# DSASync: Managing End-to-End Connections in Dynamic Spectrum Access Wireless LANs

Ashwini Kumar, *Member, IEEE*, and Kang G. Shin, *Fellow, IEEE, ACM*

***Abstract*—Wireless LANs (WLANs) have been widely deployed as edge access networks that provide the important service of Internet access to wireless devices. Therefore, performance of end-to-end connections to/from such WLANs is of great importance. The advent of Dynamic Spectrum Access (DSA) technology is expected to play a key role in improving wireless communication. With DSA capability, WLANs opportunistically access licensed channels in order to improve spectrum-usage efficiency and provide better network performance. In this paper, we identify the key issues that impact end-to-end connection performance when a DSA-enabled WLAN is integrated with the wired cloud. We propose a new network management framework, called `DSASync`, to mitigate the identified performance issues. `DSASync` achieves this objective by managing the connections at the transport layer as a third-party supervisor and targets both TCP streams and UDP flows. `DSASync` requires no modifications to the network infrastructure or the existing network stack and protocols while ensuring transport protocol (TCP or UDP) semantics to be obeyed. It mainly consists of a combination of buffering and traffic-shaping algorithms to minimize the adverse side-effects of DSA on active connections. `DSASync` is evaluated using a prototype implementation and deployment in a testbed. The results show significant improvement in end-to-end connection performance, with substantial gains on QoS metrics like goodput, delay, and jitter. Thus, `DSASync` is a promising step toward applying DSA technology in consumer WLANs.**

***Index Terms*—Approximate entropy, cognitive radio (CR), Dynamic Spectrum Access (DSA), TCP, UDP, Wi-Fi.**

## I. Introduction

The PRIMARY function served by a majority of IEEE 802.11 wireless LANs (WLANs)—e.g., home or office wireless networks—is to serve as the *first/last-mile access network* to the wired network cloud or the Internet, thus enabling the end devices to avail of networking services over the wireless medium. However, rapid proliferation in wireless coverage has exacerbated the unwanted side-effect of interference and congestion on the consumer spectrum bands, resulting in poor wireless networking performance. Moreover, the popularity of bandwidth-intensive and QoS-sensitive networking services (like video streaming) is also growing, which further stresses the performance on consumer WLANs. Unlicensed wireless operation, called Dynamic Spectrum Access (DSA) [2], is emerging as an important solution to this potential performance shortfall in WLANs.

With DSA capability, a WLAN can opportunistically communicate on spectrum bands that are licensed to a different service or operator/owner, subject to certain constraints like incumbent protection. A WLAN with DSA capability is referred to as a *DSA network* (DSAN). DSA is witnessing active research and standardization (e.g., IEEE 802.22 [3]), with the FCC's approval of commercial unlicensed operations in the TV spectrum [4]. We argue that effective integration of DSANs with existing networking infrastructure is important for the success of DSA and a key step forward toward fully cognitive wireless networks [5].

*Need for Connection Management in DSANs:* DSANs are expected to be utilized by consumers in a similar manner as most existing WLANs are—as edge access networks. Thus, DSANs must match and exceed the end-to-end performance of traditional WLANs in order to be commercially viable. Currently, there is a lack of "end-to-end" insights into DSA. In this paper, we investigate issues of integrating a DSAN with the wired network.

DSA entails additional operational constraints in a rapidly changing spectrum environment. Therefore, it involves a number of functions and events that can be disruptive to ongoing network traffic. Examples include spectrum sensing, channel switching, spectrum management and coordination, and incumbent activity. Apart from performance degradation at the lower link/PHY layers, such disruptive DSA-related phenomena can make long-term adverse impacts on the end-to-end communication. From a networking viewpoint, the transport layer is the first layer (from bottom of the networking stack) with true end-to-end semantics. We observe that the adverse impact of DSA on end-to-end connections is primarily a consequence of its negative side-effects at the transport layer. The main reason for this undesirable reaction is the ignorance of higher-layer transport protocols about lower-layer DSA semantics, as we discuss next.

*TCP Streams:* Consider the example of incumbent activity on the licensed channel. A TCP connection between a server host in the cloud and a client on the DSAN can experience timeouts when the client cannot send out ACK packets in time because of an ongoing DSA-induced quiet period on incumbent detection. Consequently, TCP's congestion control mechanism will be unnecessarily invoked leading to further performance

Manuscript received December 13, 2010; revised July 20, 2011; accepted October 03, 2011; approved by IEEE/ACM Transactions on Networking Editor K. Papagiannaki. Date of publication January 09, 2012; date of current version August 14, 2012. The preliminary work of this paper appeared in the Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON), Boston, MA, June 21–25, 2010.

A. Kumar was with the University of Michigan, Ann Arbor, MI 48109 USA. He is now with Juniper Networks, Sunnyvale, CA 94086 USA (e-mail: ashwinik@eecs.umich.edu).

K. G. Shin is with the University of Michigan, Ann Arbor, MI 48109 USA (e-mail: kgshin@eecs.umich.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2011.2178264

degradation. Other fundamental DSA functions like spectrum sensing and channel-switching also contribute to this negative impact. Such interruptions can be frequent, given: 1) the regulatory restrictions imposed on unlicensed operations; 2) DSA service provider requirements; as well as 3) unforeseen incumbent activity.

Techniques have been proposed in the past to address performance problems arising due to packet loss in the presence of high bit error rate experienced on wireless medium, particularly for TCP [6], [7]. In modern WLANs, random wireless errors are not a significant problem, as the quirks of early-era wireless protocols have been addressed through more sophisticated error detection/correction schemes. In the context of DSANs, delays and losses arise due to DSA-related events, and the disruption could last significantly longer. Unlike random wireless errors, knowledge about many of the disruptive DSA events can be deterministically obtained, thus making a proactive approach feasible in masking their side-effects at the transport level.

*UDP Flows:* While there have been few TCP connection management schemes for WLANs in the past (as noted earlier), not much work has been done for UDP-based network connections. UDP is a connectionless protocol, and by design, it does not address packet errors, delays, or losses. Thus, managing UDP flows was not considered of particular importance because applications requiring ordered delivery and reliability should use TCP instead. However, we argue that UDP connection management is now very important as UDP flows carry a significant portion of network traffic. The reason behind this is the tremendous growth in popularity of multimedia-based network services, e.g., video streaming, voice/video conferencing, etc., which typically use UDP to ensure timeliness of delivery. While such applications can tolerate some disruptions/losses, they are highly QoS-sensitive.

*Summary of Our Approach:* We propose DSASync to address the end-to-end performance issues when integrating a DSAN with the wired Internet. DSASync is a network management framework for regulating TCP and UDP connections traversing the wired–wireless boundary. DSASync comprises algorithms based on buffering and traffic shaping to minimize adverse impacts on TCP/UDP connections. During DSA-related disruptions (e.g., due to incumbent activity) the packets are buffered in order to minimize losses. Also, the traffic rate is shaped based on the expected amount of disruptions to eliminate undesirable changes to connection behavior. There are two main advantages of DSASync—it maintains the end-to-end semantics of the standard transport protocols (TCP/UDP) and ensures compatibility by not requiring any changes to their existing implementations.

DSASync exploits several built-in control knobs of TCP to provide a complete TCP connection management solution. For example, the receive window advertisement is set to 0 to throttle a prolific sender when the buffer space is full. Also, fast retransmit/recovery is utilized to signal any sudden decrease in channel capacity in case of a channel switch. Unfortunately, UDP flows, being stateless, do not provide such ready-made hooks. Therefore, it is significantly more challenging to provide complete and nonintrusive connection management for UDP flows. In DSASync, we take the *higher-layer* approach to address this problem by utilizing Real-time Transport Protocol (RTP) [8] features to manage the UDP-based connection. For example, RTP control packets are used to signal changes in channel capacity or to throttle the remote sender. Although not all UDP flows use RTP, RTP over UDP is predominantly used for most real-time streaming applications like VoIP, where 100% reliability or in-order delivery is not required, but QoS is still important. In fact, our solution can be generalized for managing any connection that uses RTP, irrespective of the underlying transport protocol.

To the best of our knowledge, this is the first attempt to consider integration issues with deployment of DSANs. DSASync is designed to be compatible, scalable, and practical—a prototype implementation is also developed and evaluated in a testbed as part of this work.

*Contributions:* The contributions of this paper are threefold. First, we identify the key challenges for the mainstream integration of DSA-based WLANs. The causes for various performance problems are identified. Second, we propose DSASync to address the identified issues in the context of TCP and UDP connections. Third, DSASync is shown to better enable DSAN integration with the Internet via a testbed-based evaluation.

*Organization:* The paper is organized as follows. Related work is discussed in Section II. We present the background and description of the problem in Section III. DSASync details are presented in Section IV, with an implementation in Section V. Experimental evaluation of DSASync is presented in Section VI. The paper concludes with Section VII.

## II. Related Work

There have been significant research efforts into the challenges and development of DSA. References [2] and [9] provide general surveys about the state-of-art in the field. Various aspects of DSA, as enabled by cognitive radios, have been discussed in [5] and [10]–[12].

Spectrum-occupancy studies [13], [14] have shown the existence of abundant spectrum white spaces across most of the licensed spectrum despite diversity in channel and incumbent characteristics. The FCC has already issued preliminary guidelines for DSA operations in TV bands [4]. Several DSA MAC/PHY protocols have already been proposed in literature, especially for TV bands [15]–[17]. Also, standardization efforts for DSA protocols are currently in progress [3], [18].

However, there have been very few publications on the end-to-end impact of DSA during actual deployment in WLANs. Adaptation to application requirements in DSA has been proposed in [19]. However, that approach is node-centric rather than network-centric and does not account for impacts on the end-to-end connections. The authors of [20] identified the important issues affecting TCP in a DSAN. They proposed a novel reliable transport protocol for DSA ad hoc networks, called TP-CRAHN. However, TP-CRAHN does not address the issues when a DSA network acts as an access network to the Internet. Another key shortcoming of this work is its deployment incompatibility—it requires a completely new transport protocol to be linked and loaded on the devices. On the other hand, DSASync integrates DSANs with the Internet without any changes to the existing applications or the protocol stack and is also comprehensive in its approach by incorporating both TCP and UDP connection management.

