Time-Slicing High Throughput WiFi Networks Using Centralized Queueing and Scheduling

Vishal Sevani, Purushothaman Saravanan, S. V. R. Anand, Joy Kuri, Anurag Kumar
ECE and ESE Departments, Indian Institute of Science (IISc), Bengaluru, India

ABSTRACT

We study fine grained (10s of ms) *overlay time-slicing*, and centralized queuing and scheduling, for the performance management of "high throughput" (HT) IEEE 802.11 standards, where cochannel interference reduces PHY rates and aggregation, causing poor performance. In overlay time-slicing, interference between AP-STA (Access Point and associated Station) links is eliminated by queuing downlink packets in a scheduler, between the wireline network and the APs, and releasing packets to a set of AP-STA links only in their time-slice. This can manage downlink and uplink FTP and HTTP transfers, and downlink packet voice traffic.

We utilize a stochastic approximation based closed-loop mechanism that releases only as much data in a time-slice as can be "served," so that the AP-STA links mapped to that time-slice are inactive at the end of their time-slice, thus, eliminating cochannel interference.

Fine-grained overlay time-slicing is demonstrated on an experimental network with two cochannel AP-STA pairs, a setting that we see in our campus WiFi network. In our approach, even for small time-slices (20ms to 50ms), for downlink and uplink TCP bulk transfers, in spite of the scheduler working with partial information, the time-slice boundaries are respected, and performance is close to network utility optimal. Fine-grained time-slicing reduces HTTP access delay, prevents TCP connections over the wide area network from reacting to the path interruptions, and also permits slicing of applications such as interactive packet voice.

ACM Reference Format:
Vishal Sevani, Purushothaman Saravanan, S. V. R. Anand, Joy Kuri, Anurag Kumar, ECE and ESE Departments, Indian Institute of Science (IISc), Bengaluru, India. 2022. Time-Slicing High Throughput WiFi Networks Using Centralized Queueing and Scheduling. In *16th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & CHaracterizatiion (WiNTECH '22), October 17, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3556564.3558237

1 INTRODUCTION

IEEE 802.11 WiFi networks are the access networks of choice in enterprises and academic campuses. However, they are performance-limited by uncontrolled internal interference between cochannel devices, so that the true benefits of High Throughput (HT) mechanisms are diminished, in practice.

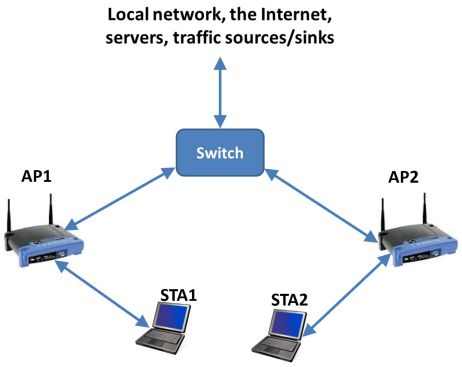

Figure 1: A 2AP-2STA cochannel subnetwork.

Consider the cochannel WiFi subnetwork in Fig. 1, arising in a WiFi installation even after automatic channel selection and power control have optimized spatial reuse, to the extent they can. In Sec. 3, we provide examples of scenarios that arise in an operational network on our academic campus. The cochannel APs in Fig. 1 are not in carrier-sense (CS) range. The transmit powers and internode distances are such that, for the best possible bit rate between the APs and their STAs, there is excessive interference from each AP to the STA associated with the other AP. Fig. 2 shows the measured TCP file transfer throughputs from a server on the local network to the STAs, when the two AP-STA links are *isolated*, or *together* (i.e., the default operation). An extensive experimental study of this network scenario is given in Sec. 4.

From the rough sketch on the (θ_1, θ_2) axes, in Fig. 2, we see that (τ_1, τ_2) reflect an inefficient sharing of the wireless medium. Given that only packet scheduling is in our control, for the network in Fig. 1 , it would be desirable that the together throughputs remain on the line joining the points $(\beta_1, 0)$ and $(0, \beta_2)$, preferably $(\frac{\beta_1}{2}, \frac{\beta_2}{2})$, the proportional fair operating point. Note that, general networks of AP-STA links (such as the ones discussed in Sec. 3) will have more complex throughput regions, and the desired operating point will have to be determined in each case.

We investigate a centralised, *overlay* approach that schedules (in this example) AP1-STA1 and AP2-STA2 in alternate time-slices, aiming to achieve "isolated" performance over each time-slice, thereby reducing the deleterious effects of contention and collisions. No changes are made to the APs and STAs; nodes that are scheduled within a time-slice, continue to use CSMA/CA for channel access. Hegde et al. [1] introduced this idea for resolving several performance anomalies in single AP IEEE 802.11g WiFi networks. The

WiNTECH '22, October 17, 2022, Sydney, NSW, Australia

ACM ISBN 978-1-4503-9527-4/22/10…$15.00
https://doi.org/10.1145/3556564.3558237

Vishal Sevani, Purushothaman Saravanan, S. V. R. Anand, Joy Kuri, Anurag Kumar
ECE and ESE Departments, Indian Institute of Science (IISc), Bengaluru, India

Setting	Mbps±95% CI	
	AP1-STA1 θ_1	AP2-STA2 θ_2
Isolated	β_1 : 79.6 ± 0.99	β_2 : 103.5±1.03
Together	τ_1 : 21.7 ± 0.62	τ_2 : 25.7 ± 0.56

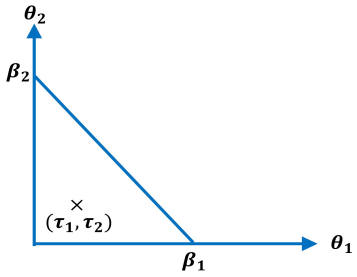

Figure 2: TCP downlink throughputs for "isolated" and "together" operation of AP1-STA1 and AP2-STA2.

idea was further developed by Sunny et al. [2], for managing TCP controlled file transfers in multiAP IEEE 802.11g networks.

