This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Secon 2010 proceedings.

Managing TCP Connections in Dynamic Spectrum Access Based Wireless LANs

Ashwini Kumar and Kang G. Shin
Real-Time Computing Laboratory, Department of EECS
The University of Michigan, Ann Arbor, MI 48109-2121
Email:{ashwinik,kgshin}@eecs.umich.edu

***Abstract*—Wireless LANs have been widely deployed as edge access networks in home/office/commercial buildings, providing connection to the Internet. Therefore, performance of end-to-end connections to/from such WLANs is of great importance to network applications and end-users. The advent of Dynamic Spectrum Access (DSA) technology is expected to play a major role in improving wireless communication. With DSA enabled, WLANs opportunistically access licensed channels in order to enhance spectrum-usage efficiency, and provide better network performance. In this paper, we explore issues and solutions in realizing this potential of DSA. We first identify the key issues that impact end-to-end TCP performance when a DSA-enabled WLAN is integrated with the wired cloud. Then, we propose a new network management framework, called `DSASync`, to eliminate or mitigate the performance issues we identified. `DSASync` requires no modifications to the fixed wired network or existing network stack, while ensuring traditional TCP semantics to be obeyed. `DSASync` uses a combination of buffering and traffic-shaping algorithms to minimize the adverse side-effects of DSA on the TCP flows. Finally, we evaluate `DSASync` through a prototype implementation and deployment in a testbed. The results show significant improvements in TCP performance, e.g., a 74% increase in downlink goodput, making it a promising step forward towards applying DSA technology in consumer WLANs.**

I. Introduction

The primary function served by a majority of 802.11 WLANs (e.g., Wifi hotspots, home/office wireless/mesh networks) is to act 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 (e.g., web-access or VoIP) seamlessly over the wireless medium. However, increase in wireless coverage has also led to crowding on the ISM spectrum bands, resulting in higher interference and poorer wireless networking performance. Further, demand for high bandwidth and QoS-sensitive networking services is also growing. Dynamic Spectrum Access (DSA) [1], based on cognitive radio (CR) technology [2], [3], is emerging as a key solution to this potential performance shortfall in existing WLANs.

DSA is witnessing active research and standardization (e.g., IEEE 802.22 [4]), with FCC having already approved commercial unlicensed operations in UHF spectrum [5]. With DSA functionality, a WLAN can opportunistically communicate on licensed spectrum bands, subject to regulatory constraints. A WLAN with DSA capability is referred to as a *DSA Network* (DSAN). We argue that effective integration of DSANs with existing networking infrastructure is important for the success of DSA.

Currently, there is lack of "end-to-end" insights into DSA. In this paper, we investigate issues of integrating a DSAN with the wired network. Such DSANs are expected to be utilized by consumers in a similar manner as most existing WLANs are—as first/last-mile access networks. Thus, DSANs must exceed the end-to-end performance of traditional WLANs in order to be commercially viable.

However, DSA entails additional operational constraints in a rapidly-changing spectrum environment. It involves a number of fundamental activities that can be disruptive to ongoing network traffic. Examples include spectrum sensing, channel switching, spectrum management & coordination, and incumbent activity. Apart from performance degradation at the link layer, such disruptive DSA-related phenomena can make long-term adverse impacts on the end-to-end communication. This is especially true for TCP streams, as DSA semantics are unknown to it.

For example, a TCP connection between a server host in the cloud and a client on the DSAN will experience timeouts when the client cannot send out ACK packets in time, because of a DSA-induced quiet period on incumbent detection. Consequently, the TCP's congestion control mechanism will be unnecessarily invoked leading to further performance degradation. Such interruptions can be frequent—given the regulatory restrictions imposed on unlicensed operations, DSA service provider requirements, as well as, incumbent activity on the spectrum band.

Techniques have been proposed in the past to address performance problems arising in TCP flows due to disruptions in the presence of high bit-error rate experienced on wireless medium. In modern WLANs, bit-error losses are a minor issue because of sophisticated error detection/correction schemes used today. In the context of DSANs, delays and losses are primarily due to side-effects of DSA-related events, which can produce a significant adverse impact. Unlike random wireless errors, knowledge about many of the disruptive DSA events can be obtained beforehand or at their onset, thus making a proactive approach feasible in masking their side-effects at the TCP level. If any information is not directly available, historical observations can be utilized.

In this paper, we provide a comprehensive end-to-end solution, called `DSASync`, to address the TCP performance issues

978-1-4244-7149-2/10/$26.00 ©2010 IEEE

when integrating a DSAN with the wired cloud. DSASync is a network management framework for regulating TCP connections traversing the wired-wireless boundary. DSASync incorporates algorithms based on buffering and traffic-shaping to minimize adverse impacts on ongoing TCP streams. There are two important advantages of DSASync—(a) it maintains the end-to-end semantics of the existing TCP protocol, and, (b) it requires no changes to TCP's existing implementations.