As mentioned earlier, there have been several prior efforts on improving TCP performance (albeit very little in context of UDP) in wireless environments [6], [7], [21]. The key ideas proposed were: 1) splitting of wired and wireless TCP connections; or 2) localized caching and retransmissions when a packet loss was identified on the wireless link. These efforts were motivated by the high error rate in wireless medium, which caused substantial performance drop in TCP connections. Such concerns have mitigated with the advent of stronger error-correction schemes and higher data-rate standards [22]. However, from these results, `DSASync` borrows the concept of packet buffering and utilizing a proxy (e.g., the base station on the wired–wireless boundary) as the central entity in executing its functions.

Our solution is related to the area of power management and wireless network selection [23], [24], where traffic flows must be protected from disruptions caused by power-saving optimization algorithms. Such power-saving mechanisms produce adverse side-effects similar to DSA, and `DSASync`—like buffering and traffic shaping—can be very useful in such scenarios.

## III. Issues, System Model, and Notation

### A. Integration Issues

As briefly mentioned earlier, unlicensed operation exhibits several characteristics that adversely impacts a DSAN's effectiveness in functioning as an edge access network. The major issues are listed as follows.

*Sensing Interruptions:* Spectrum sensing is performed to detect channel characteristics, including incumbent presence or absence. For reliable spectrum sensing, there must be no unlicensed traffic on the channel. Thus, every sensing event involves scheduling of a *quiet period* (QP), during which packet transmission is halted.[1] Typically, the underlying DSA MAC protocol schedules QPs [25]. Depending on the sensing technology and the channel characteristics, a QP typically lasts for tens of milliseconds (ms) or more and can be scheduled as frequently as every few hundred milliseconds. In general, DSA protocols schedule sensing with a higher frequency than regulatory requirement in order to improve sensing accuracy and, hence, improve DSA performance. For example, nodes can independently sense *out-of-band* channels proactively to get a better picture of spectrum conditions. While an external sensing infrastructure (e.g., geolocation sensing database) may reduce the sensing overhead, online real-time sensing will be needed for accuracy and correctness.

*Channel-Switch Delays:* Channel-switches can occur frequently during DSA. Depending on the underlying DSA protocol semantics, a channel-switch may be made to exploit a better channel, or in the event of incumbent transmission on the current channel. Each channel-switch can incur delay (e.g., due to the interface reset, coordination between nodes, etc.) of $\approx$100 ms. Channel-switches (and sensing) also contribute to the problem of "bandwidth fluctuation."

[1]Though single wireless data interface is assumed for cost and simplicity reasons, this problem is independent of the number of interfaces in the nodes for in-band sensing.

*Bandwidth Fluctuation:* During DSA, nodes can experience wide variations in available bandwidth for several reasons. Spectrum available for unlicensed usage on the current channel depends on an incumbent utilization fraction, which can change dynamically and substantially. Presence of additional WLANs in the vicinity (on the same channel) further decreases available bandwidth for application traffic in a DSAN. Furthermore, channel-switches may also contribute to bandwidth variability. This can occur because of: 1) different amount of channel usage opportunities available on the new channel; 2) different channel-access strategy resulting in more/less throughput efficiency; and 3) different spectrum widths of the new channel.

*Incumbent Activity:* Incumbent transmissions must be protected while a DSAN operates on a licensed channel. Thus, when a Primary User (PU) transmission is detected (via sensing), the DSAN nodes must not transmit and stop any ongoing transmission within a very short time. If incumbent activity on the channel is high, the DSAN's communication traffic will suffer greater delay and reduction in available bandwidth. Although the underlying DSA protocol is typically designed to take corrective actions when such a situation persists (e.g., by switching to a different channel), an incumbent activity still results in significant disruption to ongoing communication.

Note that sensing/switching may not be independent of incumbent behavior. For example, in WiMAX-type channels (e.g., upcoming 802.16h draft), the white spaces (and incumbent activities) are short ($\approx$20 ms) but abundant, which require frequent sensing to exploit for DSA.

DSA is fundamentally disruptive for ongoing wireless communication, especially on a short-term scale, which cannot be fully eliminated. Therefore, the design principle of `DSASync` is to carefully manage end-to-end connection flows in order to minimize the impact of the aforementioned disruptive events experienced when DSA is active on the WLAN. Since transport layer forms the basis of end-to-end connections and directly impact application performance, our solution targets two core transport protocols—TCP and UDP.

### B. System Model and Notation

The system under consideration is a single-hop WLAN with wireless devices (e.g., a Wi-Fi hotspot) that connect to the wired network (e.g., Internet) through a base station (or a designated node that interfaces with the wired network), as shown in Fig. 1. This edge WLAN has DSA capability and, hence, is also a DSAN. Each wireless device is equipped with a DSA-enabled wireless interface card and necessary hardware components, together with a DSA protocol. As is typical of edge wireless access networks, the base station coordinates association, authentication, and traffic to/from other nodes. Hence, it has knowledge about other nodes' important DSA MAC parameters (e.g., sleep/awake cycles or independent spectrum sensing schedules, if any). No restriction on DSA MAC protocol or spectrum sensing is assumed to retain the generality of `DSASync`.

We will use the following acronyms throughout the paper:

- *wired network* (WN): the network cloud (e.g., the Internet) to which the wireless end-devices communicate to avail of network services;

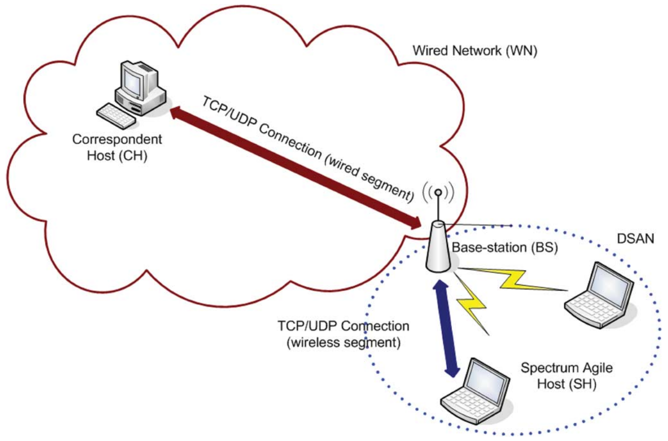

Fig. 1. System model for a DSAN.

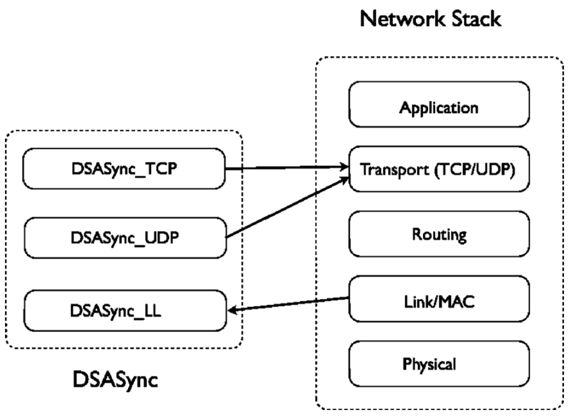

Fig. 2. Architectural overview of DSASync.

- *DSA network* (DSAN): a DSA-based wireless network that is connected to the WN;
- *spectrum-agile host* (SH): a DSA-enabled end-device in the DSAN that communicates with a device in the WN;
- *correspondent host* (CH): an end-device in the WN that communicates with an SH;
- *base station* (BS): the designated device (or access point) that connects the DSAN to the WN. All communications from DSAN to WN and vice versa must pass through BS;
- *transmission freeze period* (TFP): the duration during which packet transmission is halted by one or more SHs or by the entire DSAN due to DSA-related events.

## IV. DSASync

DSASync is logically a link-layer (LL) network management protocol (similar to Snoop Agent [6]). However, DSASync manages TCP/UDP connections—Fig. 2 shows DSASync's architecture schema.

To accomplish traffic management, DSASync sniffs the packets in transit (at the BS), and maintains state information (e.g., last ACK copy, sequence numbers, etc.) for each ongoing TCP stream it detects. Similarly, it maintains some state information (e.g., copies of latest *sender report* and *receiver report* packets) for each RTP-based UDP flow identified.

### A. DSASync: Link Layer

The DSASync LL component (DSASync_LL) is the "information monitoring" unit of DSASync, which collects and maintains information about DSA parameters required by DSASync. The parameters of interest are as follows.

1) $N$ = total number of wireless nodes associated with the BS in the DSAN.
2) $f^{\text{sense}}_{\text{DSAN}}(t)$ = the frequency of spectrum sensing by the entire DSAN. This parameter usually corresponds to the cooperative sensing schedule in which all nodes participate.
3) $t^{\text{sense}}_{\text{DSAN}}(t)$ = the duration of each spectrum sensing event scheduled by the DSAN.
4) $f^{\text{sense}}_{i}(t)$ = the frequency of additional sensing (e.g., out-of-band) sensing performed by node $i$.
5) $t^{\text{sense}}_{i}(t)$ = the duration of each node-specific sensing event at node $i$.
6) $f^{\text{switch}}_{\text{DSAN}}(t)$ = the frequency (rate) of channel switches.
7) $t^{\text{switch}}_{\text{DSAN}}(t)$ = the delay involved in each channel switch.
8) $g_{\text{PU,ON}}(t)$ = the PU's ON time distribution.
9) $\text{S}_{\text{DSAN}}$ = the Boolean parameter indicating if sensing is currently ongoing in the DSAN.
10) $\text{SW}_{\text{DSAN}}$ = the Boolean parameter indicating if the DSAN is currently performing a channel-switch.
11) $\text{S}_i$ = the Boolean parameter indicating if sensing is currently ongoing at node $i$.
12) $\text{PU}_{\text{ON}}$ = the Boolean parameter indicating if there is currently a PU activity on the current channel.

These parameters are part of any DSA MAC/PHY protocol and are typically available at the link layer. For example, most DSA protocols (specifically their sensing components) estimate PU ON/OFF distribution in order to enhance DSA performance [25]–[27].

### B. TCP Management

The main task of the TCP management component, DSASync_TCP, is to utilize the information collected by DSASync_LL in managing both downlink (from CH to SH) and uplink (from SH to CH) TCP traffic. The objective is threefold: 1) to minimize packet loss; 2) to minimize timeouts and, hence, retransmissions; 3) to adjust TCP connection parameters in response to changes in available bandwidth. In the basic design, DSASync_TCP executes only at the BS, as the BS has all the necessary information and the incoming/outgoing traffic must pass through it (see Sections III-B and IV-D). DSASync_TCP consists of three modules—DSASync_TCP_CH-SH, DSASync_TCP_SH-CH, and DSASync_TCP_CAP.

