Computer Networks 103 (2016) 101–114

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

TCP over scarce transmission opportunity in cognitive radio networks

Sang-Seon Byun*

Department of Computer Engineering, Catholic University of Pusan, Busan, South Korea

ARTICLE INFO

Article history:
Received 6 July 2015
Revised 29 January 2016
Accepted 20 March 2016
Available online 6 April 2016

Keywords:
TCP
Freeze-TCP
TCP-Freeze-CR
Cognitive radio networks
Software-defined radio
Universal Software Radio Peripheral
GNU Radio
IEEE 802.15.4

ABSTRACT

Transmission control protocol (TCP) is the most popular transport layer protocol for applications that require reliable and ordered data delivery essentially. In this paper we consider the deployment of TCP to secondary users (SUs) in overlay cognitive radio networks (CRNs), and address its performance degradation; in CRNs, SU's transmissions are frequently disrupted by the detection of primary user's transmission, and which makes the SU experience consecutive retransmission-timeout and its exponential backoff. Subsequently, the TCP in SU does not proceed with the transmission even after the disruption is over or the SU hands over to other idle spectrum. To tackle this problem, we propose a cross-layer approach called TCP-Freeze-CR; lower layer protocols send the overlying TCP two different cross-layer signals, *freeze* on the detection of primary user's transmission, and *unfreeze* after handing over to an idle spectrum. Moreover we consider a practical situation where either secondary transmitter (ST) or secondary receiver (SR) detects primary user's transmission; therefore additional message exchanges are needed between ST and SR to retrieve and resynchronize to other idle spectrum, i.e., spectrum synchronization. This situation is more complex than the case where both ST and SR detect primary user's transmission. Hereby, we develop a spectrum synchronization procedure coupled with TCP-Freeze-CR into a finite state machine. All of our proposals are implemented and evaluated on a real CRN consisting of 6 software radio platforms. In the implementation, we deploy 802.15.4 implementation as a target physical layer protocol, and couple it with TCP-Freeze-CR using Unix Domain Socket. The experimental results illustrate that standard TCP suffers from significant performance degradation in CRNs, and show that TCP-Freeze-CR can greatly alleviate the degradation; e.g., for 1200 s, ST with TCP-Freeze-CR can send about 10 times more packets than ST with standard TCP.

1. Introduction

Cognitive radio (CR) technology plays an essential role for solving the problem of spectrum scarcity as demand for emerging wireless applications is increasing: smart power grid, medical appliances, surveillance system, etc. To this end, a plentiful number of research literature have been published since Mitola and Maguire proposed the concept of CR [1]. However, majority of the research work related to CR have focused on the issues in physical or link layer, i.e., spectrum sensing, dynamic spectrum allocation, interference control, etc.; a few of them have addressed issues in routing over CR ad-hoc networks.

In this paper, we focus on significant performance degradation that transmission control protocol (TCP) suffers from in CR networks (CRNs), and study an enhancement scheme that can alleviate the performance degradation.

In Internet, TCP provides applications with reliable and ordered packet delivery over unreliable physical media. Therefore, TCP is the most widely used transport layer protocol even in wireless networks, and there is no signs of change in the foreseeable future.

In TCP-NewReno [2],[1] packet losses are regarded as the signal of network congestion, and two different signals are used for noticing the packet loss: triple duplicate acknowledgments (ACKs) and retransmission-timeout (RTO). On receiving triple duplicate ACKs, TCP performs fast retransmission and triggers congestion avoidance mechanism with halving its congestion window size (henceforth, we denote congestion window size *cwnd*); on the occurrence of an RTO, it retransmits unacknowledged packet with squeezing *cwnd* to 1 and triggers the slow start deeming the network overloaded.

An extensive research work has been conducted to resolve the problem that standard TCP experiences over conventional wireless networks: multiple random packet losses within one round-trip-time (RTT) due to interference, shadowing, fading, and collision in

* Tel.: +82 51 510 0651.
E-mail address: ssbyun@cup.ac.kr

[1] Throughout this paper, TCP-NewReno is referred as standard TCP since it has been the most widely used TCP standard [3].

http://dx.doi.org/10.1016/j.comnet.2016.03.026
1389-1286/

wireless channels, lead to consecutive RTOs and exponential back-off of retransmission-timer (RT), and standard TCP therefore suffers from drastic decrease of its throughput [4].

In CRNs, a secondary user (SU) whose applications communicate using standard TCP experiences performance decrease because of not only but the random packet loss and interference also disruption by primary user (PU) 's transmission (hereafter, referred as primary transmission briefly) especially if both of the SU and PU are accessing the same channel [5]. Generally, the random loss occurs transiently, but on the other hand, the disruption by primary transmission may last for a relatively long while. Therefore, unless SU hands over to other unused spectrum, the disruption can incur more consecutive RTOs and exponential back-off of RT; assuming such disruption triggers an RTO at an SU, and continues even until the next RT expires, the RT is backed off exponentially twice, and thus the SU does not proceed with the transmission even after the disruption is over or she succeeds to hand over to other unused spectrum. This performance degradation becomes more significant as primary transmission occurs more frequently and for a longer while.

Most of the TCPs for conventional wireless networks also tackle the same issue: discerning congestive error and channel error. However the channel error in conventional wireless networks is generally transient, and most of the TCP for conventional wireless networks control sending rate or window size with reflecting the channel error to bandwidth estimation. However SU should shut down the transmission completely as soon as detecting primary transmission (henceforth, we note primary user detection). This feature is the difference with TCPs for conventional wireless networks.

Considering an overlay-CRN[2]—whose definition is given in later section of this paper—with multiple channels, we envisage a cross-layer approach that can alleviate the aforementioned problem with modifying standard TCP very slightly. The modified TCP is referred to as *TCP-Freeze-CR*. TCP-Freeze-CR is implemented in secondary transmitter (ST) only; secondary receiver (SR) can use the standard TCP without any modifications.

As inferred in the name of the scheme, TCP-Freeze-CR is motivated by Freeze-TCP [6]. In Freeze-TCP, wireless receiver predicts impending channel degradation and mobility, and freezes its TCP via cross-layer signaling. On the prediction, wireless receiver sends zero window advertisement in order to freeze the transmission. On the detection of good channel quality, the wireless receiver sends triple ACKs then its sender resumes the transmission. As mentioned beforehand, TCP in overlay-CRNs should shut down transmission on primary user detection quickly. Therefor Freeze-TCP is the best candidate that can be applied to overlay-CRNs easily.

Besides, we consider a realistic situation where spectrum handover is triggered by either ST or SR. Therefore, any SU detecting primary transmission should notify the other SU of the detection since both ST and SR should be made perform spectrum sensing to retrieve an idle spectrum commonly accessible by themselves. To this end, we develop a spectrum synchronization mechanism using a finite state machine, and couple it with TCP-Freeze-CR.

We evaluate our scheme by implementing it onto a representative software-defined radio (SDR) platform, Universal Software Radio Peripheral (USRP E100), where many part of the lower layer operations are executed as user processes written in high level programming languages, i.e., C++ and Python, above the library called USRP Hardware Driver (UHD) [7]. Furthermore, we deploy IEEE 802.15.4 implementation [8] as the target lower layer protocols, and use orthogonal frequency-division multiplexing (OFDM) implementation for PUs.

The remainder of the paper proceeds as follows: In Section 2, we present our spectrum synchronization mechanism. In Section 3, we present our main contribution, TCP-Freeze-CR. In Section 4, we give the implementation details. In Section 5, we present the experimental results. In Section 6, we give some related work. In Section 7 we discuss an extension of our scheme to multi-hop CR networks, and conclude this paper in Section 8.

2. Spectrum synchronization

2.1. Access technique for CRNs

Access techniques for CRNs can be classified into two types [5]. In overlay-CRNs (or interference-free CRNs), SUs can access spectrums not occupied by any PUs. As a result, there should be virtually no interference to the PUs. On the other hand, in underlay-CRNs (or interference-tolerant CRNs), SUs are allowed to interfere with primary transmission up to a certain tolerable level. In this paper, we consider the overlay-CRNs only.

2.2. Spectrum sensing model

Generally, spectrum sensing is defined as the task of finding spectrum unused or underutilized by PUs [9]. Then spectrum sensing approaches can be classified into two types according to the time when the sensing task is performed: *proactive sensing* and *on-demand sensing*. When proactive sensing is applied, SUs perform the sensing task periodically and continuously even while they are communicating over an idle channel safely; on detecting primary transmission, ST and SR hand over and resynchronize to the spectrum detected as idle by both of them. On the contrary, SUs perform the sensing task only when they overhear primary transmission in on-demand sensing.

Assuming that ST finds the spectrum to hand over and resynchronize, ST should receive the list of the spectrums detected as idle by her target receiver (i.e., SR) as well. Then the ST should inform the SR of the spectrum they will hand over and resynchronize to. Therefore additional message exchanges between ST and SR are inevitable. When proactive sensing is used, the message exchanges occur periodically and continuously, and thus, proactive sensing can prevent PUs from overhearing these message exchanges. However, on-demand sensing cannot eliminate such overhearing (unless there is an extra dedicated channel for the message exchange).

In this work, we consider the spectrum synchronization coupled with on-demand sensing. Hence we need to assume an extra control channel for the message exchanges between ST and SR. By applying proactive sensing, we can relax this assumption. However we have found that proactive sensing is not adequate on our software radio platform (i.e., USRP E100) since periodic and continuous spectrum sensing overuse the processing power. Thus we decide to use on-demand sensing unavoidably, and, as a result, more processing resources are given to processing of TCP-Freeze-CR and 802.15.4 implementation. Incidentally many related work such as [10] and [11] have premised ideal proactive sensing without any implementation-based experiments.