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.

The contributions of this paper are three-fold. First, we identify the key challenges for the mainstream integration of DSA-based WLANs. Second, we propose DSASync to address the identified issues in the context of end-to-end TCP connections. Third, DSASync is shown to better enable DSAN integration with the Internet via a testbed-based evaluation.

The paper is organized as follows. Prior related work is discussed in Section II. We present the background of this work in Section III: the DSAN integration issues in Section III-A and system model & notation in Section III-B. 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. Reference [1] is a general survey about the state-of-art in this field. Cognitive radio, the platform for DSA, has been discussed in [3], [6].

FCC has approved preliminary guidelines for DSA operations in TV bands [5]. Several DSA MAC/PHY protocols have already been proposed in literature, especially for TV bands [7], [8]. The IEEE 802.22 Group [4] is standardizing Wireless Regional Area Networks (WRANs).

However, there have been very few publications on the end-to-end impact of DSA. Adaptation to application requirements in DSA has been proposed in [9]. However, that approach is node-centric rather than network-centric, and does not account for impacts on the transport layer. The authors of [10] identified the important issues afflicting 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. DSASync, on the other hand, integrates DSANs with the Internet without any changes to the existing applications or the protocol stack.

As mentioned earlier, there have been several prior efforts on improving TCP performance in wireless environment [11]–[13]. The key ideas proposed were: (a) splitting of wired and wireless TCP connections, or (b) localized caching and retransmissions when a packet loss was identified. These efforts were motivated by the high error-rate in wireless medium which caused substantial performance drop in TCP connections. With the advent of higher data-rate standards and stronger error-correction schemes [14], such concerns have been mitigated. However, from these works, DSASync borrows the concept of proxy-based packet buffering.

III. Issues, System Model and Notation

A. Integration Issues

A DSAN exhibits several characteristics that adversely impact its effectiveness in functioning as an edge access network, especially for TCP connections across the wired-wireless boundary (see system model in Fig. 1). The major issues are listed below.

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 [15], but they may also be needed for accuracy and correctness in a scenario where an external sensing infrastructure is used. 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 ms.[2] In general, DSA protocols schedule sensing with a higher frequency in order to improve sensing accuracy and hence, enhance DSA performance [17]. For example, nodes can sense *out-of-band* channels to get a better picture of spectrum conditions [18].

Channel-switch delays: Channel-switches can occur frequently during unlicensed operation. Depending on the underlying DSA protocol, a channel-switch may be made proactively to exploit a better channel, or in the event of incumbent transmission on the current channel. Each channel-switch can incur noticeable delay and QoS degradation [18], e.g., due to the interface reset or coordination between nodes. Channel-switches also contribute to the problem of "bandwidth fluctuation" discussed next.

Bandwidth fluctuation: During DSA operation, nodes can experience wide variations in available bandwidth for several reasons. Spectrum available for unlicensed usage on the current channel depends on the incumbent utilization fraction, which can change dynamically and substantially, depending on the load in the incumbent network. Presence of additional WLANs in the vicinity (on the same channel) further decreases available bandwidth for application traffic in a DSAN. Further, channel-switches may also contribute to bandwidth variability. This can occur because of: (a) different channel utilization on

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

[2]For TV bands, *fine sensing* takes around 25ms and incumbent detection must be within 2s according to FCC rules [5], [16].

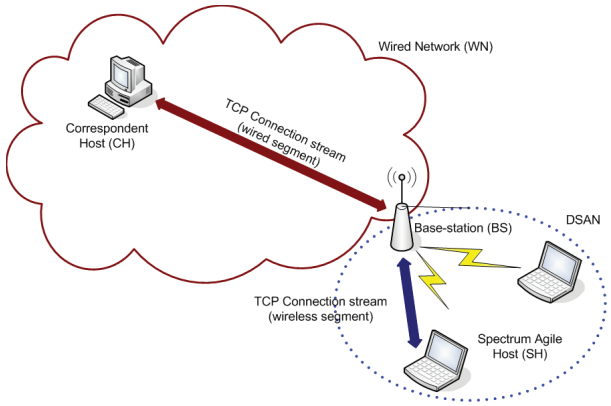

Fig. 1. The system model for a DSAN.

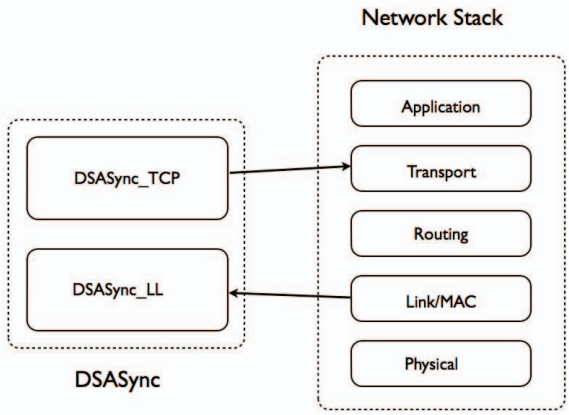

Fig. 2. Architectural overview of DSASync.

the new channel, (b) different channel-access strategy resulting in more/less throughput efficiency, and (c) different spectrum-width (or raw capacity) of the new channel.