In this paper, we consider centrally controlled time-slicing for "high throughput" (HT) IEEE 802.11 networks. We focus on *fine-grained allocation of airtime* (20ms-50ms) (to sets of AP-STA links), whereas Hegde et al. [1], and Sunny et al. [2] dealt with IEEE 802.11g networks, and 1000 ms time-slices. HT IEEE 802.11 networks use higher order modulation schemes and packet aggregation, both of which adapt *downwards* under high packet loss (see Sec. 4; also see [3]). Fine-grained time-slicing enables short scheduling frames, thereby reducing access delays for interactive traffic, and preventing time-outs in wide-area TCP connections.

The challenge to designing and implementing fine grained (say, 20 ms) time-slices is to prevent (1) data overflow between time-slices, as the ensuing interference diminishes the performance gains, and (2) underflow of data served within a slice, as that diminishes efficiency.

1.1 Related Literature

To the best of our knowledge, the literature on slicing does not consider issues in multi-AP networks. Instead, the focus is on a single AP/device, and queuing and scheduling strategies to accommodate multiple flows (or virtual WLANs) sending traffic through the AP/device, such that each flow's (or virtual WLAN's) performance demands are met [4–7]. This is not enough to assure good performance, because in practice, a STA experiences interference from cochannel APs in the neighbourhood; this aspect is not considered at all.

[4] considers a single AP through which several traffic flow-aggregates pass. Each aggregate demands a percentage of the total airtime at the AP. The authors discuss a queueing and scheduling mechanism based on Deficit Round Robin (DRR) to allocate airtime to each aggregate. [5] proposes a hypervisor running on the AP. The hypervisor is used to create multiple slices, each with its buffer and EDCA parameters. [6] proposes a new architecture for "WiFi sharing Customer Premises Equipment (CPE)" – a device that allows a physical AP to support multiple virtual WLANs (V-WLAN), with each V-WLAN being allotted configurable resources. [7] considers a Click modular router placed inside an AP and proposes a queueing structure to share the bytes transmitted from each V-WLAN according to predefined proportions.

[8] considers a multi-AP WLAN in which resources must be sliced among several tenants (Virtual Network Operators [VNO-s]) and the problem is to share the AP resources such that each VNO's SLA can be met. A central controller interacts with an agent in each AP to collect information on prevailing traffic and network conditions, and dynamically adjusts the weights of a Weighted Airtime Deficit Round Robin (WADRR) scheduler. [9] addresses the same problem, but focuses on a single AP only. Given traffic arrival statistics and target delays for slices, the authors provide an algorithm to calculate the airtime that each slice needs. The implicit assumption in both [8, 9] is that if an AP is allocated some resources, then STAs associated with it will automatically experience adequate QoS. But the problem is that a STA can experience interference from nearby co-channel APs that are transmitting packets to their own STAs.

[1] and [2] were the first to introduce the idea of centralized time-slicing to manage various performance anomalies in single-AP and multiAP WLANs. The authors evaluated their proposals in an 802.11g network and showed that performance could be managed very well. Our work in this paper follows that in [1] and [2], but solves the problem of *fine-grained (10s of ms) time-slicing for HT IEEE 802.11 networks*, which also, automatically, solves the problem of time-slicing TCP connections over the WAN, and the latency performance of HTTP and packet voice. Fine-grained time-slices also ensure that the STAs do not idle for long, thus triggering the Power Save Mode (PSM), and thereby making time-sliced operation unpredictable.

2 CONTRIBUTIONS OF THIS WORK

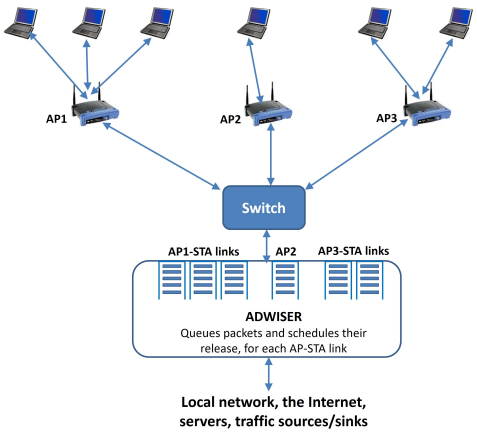

Figure 3: Queueing and time-slicing with ADWISER.

Figure 3 shows several APs and their associated STAs with a device called "ADWISER" (Advanced WiFi Service Enhancer, see also [1], [2]) inserted in the path of all packet flows between the traffic sources/sinks and the STAs. In ADWISER, there are successive, periodic *frames* (see Fig. 9) which contain *time-slices*, to each

of which some of the AP-STA pairs are mapped, such that each AP-STA pair is mapped to at least one time-slice. When managing *downlink* (AP to STA) (resp., *uplink* (STA to AP)) TCP controlled transfers, ADWISER queues the TCP data packets (resp., TCP ACK packets) destined for the STAs, and releases these to the corresponding AP only in a time-slice allotted to the AP-STA pair. This isolates in time the activity in various parts of the WiFi network that interfere under the default operation (recall Fig. 2). Such time-sliced operation, however, introduces *access delay* if a user makes a file transfer or HTTP request when their STA is not scheduled, thus, motivating the need for *fine-grained* time-slicing.

In the present work, we make the following contributions in the context of the ADWISER time-slicing architecture:
(1) In Sec. 3 we use observations from the controller of a professionally installed WiFi network, on our campus, to show that there exist sets of AP-STA links that experience internal cochannel interference, which limits their performance. The 2AP-2STA network in Fig. 1 is a simple such configuration, that serves as our experimental test-bed.
(2) In Sec. 4 we present a detailed experimental study of the 2AP-2STA network in Fig. 1, to explain the poor performance, specially in the context of the HT mechanisms of IEEE 802.11n.
(3) In Sec. 5 for bulk TCP transfers, we propose the TS-ABR-QD (Time Slicing - Adaptive Batch Release - Queue Drain) algorithm, which we implement by using the Robbins-Monro ([10] [11]) stochastic approximation technique to develop a mechanism to adaptively release TCP data packets or TCP ACKs to the AP of the AP-STA pair to which a time-slice is allotted.
(4) In Sec. 6, we provide an experimental study that demonstrates the benefits of time-slicing, with TS-ABR-QD, for bulk TCP transfers (downlink and uplink, from the LAN or from the WAN), HTTP traffic, or UDP packet voice together with bulk TCP. The idea of *dynamic* time-slicing is introduced in the context of intermittent interactive traffic such as HTTP.