2.3. Spectrum synchronization model

Majority of related work have assumed that ST or a central authority can detect all primary transmissions or idle spectrums that are actually detectable by SR only [10]. In this paper, we relax this assumption; that is, we consider the situation where ST and

[2] In this paper we consider only single-hop networks. Nonetheless, we believe that our approach can be extended for multi-hop or infra-structured networks, and which remains as one of our future task.

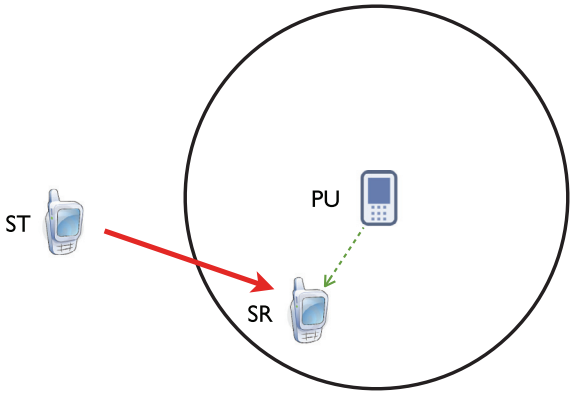


Fig. 1. Independent primary user detection. Only SR is interfered by PU.

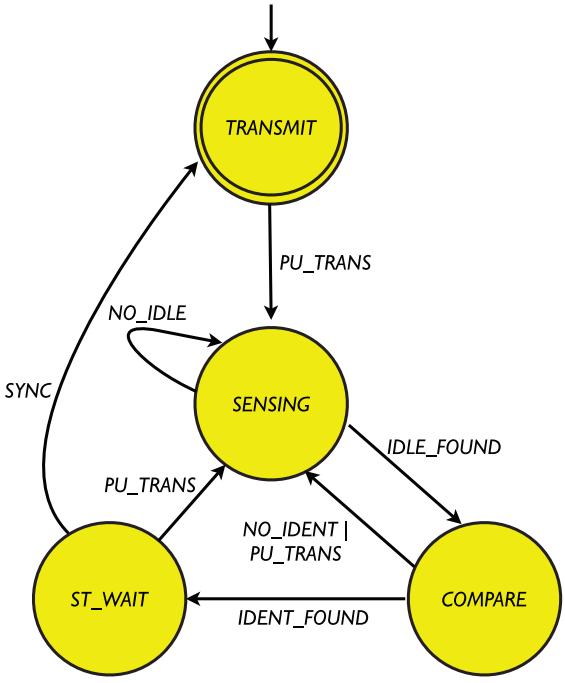


Fig. 2. Finite state machine model of the spectrum synchronization.

SR detect primary transmission independently; either ST or SR may be interfered by PU (refer to Fig. 1). As a result, a spectrum detected as idle (or busy) by ST can be detected as busy (or idle) by SR in this situation.

Considering the aforementioned issue, the spectrum synchronization is modeled using the following finite state machine (FSM).

$$M = (\Sigma, S, s_0, \delta, F)$$

where Σ is a (finite non-empty) set of input strings, S is a (finite non-empty) set of states, $s_0 (\subset S)$ is a set of initial states, δ is a state-transition function: δ: $S \times \Sigma \rightarrow S$, and $F (\subset S)$ is a set of final states. Subsequently, each tuple is given as

$$\Sigma = \{PU\_TRANS, NO\_IDLE, IDLE\_FOUND, NO\_IDENT, IDENT\_FOUND, SYNC\};$$
$$S = \{TRANSMIT, SENSING, COMPARE, ST\_WAIT\};$$
$$s_0 = F = \{TRANSMIT\}.$$

The definition of δ is illustrated on Fig. 2. Then each input string in Σ indicates:

- *PU_TRANS*: Either ST or SR detects primary transmission;
- *NO_IDLE*: ST or SR or both of them cannot detect any idle spectrum;
- *IDLE_FOUND*: ST and SR find at least one idle spectrum;
- *NO_IDENT*: There is no spectrum that both ST and SR detect as idle identically;
- *IDENT_FOUND*: There is at least one spectrum that both ST and SR detect as idle identically;
- *SYNC*: ST and SR succeed to hand over and synchronize to the idle spectrum.

Then the definition of each state in S is given as:

- *TRANSMIT*: ST is transmitting user data to SR;
- *SENSING*: ST and SR are performing spectrum sensing;
- *COMPARE*: ST is checking whether there is any spectrum that both ST and SR detect as idle identically;
- *ST_WAIT*: ST is waiting for SR's acknowledging to the request of spectrum handover.

Accordingly, the state transitions are presented as follows:

- In *TRANSMIT* state, the state transforms to *SENSING* state whenever either ST or SR detects primary transmission. That is

  $$\delta(TRANSMIT, PU\_TRANS) = SENSING.$$

- In *SENSING* state, both ST and SR perform the spectrum sensing, and the state transforms to *COMPARE* state only if ST and SR find at least one idle spectrum, i.e.,

  $$\delta(SENSING, IDLE\_FOUND) = COMPARE;$$

  otherwise, they perform the spectrum sensing again, i.e.,

  $$\delta(SENSING, NO\_IDLE) = SENSING.$$

  In our implementation, we let SR inform ST of the result of her spectrum sensing.
- In *COMPARE* state, ST checks whether there is any spectrum detected as idle by SR as well. If ST finds at least one such spectrum, the state transforms to *ST_WAIT* state, i.e.,

  $$\delta(COMPARE, IDENT\_FOUND) = ST\_WAIT;$$

  otherwise, they perform the spectrum sensing again, that is,

  $$\delta\ \ (COMPARE, NO\_IDENT) = SENSING.$$

- In *ST_WAIT* state, ST requests SR to hand over to the newly found idle channel, and awaits the acknowledgement from SR. If SR acknowledges, the state transforms to *TRANSMIT* state, i.e.,

  $$\delta(ST\_WAIT, SYNC) = TRANSMT;$$

  otherwise (that is, SR denies to hand over to the new idle spectrum due to the primary user detection), the state transforms to *SENSING* state again, i.e.,

  $$\delta(ST\_WAIT, PU\_TRANS) = SENSING.$$

- Besides, the state transforms to *SENSING* state from any states where primary transmission is detected.

3. TCP-Freeze-CR

3.1. Vs. TCP in conventional wireless networks

The main difference between TCP in CRNs and TCP in conventional wireless networks is well discussed in [5] and [12].

The major loss that occurs in conventional wireless networks is due to channel errors (caused by fading, interference, shadowing, and user mobility). The channel errors degrade the channel quality such as bandwidth. Therefore TCP in conventional wireless networks need to adapt its sending rate or window size according to the channel quality (e.g., TCP-Westwood [13]). Furthermore, the channel errors are generally transient.

In CRNs (especially in overlay-CRNs), there is an additional loss - called *service interruption loss* - due to primary transmission. The service interruption loss does not affect channel quality, but channel accessibility. SU should stop her transmission on detecting primary transmission immediately. Then she tries to find other idle channel. If no idle channel is found (because of heavy primary transmission in a given region), SU experiences service interruption loss for a longer duration. Otherwise, SU can continue her transmission after handing over to a newly found idle channel. To sum up, the strategy SU can have on detecting primary transmission is either to stop his transmission or to continue on the new idle channel. There is no need of sending rate adjustment.

TCP in conventional wireless networks may be used in underlay-CRNs, but the factor of transmission power control (in order to control interference to primary transmission within a certain tolerance) should be reflected to the sending rate adjustment. Therefore TCP in conventional wireless networks cannot work properly in CRNs without any modifications.

In this paper, we study TCP in overlay-CRNs where SU should stop her transmission and try to find other idle channel as soon as she detects primary transmission. Therefore Freeze-TCP is the proper choice in overlay-CRNs since it has already necessary mechanisms - stopping and resuming transmission by cross-layer signals according to channel quality.

3.2. Vs. Freeze-TCP

If we consider only SRs in CRNs, Freeze-TCP can be used with minor modifications. However, as mentioned in [14], Freeze-TCP suffers from unstable performance due to inaccurate prediction of impending channel degradation: too frequent freezes make the transmission retarded because of inflated RTO. Furthermore, the disruption due to primary transmission is not transient usually. And also there is no way of distinguishing disruption by primary transmission and the one by the depletion of receiver buffer: spectrum synchronization does not need to be triggered on the buffer depletion. In TCP-Freeze-CR, we resolve this problem by squeezing the RTO whenever TCP freezes (even by false alarming) and explicit separate cross-layer signaling.

The other main differences of our cross-layer scheme (TCP-Freeze-CR) to Freeze-TCP are:

(1) Not only the receiver but also the sender can freeze TCP in TCP-Freeze-CR. On the other hand, only the receiver can freeze the transmission in Freeze-TCP.
(2) TCP-Freeze-CR freezes TCP only when primary transmission is detected. However, Freeze-TCP does not have any mechanisms that distinguishes the buffer depletion and the detection of the primary transmission. Therefore the buffer depletion can yield unnecessary spectrum synchronization. Furthermore, if the zero-window advertisement is lost, the sender continues its transmission, which will interfere primary transmission until the sender receives the next zero-window advertisement successfully.
(3) In Freeze-TCP, the sender resumes transmission by receiving triple acknowledgements from receiver. Therefore, the transmission may not start immediately even if only one of the triple acknowledgements is lost. TCP-Freeze-CR stops and resumes by reliable RPC (remote procedure call).