*1) DSASync_TCP_CH-SH:* This module buffers the downlink (CH–SH) TCP packets for destination wireless nodes during the TFPs. The current state of the destination node (w.r.t. its packet-reception capability) is known from DSASync_LL. The buffered packets are then transmitted from the BS to the SH when the transmission can be resumed.

Due to limited buffer space at the BS, it is possible to run out of space before the transmission is resumed. To prevent this situation, DSASync_TCP attempts to proactively pause the sender by exploiting the built-in flow control mechanism of TCP.

Let the allocated space (at the BS) for buffering downlink TCP packets be $B^{\text{tcp}}_{\text{alloc}}$. $B^{\text{tcp}}_{\text{low}}$ and $B^{\text{tcp}}_{\text{high}}$ are the configuration parameters for TCP buffer space thresholds, where $B^{\text{tcp}}_{\text{alloc}} > B^{\text{tcp}}_{\text{high}} > B^{\text{tcp}}_{\text{low}}$. $B^{\text{tcp}}_{\text{free}}$ is the current free buffer space for TCP

**Algorithm 1:** Algorithm TCP_CH-SH-a: Handler for downlink TCP traffic

```
Require: $B^{\mathrm{tcp}}_{\mathrm{free}}$, $B^{\mathrm{tcp}}_{\mathrm{low}}$, hold
 1: $p \leftarrow$ incoming CH–SH pkt
 2: $\mathrm{dest} \leftarrow$ destination SH of $p$
 3: $\mathrm{src} \leftarrow$ source CH of $p$
 4: $\mathrm{conn} \leftarrow$ $p$'s TCP connection identifier
 5: $\mathrm{TFP} \leftarrow \mathrm{SW_{DSAN}}|\mathrm{S_{DSAN}}|\mathrm{S_{dest}}|\mathrm{PU_{ON}}$
 6: if $\mathrm{TFP} = 0$ then
 7:     Add $p$ to transmit queue
 8: else
 9:     Buffer pkt
10:     if $\mathrm{hold} = \mathrm{false}$ then
11:         if $B^{\mathrm{tcp}}_{\mathrm{free}} < B^{\mathrm{tcp}}_{\mathrm{low}}$ then
12:             $\mathrm{hold} = \mathrm{true}$
13:             for each TCP connection do
14:                 Advt. zero rwin to sender CH
15:             end for
16:         else
17:             if $\mathrm{SN^{conn}_{new}} = \mathrm{SN^{conn}_{last}} + \mathrm{rwin^{conn}}$ then
18:                 Advt. zero rwin $\mathrm{src}$ for $\mathrm{conn}$
19:             end if
20:         end if
21:     else
22:         if $p =$ window update request then
23:             Advt. zero rwin to src for conn
24:         end if
25:     end if
26: end if
```

packets. For the TCP connection $\mathrm{conn}$, $\mathrm{SN^{conn}_{last}}$ is the latest sequence number acknowledged by the SH, $\mathrm{SN^{conn}_{new}}$ specifies the sequence number of the latest data packet (coming from CH) buffered at BS, and $\mathrm{rwin^{conn}}$ is the latest advertised receive window.

The BS uses the procedure outlined in Algorithms 1 and 2—both executed in parallel—in order to manage CH–SH TCP traffic. Algorithm 1 is executed when a TCP packet is received from the CH. Algorithm 2 is executed periodically, based on a sufficiently frequent timer interrupt. A separate process updates buffer sizes (when a packet is added/removed) and also overwrites the $\mathrm{rwin}$ field to 0 in outgoing packets, if *hold* parameter (see Algorithms 1 and 2) is true.

Algorithm 1 checks for existence of a TFP currently (line 5) and buffers the incoming packet in such a scenario (line 9). Furthermore, to manage buffer space, it exploits the TCP flow control mechanism to avoid buffer overflow (and hence dropped packets) at the BS. The BS advertises a zero-size receive window on behalf of the SH when the buffer threshold is reached (lines 11–15). The same strategy is used to prevent retransmissions (due to timeouts at the sender CH) when the receive window becomes full during TFP (lines 17–19). During TFP, a window update request from CH is handled by sending out a zero-size rwin advertisement (lines 22–23). The algorithm does not prevent timeouts or retransmissions at the endpoints due to non-DSA factors, such as congestion in the network.

**Algorithm 2:** Algorithm TCP_CH-SH-b: Buffering manager for downlink TCP traffic

```
Require: $B^{\mathrm{tcp}}_{\mathrm{free}}$, $B^{\mathrm{tcp}}_{\mathrm{high}}$, hold
 1: if $\mathrm{hold} = \mathrm{true}$ then
 2:     if $B^{\mathrm{tcp}}_{\mathrm{free}} > B^{\mathrm{tcp}}_{\mathrm{high}}$ then
 3:         $\mathrm{hold} \leftarrow \mathrm{false}$
 4:     end if
 5: end if
 6: for $i \leftarrow 1$ to $N$ do
 7:     $\mathrm{TFP} \leftarrow \mathrm{SW_{DSAN}}|\mathrm{S_{DSAN}}|\mathrm{S}_i|\mathrm{PU_{ON}}$
 8:     if $\mathrm{TFP} = 0$ then
 9:         Unbuffer any $i$'s pkt to transmit queue
10:     else
11:         Buffer any $i$'s pkt from transmit queue
12:     end if
13: end for
```

However, an older packet is replaced with a newly arrived packet with the same sequence number, i.e., when a duplicate packet arrives.

It is also possible to manage CH–SH TCP traffic by sending out ACKs to the CH on behalf of the SH, or even splitting TCP connections at the BS. However, `DSASync` does not take these approaches for two reasons. First, it will violate end-to-end semantics of TCP data flow, e.g., a successful reception of ACK at CH (the source) will no longer imply that the packet has successfully reached SH (the destination). Second, sending ACKs will likely result in receiving more packets during the TFP, which may lead to buffer space getting filled up earlier. Furthermore, the resource overhead of `DSASync` will be higher.

*2) DSASync_TCP_SH-CH:* TCP performance degrades due to irregular behavior in the reception stream. For example, we observe timeouts and retransmissions when a TFP sets in, as CH often does not receive ACKs in time according to its RTT estimate—which is typically quite low as it was based on continuous packet reception during the past non-TFP period. Furthermore, a "start-and-stop" type of data packet reception also contributes to other QoS issues, such as increased application jitter.

To minimize this connection degradation for the uplink (SH–CH) TCP stream, the BS attempts to "smooth" the outgoing flow. The key idea is to spread the uplink packets over the TFPs, so that the CH sees a relatively steady stream of packets despite the disruption at the source SH.

Given the information available from `DSASync_LL`, the average fraction of TFPs for node $i$ can be estimated. Consider a time interval, say $[T - \triangle T, T]$. The total TFP for node $i$ during this $\triangle T$ time window is the sum of delays (on average) due to sensing, switching and PU activity interruptions

$$\begin{aligned}\mathrm{TFP}^{\mathrm{avg}}_i &= E\left[f^{\mathrm{sense}}_{\mathrm{DSAN}}(t) \cdot t^{\mathrm{sense}}_{\mathrm{DSAN}}(t) \cdot t\right]^{t=T}_{t=T-\triangle T} \\ &+ E\left[f^{\mathrm{sense}}_i(t) \cdot t^{\mathrm{sense}}_i(t) \cdot t\right]^{t=T}_{t=T-\triangle T} \\ &+ E\left[f^{\mathrm{switch}}_{\mathrm{DSAN}}(t) \cdot t^{\mathrm{switch}}_{\mathrm{DSAN}}(t) \cdot t\right]^{t=T}_{t=T-\triangle T} \\ &+ E[g_{\mathrm{PU,ON}}(t) \cdot t]^{t=T}_{t=T-\triangle T}\end{aligned}$$

**Algorithm 3:** Algorithm TCP_SH-CH: Smoothes data rate of uplink TCP traffic

```
Require: $\alpha_{\min}, D_i^{\text{out,tcp}}, \alpha_i, T_i, \text{dequeue}_i\ (\forall i \in N)$
 1: while SH–CH queue is nonempty do
 2:     $p \leftarrow$ 1st TCP pkt in queue
 3:     $\text{done} \leftarrow \text{false}$
 4:     $\text{count} \leftarrow 1$
 5:     while $\text{done} = \text{false}$ and $\text{count} \leq N$ do
 6:         $\text{src} \leftarrow$ source of $p$
 7:         if $\text{dequeue}_{\text{src}} = \text{false}$ then
 8:             $\beta_{\text{src}} \leftarrow \max(\alpha_{\text{src}}, \alpha_{\min})$
 9:             $T_{\text{src}} \leftarrow$ timestamp of src's last TCP pkt
                dequeue
10:             $T_{\text{curr}} \leftarrow$ current timestamp
11:             $\text{elapsed} \leftarrow T_{\text{curr}} - T_{\text{src}}$
12:             if $\{\text{size}(p)/(\text{elapsed}) < \beta_{\text{src}} D_{\text{src}}^{\text{out,tcp}}\}$ then
13:                 $\text{dequeue}_{\text{src}} \leftarrow \text{true}$
14:                 $\text{done} \leftarrow \text{true}$
15:             end if
16:         end if
17:         $\text{count} \leftarrow \text{count} + 1$
18:         $p \leftarrow$ next TCP pkt in queue
19:     end while
20: end while
```

Therefore, the fraction of non-TFP period for node $i$ during $[T - \triangle T, T]$ is given by

$$\alpha_i = 1 - \frac{\text{TFP}_i^{\text{avg}}}{\triangle T}. \tag{1}$$

In practice, historical information on TFP durations during a moving time-window of size $\triangle T$ can be utilized to compute the $\alpha_i$ value for each $i$.