Incumbent activity: Incumbent transmissions must be protected when a DSAN operates on a licensed channel. Thus, when an incumbent or *Primary User (PU)* activity is detected (via sensing), the DSAN nodes must not begin transmission 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, as well as, drop in available bandwidth. Although the underlying DSA protocol is usually 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 the ongoing communication.

Note that DSA operation is fundamentally disruptive, especially on a short-term scale, which cannot be completely eliminated. Therefore, the design principle of DSASync is to carefully manage TCP streams, in order to minimize the impact of the aforementioned disruptive events experienced when DSA is active on the WLAN.

B. System Model and Notation

The system under consideration is a single-hop WLAN with wireless devices that connect to the wired cloud through a base station, as shown in Fig. 1. The WLAN acts as the first/last-mile/edge access network. The 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).

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.
- *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 a SH. The CH is usually a fixed host in the Internet cloud (the WN).
- *Base Station* (BS): The designated device (or access point) that connects the DSAN to the WN.
- *Transmission Freeze Period* (TFP): The interval 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 network management protocol (similar to Snoop Agent [11]). However, DSASync manages TCP connections—Fig. 2 shows DSASync's architecture schema. To achieve this, DSASync sniffs the packets in transit (at the BS), and maintains state information (e.g., last ACK copy, sequence #s, etc) for each ongoing TCP stream it detects.

A. DSASync: Link Layer

The DSASync LL component (DSASync_LL) is the monitoring unit of DSASync. It collects and maintains information about DSA parameters required by DSASync_TCP. Such parameters are managed by the DSA MAC/PHY protocol, and are typically available at the link layer. The parameters of interest are as follows.

1) $f_{sense}^{DSAN}(t)$ = the frequency of spectrum sensing by the entire DSAN. This parameter usually corresponds to the cooperative sensing schedule in which all nodes participate.
2) $t_{sense}^{DSAN}(t)$ = the duration of each spectrum sensing event scheduled by the DSAN.

3) $f^i_{sense}(t)$ = the frequency of additional sensing (e.g., out-of-band) sensing performed by node i.
4) $t^i_{sense}(t)$ = the duration of each of the node-specific sensing event at node i.
5) $f^{DSAN}_{switch}(t)$ = the frequency of channel switches.
6) $t^{DSAN}_{switch}(t)$ = the delay involved in each channel switch.
7) $g^{ON}_{PU}(t)$ = the PU's ON time distribution.
8) S_{DSAN} = the Boolean parameter indicating if sensing is currently ongoing in the DSAN.
9) SW_{DSAN} = the Boolean parameter indicating if the DSAN is currently performing a channel-switch.
10) S_i = the Boolean parameter indicating if sensing is currently ongoing at node i.
11) PU_{ON} = the Boolean parameter indicating if there is currently a PU activity on the current channel.

In the rare scenario where a parameter is not directly available from the DSA protocol, `DSASync_LL` uses simple history-based estimates for each of them. Some of the parameters can be time-varying, as shown. In practice, knowledge of their current values is found to be sufficient for `DSASync`.

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 to/from the wireless clients in a DSAN. The objective is three-fold: (a) to minimize packet loss, (b) to minimize time-outs and hence, retransmissions, (c) to adjust TCP connection parameters in response to changes in available bandwidth. In the current 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 Section III-B).

a) **`DSASync_TCP_CH-SH`***:* This module buffers the downlink packets during TFPs. The buffered packets are transmitted from the BS to the SH when the transmission can be resumed. The current state of the destination wireless node (w.r.t. its packet-reception capability) is known from `DSASync_LL`. Note that the DSA MAC protocol, which is monitored by `DSASync_LL`, typically features network coordination mechanisms (e.g., via control channel [7], [19]) of information like sensing schedule, or incumbent detection events. This ensures that nodes in the DSAN are on the same page. Thus, onset and expiry of TFPs can be obtained in a sufficiently timely manner without introducing any extra overhead.

Due to limited buffer space at the BS, it is possible to run out of space before the transmission is resumed. This can happen, for example, if the DSAN is blocked from transmission for a long period due to ongoing incumbent activity. In such a situation the CH may also time out waiting for ACKs from the SH, thus incurring unnecessary retransmission overhead. To prevent this, `DSASync_TCP` proactively pauses the sender by exploiting the built-in flow control mechanism of TCP.

