWISER

Fig. 16. 1 AP, 2 STAs, 75 Mbps rate guarantee to STA21: default (blue callouts), TS-ABR-PFS(RG) (green callouts).

D. Downlink TCP Flows: Rate Guarantee to One Flow

We now employ the PF(RG) algorithm shown in Eqn. (12) and Eqn. (13). The parameter b has to be much smaller than a since these two iterations comprise a two-time scale stochastic approximation, with the the index-bias being updated at a slow time-scale.

We consider the two STAs, STA21 and STA22, associated with AP2 in the testbed (see Sec. II). STA22 was kept fixed; STA21 was placed at three locations, successively farther away from AP2.

In Fig. 16, the blue callout boxes show the downlink TCP throughputs at the three locations. The throughput to STA21 drops from 58.05 Mbps to 43.95 Mbps, while that of STA22 62.72 Mbps goes up to 65.84 Mbps as per the default scheduling policy in the AP.

We, however, want to ensure that even as STA21 moves from location to location it is provided a throughput guarantee of 75 Mbps at all these locations. We enabled the TS-ABR-PFS(RG) algorithm (Sec.IV-G) in ADWISER. We observe that at the successively farther locations, STA21 gets throughputs of 74.91, 75.77, and 74.45 Mbps while, STA22 records throughputs of 27.30, 28.29, and 14.05 Mbps.

The rate region at the middle location is shown in Fig. 9. Evidently, to provide a throughput of 75 Mbps to STA21, the throughput of STA22 has to reduce to below 30 Mbps, and further reduce to a little over 14 Mbps at the farthest location. We notice that the sum TCP throughput achieved by STA21 and STA22 at the middle location, with default Wi-Fi operation (middle blue callout box), is $57.89 + 63.16 = 121.05$ Mbps, whereas the sum throughput with TS-ABR-PFS(RG) is $75.77+28.29 = 104.06$ Mbps, about 14% smaller. This is due to the overheads discussed in Sec. V-A, due to which the rate region with ADWISER is smaller.

Fig. 17 shows the time series of the STA throughputs, and the index-bias, $\nu_{STA21}(k)$, at the middle location. The STA21 throughput fluctuates around 75 Mbps, the STA22 throughput varies between a little below 20 Mbps and 40 Mbps, averaging to 28.29 Mbps. The index-bias has an average of 0.0322, fluctuating to higher values whenever (due to channel variations) the STA21 throughput experiences a drop. As STA21 moves farther away, the index-bias will increase to compensate for a smaller rate region.

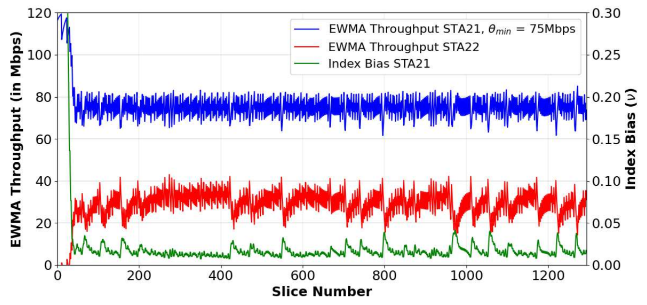

Fig. 17. 1 AP, 2 STAs, 75 Mbps rate guarantee to STA21: TS-ABR-PF(RG) time-series at Location 2.

VII. Conclusion

We have demonstrated that ADWISER's overlay queueing and scheduling approach, for TCP flows, provides network utility close to optimum, can provide throughput guarantees to designated AP-STA links, and manages the latency of downlink UDP flows that share the best-effort access category. In Sec. V we have discussed various implementation challenges and, therein, suggested directions for future research.

MultiAP coordination for Coordinated OFDMA remains to be explored. Such techniques will certainly be enabled by the standardization of a measurement and control plane between the APs and a central controller. We are exploring the management of uplink real-time UDP flows (such as for industrial sensing and robot control) by time synchronizing these devices with ADWISER, and releasing data from them in designated minislices within the time-slicing framework.

Acknowledgment

The authors are thankful to Nadeem Akhtar and Vijayalakshmi Chetlapalli for useful discussions.

References

[1] M. Hegde et al., "Experiences with a centralized scheduling approach for performance management of IEEE 802.11 wireless LANs," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 648–662, Apr. 2013.

[2] A. Sunny et al., "A generic controller for managing TCP transfers in IEEE 802.11 infrastructure WLANs," *J. Netw. Comput. Appl.*, vol. 93, pp. 13–26, Sep. 2017.

[3] V. Sevani, P. Saravanan, S. V. R. Anand, J. Kuri, and A. Kumar, "Time-slicing high throughput WiFi networks using centralized queueing and scheduling," in *Proc. 16th ACM Workshop Wireless Netw. Testbeds, Experim. Eval. CHaracterization*, Oct. 2022, pp. 53–60.

[4] Z. Shen, B. Li, M. Yang, Z. Yan, X. Li, and Y. Jin, "Research and performance evaluation of spatial reuse technology for next generation WLAN," in *Wireless Internet*. Cham, Switzerland: Springer, 2019, pp. 41–51.

[5] M. S. Afaqui, E. Garcia-Villegas, E. Lopez-Aguilera, G. Smith, and D. Camps, "Evaluation of dynamic sensitivity control algorithm for IEEE 802.11ax," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2015, pp. 1060–1065.