3 DOES INTERNAL COCHANNEL INTERFERENCE OCCUR IN PRACTICE?

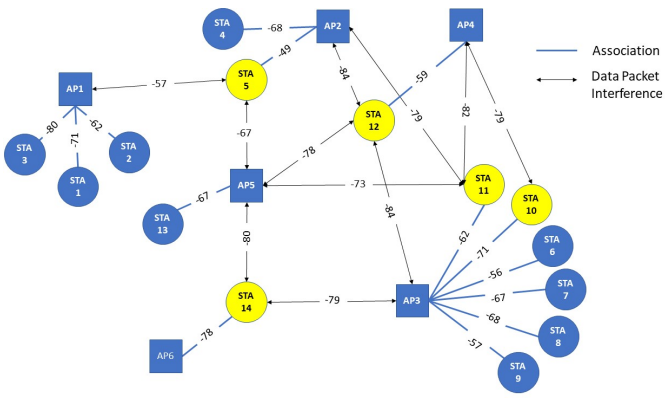

Figure 4: Cochannel fragment of a WiFi network; student housing building; 5 GHz band, 20 MHz ch. 149.

We studied a professionally installed and managed WiFi network in one of our student housing buildings: a four floor building, 400 single-occupancy rooms, with 120 APs (11ac, 2.4 GHz and 5 GHz, downlink MU-MIMO disabled), well-known manufacturer, roughly 30 APs per floor. A WiFi controller uses special purpose hardware in the APs to measure the received signal strength (RSS) from other devices in the network, and the PHY rates actually achieved between associated AP-STA pairs. In Fig. 4, STA12 is associated with AP4 at −59 dBm RSS, which, without interference, would yield a PHY rate of 156 Mbps, requiring a 29 dB SINR [12]. However, taking into account the interference from STA 10 and 11 and consulting [12], we predict an uplink PHY rate of 78 Mbps; this agrees roughly with the observed value of 76.2 Mbps (Tbl. 1). On the downlink, STA12 receives interference from APs 2, 5, and 3; further, the transmit powers vary because of power control based on the signal strengths of one AP as measured at other APs. We surmise that, at the time of measurement, the interfering APs were transmitting at higher power, leading to the next lower PHY rate in [12], viz., 52 Mbps, in rough agreement with the observed value.

Table 1: Expected and observed PHY rates in Figure 4; †: from 11ac MCS Table [12]; *: from WLAN controller.

Affected STAs	Associated RSS (dBm)	Expected PHY Rate (Mbps)†	Downlink PHY rate (Mbps)*	Uplink PHY rate (Mbps)*
STA5	-49	156	70.6	74.1
STA11	-62	130	72.0	78.5
STA12	-59	156	50.5	76.2

The network managers had "optimised" the network with interventions such as power adjustments at the APs, encouraged reassociations, and band steering. Already, automatic channel selection would have aimed to reduce interference and encourage spatial reuse. The situation shown in Fig. 4 was observed despite these optimisations. We observe that the AP4→STA12 link could be expected to get much better average performance, if APs 2, 3, and 5 were prevented from being able to transmit at the same time as when AP4 was transmitting to STA12. On the other hand, links AP1→STA3, AP2→STA4, and AP3→STA6 could be scheduled together, thus increasing spatial reuse. Such scheduling in time needs to be done with the aim of achieving better overall network performance. This is the objective of the overlay time-slicing and packet scheduling we study in this paper.

4 EXPERIMENTS ON A LAB TESTBED

Following the observations in Sec. 3, the network in Fig. 1 is a typical setting in which uncontrolled, internal cochannel interference occurs. Our 2AP-2STA lab test-bed is shown in Figure 5. The two cochannel APs and the two (stationary) STAs are positioned such that STA1 and STA2 would then be at their respective "cell" edges. The two APs are connected via an Ethernet switch to a server, which is the other endpoint of all applications initiated in the STAs, and which also emulates the WAN propagation delay and effective bit rate. The server, the switches, and ADWISER are in the same room as AP1.

Vishal Sevani, Purushothaman Saravanan, S. V. R. Anand, Joy Kuri, Anurag Kumar
ECE and ESE Departments, Indian Institute of Science (IISc), Bengaluru, India

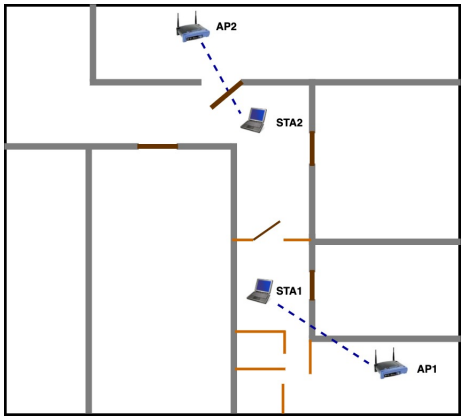

Figure 5: Laboratory layout of the setup in Fig. 1.

Equipment: APs: IEEE 802.11ax, dual-band, Tx-Rx 2x2, antenna gains: 3.8 dBi (2.4 GHz), 4.5 dBi (5 GHz); STAs: Intel i5/8GB RAM laptops with Ubuntu 20.04. Our "*sniffers*" were just laptops configured in monitor mode.
Network setup: All our experiments were done with IEEE 802.11n, Channel 100 in the 5 GHz band, or Channel 13 in the 2.4 GHz band. The physical layer used the 20 MHz bandwidth with two spatial streams. The AP-STA distances were 6-7 meters. The AP-AP distance was about 30 m. There were no active APs on Channel 13 in the 2.4 GHz band, nor on Channel 100 in the 5 GHz band. Bulk TCP transfers were emulated using *iperf*. The server was running Linux (Ubuntu 20.04) in which the latest version of TCP CUBIC was implemented.