Let there be N nodes in the DSAN and the allocated space (at the BS) for buffering CH-SH packets be B_{alloc}. B_{low} & B_{high} are the configuration parameters for buffer space thresholds ($B_{alloc} > B_{high} > B_{low}$). B_{free} is the current free buffer space. For the TCP connection j, SN^j_{last} is the latest sequence # acknowledged by the SH, SN^j_{new} specifies the sequence # of the latest data packet (coming from CH) buffered at BS, and $rwin^j$ 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

Algorithm 1 Algorithm TCP_CH-SH-a

Require: B_{free}, B_{low}, $hold$
$p \leftarrow$ incoming CH-SH pkt
$dest \leftarrow$ destination SH of p
$src \leftarrow$ source CH of p
$conn \leftarrow p$'s TCP connection identifier
$TFP \leftarrow SW_{DSAN}|S_{DSAN}|S_{dest}|PU_{ON}$
if $TFP = 0$ **then**
  Add p to transmit queue
else
  Buffer pkt
  if $hold = false$ **then**
    if $B_{free} < B_{low}$ **then**
      $hold = true$
      for each TCP connection **do**
        Advt. zero rwin to sender CH
      end for
    else
      if $SN^{conn}_{new} = SN^{conn}_{last} + rwin^{conn}$ **then**
        Advt. zero rwin src for $conn$
      end if
    end if
  else
    if p=window update request **then**
      Advt. zero rwin to src for $conn$
    end if
  end if
end if

Algorithm 2 Algorithm TCP_CH-SH-b

Require: B_{free}, B_{high}, $hold$
if $hold = true$ **then**
  if $B_{free} > B_{high}$ **then**
    $hold \leftarrow false$
  end if
end if
for $i \leftarrow 1$ to N **do**
  $TFP \leftarrow SW_{DSAN}|S_{DSAN}|S_i|PU_{ON}$
  if $TFP = 0$ **then**
    Unbuffer any i's pkt to transmit queue
  else
    Buffer any i's pkt from transmit queue
  end if
end for

TCP traffic. Algorithm 1 is executed when a TCP packet is received from the CH and destined for a node in the DSAN, i.e., for each TCP packet about to be added to the outgoing queue at the BS's wireless interface. Algorithm 2 is executed periodically, based on a sufficiently frequent timer like the system timer. A separate process updates buffer sizes (when a packet is added/removed) and also overwrites the $rwin$ field to 0 in outgoing packets, if the $hold$ parameter (see Algorithms 1 & 2) is true.

Algorithm 1 exploits the TCP flow control mechanism to avoid buffer overflow (and hence dropped packets) at the BS. The BS advertises a 0-size receive window on behalf of the SH, when the buffer threshold is reached. The same strategy is used to prevent retransmissions (due to timeouts at the sender CH), when the receive window becomes full while still in TFP. The algorithm does not address non-DSA factors, such as congestion in the network. However, an older packet is replaced with a newly-arrived packet with the same sequence #, 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. Further, the resource overhead will be higher.

b) **`DSASync_TCP_SH-CH`***:* TCP performance will also degrade due to irregular behavior in the opposite direction. For example, there will be timeouts and retransmissions when a TFP sets in, as the CH may not receive ACKs in time according to its RTT estimate—which would be quite low as it was based on continuous packet reception during the past non-TFP period. Further, a "start-and-stop" type of data packet reception would also contribute to other QoS issues, such as increased application jitter.

Thus, for the uplink TCP stream, i.e., from SH to CH, the BS attempts to "smooth" the flow. The key idea is to spread the packets transmitted from the SH to CH over the TFPs, so that the CH sees a relatively steady stream of packets despite the disruption at the source SH. Thus, the temporal discontinuities in packet reception are masked. This is accomplished as follows.

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.

$$\begin{aligned} TFP_i^{avg} &= E[f_{sense}^{DSAN}(t).t_{sense}^{DSAN}(t).t]_{t=T-\triangle T}^{t=T} \\ &+ E[f_{sense}^{i}(t).t_{sense}^{i}(t).t]_{t=T-\triangle T}^{t=T} \\ &+ E[f_{switch}^{DSAN}(t).t_{switch}^{DSAN}(t).t]_{t=T-\triangle T}^{t=T} \\ &+ E[g_{PU,ON}(t).t]_{t=T-\triangle T}^{t=T} \end{aligned}$$

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

$$\alpha_i = 1 - \frac{TFP_i^{avg}}{\triangle T}. \quad (1)$$

Historical information on TFP durations during a moving time-window of size $\triangle T$ can be utilized to compute α_i.

Let α_{min} be the configuration parameter to limit the extent of traffic shaping. In order to manage SH-CH 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 uplink TCP packets as:

$$D_{eff,src} = \beta_{src}.D_{src} \quad (2)$$

where $\beta_{src} = max(\alpha_{src}, \alpha_{min})$, and D_{src} 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 SH-CH TCP traffic. The α_{min} parameter provides control over (a) excessive delays (and hence very high response-times for applications), and (b) buffer space run-out.

