Towards Fast Flow Convergence in Cognitive Radio Cellular Networks

Fan Zhou*, Marco Di Felice†, Benjamin Drozdenko‡ and Kaushik Chowdhury*
* Dept. of Electrical and Computer Engineering, Northeastern University, Boston, MA, USA
† Dept. of Computer Science, University of Bologna, Bologna, Italy
‡ Louisiana Tech University, Ruston, LA, USA

***Abstract*—Cognitive radio (CR) is an enabling technology that allows opportunistic use of under-utilized licensed spectrum allocated to primary users (PUs). However, the frequent channel sensing and switching interferes with the transport layer functions, leading to slow flow convergence during active transmissions by the CR. In this paper, we propose TCP C^2, a method that greatly improves the flow responsiveness to abrupt variation of underlying layer spectrum availability in cellular CR architectures. The key idea of C^2 is to allow the sender to estimate the current bottleneck link bandwidth and network load by observing variance in the throughput and round trip time. Following this, fast congestion window scaling allows the flow to converge quickly to the optimal sending rate. Analytic derivations and packet-based simulation results show the increased resiliency of our approach over classical end-to-end TCP protocols in the presence of intermittent spectrum sensing and disruptions caused by PU arrival. Additionally, we show that C^2 enforces fairness among flows, and also coexists well with classical TCP flavors.**

I. Introduction

The design of transport layer protocol remains an open challenge in cognitive radio (CR) cellular networks, owing to the difficulties in interpreting the impact of various physical and link-layer decisions in response to changing spectrum conditions. While the problem of bandwidth limitation and high link latency is already difficult to solve in cellular networks, CR adds more complexity because of the higher priority channel access by primary users (PUs). At such times, the sending rate of the CR users must be scaled back, and when the PU vacates the spectrum, the channel must be efficiently utilized. In this paper, we show that the binary nature of the spectrum availability is not handled well by AIMD model of classical TCP, and we propose a new end-to-end congestion control approach that allows flows to quickly respond to spectrum interruptions and achieve high link utilization.

CR networks have several unique features that restrict the end-to-end performance. First, the path is disconnected during spectrum sensing and switching. Moreover, the time taken by CR users to resume transmission is usually unknown to the server. Since TCP cannot distinguish between disconnections caused by network congestion and loss of spectrum availability, frequent transmission timeout events are triggered, causing congestion window (`cwnd`) to be reduced to 1. Second, as we show in Sec. III, the additive-increase/multiplicative-decrease (AIMD) mechanism during congestion avoidance stage of standard TCP does not enable `cwnd` to ramp up quickly during the short consecutive transmission periods, which greatly limits the flow throughput. Finally, the situation is further deteriorated by the abrupt bandwidth variation after channel sensing, as CR users may choose to switch to a new frequency band for higher capacity. AIMD does not respond in a timely manner to large changes in channel capacity, causing the available spectrum resources to be under-utilized.

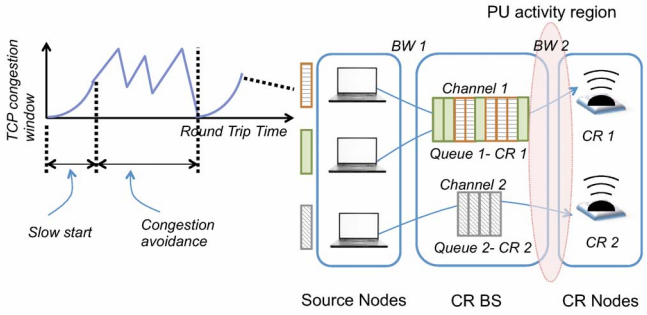


Fig. 1. Network architecture of a CR cellular BS serving multiple CR nodes.

To address these problems, we present C^2: a new congestion control approach aimed at efficiently utilizing the channel resources given rapid variance in spectrum availability in Cognitive Radio Cellular networks. C^2 significantly increases the transport layer responsiveness by quickly scaling `cwnd` to an appropriate value, according to the estimation of available bandwidth and number of active flows sharing the bottleneck link. This guarantees fast convergence among flows given abrupt spectrum fluctuations, since we skip the conservative bandwidth probing step of AIMD in traditional TCP. Another unique feature of C^2 is that it only requires minimum feedback from destination during connection startup stage. This lightweight design reduces the operation overhead and deployment barrier, since it does not require modifications to lower layer or intermediate switches.

This paper makes three main contributions:

- We mathematically demonstrate the reduction of flow throughput due to poor interaction between TCP and CR operation through performance modeling (Sec. IV-A).
- We propose C^2, which completely alters the design of *slow start* and *congestion avoidance* stages of standard TCP. C^2 allows each sender to estimate its own optimal sending rate in a distributed manner by interpreting the number of active competing flows and total link capacity. Then, C^2 scales `cwnd` quickly, according to the estimation results, to achieve high

978-1-5090-5019-2/17/$31.00 ©2017 IEEE

spectrum utilization and low queuing delay (Sec. IV).