3.3. Operations with the spectrum synchronization

As mentioned in Section 1, TCP-Freeze-CR is aimed at being implemented in ST side only, and we can use standard TCP in SR without any modification. Therefore SR should notify ST of primary transmission she detects. In our implementation, we let SR's lower layer protocol inform ST's lower layer protocol of the primary user detection. Then ST's lower layer in turn issues the cross-layer signal, *freeze*, to its overlying TCP-Freeze-CR for halting the transmission. On finding a new idle spectrum, ST's lower layer issues the other cross-layer signal, *unfreeze*, to her TCP-Freeze-CR in order to resume the transmission.

The operations of TCP-Freeze-CR are given as follows:

(1) If an SU detects primary transmission, her TCP state is frozen until she finds and hands over to any other idle spectrum. Usually, the start and finish of primary transmission are detected through spectrum sensing procedure - which is the process of detecting primary transmission and finding other idle spectrum - in lower layer (MAC or physical layer) protocols.
(2) On detecting primary transmission, lower layer protocol sends the overlying TCP a cross-layer signal (i.e., *freeze*).
(3) On receiving the cross-layer signal, the TCP stores its current *cwnd* and RTT, and cancels any impending RT, and it stops transmitting user data by setting $cwnd = 0$.
(4) As soon as the lower layer protocol finds and hands over to a newly found idle spectrum, it gives the TCP the cross-layer signal again (i.e., *unfreeze*), and then, the TCP resumes transmitting the user data by restoring its *cwnd* and RTT and resetting the RT.
(5) By resetting the RT, any impending retransmission that is led to by either packet loss or acknowledgement loss can proceed after a short while. Furthermore, if restored $cwnd = 0$, the transmission cannot restart immediately. Therefore, *cwnd* is set 1 if restored $cwnd = 0$. These additional mechanisms help TCP to perform the transmission as soon as possible even after a false alarm.
(6) All the transmissions impending in lower layer are cancelled immediately as soon as primary transmission is detected in order to prevent inaccurate estimation of current RTT.

Coupling TCP-Freeze-CR with the FSM of spectrum synchronization mentioned in Section 2.3, the following two transition functions incur the cross-layer signals:

$$\delta(TRANSMIT, PU\_TRANS) = SENSING$$

and

$$\delta(ST\_WAIT, SYNC) = TRANSMIT.$$

The former issues the freeze signal to TCP, and the latter issues the unfreeze signal. On receiving the freeze signal, TCP-Freeze-CR stores current *cwnd* and RTT and cancel impending RT; on receiving the unfreeze signal, it restores *cwnd* (only if $cwnd > 0$) and RTT, and resets RT with minimum RTO value. If $cwnd = 0$, it sets $cwnd = 1$.

Algorithmically, the main behavior of TCP-Freeze-CR can be written as:

```
procedure OnCrossLayerSignal(signal = {freeze, unfreeze})
    if signal == freeze then
        Cancel impending RT;
        old_cwnd = cwnd;
        old_rtt = rtt;
        cwnd = 0;
    else if signal == unfreeze then
        rtt = old_rtt;
        RTO_value = min_RTO_value;
        if old_cwnd > 0 then
            cwnd = old_cwnd;
        else                                   ▷ if old_cwnd = 0
            phase = SLOW_START;
            cwnd = 1;
        end if
        Restart RT;
    end if
end procedure
```

The detailed step-by-step operations of TCP-Freeze-CR coupled with the spectrum synchronization are depicted in Fig. 3; Fig. 3(a) and (b) depict the steps in case primary transmission is detected by ST only, and in case detected by SR only, respectively. It is assumed that ST and SR share predefined channel list that enumerates the center frequency of each channel accessible by both ST and SR. Therefore ST and SR can specify the center frequency of each channel by only notifying the corresponding channel index.

When ST detects primary transmission first, she first freezes her TCP, notifies SR of the detection, and performs spectrum sensing (step (1)–(3) in Fig. 3(a)). In reply to the notification, SR sends the bitmap of her channel list that identifies the idleness of each channel as soon as she completes spectrum sensing (step (3)–(4) in Fig. 3(a)). In case SR detects primary transmission first, she triggers her spectrum sensing task, and then sends ST the bitmap of her channel list (step (1)–(2) in Fig. 3(b)). On receiving the bitmap from SR, ST identifies the channel detected as identically idle by SR and herself (step (5) in Fig. 3(a) and step (4) in Fig. 3(b)). If SR detects primary transmission first, ST should freeze her TCP and perform spectrum sensing (step (3) in Fig. 3(b)) prior to identifying the idle channel accessible by SR as well as herself. Successfully identifying a new idle channel, ST requests SR to hand over to the new channel (step (6) in Fig. 3(a) and step (5) in Fig. 3(b)), and herself hands over to the channel as well. As soon as SR completes the handover, she acknowledges to ST, and then ST unfreezes her TCP (step (6) ∼ (7) in Fig. 3(a) and step (7) ∼ (8) in Fig. 3(b)).

3.4. Additional enhancement schemes

In this subsection, we present the two additional enhancement schemes used in TCP-Freeze-CR.

On receiving an unfreeze signal,

(1) It restores *cwnd* only if $cwnd > 0$, and $cwnd = 1$ otherwise;
(2) It restarts RT with the minimum RTO value.

TCP in ST may get the freeze signal when its current $cwnd = 0$; this situation occurs if the ST was waiting for the arrival of ACKs from her receiver. In this situation, TCP cannot resume the transmission even after it gets the unfreeze signal (refer to Fig. 4(a)). In order to prevent this situation, we let TCP set its $cwnd = 1$ if the old *cwnd* is 0. Furthermore, the phase is set slow-start in order to achieve the proper sending rate more quickly.

Basically, CRNs consider wireless environments. Hence either the original packets, which are transmitted right after TCP becomes unfrozen, or the ACKs, which are issued in reply to the reception of the original packets, can be lost due to conventional channel error. In this case, TCP can be frozen again without retransmitting the lost packets if its current RTO value is set too high. This situation is depicted in Fig. 4(a). By the experiments, we verify that this problem becomes more dominant under more frequent disruption. To mitigate this problem, we let RT restart with minimum RTO value (40 ms in our implementation). However, this scheme results in another problems: more frequent RTO and shrinkage of *cwnd*, i.e., $cwnd = 1$. We believe these problems can be tackled by coupling with the schemes of differentiating packet loss in wireless networks such as Snoop agent [15] and link layer recovery [16], and which will be one of our future work.

Besides, all the transmissions impending in lower layer are cancelled immediately as soon as primary transmission is detected; all the packets queued in lower layer buffer (ACKs in SR and user data in ST) are discarded immediately. This action prevents inaccurate estimation of current RTT.

4. Implementation details

Usually, the implementation of CR is conducted in SDR since SDR is better suited to deal with various protocols and cross-layering [17]. CR is often regarded as the goal SDR ultimately need to achieve: a dynamically reconfigurable and fully flexible radio system that can adapt itself according to user demands, frequency scarcity, available communication protocols, etc.. It is very difficult to implement the reconfigurability, especially, spectrum sensing, dynamic spectrum access, channel synchronization and protocol (or vertical) handoff, in current DSP-based legacy radio architecture. Furthermore the reconfiguration process should be done with gathering relevant information regularly (e.g., spectrum sensing results and communication quality), and the information and decisions made based on the information should be shared by nodes. Therefore more flexible general purpose processor-based SDR platforms are considered for CR.

We implement our schemes on a representative SDR device, USRP E100, and verify the performance gained with TCP-Freeze-CR.

In this following subsections, we present the implementation details.

4.1. The platform

USRP E100 is a standalone version of USRPs, and its main hardware consists of TI OMAP Beagle Board and ARM Cortex A8 core. It uses Embedded Linux operating system, and can hold one RF daughter board that plays the role of RF front-end. Besides, it holds one FPGA chip where DAC/ADC and interpolator/decimator are programmed.

GNU Radio [18] and UHD [7] are the most commonly used radio softwares in the USRPs. GNU Radio is an open source project, and supports hardware-independent signal processing functionalities. In GNU Radio, the signal processing blocks are written in C++ while the signal flow interfaces are built using Python. UHD—developed by Ettus Research LLC, the manufacturer of the USRP—can work with or replace GNU Radio, and its signal processing blocks are exclusively optimized for the USRP devices.

In USRP E100, most of the signal processing operations are executed on the general purpose processor (GPP). Therefore, it has much less performance than legacy DSP-based radio system. However, the devices fit to the purpose of evaluating prototypical protocols and provide great flexibility such being able to implement message exchanges between SUs with remote procedure call (RPC).