IEEE 802.11ax can be viewed as a high performance extension to IEEE 802.11n, with uplink and downlink MU-MIMO and OFDMA being major new features, that could help in managing interference. Without interAP coordination, however, the 11ax MU-MIMO-OFDMA cannot consistently mitigate interAP interference. Hence, we have considered only 11n in our work.

The throughputs shown in Fig. 2 were obtained from this setup. The confidence interval (CIs) capture the variability over several repetitions.

With ideal time-slicing, during a time-slice, an AP-STA link will obtain its isolated throughput (i.e., $\beta_i, i = 1, 2$). If the fraction of air-time allotted to APi-STAi is $\alpha_i, i = 1, 2, \alpha_1 + \alpha_2 = 1$, we evaluate the network performance with the *log utility*

$$\ln \alpha_1\beta_1 + \ln \alpha_2\beta_2$$

It follows that, utility optimal scheduling will give the throughputs $\frac{79.6}{2}$ Mbps and $\frac{103.5}{2}$ Mbps. Note that without the *log* function we would evaluate network utility as the *sum throughput* which would be optimized by giving all airtime to the AP-STA pair with the higher isolated throughput, an unfair operating regime.

4.1 Packet Loss and Consequences

To understand the performance in Fig. 2, we carried out pairwise RSS measurements in the 2AP-2STA lab setup; the results are shown in Tbl. 2. 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 sniffer.

Table 2: RSS measurements (in dBm ± 95% CI) for the setup in Fig. 5; cell (i, j) is RSS at i from j.

	AP1	AP2	STA1	STA2
AP1	-	-87.1±0.81	-64.1±0.93	-75.1±0.7
AP2	-88.2±1.00	-	-76.8±1.31	-62.9±0.8
STA1	-64.6±0.88	-77.1±0.86	-	-61.4±0.97
STA2	-74.2±1.22	-62.6±0.78	-62.3±0.71	-

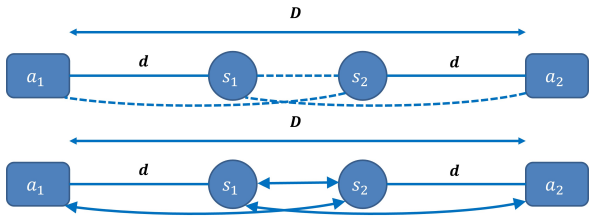

Figure 6: Carrier sense and interference relationships between the devices in the 2AP-2STA network. Here, a_i is APi, and s_i is STAi. See text for explanation.

From the measurements in Tbl. 2, the two diagrams in Figure 6 show the pairs of nodes in (i) carrier-sense (CS) range (dashed lines; two devices are in CS range if the RSS between them is greater than −82 dBm), and (ii) data packet interference range (solid lines with arrows) for the network shown in Figure 5. The solid lines without arrows are the AP-STA associations. The association and interference can be understood using the tables in [12]. For example, the AP1→STA1 RSS (for 20 MHz and 2 spatial streams) would support an MCS with PHY rate 117 Mbps, without any interference at STA1. This, however, nominally requires an SINR of 20 dBm, which cannot be sustained if either STA2 or AP2 interfere with the AP1→STA1 transmission, giving rise to the solid double-arrow lines between STA2 to STA1 and AP2 to STA1.

Fig. 6 helps understand Fig. 2. With Basic Access (BA), assuming that simultaneous attempts are rare, when an STA is transmitting, the other AP and STA are CS blocked, and the transmission completes successfully. On the other hand, when an AP is transmitting, the other AP is not CS blocked, and can start its own transmission, thereby both the APs' transmissions are corrupted (indicated by the solid arrows between the APs and the STAs in the lower panel of Fig. 6). This results in packet loss.

Could this packet loss lead to a drop in the TCP window, which could, in turn, starve the node AP buffers, not permitting sufficient aggregation? We note that there is no dearth of packets at the AP transmit buffers. We can assert this by watching the TCP *cwnd* at the sender; see Fig. 7. There is hardly any difference between cwnd evolution for isolated and together operation, indicating that the TCP sender does not see significantly more packet losses during together operation. Thus, MAC level retransmissions prevent the TCP sender from noticing TCP packet losses.

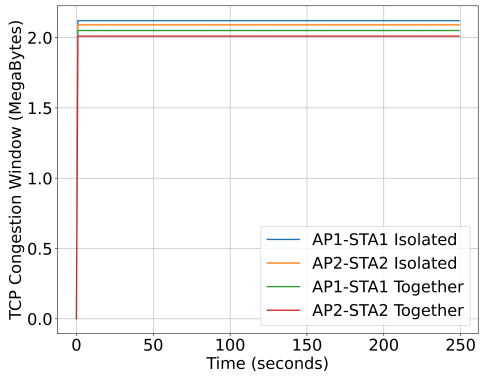

Figure 7: Time series of TCP congestion window (*cwnd*) at 1sec intervals via *iperf*, for isolated and together operation of the AP-STA links, in the 2AP-2STA network.

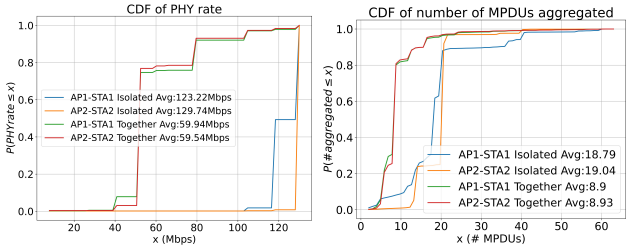

Figure 8: CDFs of PHY rates and number of MPDUs aggregated for isolated and together operation.