D_{src} 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,

Algorithm 3 Algorithm TCP_SH-CH

Require: α_{min}, $\forall i \in N$, D_i, α_i, T_i, $dequeue_i$

while SH-CH queue is non-empty **do**
  $p \leftarrow$ 1st TCP pkt in queue
  $done \leftarrow false$
  $count \leftarrow 1$
  while $done = false$ and $count \leq N$ **do**
    $src \leftarrow$ source of p
    if $dequeue_{src} = false$ **then**
      $\beta_{src} \leftarrow$ max(α_{src}, α_{min})
      $T_{src} \leftarrow$ timestamp of src's last pkt. dequeue
      $T_{curr} \leftarrow$ current timestamp
      $elapsed \leftarrow T_{curr} - T_{src}$
      if $\{size(p)/(elapsed) < \beta_{src} D_{src}\}$ **then**
        $dequeue_{src} \leftarrow true$
        $done \leftarrow true$
      end if
    end if
    $count \leftarrow count + 1$
    $p \leftarrow$ next TCP pkt in queue
  end while
end while

Algorithm 4 Algorithm TCP_SH-CH-OPT

Require: α_{dsan}, D_{dsan}, T_{dsan}, $dequeue_{dsan}$

while SH-CH queue is non-empty **do**
  if $dequeue_{dsan} = true$ **then**
    Wait for dequeue to complete
  end if
  $p \leftarrow$ 1st TCP pkt in queue
  $\beta_{dsan} \leftarrow$ max(α_{dsan}, α_{min})
  $T_{dsan} \leftarrow$ timestamp of last pkt dequeue
  $T_{curr} \leftarrow$ current timestamp
  $elapsed \leftarrow T_{curr} - T_{dsan}$
  if $\{size(p)/(elapsed) < \beta_{dsan} D_{dsan}\}$ **then**
    $dequeue_{dsan} \leftarrow true$
  else
    Wait $\{(size(p)/\beta_{dsan} D_{dsan}) - (elapsed)$ interval
  end if
end while

T_{src} and $dequeue_{src}$ data structures (used in Algorithm 3) are updated by monitoring the dequeue events. Note that the packets enter the queue in the temporal order they are received, as before. The local variables $done$ and $count$ ensure that the number of iterations is bounded while searching for a packet to be dequeued.

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 DSAN. Thus, each $\alpha_i, \forall i \in N$ can be closely approximated by the average of α_i values, say α_{dsan}. Further, since the packets from nodes are queued on a first-come-first-serve basis, the individual data rates can now also be replaced by the overall incoming data-rate D_{dsan}.

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

c) **`DSASync_TCP_CAP`***:* TCP's flow and congestion control mechanisms allow 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. [3]

For CH-SH downlink traffic, the procedure outlined in Algorithm 5 is executed when a channel-switch event is indicated (through `DSASync_LL component`). In this algorithm, C is the raw physical-layer bandwidth on the new channel, while e is the data transfer efficiency for TCP when paired with the MAC/PHY protocol to be used in the new channel. D_i^{in} denotes the incoming (or downlink) data-rate for node i. Note that D_i^{in} can be calculated at the BS by a simple sliding time-window based historical average of the received packets destined for node i. Algorithm 5, again, uses the built-in congestion control of TCP to trigger fast retransmit/recovery by sending at least 3 duplicate ACKs to the CH, if the current downlink data-rate for a node cannot be sustained on the new channel. The objective is to prevent slow recovery where $cwnd$ is reduced to 1, instead of half of the current value as in fast recovery. Thus, the sender will automatically reduce its sending data-rate with less severe 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.

[3]When the channel capacity increases, the TCP performance gradually improves by itself.

Algorithm 5 Algorithm TCP_CAP

Require: C, e, $D_i^{in}, \forall i \in N$

for $i \leftarrow 1$ to N **do**
  if $D_i^{in} > eC$ **then**
    Send 3 duplicate ACKs to i's CHs
  end if
end for

`DSASync` does not take any action for SH-CH uplink traffic when capacity decreases on a channel-switch, as the uplink data-rate from the source SH is automatically curtailed (due to change in raw network capacity) to reflect the change.

V. Implementation

We evaluate `DSASync` by implementing it as a Linux kernel module. The MadWifi device driver (madwifi-0.9.4) [20] 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).

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` and `DSASync_TCP` need to execute only at the BS. There are two contributing factors. 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 full knowledge about the network state (including the required parameters of other nodes).[4]

VI. Evaluation

A. Testbed Setup

A testbed is built according to our system model (see Fig. 1), and consists of a WLAN cell with 6 client laptops (the SHs),

[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.

each equipped with an Atheros-based Linksys WPC 55AG wireless card. Another laptop acts as the AP (the BS) which interfaces with the wired LAN of our University. The CH is deployed on the wired segment of the University LAN. Thought both 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 between the WLAN and the incumbent—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. The average of ON/OFF duration for the distribution is varied to change the incumbent channel utilization.