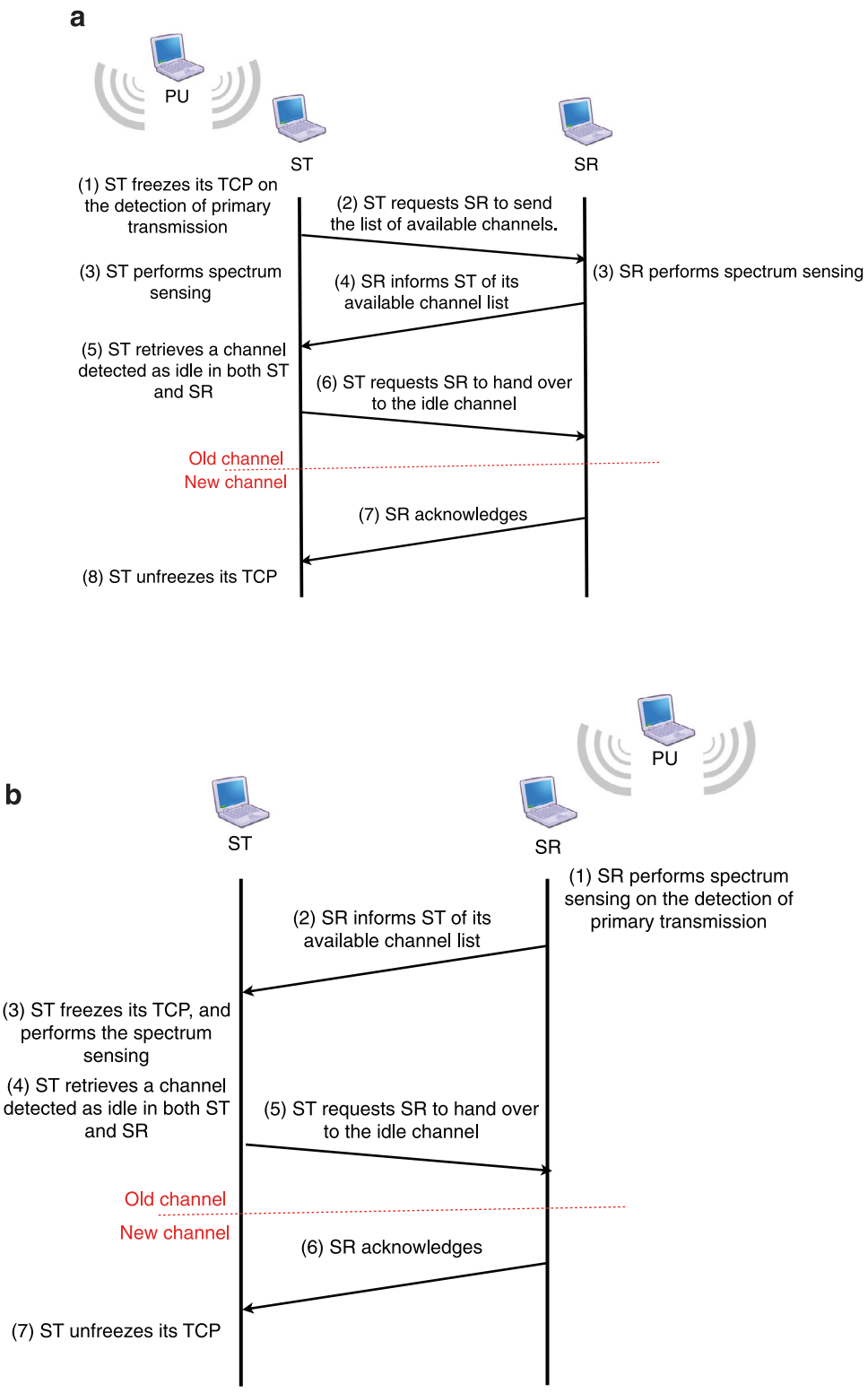


Fig. 3. Illustrations of TCP-Freeze-CR operations coupled with the spectrum synchronization task. (a) In case ST detects primary transmission first. (b) In case SR detects primary transmission first.

4.2. Implementation of spectrum synchronization

The spectrum synchronization scheme is implemented as Python functions and added to the interface block of 802.15.4 implementation.

The main reason of choosing IEEE 802.15.4 as the target lower layer is that, at first, it works perfectly on our software radio platform (USRP E100), and there are many issues related to the coexistence of IEEE 802.15.4 with other major protocols operating on the unlicensed 2.4 GHz ISM band, notably IEEE 802.11 (WLAN) and IEEE 802.15.1 (Bluetooth) [19–21]. Furthermore, it is not unusual to address the reliable data transmission of IP-enabled small sensor devices [22–25].

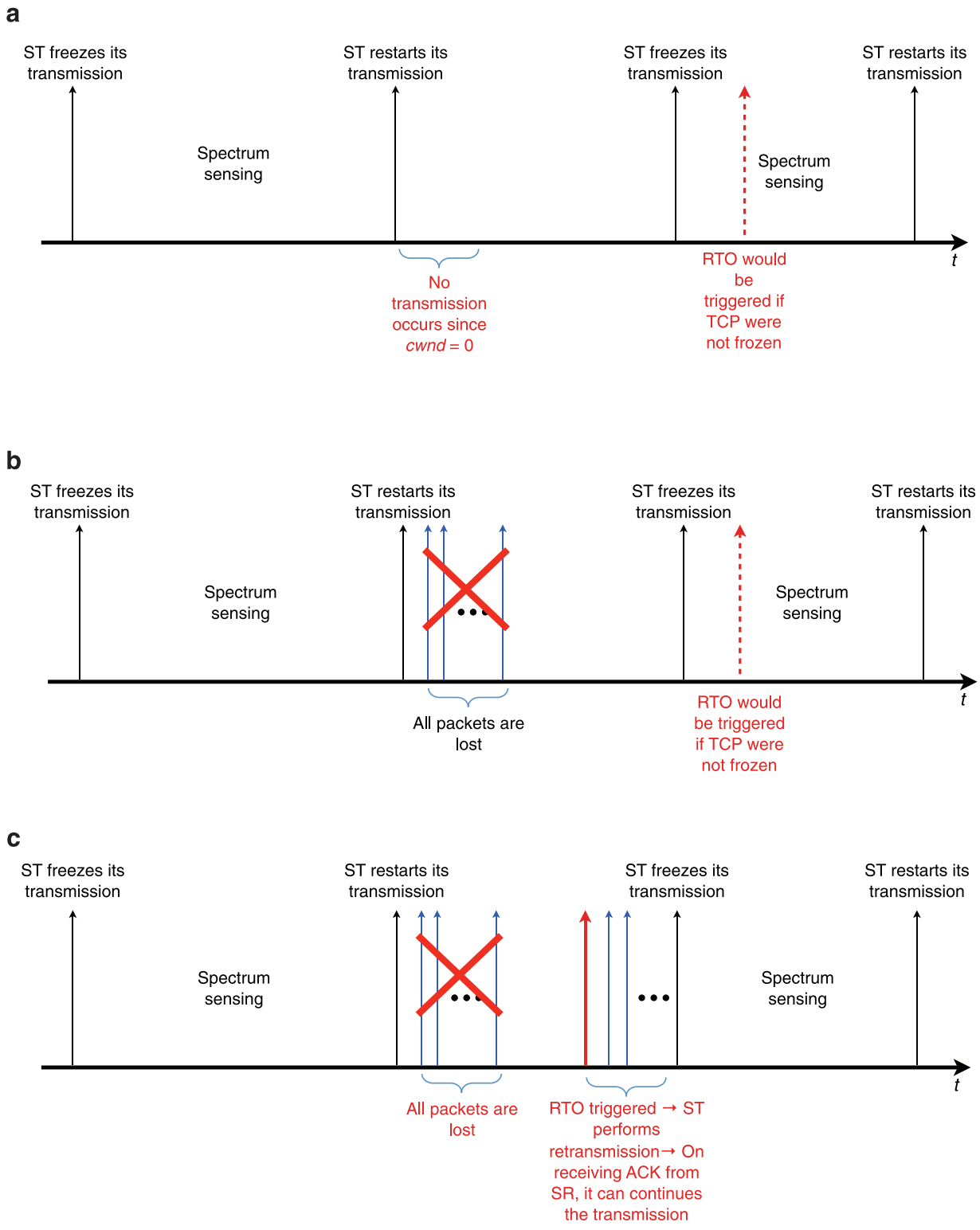


Fig. 4. Illustrations of the situations that TCP may encounter after it is unfrozen. (a) If restored *cwnd* = 0, the transmission can be deterred. (b) If either the packets, which are transmitted right after TCP becomes unfrozen, or the ACKs, which are issued in reply to the transmitted packets, are lost, the retransmission may be delayed due to a large RTO value. (c) The retransmission can be performed within the current transmission round if RTO value is set to the minimum RTO value.

We add the functionalities of spectrum sensing and hand-over to the retrieved idle spectrum in the IEEE 802.15.4 implementation. Then we couple TCP-Freeze-CR and the spectrum synchronization mechanism using an interprocess communication (IPC) mechanism. The functionalities of spectrum sensing and hand-over are implemented in SR as well as ST since we consider the situation where the primary user detection can takes place in either ST or SR.

4.2.1. Primary user detection

Actually, it is highly recommended to let spectrum sensing mechanism fully independent from any waveform standards.

However, in this paper, we focus on the performance of transport layer, i.e., TCP, and therefore need to minimize the influence of the spectrum sensing onto the performance of TCP. Therefore we implement and use simple energy detection mechanism that works efficiently in small CR topology. Other spectrum sensing mechanisms are found in [26]. Besides, we utilize OFDM implementation (included in the bundle of UHD examples) for PUs' transmissions.