However, as a consequence of packet loss, at the node level, the adaptive PHY rate mechanism reduces the supportable PHY rate. The CDFs of PHY rates for together and isolated operation, for both AP-STA pairs, are shown in the left panel of Figure 8. Whereas, for isolated operation the mean PHY rates exceed 120 Mbps, for together operation the mean PHY rates drop to below 60 Mbps. Indeed, an entire AMPDU transmitted by an AP will be lost, due to the loss of the PLCP header that precedes the AMPDU, if this transmission starts when the other AP is already transmitting an AMPDU. Thus, large AMPDUs become detrimental, and, in addition to the PHY rate reduction, the aggregation also reduces (see also [3]). The CDFs of the number of MPDUs aggregated into an AMPDU are shown in the right panel of Figure 8. The mean AMPDU size is close to 19 MPDUs for isolated operation, but drops to below 9 MPDUs in together operation. The combined effects of the steep drops in the PHY rate from the APs to the STAs, and the number of MPDUs in an AMPDU, are the reasons for the poor together TCP throughputs shown in Fig. 2.

In the setup of Fig. 5, RTS-CTS would help to eliminate simultaneous transmissions by the two APs. However, we found that the RTS-CTS mechanism is not used liberally enough. The RTS transmission logic is opaque to us, so we conducted experiments with the RTS threshold set to low values, in APs from two manufacturers, expecting that nearly every AMPDU transmitted by an AP would be preceded by an RTS. Manufacturer 1 APs transmit RTS packets sparingly, with only about 3.7% to 18.5% of the AMPDUs being protected by RTS-CTS, with greater RTS-CTS use for higher MPDU error rates. For APs from Manufacturer 2, the percentages were 17.5% to 36.8%. Thus, neither AP executes the RTS-CTS exchange aggressively enough to "reserve" the wireless medium for the AMPDU to follow.

5 THE TIME-SLICING ALGORITHM

We have shown that, the default operation of our 2AP-2STA testbed network does not eliminate internal cochannel interference in the network, and such interference leads to poor AP-STA TCP throughputs due to reduction in the PHY rates and in aggregation levels, the two mechanisms that enable high throughputs.

We eliminate internal interference by using the overlay time-slicing approach, with centralized queueing and scheduling, as described in Sec. 2. To handle TCP bulk transfers in the network of Fig. 5, each time-slicing frame has two time-slices, in one of which we schedule AP1-STA1, and AP2-STA2 in the other. ADWISER has a queue of packets to be sent to each of the APs: TCP data packets if the TCP transfer is downlink, and TCP ACKs if the transfer is uplink. Packets are released to AP1, say, in the slice for AP1-STA1, and the aim is that, at the end of the time-slice, all activity in the WiFi channel, due to AP1-STA1, comes to an end. The next time-slice starts and packets are released to the AP2-STA2 pair. During this time-slice, any packets arriving at ADWISER, destined to AP1-STA1, are queued in ADWISER, waiting for the next time-slice for AP1-STA1.

In implementing such a system, we had to resolve two main design problems: (1) How do we release data to an AP-STA pair, in its time-slice, so that its throughput is as close as possible to its isolated throughput? (2) How do we determine, in ADWISER, without probing any of the network devices, that the WiFi channel activity due to the release of data/acks has come to an end?

After considerable experimentation (in which we studied constant rate packet release during the slice, with time being left, before the end of the slice, for draining the queues), we designed a data release algorithm, for bulk TCP transfers, that has two design principles:
(1) In each slice, *the data is released in a batch*, so as to promote a high level of aggregation right from the beginning of the time-slice.
(2) The amount of data released must be such that the AP-STA link to which it is released (in possible contention with other AP-STA links assigned to the same slice) can *drain out that data within the time-slice*, while staying busy right up to the end of the slice.

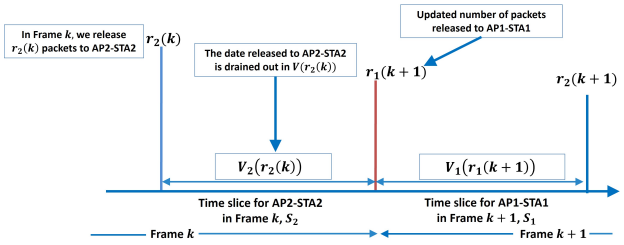

Figure 9: TS-ABR-QD: Frame structure and various processes; the vertical lines labeled $r_i(k)$ indicate variable size batch release at the beginning of time-slices.

For the 2AP-2STA network, Fig. 9 is a depiction of the data release algorithm. We will describe the details for downlink TCP transfers at both AP-STA pairs. Frame $k, k \geq 1$, comprises two time-slices, one for each AP-STA pair. In Frame k, the amount of data released is $r_i(k)$ for APi-STAi. Define $V_i(r)$ as the random time taken by APi-STAi to drain the amount of data r.

With time-slice S_i (milliseconds) assigned to APi-STAi, in a total frame time of $S = S_1 + S_2$, we adapt the amount of data released as follows

$$r_i(k+1) = \left(r_i(k) + a(S_i - V_i(r_i(k))\right)^+ \quad (1)$$

where, as usual, $(x)^+ = \max\{x, 0\}$. This is the Robbins-Monro stochastic approximation algorithm, which, without knowing the structure of the random variable $V_i(r)$, adjusts the release batches to attempt to ensure that the average queue drain time matches the available slice time, S_i. We call this algorithm TS-ABR-QD (Time Slicing with Adaptive Batch Release followed by Queue Drain).

Effect of a: Assume that $\mathbb{E}(V_i(r)|r) = \frac{r}{\beta_i}$ and $\mathbb{V}ar(V_i(r)|r) = \sigma_i^2 r$, where β_i and σ_i are unknown system parameters; β_i can be interpreted as the service rate at which APi-STAi serves the released data. We can show that, for $|1 - \frac{a}{\beta_i}| < 1$, $\lim_{k\to\infty} \mathbb{E}(r_i(k)) = \beta_i S_i$ and $\lim_{k\to\infty} \mathbb{V}ar(r_i(k)) = \frac{\sigma_i^2 \beta_i^2 S_i}{\frac{2}{a} - \frac{1}{\beta_i}}$.