Let $\alpha_{\min}$ be the administrative configuration parameter to limit the extent of traffic shaping. In order to manage uplink TCP traffic the BS executes Algorithm 3. Algorithm 3 outlines the dequeueing process for the outgoing queue at the BS's wired interface. The algorithm modifies the rate of a wireless node src's outgoing TCP packets as

$$D_{\text{eff,src}} = \beta_{\text{src}} \cdot D_{\text{src}}^{\text{out,tcp}} \tag{2}$$

where $\beta_{\text{src}} = max(\alpha_{\text{src}}, \alpha_{\min})$, and $D_{\text{src}}^{\text{out,tcp}}$ is the actual data rate at which src's outgoing TCP packets are received at the BS. Thus, linear traffic shaping is applied to the uplink TCP traffic. Algorithm 3 maintains the timestamp of last dequeue for each src (line 9) and dequeues a packet if the elapsed time since then will not violate the "smoothed" data rate (lines 11–15). The $\alpha_{\min}$ configuration parameter provides administrative control over: 1) excessive delays (and hence very high response-times for applications); and 2) buffer space runout.

$D_{\text{src}}^{\text{out,tcp}}$ is easily estimated by monitoring the rate at which node src's TCP data packets enter the BS's outgoing queue (at its wired interface) in the moving time-window $\triangle T$. Similarly, $T_{\text{src}}$ and $\text{dequeue}_{\text{src}}$ are local variables (used in Algorithm 3)

**Algorithm 4:** Algorithm TCP_SH-CH-OPT: Optimized version of uplink TCP traffic shaping

```
Require: $\alpha_{\text{dsan}}, D_{\text{dsan}}^{\text{out,tcp}}, T_{dsan}, \text{dequeue}_{\text{dsan}}$
 1: while SH–CH queue is nonempty do
 2:     if $\text{dequeue}_{\text{dsan}} = \text{true}$ then
 3:         Wait for dequeue to complete
 4:     end if
 5:     $p \leftarrow$ 1st TCP pkt in queue
 6:     $\beta_{\text{dsan}} \leftarrow \max(\alpha_{\text{dsan}}, \alpha_{\min})$
 7:     $T_{\text{dsan}} \leftarrow$ timestamp of last TCP pkt dequeue
 8:     $T_{\text{curr}} \leftarrow$ current timestamp
 9:     $\text{elapsed} \leftarrow T_{\text{curr}} - T_{\text{dsan}}$
10:     if $\{\text{size}(p)/(\text{elapsed}) < \beta_{\text{dsan}} D_{\text{dsan}}^{\text{out,tcp}}\}$ then
11:         $\text{dequeue}_{\text{dsan}} \leftarrow \text{true}$
12:     else
13:         Wait $\{(\text{size}(p)/\beta_{\text{dsan}} D_{\text{dsan}}^{\text{out,tcp}}) - (\text{elapsed})$
            interval
14:     end if
15: end while
```

that are updated by monitoring the dequeue events. Note that the packets enter the queue in the temporal order they are received, as before.

In practice, the node-specific sensing duration will not vary significantly for different nodes. This is because the sensing technology across devices is expected to be similar, and the spectrum environment is also similar across the single-hop edge DSAN. It may also turn out to be the least dominating fraction in the $\alpha_i$ calculation (1)—$E[f_i^{\text{sense}}(t) \cdot t_i^{\text{sense}}(t) \cdot t]_{t=T-\triangle T}^{t=T}$ can be very small compared to other terms like incumbent activity and collective sensing duration. Thus, each $\alpha_i$, $\forall i \in N$, can be closely approximated by the average of $\alpha_i$ values, say $\alpha_{\text{dsan}}$. Furthermore, since the packets from nodes are queued on a first-come–first-served basis, the individual data rates can now also be replaced by the overall incoming data rate $D_{\text{dsan}}^{\text{out,tcp}}$.

Hence, Algorithm 3 can be further optimized to yield Algorithm 4. The revised algorithm has a lower implementation and runtime overhead because it has to maintain fewer state variables. However, the most significant gain is due to reverting back to traditional queue semantics (which has an $O(1)$ dequeueing process, albeit at the "traffic-shaped" rate) for the uplink (SH–CH) queue in Algorithm 4.

*3) DSASync_TCP_CAP:* TCP's flow and congestion control mechanism allows its adaptation to gradual capacity changes in the network. Thus, small capacity fluctuations, typically encountered on the same channel, do not warrant any special handling. However, during channel-switches—where substantial and sudden capacity decrease may occur, this adaptation can be prolonged. When there is a significant loss of capacity, there can be substantial packet losses and retransmissions in the process.[2]

For the downlink (CH–SH) traffic, the procedure outlined in Algorithm 5 is executed when a channel-switch event is

[2]When the channel capacity increases, the TCP performance gradually improves by itself. Therefore, DSASync does not take any action in this case.

**Algorithm 5:** Algorithm TCP_CAP: Signaling unsustainable decrease in local capacity to a remote TCP sender

```
**Require:** $C$, $e_{\text{tcp}}$, $D_i^{in,\text{tcp}}$ $(\forall i \in N)$
1: **for** $i \leftarrow 1$ to $N$ **do**
2:     **if** $D_i^{in,\text{tcp}} > e_{\text{tcp}}C$ **then**
3:         Send 3 duplicate ACKs to $i$'s CHs
4:     **end if**
5: **end for**
```

indicated (through DSASync_LL component). In this algorithm, $C$ is the raw physical-layer bandwidth on the new channel, while $e_{\text{tcp}}$ is the data transfer efficiency for TCP with the DSA MAC/PHY protocol to be used in the new channel. For example, various studies have shown that $e_{\text{tcp}} \approx 0.5$ over 802.11. $D_i^{in,\text{tcp}}$ denotes the downlink TCP data rate for node $i$. $D_i^{in,\text{tcp}}$ is calculated by a sliding time-window-based historical averaging of TCP packets received for node $i$ at the BS.

Algorithm 5 triggers TCP's fast retransmit/recovery by sending at least three duplicate ACKs to the CH if the current downlink data rate for a node cannot be sustained on the new channel (lines 2–4). The objective is to prevent slow recovery where $\text{cwnd}$ is reduced to 1 instead of half of the current value as in fast recovery. Thus, the sender will automatically reduce its data rate, resulting in lesser impact than would otherwise occur. Note that we avoided the TCP window scale option to manage such capacity changes, as they are optional—many network routers and firewalls do not implement this feature. In contrast, fast retransmit/recovery feature is a part of most TCP implementations (e.g., TCP Reno) that are commonly used in modern operating systems.

## C. UDP Management

UDP traffic is managed along similar lines as TCP streams, through the DSASync_UDP component. Although certain TCP connection management techniques like packet buffering are applicable to UDP flows, UDP connections cannot be managed intrinsically because they are stateless and do not provide built-in connection management knobs (unlike TCP). We do not wish to modify the UDP protocol itself (or introduce a new one), as it goes against DSASync's key design principles of compatibility and easy deployment.

We observe that most QoS-sensitive UDP-based network applications rely on the Real-time Transport Protocol [8]. RTP is an application-layer component that consists of two components: 1) RTP Data Transfer Protocol is responsible for application-level framing and delivery; 2) RTP Control Protocol (RTCP) provides QoS feedback of the data stream. Clearly, for UDP flows that are part of RTP-based communication, we can utilize the higher-layer RTP information to manage the connections to a significant extent.

DSASync_UDP manages UDP connections by using a combination of buffering at the BS and opportunistic modification/generation of RTCP packets—*Receiver Report* (RR) and *Sender Report* (SR) (for those UDP flows that are based on RTP). Like DSASync_TCP, DSASync_UDP sniffs packets in transit to

**Algorithm 6:** Algorithm UDP_CH-SH-a: Handler for downlink UDP traffic

```
**Require:** $B_{\text{free}}^{\text{udp}}$
1: $p \leftarrow$ incoming CH–SH UDP pkt
2: $\text{dest} \leftarrow$ destination SH of $p$
3: $\text{src} \leftarrow$ source CH of $p$
4: $\text{conn} \leftarrow$ $p$'s RTP connection id, if RTP-based flow
5: $\text{ctime} \leftarrow$ current time
6: $\text{rr\_time} \leftarrow$ timestamp of last RR for *conn*
7: $\text{rr\_int} \leftarrow$ avg. RR transmit interval for *conn*
8: $\text{TFP} \leftarrow \text{SW}_{\text{DSAN}}|\text{S}_{\text{DSAN}}|\text{S}_{\text{dest}}|\text{PU}_{\text{ON}}$
9: **if** $\text{TFP} = 0$ **then**
10:     Add $p$ to transmit queue to DSAN
11: **else**
12:     **if** $B_{\text{free}}^{\text{udp}} = 0$ **then**
13:         Flush oldest packets from buffer, to accommodate $p$
14:         Update any SRs, RRs in the outgoing SH-CH queue
15:         **if** *conn* is valid **then**
16:             **if** $\text{ctime} - \text{rr\_time} \geq \text{rr\_int}$ **then**
17:                 **if** No RR for *conn* exist in SH-CH outgoing
                    queue **then**
18:                     Generate a new RR for *conn*
19:                 **end if**
20:             **end if**
21:         **end if**
22:     **else**
23:         Buffer $p$
24:     **end if**
25: **end if**
```

identify active RTP sessions and maintains their metadata. This passive RTP connection identification has some limitations, which we discuss in Section IV-E.

Note that unlike TCP, RTCP cannot directly influence the ongoing UDP-based connection. It is a passive feedback mechanism, and it is up to the applications themselves to take any action in the event of RTCP feedback. Thus, the connection management for RTP-based UDP flows cannot be as responsive as that for TCP streams.

DSASync_UDP consists of three modules: DSASync_UDP_CH-SH, DSASync_UDP_SH-CH, and DSASync_UDP_CAP.

*1) DSASync_UDP_CH-SH:* DSASync_UDP_CH-SH module manages the downlink UDP traffic by using the procedure outlined in Algorithm 6. Like its TCP counterpart, Algorithm 6 is executed when a new UDP packet is received from the CH. Like DSASync_TCP_CH-SH, DSASync_UDP_CH-SH buffers (and later transmits) the incoming UDP packets during the TFPs, based on information available from DSASync_LL. However, there is a key difference in buffering semantics between DSASync_TCP_CH-SH and DSASync_UDP_CH-SH. As stated earlier, applications using UDP traffic emphasize timeliness and are somewhat loss-tolerant. Thus, Algorithm 6 favors new packets over the older (and already buffered) packets when the buffer space gets

**Algorithm 7:** Algorithm UDP_CAP: Signaling unsustainable decrease in local capacity to remote UDP sender

```
Require: $C$, $e_{udp}$, $D_i^{in,udp}$, $(\forall i \in N)$
1: for $i \leftarrow 1$ to $N$ do
2:   if $D_i^{in,udp} > e_{udp}C$ then
3:     Update loss parameters for $i$'s RRs/SRs in outgoing (CH-SH) queue
4:   end if
5: end for
```

full. Therefore, the oldest packets are purged from the buffer to create space for the newly arrived packets (lines 12 and 13).