As mentioned in Section 2.3, we consider the scenario where ST and SR detect primary transmission independently. To this end, we locate PUs quite closely to SUs and let them transmit with relatively low power (0.5 dBm in our experiments) in order to make a PU paired with one SU interfere with the other SU minimally. However, a PU dedicated to interfere ST (or SR) may still interfere SR (or ST). Hence we let each SU run OFDM receiver implementation to decode which PU is on transmission (more precisely, to decode transmitting PU's hardware address). In this manner, each SU can distinguish whether the transmitting PU is dedicated to interfere with her or not. More detailed explanations on this setting are given in the next section.

4.2.2. Synchronization through RPC

As shown in Fig. 3, there are a few message exchanges between ST and SR during the procedure of the spectrum synchronization; more precisely, either of ST and SR need to call functions implemented in the other. Therefore the message exchanges can be implemented easily using a sequence of remote procedure calls (RPCs): for instance, ST calls the function of spectrum sensing located in SR (step (2) in Fig. 3(a)), and SR returns the function with the bitmap as a return value that is to be delivered to ST (step (4) in Fig. 3(a)). Furthermore, all the message exchanges should be performed robustly and reliably. Hereby we implement the message exchange procedure using a Python RPC extension, Pyro [27].

Besides, we do not incorporate IP layer, and let the TCP-Freeze-CR process itself generate user data in order to remove side factors that can influence on the performance of TCP-Freeze-CR. Therefore, we let Pyro RPC communicate through TCP/IP stack in Linux kernel via network tunnel (TUN) interface.

4.3. Implementation of TCP-Freeze-CR

TCP-Freeze-CR is implemented based on TCP-NewReno. Therefore, we implement all the mechanisms related to the congestion control algorithm of TCP-NewReno: fast retransmission, fast recovery, retransmission by RTO, Jacobson's algorithm [28], and cumulative ACKs.

Like other software radio modules implemented in the USRPs, the 802.15.4 implementation runs in user space as a single Python process. TCP-Freeze-CR is written in C programming language and runs in user space as a single processor as well. Then they communicate through Unix Domain Socket for the delivery of user data, ACKs, and the cross-layer signals.

5. Experimental results

5.1. Experimental settings

We locate six USRPs totally as shown in Fig. 5: one for ST, one for SR, and the other four for PUs. We let the USRPs for the ST and SR be able to hand over between two different channels: if primary transmission is detected on the channel they are using, they try hand-over to the other channel. Each SU is interfered by two PUs; each channel is interfered by each PU. We let every PU generate OFDM signal for an exponentially distributed time with mean T_{on} seconds, and turn off the signal generation for an exponentially distributed time with mean T_{off} seconds. The ST and SR deem a channel is idle if no primary transmission is overheard for at least 500 ms. USRP E100 supports simultaneous listening on two different channels. We let the ST and SR listen on both the channels when no idle channel is detected, and also during the spectrum synchronization process as presented in Section 3.3.

Actually, it is possible that the ST overhears the transmission of the PU that is dedicated to interfere with the SR, and also the SR overhears the primary transmission that is set to interfere with the ST. Therefore, if ST or SR overhear from a PU that is not dedicated to interfere with her, simply she ignores what she overhears; every SU can distinguish whether the overheard transmission is interference or not by simply decoding the hardware address of the overheard packets. This setting is essential for making the ST and SR detect primary transmission independently.

We let all the USRPs for ST and SR transmit with 1 dBm of transmission power over 2 Mbps of physical bandwidth, and receive with 200 Kbps of sampling rate. We let all the USRPs playing the role of PU transmit with 0.5 dBm of transmission power. The central frequencies used for the two channels are 2.085 GHz and 2.105 GHz.[3]

In the 802.15.4 implementation, the size of MAC protocol data unit (MPDU) is given as 128 bytes including 19 bytes of the header and 2 bytes of the tail, and therefore, 107 bytes are assigned to the payload. We assign 12 bytes for the TCP header, and 4 bytes and 8 bytes for the sequence number and RTT, respectively. Accordingly, 95 bytes of user data can be transmitted without being fragmented.

We carry out two sets of experiments. The first set compares the performance of TCP-Freeze-CR with standard TCP, and the second set illustrates the performance improvement yielded by the enhancement scheme addressed in Section 3.4. All the experiments are performed for 1200 s.

5.2. Standard TCP vs. TCP-Freeze-CR

5.2.1. Goodput comparison

In order to measure the performance gain of TCP-Freeze-CR under various PU activities, we perform the experiments applying three different patterns of primary transmission: $(T_{on}, T_{off}) =$ (2 s, 8 s), (5 s, 5 s), and (8 s, 2 s), representing light PU activity, medium PU activity, and heavy PU activity, respectively. Then we first measure the goodput (bits/s) of each TCP implementation and plot it on Fig. 6. We measure the average goodput over every 8 s. As shown in Fig. 6(a), even standard TCP yields somewhat fair goodput under less intensive primary transmission; yet TCP-Freeze-CR yields better goodput. However we observe that the goodput of standard TCP drastically decreases as primary transmission becomes more heavy; standard TCP has transmitted only 18 packets successfully for 1200 s when $(T_{on}, T_{off}) = (8, 2)$ (shown in Fig. 7(c)). As described earlier, this is due to the fact that standard TCP continues to transmit or retransmit user data even though the lower layer transmission is blocked. As a result, RTOs are triggered consecutively, and the retransmission is postponed until the next RTO even when the lower layer transmission is available. Surely, TCP-Freeze-CR yields also lower goodput as primary transmission becomes more aggressive, but the decrement is much less than standard TCP (refer to Fig. 6(b) and (c)).

Incidentally, the goodputs measured in this set of experiments are relatively poor: at most up to 390 bits/s. This is resulted from the fact that, in USRP, most of physical layer operations are executed on GPP that is not optimized for signal processing.

[3] Actually, it is recommended to use the frequency range of 2.4 GHz for multi-channel communications with IEEE 802.15.4 standard. However, the daughter board, WBX daughter board, installed in our USRPs can cover up to only 2.2 GHz.

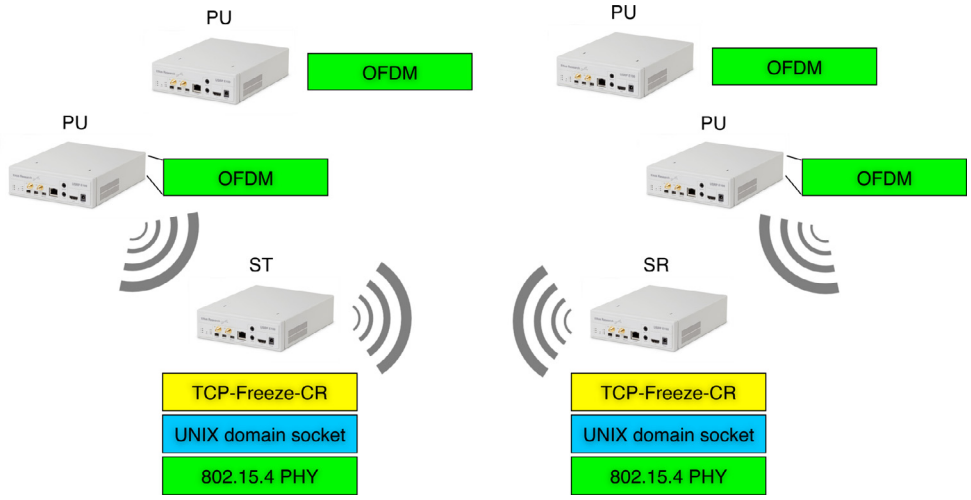


Fig. 5. Experimental topology.

5.2.2. Comparison of trace of packet sequence

In order to illustrate the contribution of TCP-Freeze-CR more clearly, we trace the sequence of the packets the SR receives, and plot the measured results on the graph in Fig. 7. As shown in the graphs, TCP-Freeze-CR yields far faster increase of the packet sequence than standard TCP does. Especially, at the end of the experiment, it is measured that TCP-Freeze-CR makes the SR receive about 10 times more packets than standard TCP when $(T_{on}, T_{off}) = (5, 5)$; it is easily anticipated that further tracing would yield larger difference.

Consequently, we conclude that the performance of standard TCP degrades more significantly as primary transmissions become heavier.

In Fig. 8, we plot the trace of the packet sequence along with the channel availability in order to observe the behavior of each TCP implementation on the occurrence of primary transmission or the detection of unused channel. In the graphs, the legend 'Primary Transmission' corresponds to the right *y*-axis, and it describes the occurrence of the primary user detection (expressed as value 2 in the graphs) or the detection of a unused channel (expressed as value 1).

Fig. 8(a) and (b) plot the measured results with standard TCP and TCP-Freeze-CR, respectively, when we apply $(T_{on}, T_{off}) = (2, 8)$. We plot the results over a portion of the interval for better readability. As shown in these graphs, it is frequently observed that the SR with standard TCP does not receive packets even after it completes the spectrum synchronization whereas the SR with TCP-Freeze-CR immediately resumes receiving packets usually right after the spectrum synchronization.

Fig. 8(c) and (d) plots the measured results with standard TCP and TCP-Freeze-CR, respectively, when $(T_{on}, T_{off}) = (5, 5)$ is applied. As shown in the graphs, the transmission opportunities become scarcer, and therefore both of the TCP implementations receive the packets in relatively lower speed. However, we observe that TCP-Freeze-CR can grab the transmission chance whenever it sees while the SR with standard TCP cannot.