Thus, the downlink TCP throughput of the APi-STAi pair becomes $\frac{\beta_i S_i}{S}$; the network utility converges to $\ln \frac{\beta_1 S_1}{S} + \ln \frac{\beta_2 S_2}{S}$ which is maximised when $S_1 = S_2 = \frac{S}{2}$.

Time-slicing general networks: For bulk TCP transfers in the 2AP-2STA network, there are just two slices in a frame. The formulation for general networks is in [2, Sec. 4]. With automatic channel selection and power control in place, we have found that the number of slices required in a frame is not large. The focus of the present paper is *fine-grained* time-slicing, so that the total frame time can remain small.

Choice of a: For a TCP throughput of, say, 79.6 Mbps (see Fig. 2), the MPDU service rate (β) is about 6.87 packet per millisecond (for 1448 byte TCP payload). By the condition above, we need $0 < a < 2\beta = 13.74$. In the experimental results we report, we have used $a = 1$.

Determining when $V_i(r_i(k))$ ends: ADWISER avoids any communication with the APs and the STAs, and determines the end of $V_i(r_i(k))$, i.e., the drain time in the k^{th} time-slice for APi-STAi, with the information it has. For downlink TCP transfers, TCP data packets are released to the AP-STA pair, and ADWISER waits until the corresponding TCP ACKs have passed through it, on the way back to the server. For uplink TCP transfers, TCP ACKs are released to the AP-STA pair, and the end of the drain time is estimated as the time taken for the corresponding number of bytes worth of TCP data packets to pass through ADWISER on the way back to the server.

6 PERFORMANCE OF TS-ABR-QD

In this section we report the performance of TS-ABR-QD for bulk TCP transfers and HTTP transfers. We consider $S_1 = S_2 = 100$ ms or 20 ms. Smaller time-slices are better for reducing HTTP response times, and also voice packet latency, if voice is also time-sliced.

6.1 Downlink TCP Transfers

Table 3: TS-ABR-QD: Downlink TCP throughputs.

Setting	θ_1 (Mbps ±95% CI)	Avg. V_1 (ms)	θ_2 (Mbps ±95% CI)	Avg. V_2 (ms)	Utility
Isolated	79.6 ± 0.99	-	103.5 ± 1.03	-	Bound: 7.63
Together	21.7 ± 0.62	-	25.7 ± 0.56	-	6.32
S_i=100ms	38.3 ± 0.38	99.64	50.9 ± 0.33	99.28	7.59
S_i=20ms	36.6 ± 0.35	19.65	47.5 ± 0.15	19.51	7.46
WAN S_i=20ms	34.9 ± 0.63	19.6	46.7 ± 0.11	19.53	7.4

Tbl. 3 shows the isolated and together throughputs for downlink bulk TCP transfers (see also Fig. 2). The utility bound is obtained as $\ln \frac{79.6}{2} + \ln \frac{103.5}{2} = 7.63$ (up to two decimal places); this bound would be achieved if, in its time slice, each AP-STA pair obtained its isolated throughput, *over the entire time-slice*. With TS-ABR-QD, and $S_i = 100$ ms, the TCP throughputs (averaged over the time-slices) are 1 to 1.5 Mbps less than the isolated throughputs, leading to a network utility of $7.59 (< 7.63)$. This reduction is due to inaccuracies in determining the end of $V_i(r_i(k))$ in ADWISER (see Eqn. (1)), and the resulting inefficiency in utilising the 100 ms time-slice. Shown also are the means of $V_i(r_i(k))$, as determined at ADWISER. For $S_i = 20$ ms, the TCP throughputs are lower, due to the larger effect of inaccuracy. Yet, the utility is 7.46, still substantially larger (note that we have log utility) than 6.32, the utility with the default network performance.

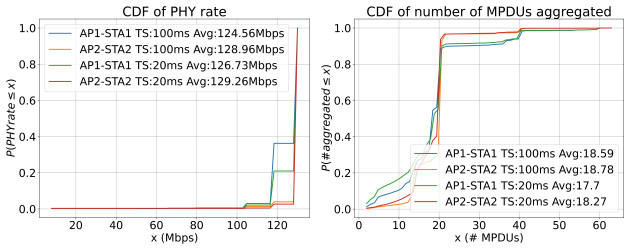

Figure 10: CDF of PHY rates and number of MPDUs aggregated, for 100ms and 20ms time-slice.

Fig. 10 shows the CDFs of PHY rates and aggregation sizes with TS-ABR-QD, and must be compared with Fig. 8. With time-slicing, the PHY rates remain above 120 Mbps and there are, on the average, close to 19 MPDUs in the AMPDUs. Thus, 100 ms or 20 ms time-slices can both achieve high utility, if they fully utilise each time-slice.

We emulated the WAN environment by using the *netem* facility in the Linux based server to create a propagation delay of 75 ms, each way, and a WAN bottleneck bitrate of 200 Mbps, each way. With small time-slices, such as 20 ms, the performance with the WAN is close to that without the WAN. This observation is significantly different from those in [2], where the 1000 ms time-slices, though simplifying the time-slicing, required the use of a TCP proxy and the F-RTO mechanism (*https://datatracker.ietf.org/doc/html/rfc5682*), to prevent the TCP sender from timing out during the long interruptions due to time-slicing.