Iperf is used to generate TCP traffic. *Tcpdump* is also used to observe the TCP traffic and record statistics. The default PHY data-rate is 24Mbps, while the buffer capacity at the AP is kept at 500MB. The default average incumbent channel utilization is 20%, and the average sensing overhead for each SH is 5% of the runtime. The initial TCP send and receive window size is 256KB and each experiment run lasts 20s. Other default values are: $\alpha_{min} = 0.5$, $B_{high} = 500$MB, and $B_{low} = 400$MB.

B. Performance Metrics

Application-layer throughput is the main performance metric used to evaluate DSASync. End-to-end delay, as observed from the receiver's perspective, is also used for analysis. For each of the experiments, we compare the performance metrics for two cases: (a) DSA operating with DSASync ("DSASync"), (b) DSA operating without DSASync ("Regular").

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 (≈1.1ms). However, we observe that gains from using DSASync when DSA is employed (which are discussed next in Section VI-C2) far outweighs its overhead impact. Thus, DSASync must be activated only when the edge WLAN is actively employing DSA.

2) Microbenchmarks: To establish the basic performance trends with DSASync, we first evaluate it using a single wireless client in the WLAN cell operating on a single channel (i.e., no switching overhead). Fig. 3 shows the average goodput variation in the time-window of 0-20s.

It is seen that DSASync results in better goodput as compared to regular DSA, especially in downlink CH-SH

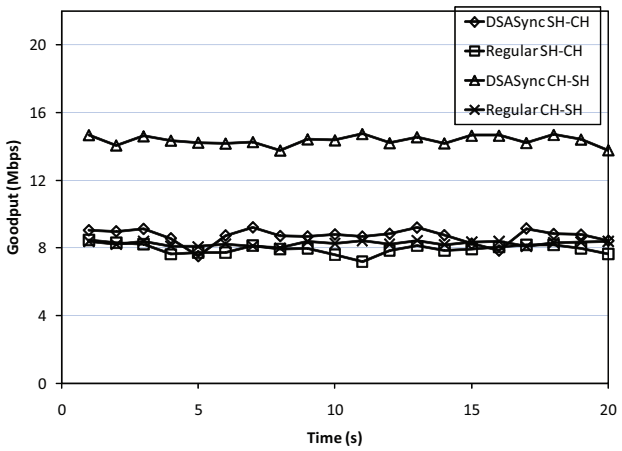

Fig. 3. Average goodput, each over last 1s-period, during 0-20s intervals.

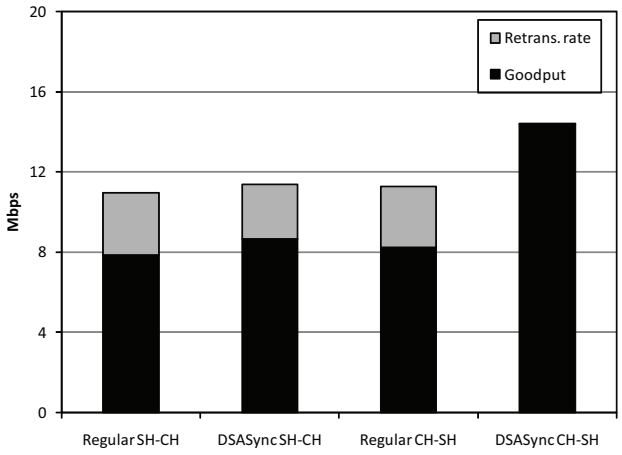

Fig. 4. Average goodput and retransmission rate.

direction. For this scenario, the average 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 (≈0.018Mbps), 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 (3.1Mbps). Thus, the

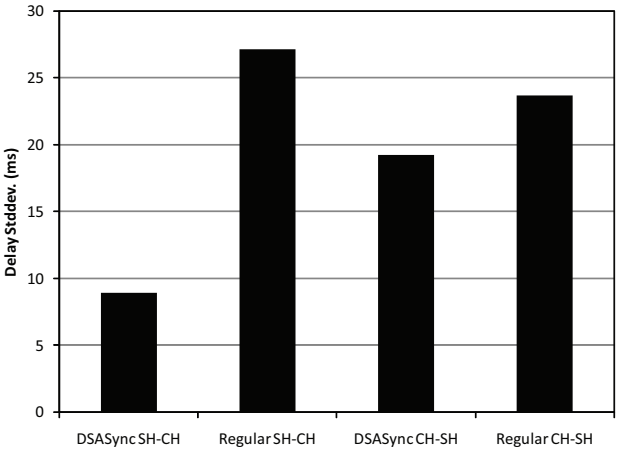

Fig. 5. Average end-to-end delay standard deviation at the receiver.