Then we make the transmission opportunities more scarcer (i.e., we apply $(T_{on}, T_{off}) = (8, 2)$), and the measured results are plotted on the graphs in Fig. 8(e) (for standard TCP) and Fig. 8(f) (for TCP-Freeze-CR). As expected, the SR has much less transmission opportunities regardless of the TCP implementation she uses. However, we observe that TCP-Freeze-CR can seize the transmission chance more frequently (but not always) than standard-TCP.

We also observe that the SR with TCP-Freeze-CR also fails to receive packets due to the burst packet loss over wireless channel and the exponential backoff of RT (for instance, we observe this situation at around 870 s on the graph in Fig. 8(b)).

5.3. Effects of enhancement schemes of TCP-Freeze-CR and evaluation of Freeze-TCP

The next set of experiments is performed in order to present the performance improvement yielded by the enhancement scheme addressed in Section 3.4; the enhancement schemes are (a) letting $cwnd = old\_cwnd$ if $old\_cwnd > 0$, $cwnd = 1$ otherwise, and (b) setting RT value = minimum RT value. The measured results are plotted on the graph in Fig. 9. For the purpose of comparison, we also trace the packet sequences without the enhancement schemes (i.e., TCP resumes its transmission with simply restoring the old *cwnd, RTT*, and RTO regardless the values they have when it receives the unfreeze signal), and plot them with the legend named 'New Reno + Cross-layer' on the graph. For these experiments, we apply $(T_{on}, T_{off}) = (5, 5)$, and let only ST be affected by primary transmission.

Also we evaluate Freeze-TCP coupling with the cross-layer signaling scheme: receiver issues zero-window advertisement on hearing freeze signal, and triple-acknowledgement on hearing unfreeze signal.

As shown in the graphs, TCP-Freeze-CR (with the enhancement schemes) outperforms the other two implementations. Furthermore, surprisingly, 'New Reno + Cross-layer' implementation yields similar performance with standard TCP. Therefore we conclude that the enhancement schemes improve the TCP performance significantly in scarce transmission opportunity.

We also observe that Freeze-TCP with the cross-layer signals shows low performance since the loss of the zero-window advertisement packet makes the ST strongly interfered by primary transmission, and the loss of triple acknowledgements stalls the transmission. Furthermore, it is sure the ST interferes with the primary transmission. So the loss of zero-window advertisement and triple acknowledgements can make disastrous results, which make the original Freeze-TCP difficult to be deployed in CRNs. We have not tested the mis-triggered spectrum synchronization by the buffer depletion since we have not implemented the flow control mechanism in any of the TCPs used for the experiments.

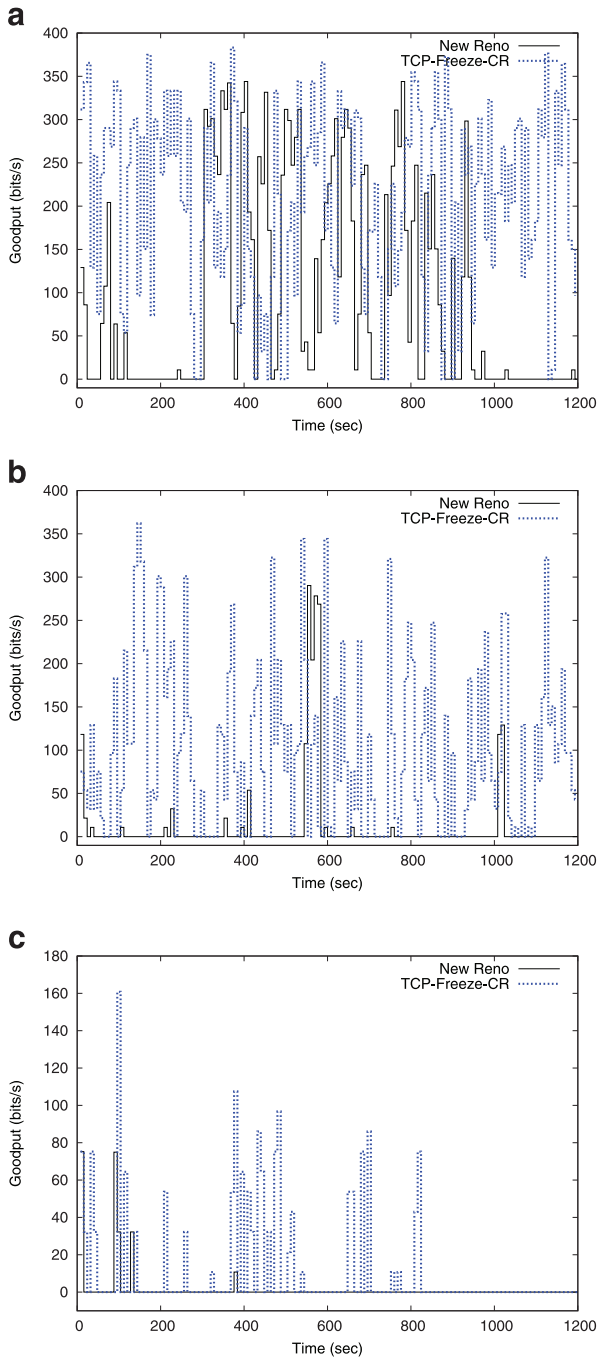


Fig. 6. Goodput of each TCP implementation with three different primary transmission patterns. 'New Reno' indicates standard TCP. (a) In case $(T_{on}, T_{off}) = (2, 8)$. (b) In case $(T_{on}, T_{off}) = (5, 5)$. (c) In case $(T_{on}, T_{off}) = (8, 2)$.

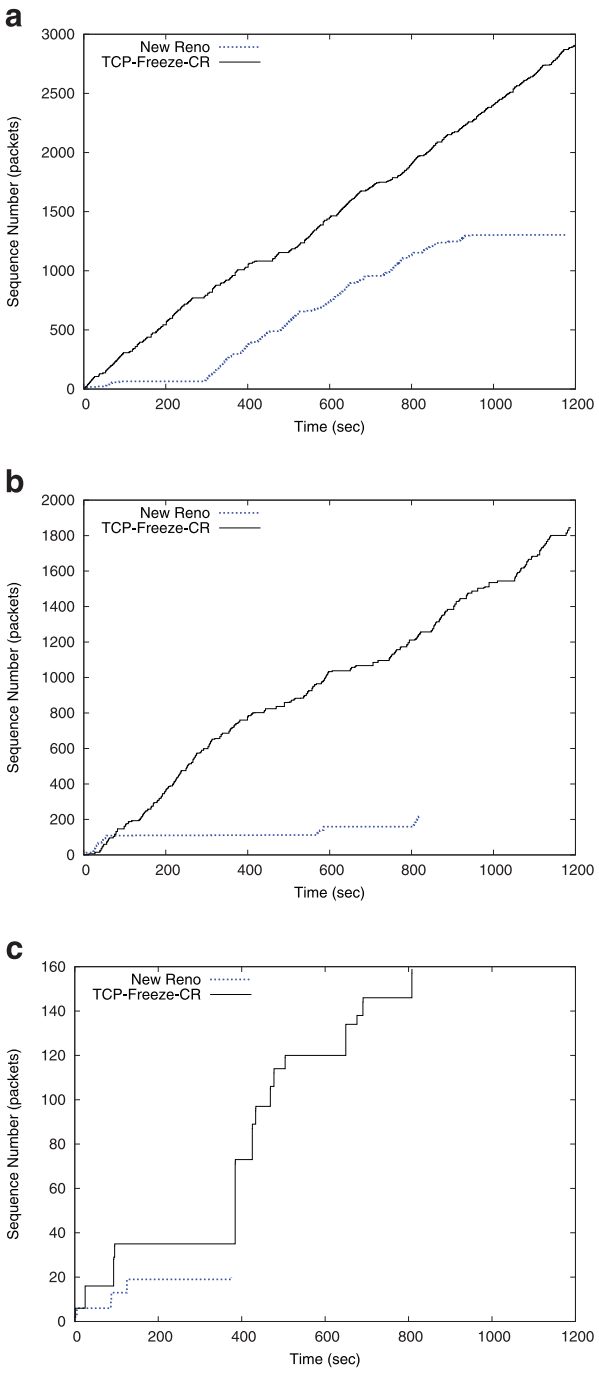
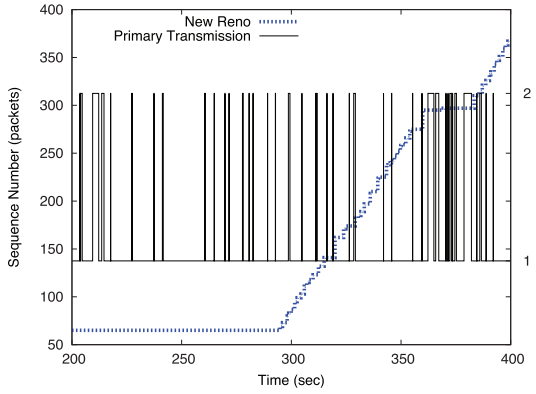


Fig. 7. Trace of the packet sequence measured in the SR (a) In case $(T_{on}, T_{off}) = (2, 8)$. (b) In case $(T_{on}, T_{off}) = (5, 5)$. (c) In case $(T_{on}, T_{off}) = (8, 2)$.