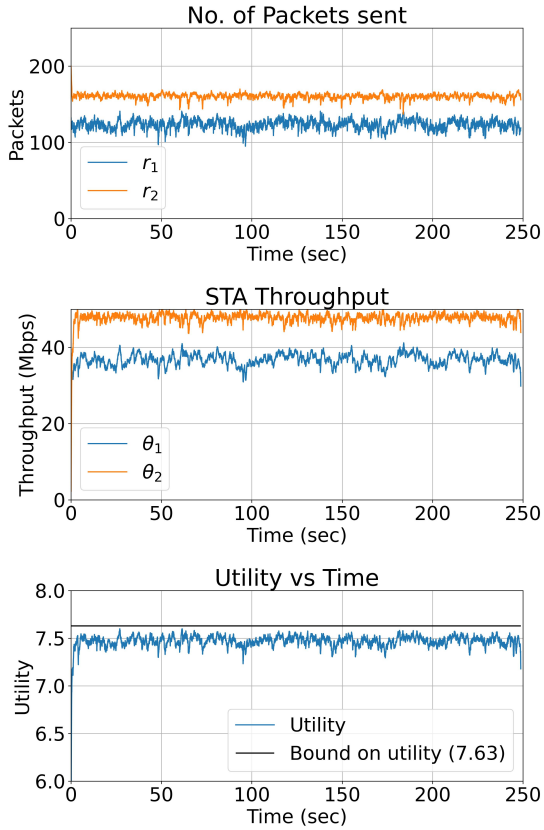

Figure 11: 2AP-2STA network, downlink TCP transfers, controlled with ADWISER TS-ABR-QD: Time-series of the number of packets released in each time-slice, the average throughputs, and the average utility, for 20 ms time-slice.

Fig. 11 shows the time-series of some processes in TS-ABR-QD. The top panel shows the number of TCP data packets ADWISER releases in each slice, for each AP-STA pair. The middle panel shows the TCP throughputs for each AP-STA pair (exponentially weighted moving average (EWMA)), and the bottom panel shows an EWMA plot for the network utility. The stochastic approximation algorithm converges rapidly, and there is steady-state noise due to constant gain a in Eqn. (1).

Table 4: TS-ABR-QD: Uplink TCP throughputs.

Setting	θ_1 (Mbps ±95% CI)	Avg. V_1 (ms)	θ_2 (Mbps ±95% CI)	Avg. V_2 (ms)	Utility
Isolated	75.6 ± 0.39	-	103.0 ± 0.73	-	Bound: 7.56
Together	29.7 ± 0.59	-	41.2 ± 0.51	-	7.10
S_i=100ms	36.5 ± 0.52	99.80	49.1 ± 0.36	99.70	7.49
S_i=20ms	31.9 ± 0.46	19.9	44.4 ± 0.72	19.56	7.25
WAN S_i=20ms	31.8 ± 0.46	19.72	43.9 ± 0.52	19.83	7.24

6.2 Uplink TCP Transfers

With uplink TCP transfers at both the AP-STA pairs, the AP→STA interference affects TCP ACKs being sent by the APs to the STAs. In TCP, ACKs are cumulative, so the information lost in a corrupted ACK could be available in the next received ACK. With this in mind, we can understand the together throughputs in Tbl. 4[1]. Whereas the isolated throughputs are almost the same as those obtained in the downlink (Tbl. 3), the together throughputs in the uplink are higher, leading to a smaller drop in network utility (from 7.56 to 7.10). Using the TS-ABR-QD algorithm, ADWISER is able to increase the utility to 7.49 with 100 ms time-slices, and to 7.25 with 20 ms time-slices. Over the emulated WAN, the network utility, for 20 ms time-slices is 7.24. Time series like those in Fig. 11 were obtained for uplink TCP transfers as well, and looked very similar; we are unable to include them due to lack of space.

6.3 HTTP with Bulk TCP Transfers

Table 5: TS-ABR-QD: HTTP (STA2), bulk TCP (STA1)

Setting	TCP thpt. θ_1 (Mbps±95% CI)	Avg. HTTP response time at STA2
Without ADWISER	37.2 ± 0.9	1000
ADWISER S_i = 20 ms	43.8 ± 0.71	549

Tbl. 5 shows the performance of downlink HTTP transfers being done by STA2, along with a downlink bulk TCP transfer at STA1. HTTP transfers at STA2 were emulated by repeatedly downloading an HTML webpage, alternating with user "think" times. Each download involved 10 HTTP requests: 1 of size 1MB, 3 of size 10KB, 3 of size 100KB, and 3 of size 500KB, a total of 22.640 Mb; all these objects were downloaded by several parallel TCP connections. The think time was uniformly distributed between 100 and 500ms.

We see from Tbl. 5 that, without ADWISER, the mean TCP throughput at STA1 is 37.2 Mbps, as opposed to the isolated throughput of 79.6 Mbps (Tbl. 3), and the average HTTP page response

[1]In obtaining these results, we disabled TCP segmentation offload (a facility for reducing CPU load) in Linux, by setting the parameter *tcp_tso_win_divisor* to 0. With this setting, the uplink performance in Linux was similar to that we obtained with Windows and MAC OS STAs.

time was 1000 ms. The higher TCP throughput (than the together throughput in Tbl. 3) is due to the TCP transfer at STA1 being able to utilise the think times of the HTTP transfer, during which times it gets the isolated throughput. Since there is a bulk TCP transfer, when the HTTP page is being transferred, it gets a throughput of 25.7 Mbps (see Tbl. 3), yielding a high response time. On the other hand, with ADWISER (using a 20 ms time-slice), the TCP throughput at STA2 increases to 47.5 Mbps, and the HTTP response time reduces to 549 ms, including the delay due to time-slicing.

Since the user has a think time after getting each page, ADWISER implements *dynamic* time-slicing so that, during the think times, *every* time-slice, in *every* frame, is used by the TCP transfer at STA1. This is implemented as follows. During TCP transfers the corresponding ADWISER queue is nonempty with high probability. Thus, ADWISER identifies that an HTTP user is in a "think" time, when the corresponding packet queue in ADWISER becomes empty. If a time slice is alloted to the HTTP flow, but the corresponding packet queue (in ADWISER) is empty, that slice is allotted to the TCP flow of the other AP-STA pair. This dynamic time slicing is the reason why the AP1→STA1 TCP transfer, when competing with an HTTP flow at AP2→STA2, obtains a higher throughout with ADWISER (i.e., 43.8 Mbps) than when competing with another TCP flow at AP2→STA2 (i.e., 36.6 Mbps, see Tbl. 3).

6.4 Real-time Packet Voice and TCP

Table 6: Bidirectional packet "voice" at AP2-STA2, with a bulk downlink TCP transfer at AP1-STA1.