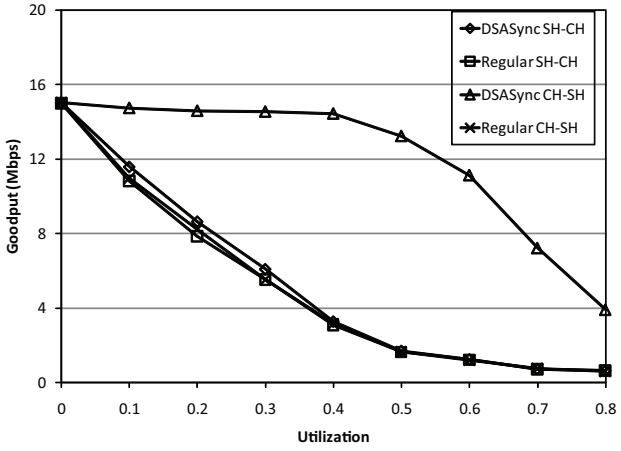

Fig. 6. Goodput comparison with varying amount of DSA impact.

goodput is much lower.

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

An interesting trend is seen with the delay variation from the receiver's viewpoint, which is shown in Fig. 5. Variation in delay at the receiver's end is a direct indicator of application jitter. Despite not producing substantial improvement in goodput, usage of DSASync results in a significant reduction in the average delay variation at the CH for the uplink traffic. This is a result of managing the uplink traffic at the BS. The delay variation for CH-SH downlink data stream remains higher, 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.

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. This is a part of our future work. However, a DSASync agent at the BS will still be required.

Fig. 6 shows 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, 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).

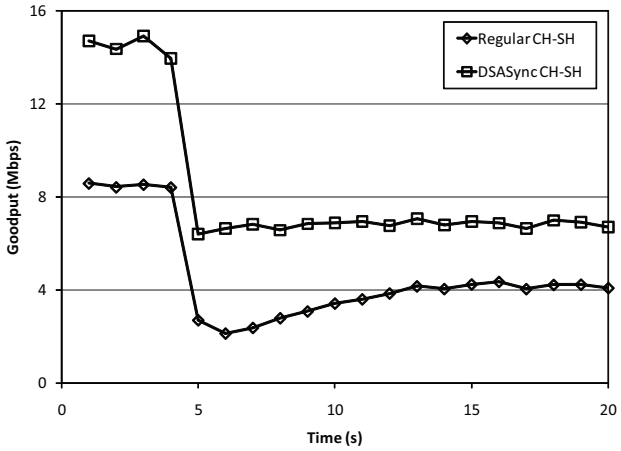

Fig. 7. Effect of PHY capacity change.

DSASync leads to marginally better performance ($\approx 10\%$) for the SH-CH data stream. However, the performance drops quickly with increase in utilization, which again suggests the need for a local DSASync agent at each WLAN node.

Fig. 7 shows the effect of reducing the network capacity, which can occur when the DSAN changes channels. Here the PHY-layer capacity is reduced to 12Mbps from 24Mbps at 5s. As seen in the plot, without DSASync, the CH-SH goodput reduces by almost 70% (the capacity reduction is 50%) and takes some time (6-7s) 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.

3) Macrobenchmarks: To check the scalability of DSASync, 4 TCP connections are started on each of the 6 clients—thus, there are 24 parallel TCP data streams. The spectrum consists of 4 channels, each with time-varying incumbent activity. Fig. 8 shows the average performance of the TCP connections in terms of goodput and retransmission rate. The trends are similar to those noted in Fig. 4. DSASync is found to perform even better in a larger-scale situation,

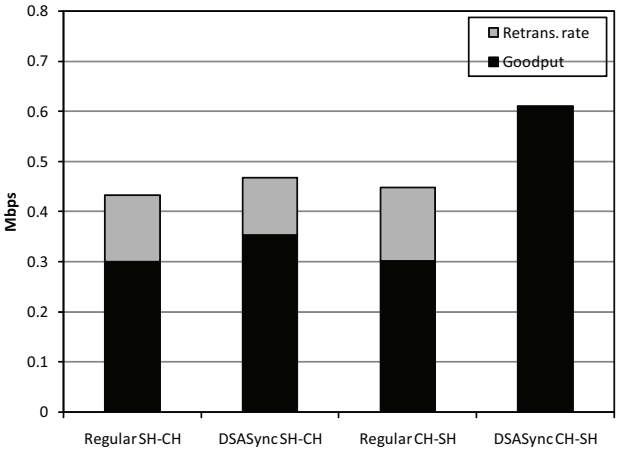

Fig. 8. Performance in a large-scale scenario.

especially in the downlink CH-SH direction where goodput improves by 102%. Similar results, as those noted for microbenchmarks, are observed for other corresponding experiments.