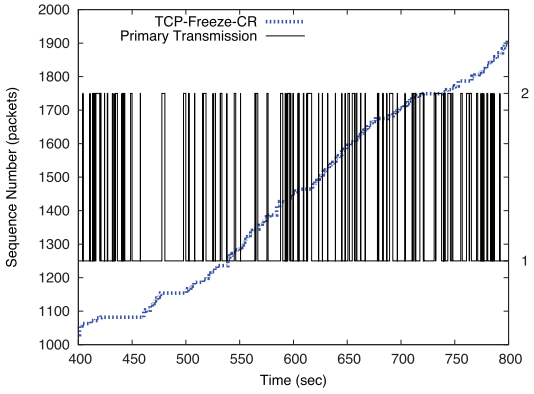
6. Related work

Lately, several SDR-based experimental platforms and testbeds for CRNs have shown up in the research field, and have been paid attention to by researchers as well as practitioners. In [29] and [30], the authors have reviewed the most popular SDR platforms and introduced exemplary CRNs. Mainly, those SDR-based platforms are designed to carry out the majority of lower layer operations in software with minimal hardware RF front-end; hence, we can achieve cross-layer implementation on these platforms more conventionally since lower layers as well as transport layers can be implemented using high level programming languages, and which enables easy implementation of cross-layer signaling between layers.

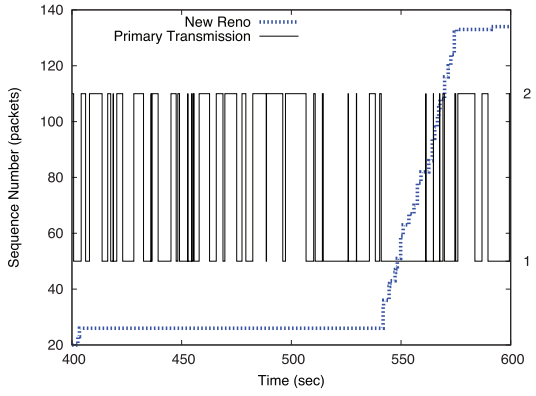
Issariyakul et al. [5] have initiated the issues of the TCP performance in CRNs. Then a few approaches [10–12,31,32] have addressed the same issues over cognitive radio ad-hoc networks. However, all of them are evaluated via simulations or numerical tests with somewhat optimistic and convenient assumptions: for instance, perfect knowledge on the channel state [31,33], an error-free feedback channel [10,12], statistical knowledge on primary transmission [10,12,32], static channel gain during a single RTT [31,33], perfect knowledge on the available bandwidth [11], etc.



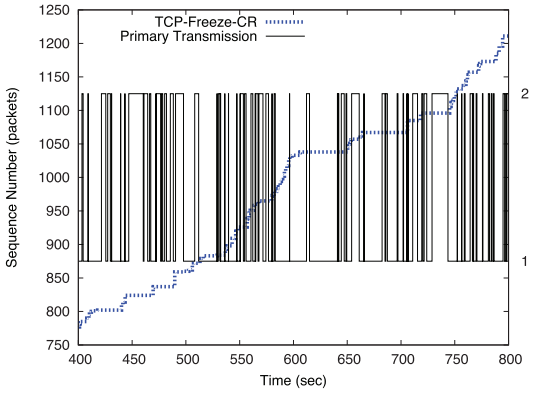
(a) standard TCP in case $(T_{on}, T_{off}) = (2, 8)$



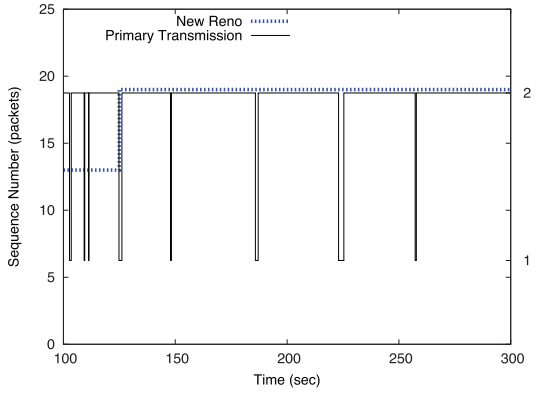
(b) TCP-Freeze-CR in case $(T_{on}, T_{off}) = (2, 8)$



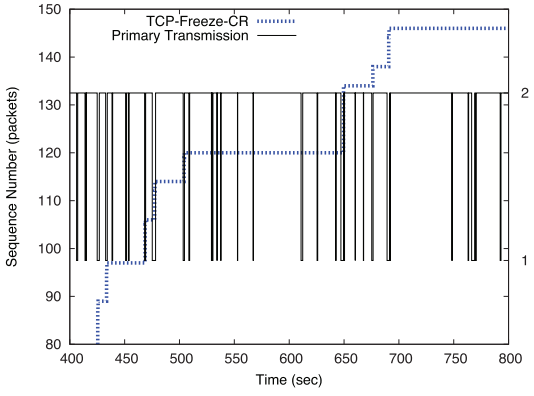
(c) standard TCP in case $(T_{on}, T_{off}) = (5, 5)$



(d) TCP-Freeze-CR in case $(T_{on}, T_{off}) = (5, 5)$



(e) standard TCP in case $(T_{on}, T_{off}) = (8, 2)$



(f) TCP-Freeze-CR in case $(T_{on}, T_{off}) = (8, 2)$

Fig. 8. Trace of the packet sequence along with the channel availability. The channel availability (denoted as 'Primary Transmission' in the legends) corresponds to the right *y*-axis: 1 indicates the ST and SR succeed to secure and synchronize to the other idle channel, and 2 indicates lower layer transmission stops since either ST or SR detects primary transmission.

For all that, it is still challenging to implement cross-layer schemes in general computer systems since it is inevitable to manipulate the TCP stack and device driver in operating system. In [34] and [35], the authors have implemented cross-layer approaches to improve TCP performance over multi-hop networks; their approaches were implemented in Linux kernel and MadWiFi driver.

Kumar and Shin [36] have proposed a scheme called DSASync-TCP targeting infra-structured CRNs. Their scheme is motivated by the scheme of Snoop agent [15]. Hence, DSASync-TCP is imple-

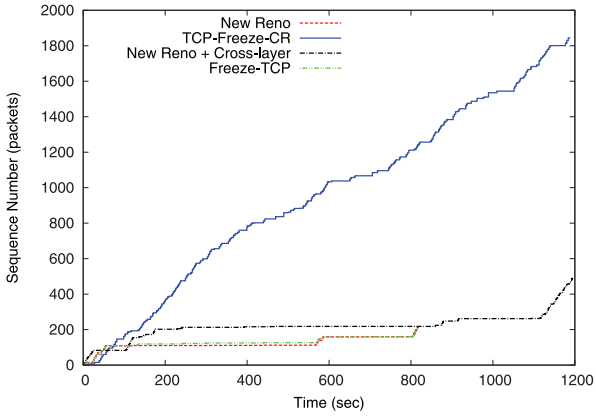


Fig. 9. Trace of packet sequence.

mented in base station (BS), and standard TCP can be used in both of corresponding host (CH) and SU (defined as spectrum-agile host in the paper) without any modification. In DSASync-TCP, BS buffers downlink packets during the interval of spectrum sensing/synchronization, and performs local transmission as soon as the sensing and synchronization processes are over. In order to prevent the buffer overflow in BS, BS advertises zero window size to CH when she detects no more free space in her buffer. Furthermore, the authors have addressed a regulation scheme to overcome the high jitter. They have implemented DSASync-TCP as Linux kernel module and proceeded with the evaluation on a real testbed. However, they have not considered the performance degradation inflicted by the consecutive RTOs.

More recently, Al-Ali and Chowdhury [37] have proposed an equation-based rate control for CRN: TFRC-CR. They suppose that available channels are stored in a designated databases maintained by a central authority such as FCC. They modify TFRC in order to make it agile to the PU activity, that is, faster rate recovery after PU departure. However, their method is strongly dependent on the central authority for detecting the PU activity and idle spectrum: the SU should be continuously informed of the PU activity by the central authority.

7. Cognitve radio multihop networks

TCP-Freeze-CR scheme itself can be applied to multi-hop networks in two different ways: hop-by-hop and end-to-end. It is essential to add the store-and-forward capability if we consider hop-by-hop deployment. If we intend to maintain the true end-to-end TCP, the cross-layer signaling should be relayed to source whenever it is either frozen or unfrozen.

The hop-by-hop deployment will suffer from inefficient layer traversing and the waste of buffer space: all TCP packets should be delivered to, and stored by TCP layer in every intermediate node until they are acknowledged from the next hop. However, the disruption due to the primary user detection experienced by intermediate nodes can be entirely hidden and the cross-layer signals do not need to be sent to the source. Therefore the hop-by-hop deployment is beneficial in terms of channel utilization.

There is neither layer traversing nor waste of buffer space in the end-to-end deployment. However it suffers from large delay on freezing TCP transmission since the cross-layer signals should always be relayed to source. This large delay may incur unwanted interference to transmitting PUs. Moreover the transmission should be frozen even when only one intermediate node detects primary transmission, and which results in poor channel utilization.

As a conclusion, we believe that hop-by-hop deployment will be more beneficial in terms of throughput as well as channel utilization. Therefore we have a plan to implement the hop-by-hop deployment and investigate the performance gain over the end-to-end deployment extensively.

8. Concluding remarks and future work

In this paper, we address the problem that standard TCP in SU side encounters in overlay-CRNs; primary transmission disrupts SUs' transmission, which results in drastic decrease of TCP performance following consecutive RTOs and exponential backoff of RT. Inspired by Freeze-TCP, we propose a cross-layer approach called TCP-Freeze-CR to tackle this problem. We verify the performance improvement by implementing our approach on a real testbed consisting of the software radio platforms, USRP E100.