Apart from buffering (as seen from Algorithm 6), DSASync_UDP_CH-SH provides quick feedback to the sender CH about ongoing QoS degradation on a RTP session, whenever feasible,[3] to limit the packet losses when the buffer space fills. This feedback is provided by updating an existing RR in the outgoing queue toward CH (line 14), or in its absence, generating a completely new RR (lines 17–19). To avoid unnecessary overhead during packet reception, only the RR for the session corresponding to the newly received packet is generated. Also, a separate process periodically updates all the uplink RRs to accurately reflect the loss, delay, and jitter encountered in the DSAN.

Updates of RTCP feedback packets are done without violating RTP semantics and its end-to-end principle. The procedure to update the parameters in SR/RR packets is based on the formulas described in [8].

*2) DSASync_UDP_SH-CH:* The strategy for the uplink RTP-based UDP flows is identical to that for TCP streams. The key idea is to apply linear traffic-shaping to UDP flows in order to mask the DSA-induced interruptions. The goal is to improve the QoS metrics, especially jitter, for the uplink flow. This is particularly useful when the SH is the main sender of the end-to-end connection.

The traffic-shaping algorithm employed is same as that for TCP (described in Section IV-B.2, Algorithm 3) and is based on (1) and (2). The optimized version of the algorithm is also identical (see Algorithm 4). We leave out the pseudocode in the interest of space.

*3) DSASync_UDP_CAP:* Algorithm 7 is executed for UDP flows (that are RTP-based) on a channel-switch, along similar lines as Algorithm 5 for TCP. The difference lies in the capacity change feedback mechanism. While it is possible to exploit fast retransmit/recovery mechanism for TCP, we rely on RTCP's feedback mechanism for UDP. When the current UDP data rate for a node is found to be definitely unsustainable on the new channel (line 2 of Algorithm 7), then its outgoing SRs and RRs are proactively updated by the predicted increase in loss due to reduced bandwidth, i.e., by a factor of $e_{\mathrm{udp}}C/D_i^{in,\mathrm{udp}}$. Here, $e_{\mathrm{udp}}$ is the data transfer efficiency for UDP over the DSA MAC/PHY protocol being used in the new channel. The objective is to provide quick notification to the sender CH about imminent losses, so that it can take corrective action. Other optimizations for this algorithm are discussed in Section IV-D.

[3]There are limits on the amount of RTCP feedback (RR/SR packets), typically limited to 5% of the session bandwidth [8].

## D. Extensions

Several additional optimizations and enhancements can be built upon the basic DSASync platform presented earlier. We discuss two such possible extensions.

*1) Per-Node DSASync:* DSASync agents on wireless client nodes in the DSAN can further help in minimizing losses and improving other QoS metrics. This approach essentially amounts to a distributed architecture of DSASync. Per-node DSASync agents can be implemented using a similar buffer management strategy as the DSASync buffer on the BS. Local DSASync agents will help uplink (SH–CH) traffic bandwidth in particular, as the outgoing packets will not be as easily lost or dropped at the SH itself (during TFPs). However, wireless client nodes can have significantly low resource availability (e.g., a basic smartphone), and this feature may not be feasible or be very limited in usefulness.

*2) QoS Feedback Optimization:* Depending on the traffic characteristics of the DSAN, the QoS feedback policy of DSASync can be optimized. For example, if there is a substantial presence of non-TCP (or non-UDP) type of traffic through a DSAN, then their data rate must be taken into account in determining if the channel capacity is sufficient in Algorithms 5 and 7 (line 2). This is useful and more accurate because the channel capacity is shared between TCP (or UDP) and other types of traffic. Along similar reasoning, if the amount of both TCP and UDP traffic are seen to be similar (and in majority) for a particular DSAN, then their cumulative data rate (i.e., $D_i^{in,\mathrm{tcp}} + D_i^{in,\mathrm{udp}}$) should be used for comparison with new channel capacity in Algorithms 5 and 7.

## E. Limitations

While DSASync's architecture as a nonintrusive network management entity has numerous important benefits like compatibility and ease of deployment, it also leads to certain limitations. We discuss two key limitations with the current design of DSASync.

*1) Identifying RTP-Based UDP Flows:* Proactive connection management for the QoS-sensitive RTP-based UDP flows is an important feature of DSASync. However, in practice, DSASync's passive connection identification—by sniffing packets-in-transit—may not be able to identify all the RTP-based UDP flows. Though it is trivial to check for a UDP or TCP packet using the *protocol* or *next header* field of IP header, no such standard mechanism exists for identifying RTP header that is part of application layer payload. RTP packets do not have preassigned specific port number and do not have standard signature, which further limits the ability to identify RTP sessions. In our implementation, we use a method similar to that of packet sniffing tool *Ethereal* [28], which uses packets seen earlier (e.g., SIP or RTSP packets) during the setup of connection to identify the RTP sessions. We improve this approach by looking for specific port ranges that are typically used by applications for RTP session setup and subsequent data transfer. Though this approach works fairly well, it cannot capture all RTP-based UDP flows.

*2) Connections With Encrypted Traffic:* DSASync, in its current form, cannot be utilized for traffic that is based on encryption below the transport layer, e.g., IPSec—which encrypts IP payload including transport/application headers. Since the encryption is end-to-end, DSASync, as a third entity, cannot sniff or classify the packet in transit. Thus, it is unable to recognize and manage such connections comprehensively, though buffering strategy can still be used. Most applications, however, utilize encryption at higher layers (e.g., TLS/SSL), which has no impact on DSASync's connection identification process. However, RTP-based connection management strategy for UDP may not be feasible because application payload is encrypted, and hence, RTP headers cannot be identified.

## V. Implementation

We evaluate DSASync by implementing it as a Linux kernel module. The MadWifi device driver (madwifi-0.9.4) [29] is augmented to emulate DSA protocol features (e.g., spectrum sensing and channel-switches) over 802.11 MAC. Incumbent transmissions are also emulated through a modified MadWifi-based 802.11 MAC, but with backoff features disabled (e.g., *TXOP backoff* is 0). We implemented a generic functional abstraction of DSA protocols rather than a specific DSA protocol because currently no DSA standard exists. However, the emulation is mainly derived from the 802.22 draft [3]. Note that a consensus on DSA protocol proposals is yet to emerge in the wireless networking research community. An advantage of this evaluation methodology is that it shows our solution to be generic and applicable to any DSA protocol. Also, we use Wi-Fi channels for experiments due to lack of transmission license on licensed channels.

The implementation of DSASync is simplified, as it has been developed with realistic deployment as a key design goal. Since the DSAN is a single homogeneous wireless cell (see Section III-B) with nodes operating on the same DSA MAC/PHY protocol, both DSASync_LL as well as DSASync_TCP and DSASync_UDP need to execute only at the BS. There are two factors enabling this. First, as mentioned earlier, the required parameters are easily accessible from the link-layer module. Second, the BS, in its role as the "manager" of the DSAN, has necessary knowledge about the network state (including the state of other nodes).[4]

The key challenge faced during the implementation involves sniffing of RTP connections transiting through the BS, which is required by DSASync_UDP. As discussed in Section IV-E, we rely on detection of session setup packets (e.g., SIP/SDP packets) and typical port numbers used by VoIP applications to identify RTP-based UDP connections.

## VI. Evaluation

### A. Testbed Setup

A testbed is built according to our system model (see Fig. 1) and consists of a WLAN cell with six client laptops (the SHs), each equipped with an Atheros-based Linksys WPC 55AG wireless card. Another laptop acts as the AP (the BS) that interfaces with the wired LAN of our university. The CH is deployed on the wired segment of the university LAN. Though both the SH and CH are part of the same local network, resulting in lower end-to-end latencies than what is typically experienced on the Internet, this setup is adequate for testing DSASync. An additional laptop, acting as the incumbent transmitter, is placed in the vicinity of the WLAN. To further ensure correct incumbent operation, we establish transmission range asymmetry—secondary WLAN nodes can hear incumbent node's transmissions, but not vice versa. The incumbent produces ON/OFF patterns of random durations according to an exponential distribution.[5] The average of ON/OFF duration for the distribution is varied to change the incumbent channel utilization (e.g., avg. $\mathrm{ON/OFF} = 100$ ms/400 ms for 20% utilization). As mentioned earlier, the DSA parameters are derived mainly from the 802.22 draft [3], e.g., we keep the sensing duration in the range of 5–100 ms, and its frequency is every 200 ms to 2 s.

There are five 802.11a channels in our spectrum, and we initiate experiments with the secondary WLAN in channel 36. There is one PU transmitter (as described above) in each channel. *Iperf* (ver. 2.0.4) [30], a commonly used open source network testing tool, is used to generate TCP/UDP traffic for microbenchmark experiments. For macrobenchmarks involving RTP-based UDP connections, we use the open source VoIP application *ekiga* (ver. 3.2.6) [31]. Ekiga (formerly known as GnomeMeeting) is a feature-rich softphone and supports multiple signaling protocols (like SIP, H.323) and commonly used audio/video codecs. We instrumented the ekiga source code, which allows us to exercise fine-grained control over connection parameters as well as observe key events and statistical information about its ongoing connections. Tcpdump [32] is also used to observe the traffic and verify statistics.

The default PHY data rate is 24 Mb/s, while the buffer capacity at the AP is kept at 500 MB each for both TCP and UDP. The default average incumbent channel utilization is 20%, and the average sensing overhead for each Secondary User (SU) is 5% of the runtime. The initial TCP send and receive window size is 256 kB, and each experiment run lasts 20 s. Other default values are: $\alpha_{\min} = 0.5$, $B^{\rm tcp}_{\rm high} = 500$ MB, and $B^{\rm tcp}_{\rm low} = 400$ MB. Saturation level traffic is used for both TCP and UDP, unless otherwise noted.

### B. Performance Metrics

Application-layer goodput is the fundamental performance metric used to evaluate DSASync. End-to-end delay and jitter are other metrics used for analysis. Our metrics are of a higher-layer focus (application traffic performance), as the goal of DSASync is to manage the adverse impact of DSA on ongoing communication. Since DSA (and the underlying protocol) is unaffected, we do not consider DSA-related metrics directly. For each of the experiments, we compare the performance metrics for two cases: 1) DSA operating with DSASync ("DSASync"); 2) DSA operating without DSASync ("Regular").