[6] M. Nurchis and B. Bellalta, "Target wake time: Scheduled access in IEEE 802.11ax WLANs," *IEEE Wireless Commun.*, vol. 26, no. 2, pp. 142–150, Apr. 2019.

[7] E. Khorov, I. Levitsky, and I. F. Akyildiz, "Current status and directions of IEEE 802.11be, the future Wi-Fi 7," *IEEE Access*, vol. 8, pp. 88664–88688, 2020.

[8] W. Ahn, "Novel multi-AP coordinated transmission scheme for 7th generation WLAN 802.11be," *Entropy*, vol. 22, no. 12, p. 1426, Dec. 2020. [Online]. Available: https://www.mdpi.com/1099-4300/22/12/1426

[9] D. Nunez, F. Wilhelmi, S. Avallone, M. Smith, and B. Bellalta, "TXOP sharing with coordinated spatial reuse in multi-AP cooperative IEEE 802.11be WLANs," 2021, *arXiv:2112.00515v2*.

[10] G. Lacalle, I. Val, O. Seijo, M. Mendicute, D. Cavalcanti, and J. Perez-Ramirez, "Multi-AP coordination PHY/MAC management for industrial Wi-Fi," in *Proc. IEEE 27th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2022, pp. 1–8.

[11] C. Deng et al., "IEEE 802.11be Wi-Fi 7: New challenges and opportunities," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2136–2166, 4th Quart., 2020.

[12] D. Nunez, M. Smith, and B. Bellalta, "Multi-ap coordinated spatial reuse for Wi-Fi 8: Group creation and scheduling," 2023, *arXiv:2305.04846v1*.

[13] H. J. Kushner and P. A. Whiting, "Convergence of proportional-fair sharing algorithms under general conditions," *IEEE Trans. Wireless Commun.*, vol. 3, no. 4, pp. 1250–1259, Jul. 2004.

[14] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control, and Stochastic Networks Perspective*. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[15] S. Seytnazarov, J.-G. Choi, and Y.-T. Kim, "Enhanced mathematical modeling of aggregation-enabled WLANs with compressed BlockACK," *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1260–1273, Jun. 2019.

[16] S. Zehl, A. Zubow, and A. Wolisz, "Hotspot slicer: Slicing virtualized home Wi-Fi networks for air-time guarantee and traffic isolation," in *Proc. IEEE 18th Int. Symp. A World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2017, pp. 1–3.

[17] R. Bhattacharyya et al., "QFlow: A learning approach to high QoE video streaming at the wireless edge," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 32–46, Feb. 2022.

[18] *MCS Table and How to Use It*. Accessed: Sep. 21, 2023. [Online]. Available: https://wlanprofessionals.com/mcs-table-and-how-to-use-it

[19] S. Ray and D. Starobinski, "On false blocking in RTS/CTS-based multihop wireless networks," *IEEE Trans. Veh. Technol.*, vol. 56, no. 2, pp. 849–862, Mar. 2007.

[20] A. Kumar, D. Manjunath, and J. Kuri, *Wireless Networking*. San Francisco, CA, USA: Morgan-Kaufmann, Apr. 2008.

[21] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. (1996). *TCP Selective Acknowledgment Options*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2018

[22] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky. (2000). *An Extension to the Selective Acknowledgement (SACK) Option for TCP*. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2883

[23] V. S. Borkar, *Stochastic Approximation: A Dynamical Systems Viewpoint*. New Delhi, India: Hindustan Book Agency, 2008.

[24] A. L. Stolyar, "On the asymptotic optimality of the gradient scheduling algorithm for multiuser throughput allocation," *Oper. Res.*, vol. 53, no. 1, pp. 12–25, Feb. 2005.

[25] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications*. Cham, Switzerland: Springer, 2003.

[26] A. Gopalan, C. Caramanis, and S. Shakkottai, "On the value of coordination and delayed queue information in multicellular scheduling," *IEEE Trans. Autom. Control*, vol. 58, no. 6, pp. 1443–1456, Jun. 2013.

[27] M. S. Bazaara, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*. Hoboken, NJ, USA: Wiley, 1993.

[28] S. Mandelli, M. Andrews, S. Borst, and S. Klein, "Satisfying network slicing constraints via 5G MAC scheduling," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 2332–2340.

[29] A. Kumar and R. Sundaresan, "Utility optimal scheduling with a slow time-scale index-bias for achieving rate guarantees in cellular networks," 2024, *arXiv:2408.09182v1*.

[30] R. Krashinsky and H. Balakrishnan, "Minimizing energy for wireless web access with bounded slowdown," in *Proc. 8th Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2002, pp. 119–130.

[31] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," in *Proc. IEEE INFOCOM Conf. Comput. Commun. 17th Annu. Joint Conf. IEEE Comput. Commun. Societies. Gateway 21st Century*, vol. 2, Mar. 1998, pp. 533–539.

[32] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Trans. Signal Process.*, vol. 66, no. 20, pp. 5438–5453, Oct. 2018.

[33] B. Sredojev, D. Samardzija, and D. Posarac, "WebRTC technology overview and signaling solution design and implementation," in *Proc. 38th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2015, pp. 1006–1009.

[34] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. (2003). *RTP: A Transport Protocol for Real-Time Applications*. [Online]. Available: https://www.rfc-editor.org/info/rfc3550