In this work, we consider only single-hop network. However, we need to show that our approach can be extended to multi-hop network. It is also possible to couple the cross-layer mechanism with one of TCPs for mobile ad-hoc network (MANET) or other TCP variants that consider fair bandwidth sharing among multiple connections (e.g., TFRC).

We consider overlay-CRN only in this work. Our approach can be further developed for supporting underlay-CRNs coupled with transmission power control mechanism. Also it is possible to deploy proactive spectrum sensing that can relax the constraint that PUs are tolerant on the message exchanges for the spectrum synchronization, and incorporate packet loss differentiation and link layer retransmission schemes in order to achieve faster packet recovery right after ST is unfrozen. It is also worthwhile to study a TCP regulator scheme, whose requirement has already been addressed in [36], in order to cope against the high jitter inflicted by consecutive freeze and unfreeze operations.

Last but not least, we have considered here a simple energy detection mechanism only for detecting primary transmission, and configure the experimental network where SUs never fail to detect primary transmission. We admit that this setup is far from reality, and it is more valuable to consider more complex and realistic CR environments with false alarming and detection fail, which definitely yields poorer TCP performance or unwanted interference to primary transmission. Therefore we need to consider imperfect spectrum sensing and couple more advanced spectrum sensing scheme such as statistical hypothesis testing [38] with our TCP.

References

[1] J. Mitola III, G. Maguire Jr., Cognitive radio: making software radios more personal, IEEE Personal Commun. 6 (4) (1999) 13–18.
[2] S. Floyd, T. Henderson, A. Gurtov, The NewReno Modification to TCP's Fast Recovery Algorithm, 2004 (RFC 3782 (Proposed Standard)).
[3] N. Parvez, A. Mahanti, C. Williamson, An analytic throughput model for TCP newreno, IEEE/ACM Trans. Netw. 18 (2) (2010) 448–461.
[4] Y. Tian, K. Xu, N. Ansari, TCP in wireless environments: problems and solutions, IEEE Commun. Mag. 43 (3) (2005) S27–S32.
[5] T. Issariyakul, L. Pillutla, V. Krishnamurthy, Tuning radio resource in an overlay cognitive radio network for TCP: greed isn't good, IEEE Commun. Mag. 47 (7) (2009) 57–63.
[6] T. Goff, J. Moronski, D. Phatak, V. Gupta, Freeze-TCP: a true end-to-end TCP enhancement mechanism for mobile environments, in: Proceedings of IEEE INFOCOM, vol. 3, 2000, pp. 1537–1545.
[7] UHD (USRP Hardware Driver).
[8] T. Schmid, GNU Radio 802.15.4 en- and Decoding, NESL Technical Report, 2006.
[9] S. Haykin, D. Thomson, J. Reed, Spectrum sensing for cognitive radio, Proc. IEEE 97 (5) (2009) 849–877.
[10] K. Chowdhury, M. Di Felice, I. Akyildiz, TP-CRAHN: a transport protocol for cognitive radio ad-hoc networks, in: Proceedings of IEEE INFOCOM, 2009, pp. 2482–2490.
[11] D. Sarkar, H. Narayan, Transport layer protocols for cognitive networks, in: Proceedings of IEEE INFOCOM Workshops, 2010, 2010, pp. 1–6.
[12] K.R. Chowdhury, M. Di Felice, I.F. Akyildiz, TCP CRAHN: a transport control protocol for cognitive radio ad hoc networks, IEEE Trans. Mobile Comput. 12 (4) (2013) 790–803.
[13] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, R. Wang, TCP Westwood: bandwidth estimation for enhanced transport over wireless links, in: Proceedings of ACM MobiCom, 2001.

[14] M. Lee, M. Kang, M. Kim, J. Mo, A cross-layer approach for TCP optimization over wireless and mobile networks, Comput. Commun. 31 (11) (2008) 2669–2675.
[15] H. Balakrishnan, S. Seshan, E. Amir, R.H. Katz, Improving TCP/IP performance over wireless networks, in: Proceedings of ACM MOBICOM, 1995, pp. 2–11.
[16] H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Trans. Netw. 5 (6) (1997) 756–769.
[17] Y. Zhao, S. Mao, J. Neel, J. Reed, Performance evaluation of cognitive radios: metrics, utility functions, and methodology, Proc. IEEE 97 (4) (2009) 642–659.
[18] GNU Radio.
[19] J.-H. Hauer, V. Handziski, A. Wolisz, Experimental study of the impact of WLAN interference on IEEE 802.15.4 body area networks, in: Proceedings of European Conference on Wireless Sensor Networks (EWSN), 2009, pp. 17–32.
[20] S. Chepuri, R. de Francisco, G. Leus, Performance evaluation of an IEEE 802.15.4 cognitive radio link in the 2360–2400 MHz band, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), 2011, pp. 2155–2160.
[21] M. Timmers, S. Pollin, A. Dejonghe, L. Van der Perre, F. Catthoor, Exploring vs. exploiting: enhanced distributed cognitive coexistence of 802.15.4 with 802.11, in: Proceedings of IEEE Sensors, 2008, pp. 613–616.
[22] A. Ayadi, P. Maille, D. Ros, TCP over low-power and lossy networks: tuning the segment size to minimize energy consumption, in: Proceedings of IFIP International Conference on New Technologies, Mobility and Security, 2011.
[23] A. Dunkels, T. Voigt, J. Alonso, Making TCP/IP viable for wireless sensor networks, in: Proceedings of European Workshop on Wireless Sensor Networks (EWSN), Work-in-Progress Session, 2004.
[24] A. Dunkels, Full TCP/IP for 8 bit architectures, in: Proceedings of ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys), 2003.
[25] F.L. Piccolo, D. Battaglino, L. Bracciale, A. Bragagnini, M.S. Turolla, N.B. Melazzi, On the IP support in IEEE 802.15.4 LR-WPANs: self-configuring solutions for real application scenarios, in: Proceedings of IFIP Ad Hoc Networking Workshop (Med-Hoc-Net), 2010, pp. 1–10.
[26] T. Yucek, H. Arslan, A survey of spectrum sensing algorithms for cognitive radio applications, IEEE Commun. Surv. Tutor. 11 (1) (2009) 116–130.
[27] Pyro: Python Remote Objects 4.x.
[28] V. Jacobson, Congestion avoidance and control, SIGCOMM Comput. Commun. Rev. 18 (4) (1988) 314–329.
[29] K. Chowdhury, T. Melodia, Platforms and testbeds for experimental evaluation of cognitive ad hoc networks, IEEE Commun. Mag. 48 (9) (2010) 96–104.
[30] P. Pawelczak, K. Nolan, L. Doyle, S.W. Oh, D. Cabric, Cognitive radio: ten years of experimentation and development, IEEE Commun. Mag. 49 (3) (2011) 90–100.
[31] D. Chen, H. Ji, V. Leung, Distributed optimal relay selection for improving TCP throughput over cognitive radio networks: a cross-layer design approach, in: Proceedings of IEEE International Conference on Communications (ICC), 2011, pp. 1–5.
[32] C. Luo, F. Yu, H. Ji, V. Leung, Cross-layer design for TCP performance improvement in cognitive radio networks, IEEE Trans. Veh. Technol. 59 (5) (2010) 2485–2495.
[33] D. Chen, H. Ji, V. Leung, Distributed best-relay selection for improving TCP performance over cognitive radio networks: A cross-layer design approach, IEEE J. Sel. Areas Commun. 30 (2) (2012) 315–322.
[34] A. Warrier, S. Janakiraman, S. Ha, I. Rhee, Diffq: practical differential backlog congestion control for wireless networks, in: Proceedings of IEEE INFOCOM, 2009, pp. 262–270.
[35] S. ElRakabawy, C. Lindemann, A practical adaptive pacing scheme for TCP in multihop wireless networks, IEEE/ACM Trans. Netw. 19 (4) (2011) 975–988.
[36] A. Kumar, K. Shin, Managing TCP connections in dynamic spectrum access based wireless lans, in: Proceedings of IEEE Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010, pp. 1–9.
[37] A. Al-Ali, K. Chowdhury, TFRC-CR: An equation-based transport protocol for cognitive radio networks, in: Proceedings of International Conference on Computing, Networking and Communications (ICNC), 2013, pp. 143–148.
[38] G. Chaitanya, P. Rajalakshmi, U. Desai, Real time hardware implementable spectrum sensor for cognitive radio applications, in: International Conference on Signal Processing and Communications (SPCOM), 2012, pp. 1–5.

Sang-Seon Byun. He received B.S., M.S., and Ph.D. degrees in Computer Science from Korea University, Seoul, South Korea in 1996, 2002, and 2007, respectively. He was an assistant research professor of the Graduate School of Embedded Software, Korea University, in 2007. From 2007 to 2012, he has served as postdoctoral researcher and research scientist of the Department of Electronics and Telecommunications, Norwegian University of Science and Technology, Trondheim, Norway. He has also worked as a research professor in the School of Information and Communications, Gwanju Institute of Science and Technology (GIST), Gwangju, South Korea, and a senior researcher of Deagu-Gyeongbuk Medical Innovation Foundation (DGMIF), Daegu, South Korea. Currently, he is an assistant professor of Department of Computer Engineering, Catholic University of Pusan, Busan, South Korea. His research interests include the fields of cognitive radio networks and software-defined radio architecture.