4) Remarks: Many other results have been omitted due to lack of space. From the experiments, we conclude that the buffer size at the BS must be established based on the expected amount of DSA overhead on a channel. A simple dynamic buffer allocation scheme seems well-suited for this task. Further, as noted from the results, there is a good case for deploying local DSASync agents, in addition to the proxy-type DSASync agent at the BS. In conclusion, DSASync, though simple and non-intrusive, promises to be an effective network management tool to improve TCP 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 analyzed the impact of DSA-related disruptions on TCP flows to and from the wired cloud. To address the identified problems, we 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 TCP streams. Consequently, DSASync requires no changes to the TCP protocol or its existing implementation, and maintains the end-to-end semantics necessary for TCP fidelity. We evaluated DSASync in a testbed based on our prototype implementation for Linux kernel. The evaluation results indicate that DSASync makes a significant improvement in goodput and jitter for TCP streams. Further, DSASync shows resilience in maintaining good TCP performance with increase in DSA-related disruptions.

Future work includes development of a distributed DSASync architecture with a local DSASync agent on each wireless node. Our results indicate that even better performance may be achieved in some cases (especially for uplink TCP streams) with such a model. Further, we are looking into extension of DSASync to manage UDP traffic, which forms a significant and growing fraction of Internet traffic, especially multimedia data.

Acknowledgment

The work reported in this paper was supported in part by the NSF under Grant No. CNS-0721529.

References

[1] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt Generation/Dynamic Spectrum Access/Cognitive Radio Wireless Networks: A survey," *Computer Networks Journal (Elsevier)*, vol. 50, pp. 2127–2159, September 2006.

[2] J. Mitola and G. Q. Maguire, "Cognitive radio: Making Software Radios More Personal," in *IEEE Personal Communications*, August 1999, pp. 13–18.

[3] S. Haykin, "Cognitive Radio: Brain-empowered Wireless Communications," *IEEE JSAC*, vol. 23, no. 2, pp. 201–220, February 2005.

[4] *IEEE 802 LAN/MAN Standards Committee 802.22 WG on WRANs*, http://www.ieee802.org/22/.

[5] *FCC Adopts Rules for Unlicensed Use of Television White Spaces*, Federal Communications Commission (FCC) News 202/418-0500, November 2008.

[6] S. Shankar, C. Cordeiro, and K. Challapali, "Spectrum Agile Radios: Utilization and Sensing Architectures," in *IEEE DySPAN*, November 2005, pp. 160–169.

[7] B. Hamdaoui and K. G. Shin, "OS-MAC: An Efficient MAC Protocol for Spectrum-Agile Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 7, July 2008.

[8] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh, "White Space Networking with Wi-Fi like Connectivity," in *ACM SIGCOMM*, August 2009.

[9] A. Kumar and K. G. Shin, "Extended Abstract: Towards Context-Aware Wireless Spectrum Agility," in *13th ACM MOBICOM*, September 2007, pp. 318–321.

[10] K. R. Chowdhury, M. D. Felice, and I. F. Akyildiz, "TP-CRAHN: A Transport Protocol for Cognitive Radio Ad-hoc Networks," in *28th IEEE INFOCOM*, April 2009.

[11] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *1st ACM MOBICOM*, November 1995, pp. 2–11.

[12] H. Balakrishnan, S. Seshan, and R. H. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *Wireless Networks*, vol. 1, no. 4, pp. 469–481, December 1995.

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

[14] *IEEE Std 802.11 - 2007*, March 2007, lAN/MAN Committee of IEEE Computer Society, http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

[15] H. Kim and K. G. Shin, "Efficient Discovery of Spectrum Opportunities with MAC-Layer Sensing in Cognitive Radio Networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 5, May 2008.

[16] C. Cordeiro, K. Challapali, D. Birru, and S. Shankar, "IEEE 802.22: An Introduction to the First Wireless Standard based on Cognitive Radios," *IEEE JSAC*, vol. 1, no. 1, April 2006.

[17] H. Kim and K. G. Shin, "In-band Spectrum Sensing in Cognitive Radio Networks: Energy Detection or Feature Detection?" in *14th ACM MOBICOM*, September 2008, pp. 14–25.

[18] A. Kumar, J. Wang, K. Challapali, and K. G. Shin, "A Case Study of QoS Provisioning in TV-band Cognitive Radio Networks," in *18th IEEE ICCCN (2nd COGCOM)*, August 2009, pp. 1–6.

[19] C. Cordeiro and K. Challapali, "C-MAC: A Cognitive MAC Protocol for Multi-Channel Wireless Network," in *IEEE DySPAN*, April 2007, pp. 147–157.

[20] *The MadWifi Project*, http://madwifi-project.org.