[4]In the case where all the required information is unavailable at BS, the DSASync_LL component may need to be deployed at the wireless nodes. Control packets can then be used to transmit information to the BS.

[5]Results were statistically similar when other types of probability distributions, like uniform or log-normal distributions were used.

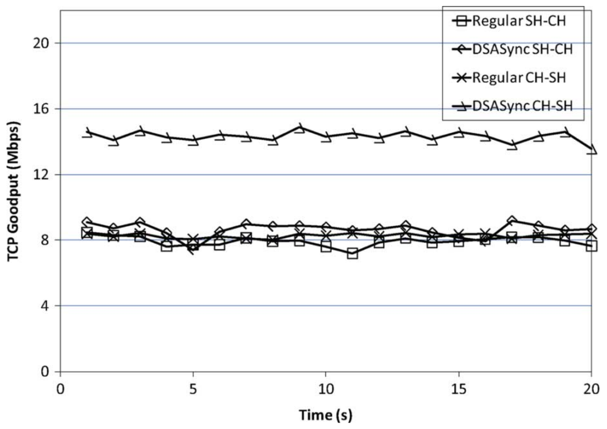

Fig. 3. Average goodput for TCP, each over last 1-s period, during 0–20-s intervals.

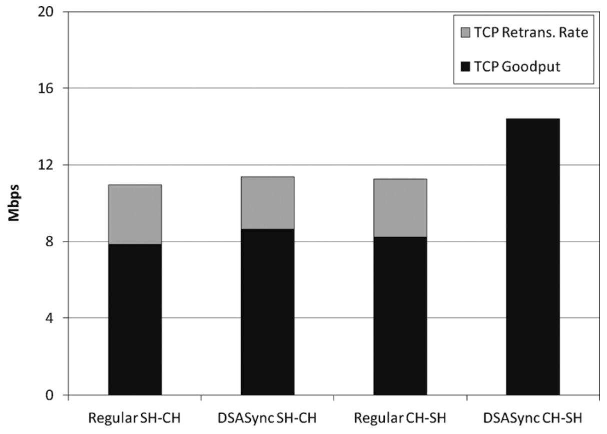

Fig. 4. Average TCP goodput and retransmission rate.

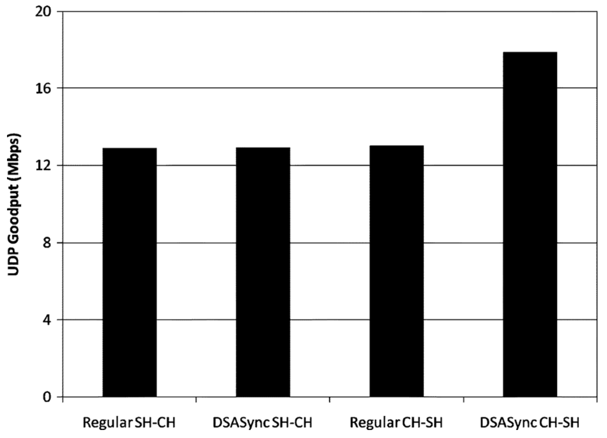

Fig. 5. Average UDP goodput.

### C. Results and Discussion

*1) Overhead Characterization:* To analyze DSASync's runtime overhead, we compare the goodput achieved using unmodified 802.11a with the scenario where DSASync agent is active at the BS. On the basis of 100 experimental runs, the extra overhead with DSASync is found to result in an average of 1.9% reduction in goodput compared to the best case, i.e., the goodput when there is zero DSA overhead. The overhead on end-to-end delay is found to be very minor ($\approx$1.1 ms). However, we observe that gains from using DSASync when DSA is employed (which are discussed next in Section VI-C.2) far outweighs its overhead impact. Thus, DSASync must be activated only when the edge DSAN is actively using DSA.

*2) Microbenchmarks:* To establish the basic performance trends with DSASync, we first evaluate it using a single wireless client in the WLAN cell. Fig. 3 shows the average TCP goodput variation in the time window of 0–20 s. UDP traffic is found exhibit a similar pattern, though the absolute values for goodput are higher because of greater efficiency of UDP resulting from its connectionless nature (no retransmissions, congestion backoff, etc.).

It is seen that employment of DSASync results in better goodput as compared to regular DSA, especially in downlink CH–SH direction. For this scenario, the average TCP goodput improvement is 74% over regular DSA (see Fig. 4). This is a result of DSASync's ability to effectively mask the TFPs (which is 25% of the total runtime) by buffering the incoming packets at the BS and proactively signaling the sender to cease transmission when necessary (see Algorithms 1 and 2). Thus, unnecessary reduction in the send window at CH is avoided, and there is negligible packet loss. Consequently, there is very little retransmission overhead (0.018 Mb/s), contributing to a much improved goodput. Through a packet-level analysis in tcpdump, we notice that the downlink data stream (CH–SH) also benefits from the traffic shaping in the reverse direction (SH–CH). This is because the ACKs are sent to the CH at a lower but steady rate, even during TFPs, which allows CH to continue sending the data packets by advancing its send window.

On the other hand, in absence of DSASync, packets get dropped at the BS during the TFPs. This results in reduction of send window (the sender perceives losses as congestion) and significant retransmission overhead (3.1 Mb/s). Thus, the goodput is much lower.

As seen in Fig. 4, the gains associated with DSASync for uplink (SH–CH) data stream is lower as compared to the downlink direction. Here, the goodput improves by 10% on average. This is because during the TFPs, the data packets originating from the SH side are essentially lost at the SH itself. Thus, the packets do not even reach the BS during interruptions. However, there is still some improvement because the BS shapes the uplink traffic (see Algorithm 4) and also buffers the inbound ACKs for SH.

Similar observations are made for UDP connections, where the average goodput comparison is shown in Fig. 5. Again, the improvement is much higher in the downlink direction (about 38%) as compared to uplink direction due to reasons mentioned above. Since there is no extra burden of retransmissions (even if packets are lost) in UDP, the absolute percentage improvement is lower. Newer packets continue to be transmitted and contribute to UDP goodput. Note that we are not using RTP-based UDP flows for these microbenchmark experiments to eliminate the application-dependent behavior in these results. Thus, only generic and always-guaranteed UDP management benefits are visible here. Depending on how the higher-layer application chooses to respond to RTCP control messages, the advantages of DSASync can be greater, as we evaluate in Section VI-C.3.

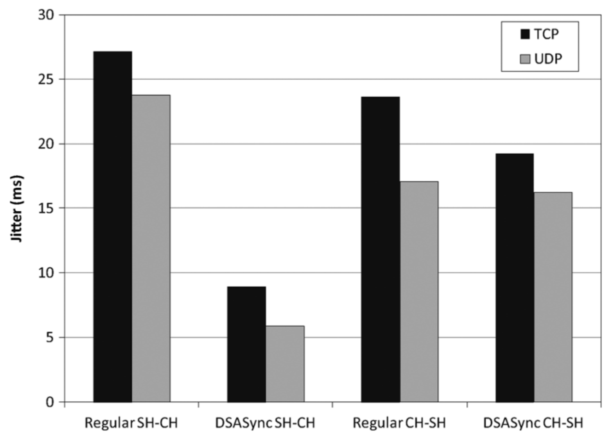

Fig. 6. Average end-to-end jitter at the receiver.

An interesting trend is seen with the delay variation, which is shown in Fig. 6. Variation in delay (at the receiver's end) is a direct indicator of the level of jitter at the application level, which is an important QoS metric. Deployment of DSASync produces a significant reduction in the average jitter at the receiving CH for the uplink traffic, for both TCP and UDP. This is, again, as a result of managing the uplink traffic at the BS. Note that the jitter for downlink data stream remains high, despite substantial improvement in corresponding goodput. This is expected because the SH cannot receive any data packet during TFPs, even though the BS buffers them for it. Also, note that jitter performance for RTP-based UDP connections (not used in this microbenchmark experiment) can be even better, depending on the application-specific behavior.

These observations suggest that deploying a local DSASync agent at each WLAN node would help in reducing the CH–SH delay variation while also improving SH–CH goodput. Our preliminary results with per-node DSASync agent indicate the validity of above conclusion. However, there are also some drawbacks associated with distributed DSASync model, as discussed earlier in Section IV-D.2. We treat this as an optional extension. Note that a DSASync agent at the BS will still be required in the distributed DSASync architecture.

Figs. 7 and 8 show the goodput variation with changes in the magnitude of DSA-related interruptions. The DSA impact is represented by "utilization," which includes sensing overhead and incumbent activity. As expected, the goodput decreases when the DSA behavior becomes more aggressive. However, we note that with DSASync, CH–SH goodput improvement is even better at higher utilizations. The goodput drops significantly only when the utilization factor is greater than 0.5. Note that DSA is not suitable for channels that exhibit very high incumbent utilization. Thus, a good DSA MAC protocol would not select such channels anyway (or would switch away from such channels). DSASync leads to marginally better performance (≈10%) for the uplink data stream. However, the performance drops quickly with increase in utilization, which again highlights the usefulness of a local DSASync agent at each WLAN node. Our experiments also reveal that larger buffer space at the BS improves the resilience provided by DSASync, especially for UDP flows.

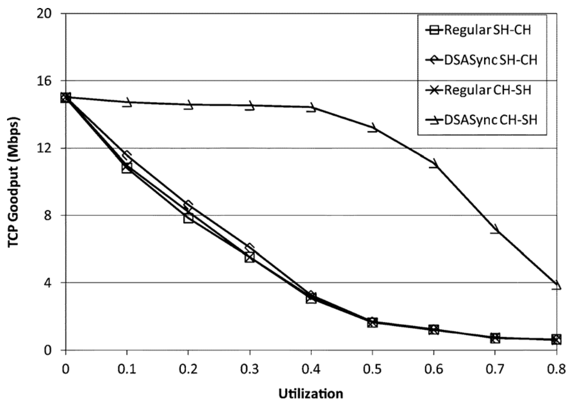

Fig. 7. TCP goodput with varying amount of DSA disruptions.