Setting	θ_1 (Mbps ±95% CI)	"Voice" pkt. mean delay (ms±95% CI); P(delay>20ms)
Only Voice	-	D/L: 1.6 ± 0.05; 0.001
		U/L: 1.72 ± 0.04; 0.001
Without ADWISER	61.3 ± 1.81	D/L: 7.89 ± 0.16; 0.02 378 pkts dropped
		U/L: 5.05± 0.1; 0.007
ADWISER S_i = 20 ms	70.7 ± 0.62	D/L: 11.63 ± 0.27; 0.08
		U/L: 5.02± 0.06; 0.00

We emulated bidirectional pkt. voice by sending 48 byte UDP packets (20 bytes payload), every 20 ms, alternately, from the server to STA2 (for 25 sec) and vice versa (cf., 8000 bps G729 voice). In AP2, the voice was mapped into the IEEE 802.11e AC-VO access category. Tbl. 6 shows that, by itself, the voice-like stream gets < 2 ms mean packet delay, and Prob(delay > 20 ms) = 0.001. With a downlink bulk TCP transfer at STA1 (see Tbl. 6), in together operation, TCP throughput drops from 79.6 Mbps (Tbl. 3) to 61.3 Mbps, with substantial downlink UDP packet loss (378 packets lost out of 10,000), due to mutual interference in the downlink. Uplink UDP has its mean delay increased with no packet loss. With ADWISER, in this traffic setting, *every* 20 ms time-slice was allotted to the AP1-STA1 TCP transfer. In each time-slice, ADWISER releases downlink voice packets to AP2, and then releases the batch of TCP packets to AP1, as per the TS-ABR-QD algorithm. Uplink UDP is not time-sliced. With ADWISER, the voice packet loss reduces to nil, but there is an increase in downlink voice packet delay due to time-slicing (10 ms of which is due to half the time-slice). The uplink UDP flow is hardly affected. "Voice" performance is well within that required from such networks (Table in Sec 2.3.4, [13]). With time-slicing, the TCP throughput also increases from 61.3 Mbps to 70.7 Mbps, the drop (from 79.6 Mbps) occurs due the airtime being used by the UDP stream, and occasional packet loss.

7 SUMMARY AND FUTURE WORK

We aimed to make time-slicing of WiFi networks practical for HT WiFi. By carefully controlling the release of data from ADWISER to the APs, so that little data overflows from one slice to the other, we reduced the time-slices to 10s of ms. We have reported performance results with 20 ms time-slices, for uplink and downlink bulk TCP transfers, over the local network and the WAN, HTTP transfers, and a voice-like application over UDP. In each case, the overlay time-slicing approach substantially improves the performance of the applications. The work reported in this paper has been from experiments on a laboratory setup. Our future work will demonstrate the overlay time-slicing approach in a general, operational WiFi network.

ACKNOWLEDGEMENTS: We thank Prof. Vivek Borkar, for discussions on various control techniques, Ayush Gupta, Amrit Priydarshi, Sindhu Srinivas, and Gopika Gopikrishnan. Arista Networks and the Centre for Networked Intelligence, IISc, provided funding for this research project.

REFERENCES

[1] M. Hegde, P. Kumar, K. R. Vasudev, N. N. Sowmya, S. V. R. Anand, A. Kumar, and J. Kuri. Experiences with a centralized scheduling approach for performance management of IEEE 802.11 wireless LANs. *IEEE/ACM Transactions on Networking*, 21(2):648–662, 2013.
[2] A. Sunny, S. Panchal, N. Vidhani, S. Krishnasamy, S.V.R. Anand, M. Hegde, J. Kuri, and A. Kumar. A generic controller for managing TCP transfers in IEEE 802.11 infrastructure WLANs. *Journal of Network and Computer Applications (Elsevier)*, 93:13–26, 2017.
[3] S. Seytnazarov, J-G Choi, and Y-T Kim. Enhanced mathematical modeling of aggregation-enabled WLANs with compressed BlockACK. *IEEE Transactions on Mobile Computing*, 18(6):1260–1273, 2019.
[4] M. Richart, J. Baliosian, J. Serrat, J-L. Gorricho, and R. Agüero. Slicing with guaranteed quality of service in WiFi networks. *IEEE Transactions on Network and Services Management*, 17(3):1822–1837, 2020.
[5] E. Coronado, R. Riggio, J. Villalon, and A. Garrido. Lasagna: Programming abstractions for end–to–end slicing in software–defined WLANs. In *Proceedings WoWMoM*, 2018.
[6] M. Carmo, S. Jardim, A. Neto, R. Aguiar, D. Corujo, and J. Rodrigues. Slicing WiFi WLAN-sharing access infrastructures to enhance ultra-dense 5G networking. In *IEEE ICC*. IEEE, 2018.
[7] K. Katsalis, K. Choumas, T. Korakis, and L. Tassiulas. Virtual 802.11 wireless networks with guaranteed throughout sharing. In *IEEE ISCC*. IEEE, 2015.
[8] K. Koutlia, A. Umbert, R. Riggio, I. Vila, and F. Casadevall. A New RAN Slicing Strategy for Multi-Tenancy Support in a WLAN Scenario. In *IEEE NetSoft*, pages 64–70. IEEE, 2018.
[9] P. H. Isolani, N. Cardona, C. Donato, G. A. Pérez, J. M. Marquez-Barja, L. Z. Granville, and S. Latré. Airtime-based Resource Allocation Modelling for Network Slicing in IEEE 802.11 RANs. *IEEE Communications Letters*, 24(5):1077–1080, 2020.
[10] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Math. Stats.*, 1951.
[11] V. S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Hindustan Book Agency, 2008.
[12] MCS table and how to use it. https://wlanprofessionals.com/mcs-table-and-how-to-use-it.
[13] Diffserv to QCI mapping-01. https://tools.ietf.org/id/draft-henry-tsvwg-diffserv-to-qci-01.html#rfc.section.2.2.