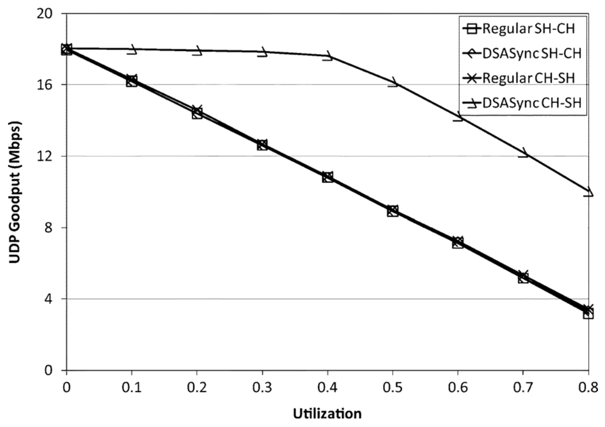

Fig. 8. UDP goodput with varying amount of DSA disruptions.

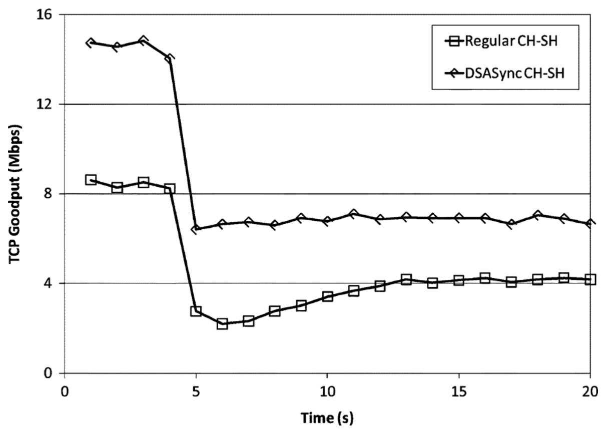

Fig. 9. Effect of PHY capacity change on TCP connection.

Fig. 9 shows the effect of reducing the network capacity, which can occur when the DSAN changes channels. Here, the PHY-layer capacity is reduced to 12 Mb/s from 24 Mb/s at 5 s. As seen in the plot, without DSASync, the CH–SH goodput reduces by almost 70% (the capacity reduction is 50%) and takes some time (6–7 s) to recover. However, with DSASync, there is no perceptible extra reduction in throughput beyond the expected decrease. This is attributed to proactive sender notification through Algorithm 5.

On the other hand, there is no appreciable difference in behavior for UDP connections between the *Regular* and *DSASync*

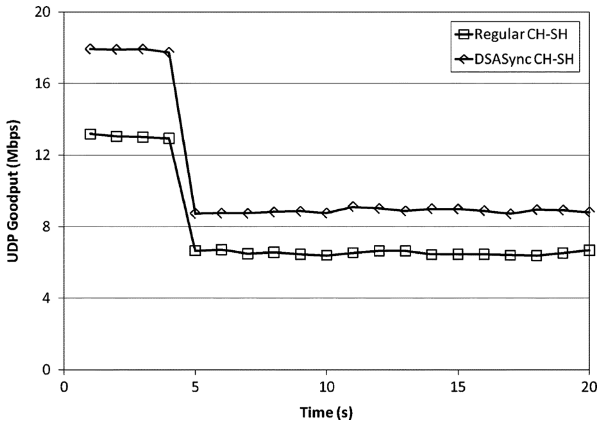

Fig. 10. Effect of PHY capacity change on UDP flow.

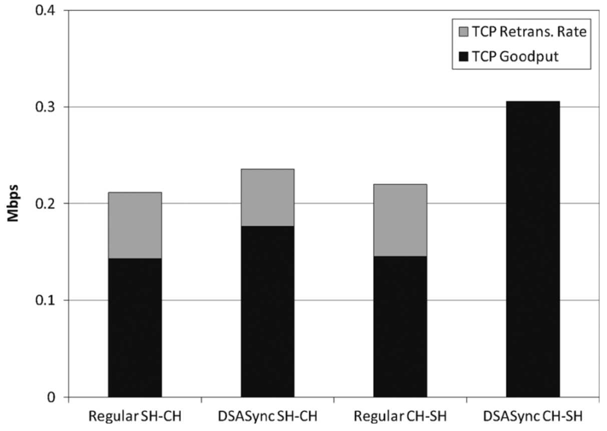

Fig. 11. Average goodput across multiple TCP connections.

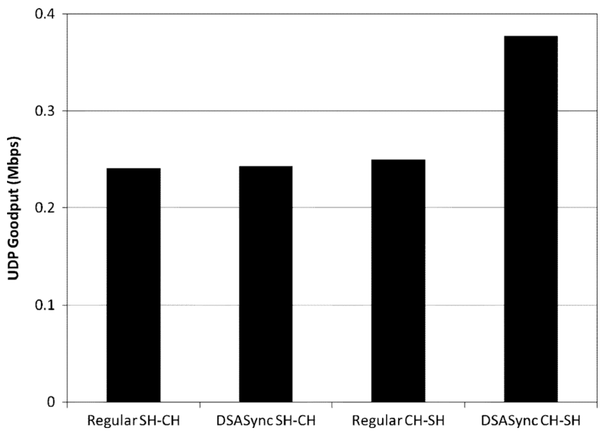

Fig. 12. Average goodput across multiple UDP connections.

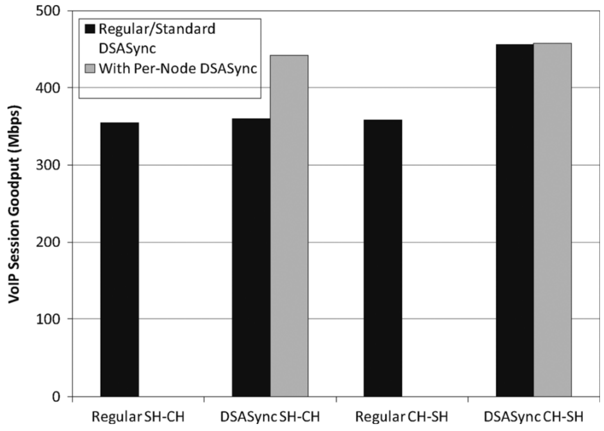

Fig. 13. Average connection goodput for ekiga VoIP sessions.

case, as seen from Fig. 10. This is because the UDP goodput metric is decreased by an equal amount in both cases (note that here losses and retransmissions do not matter), although the buffering mechanism and traffic shaping contribute to higher goodput values when DSASync is used. Again, the RTCP-based proactive feedback mechanism is not active in this microbenchmark experiment. If RTP-based UDP connections are present, such feedback (Algorithm 7) may lead to change in UDP connection behavior depending on application's reaction, e.g., changing to a low-bandwidth codec, etc. We observe this phenomena with ekiga during our macrobenchmark experiments discussed next.

*3) Macrobenchmarks:* To check the scalability of DSASync, four TCP and four UDP connections are started on each of the six clients—thus, there are 48 parallel ongoing connections. Fig. 11 shows the average performance experienced by TCP connections in terms of goodput and retransmission rate. Fig. 12 shows the performance for UDP connections. The trends are similar to those noted in Figs. 4 and 5. DSASync is found to perform even better in a larger-scale situation, especially in the downlink CH–SH direction where goodput improves by about 102% for TCP and 51% for UDP. Similar results, as those noted for microbenchmarks, are observed for other corresponding experiments.

To see the benefits of the DSASync in action, especially for the QoS-sensitive RTP-based UDP traffic, we use ekiga [31] softphone to generate videoconferencing sessions using G.711 (audio) and H.261 (video) codecs with the call speed at 384 kb/s. This requires an actual link bandwidth of around 460 kb/s each way with low jitter for optimum performance. We randomly create between 5–15 sessions in each experiment run, which are distributed among six wireless nodes in the DSAN, with each node communicating with a fixed host in the university network.

Fig. 13 shows the average goodput achieved for the communication sessions, while Fig. 14 shows the average jitter encountered. Deploying DSASync enables better overall bandwidth and significantly lower jitter for the videoconferencing session, which confirms our end-user experience during the active session. Both audio and video quality were found to be perceptibly better when DSASync was active. However, the advantage is skewed toward the downlink (CH–SH) direction, and DSASync can achieve very close to the required bandwidth (460 kb/s) despite DSA disruptions.

The graphs also shows the benefits of deploying a per-node DSASync agent. For this purpose, we implemented and deployed a local DSASync agent at each of the wireless node to manage the downlink/uplink traffic at the node itself. The goodput for uplink (SH–CH) traffic is found to improve, while also reducing jitter substantially for downlink (CH–SH) traffic. Thus, BS-based central DSASync agent together with local DSASync agents seem to be a complete solution to manage DSA-related disruptions on application traffic.

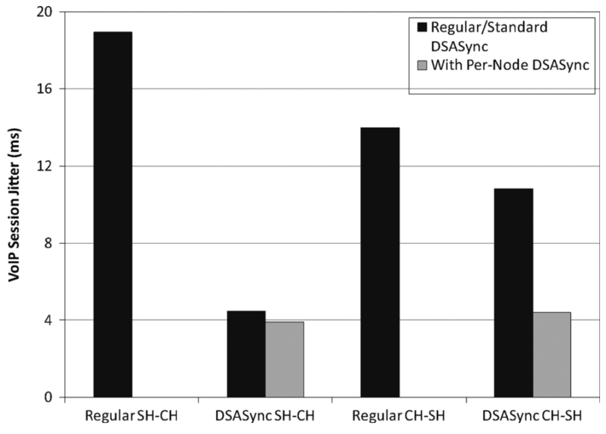

Fig. 14. Average jitter experience by ekiga VoIP sessions.

We conducted a limited set of experiments with Ekiga over the open Internet, where the remote host is on a different ISP. The results obtained were very similar to those described above and supported our observations and inferences obtained on our local testbed.

*4) Remarks:* Many results have been omitted in view of space. But we mention some important observations below.

The usefulness of DSASync is more prominent when DSA-related disruptions are frequent (as seen in Figs. 7 and 8). However, DSASync is designed to gracefully handle DSA side-effects whenever they occur, especially in the short-term before (and during) any generic DSA response (e.g., a channel switch) that kicks in. This property of DSASync is useful in any spectrum environment where DSA is deployed, regardless of the amount, or rate, of disruptions.

Our experiments indicate that the scalability of DSASync depends on the amount of resources, especially buffer space, at the BS. Modern base stations/access points are increasingly getting powerful, and memory is getting cheaper, so it is not a significant constraint. Large buffers will still be very important for supporting large DSANs and would constitute a part of network planning. To optimize this aspect, the buffer size at the BS can be established based on the expected amount of DSA overhead as well as traffic characteristics. A simple dynamic buffer allocation scheme can also be applied for this task.

Furthermore, as seen from the results, there is a good case for deploying local per-node DSASync agents, in addition to the proxy-type DSASync agent at the BS. However, we consider it to be an optional extension of DSASync because it is difficult to ensure that all the wireless clients of a DSAN (like a wireless hotspot) implement this feature. Furthermore, limited resources on many mobile devices (e.g., cellphones) may prevent its deployment. For consumer-oriented WLANs, surveys show that the majority traffic is inbound (downlink) due to dominance of downloads [33]. Thus, standard DSASync platform will still be sufficient for most deployments.

In conclusion, we argue that with the trend of increasing computing power and memory availability at low cost, the extra space/computation overhead associated in running DSASync is insignificant, especially considering the impressive performance gains achieved in this process. More importantly, DSASync accomplishes this with 100% compatibility to existing protocols. Thus, DSASync promises to be an effective network management tool to improve end-to-end connection performance in edge DSANs.

## VII. Concluding Remarks

We identified the important end-to-end communication performance issues when an edge WLAN features DSA capability. In this context, we studied the impact of DSA-related disruptions on TCP/UDP connections to/from the wired cloud. To address the identified problems, we have proposed a novel network management framework called DSASync. DSASync primarily comprises an agent on the wired–wireless interface node (e.g., the base station) of the WLAN, which executes algorithms based on buffering and traffic shaping to minimize the adverse effect on ongoing connections. DSASync features compatibility and ease of deployment as its chief design goals. Consequently, DSASync requires no changes to the TCP/UDP protocols or their existing implementation and maintains the end-to-end semantics. We evaluated DSASync in a testbed based on our prototype implementation for Linux kernel. The testbed consists of an edge DSA-based WLAN interfaced with our university's wired network. The evaluation results indicate that DSASync makes a significant improvement of performance for end-to-end connections, e.g., the downlink goodput increases by 74% for TCP and 38% for UDP in a single-connection environment, with even greater gains (102% for TCP and 51% for UDP) when multiple connections are active in the DSAN. Other QoS metrics are also found to improve significantly, e.g., jitter is reduced by more than 75% for VoIP sessions. Furthermore, DSASync shows resilience in maintaining good end-to-end connection performance with increase in DSA-related disruptions.

We plan to study the possible extensions of the basic DSASync architecture in order to further optimize connection management. We would also like to study QoS-control issues, such as prioritizing connections based on QoS demands of the nodes or applications.

## References

[1] A. Kumar and K. G. Shin, "Managing TCP connections in dynamic spectrum access based wireless LANs," in *Proc. 7th IEEE SECON*, Jun. 2010, pp. 100–108.

[2] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: A survey," *Comput. Netw. J.*, vol. 50, pp. 2127–2159, Sep. 2006.

[3] "IEEE 802 LAN/MAN Standards Committee 802.22 WG on WRANs," 2011 [Online]. Available: http://www.ieee802.org/22/

[4] "Unlicensed operation in the TV broadcast bands: Second memorandum opinion and order," FCC, Washington, DC, ET Docket No. FCC 10-174, Sep. 2010.

[5] R. W. Thomas, L. A. DaSilva, and A. B. MacKenzie, "Cognitive networks," in *Proc. 1st IEEE DySPAN*, Nov. 2005, pp. 352–360.

[6] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proc. 1st ACM MobiCom*, Nov. 1995, pp. 2–11.

[7] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th IEEE ICDCS*, May 1995, pp. 136–143.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," Internet RFC 3550, Jul. 2003 [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[9] M. Buddhikot, “Understanding dynamic spectrum access: Models, taxonomy and challenges,” in *Proc. IEEE DySPAN*, Apr. 2007, pp. 649–663.
[10] S. Haykin, “Cognitive radio: Brain-empowered wireless communications,” *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 201–220, Feb. 2005.
[11] S. Shankar, C. Cordeiro, and K. Challapali, “Spectrum agile radios: Utilization and sensing architectures,” in *Proc. IEEE DySPAN*, Nov. 2005, pp. 160–169.
[12] S. Hamouda and B. Hamdaoui, “Dynamic spectrum access in heterogeneous networks: HSDPA and WiMAX,” in *Proc. IEEE IWCMC*, Jun. 2009, pp. 1253–1257.
[13] M. A. McHenry and K. Steadman, “Spectrum occupancy measurements,” Shared Spectrum Company, Vienna, VA, 2004 [Online]. Available: http://www.sharedspectrum.com/measurements
[14] D. Chen, S. Yin, Q. Zhang, M. Liu, and S. Li, “Mining spectrum usage data: A large-scale spectrum measurement study,” in *Proc. ACM MobiCom*, Sep. 2009, pp. 13–24.
[15] B. Hamdaoui and K. G. Shin, “OS-MAC: An efficient MAC protocol for spectrum-agile wireless networks,” *IEEE Trans. Mobile Comput.*, vol. 7, no. 8, pp. 915–930, Aug. 2008.
[16] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh, “White space networking with Wi-Fi like connectivity,” in *Proc. ACM SIGCOMM*, Aug. 2009, pp. 27–38.
[17] C. Cordeiro and K. Challapali, “C-MAC: A cognitive MAC protocol for multi-channel wireless network,” in *Proc. IEEE DySPAN*, Apr. 2007, pp. 147–157.
[18] “IEEE Standards Coordinating Committee 41 (Dynamic spectrum access networks),” [Online]. Available: http://www.ieeep1900.org
[19] A. Kumar and K. G. Shin, “Extended abstract: Towards context-aware wireless spectrum agility,” in *Proc. 13th ACM MobiCom*, Sep. 2007, pp. 318–321.
[20] K. R. Chowdhury, M. Di Felice, and I. F. Akyildiz, “TP-CRAHN: A transport protocol for cognitive radio ad-hoc networks,” in *Proc. 28th IEEE INFOCOM*, Apr. 2009, pp. 2482–2490.
[21] H. Balakrishnan, S. Seshan, and R. H. Katz, “Improving reliable transport and handoff performance in cellular wireless networks,” *Wireless Netw.*, vol. 1, no. 4, pp. 469–481, Dec. 1995.
[22] *LAN/MAN Committee of IEEE Computer Society*, IEEE Std 802.11-2007, Mar. 2007 [Online]. Available: http://standards.ieee.org/getieee802/download/802.11-2007.pdf
[23] M. Anand, E. B. Nightingale, and J. Flinn, “Ghosts in the machine: Interfaces for better power management,” in *Proc. 2nd Annu. MobiSys*, 2004, pp. 23–35.
[24] B. Higgins, A. Reda, T. Alperovich, J. Flinn, T. J. Giuli, B. Noble, and D. Watson, “Intentional networking: Opportunistic exploitation of mobile network diversity,” in *Proc. 16th ACM MobiCom*, Sep. 2010, pp. 73–84.
[25] H. Kim and K. G. Shin, “Efficient discovery of spectrum opportunities with MAC-layer sensing in cognitive radio networks,” *IEEE Trans. Mobile Comput.*, vol. 7, no. 5, pp. 533–545, May 2008.
[26] M. Gandetto and C. Regazzoni, “Spectrum sensing: A distributed approach for cognitive terminals,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 3, pp. 546–557, Apr. 2007.
[27] N. B. Chang and M. Liu, “Optimal channel probing and transmission scheduling for opportunistic spectrum access,” in *Proc. 13th ACM MobiCom*, Sep. 2007, pp. 27–38.
[28] “Ethereal,” 2007 [Online]. Available: http://www.ethereal.com/
[29] “The MadWifi project,” 2009 [Online]. Available: http://madwifi-project.org
[30] “Iperf,” 2011 [Online]. Available: http://sourceforge.net/projects/iperf
[31] “Ekiga,” 2011 [Online]. Available: http://ekiga.org
[32] “Tcpdump,” 2011 [Online]. Available: http://www.tcpdump.org/
[33] “Advanced RF management for wireless grids,” Aruba Networks, San Jose, CA, 2004 [Online]. Available: http://www.reacttechnologies.com/cgi-script/csNews/news_upload/React_20News_2edb.RF-for-Grids.pdf

**Ashwini Kumar** (M’11) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2010.

He currently works with Juniper Networks, Inc., Sunnyvale, CA. His research interests are QoS issues and resource management in wireless networks, including cognitive radio networks.

**Kang G. Shin** (S’75–M’78–SM’83–F’92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is the Kevin and Nancy O’Connor Professor of Computer Science with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor. He has supervised the completion of 68 Ph.D.’s and authored or coauthored about 760 technical articles, one textbook, and more than 20 patents or invention disclosures. He has also cofounded a couple of startups. His current research focuses on computing systems and networks as well as on embedded real-time and cyber-physical systems, all with emphasis on timeliness, security, and dependability.

Prof. Shin is a Fellow of the Association for Computing Machinery (ACM). He served on Editorial Boards, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and the *Transactions on Embedded Systems*. He has also served or is serving on numerous government committees, such as the US NSF Cyber-Physical Systems Executive Committee and the Korean Government R&D Strategy Advisory Committee. He has chaired several major conferences, including ACM MobiCom 2009, IEEE SECON 2008, ACM/USENIX MobiSys 2005, IEEE RTAS 2000, and IEEE RTSS 1987. He has received numerous best paper awards, including the Best Paper Awards from the 2011 IEEE International Conference on Autonomic Computing, the 2010 and 2000 USENIX Annual Technical Conferences, as well as the 2003 IEEE Communications Society William R. Bennett Prize Paper Award and the 1987 Outstanding IEEE TRANSACTIONS OF AUTOMATIC CONTROL Paper Award. He has also received several institutional awards, including the Research Excellence Award in 1989, the Outstanding Achievement Award in 1999, the Distinguished Faculty Achievement Award in 2001, and the Stephen Attwood Award in 2004 from the University of Michigan (the highest honor bestowed to Michigan Engineering faculty); a Distinguished Alumni Award from the College of Engineering, Seoul National University, in 2002; the 2003 IEEE RTC Technical Achievement Award; and the 2006 Ho-Am Prize in Engineering (the highest honor bestowed to Korean-origin engineers).