• We implement and evaluate C^2 using *ns3*. We show that C^2 achieves 2x better throughput compared to TCP NewReno and TCP Westwood+, and approximately 20% improvement in throughput while also incurring 4X shorter queuing delay against Cubic (default implementation in Linux). We also show C^2 can co-exist fairly with loss-based TCP, owing to its novel *tit-for-tat* competition strategy. (Sec. V).

II. Related Work

TCP has been extensively studied over last three decades. The first widely deployed TCP is NewReno [2], with the classical AIMD rule for updating the `cwnd`. Since then, many other TCP flavors designed for specific network architectures have been proposed. For example, Westwood+ [3] uses bandwidth estimation to cut down `cwnd` intelligently after packet loss. It also performs better in high link loss wireless networks. Cubic [4] acts more aggressively than NewReno and can achieve higher link utilization in high speed networks. Other more recent end-to-end TCP variants include TCP FAST [6], TCP Compound [7], and Sprout [9]. However, none of these protocols target CR networks, nor do they adapt to the high variation in spectrum availability.

Transport layer protocols designed for CR network include TP-CRAHN, which distinguishes various network events that may lead to performance degradation by a combination of explicit feedback from the intermediate nodes and the destination [10]. Another CR-specific TCP variant, TFRC-CR, was proposed in [11]. The idea behind TFRC-CR is to enable a sender to identify reasons for decreasing throughput, such as network congestion or PU activity, by acquiring spectrum availability information from the FCC's spectrum database. The sender can then directly adjust the sending rate according to an improved rate control equation based on TFRC. A different approach to improve throughput in a CR network is proposed in [12], which involves adapting the functions at the PHY/MAY layers such as spectrum sensing/access, modulation, and coding, while leaving the behavior of standard TCP untouched.

However, all these works involve extensive use of cross-layer or intermediate-node information. While this approach can provide performance improvement by having more accurate insight into the network condition, its generality is greatly limited. The reason for this is that the transport layer operates end-to-end from a classical networking viewpoint. Thus, enforcing dependence on the choices made at the lower layers, or mandating feedback from other intermediate nodes, is in opposition to the commonly accepted end-to-end paradigm. In contrast, C^2 leverages modifications only at the end-server/mobile client nodes, and does not require any explicit notification from intermediate base stations.

III. Network and Performance Model

The network scenario we consider is shown in Fig. 1. Here, the CR cellular base station (BS) serves multiple mobile CR users through wireless links. The connectivity from the BS to the source nodes are high-speed wired links. Each CR node must perform spectrum sensing and channel switching at pre-decided intervals due to the interruption of primary users. However, the CR nodes do not collaborate with each other, and the BS also does not return any feedback to the source nodes. Thus, this is a general scenario because the BS here does not inject itself into the protocol operation.

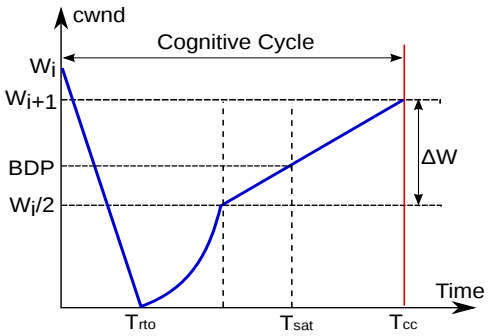


Fig. 2. Performance model of AIMD congestion control in 1 cognitive cycle.

We first study the expected reduction in throughput for a TCP flow that uses classical AIMD congestion control in a CR network. We define one Cognitive Cycle (CC) as the duration from the moment the CR node starts channel sensing/switching to the next transmission interruption. For simplicity, we assume a constant duration (T_{cc}) and availability of the wireless link bandwidth (BW) during each CC. While this model ignores the randomness of CR activities, it helps us to understand the basic trends in performance reduction, which we use to evaluate the influences of varying CC duration and bandwidth to the flow performance in Sec. V.

Convergence of `cwnd`: We first show that the `cwnd` will quickly converge to a low level during each CC. As shown in Fig.2, we assume W_i is the `cwnd` before the flow enters $i's$ CC. The bursty packet loss during channel sensing/switching triggers TCP retransmission timeout (RTO) and reduces W_i to 1 and `ssthresh` to $W_i/2$. Following this, the flow will enter the standard slow start and congestion avoidance. Thus, we can compute the W_{i+1} as $W_{i+1} = \frac{W_i}{2} + \Delta W$.

Here, ΔW is the incremental `cwnd` during congestion avoidance in each CC. We consider ΔW to be approximately constant during each CC, since it is decided by CC duration and RTT. Thus, assuming the initial `cwnd` before the first cognitive cycle is W_0, with W_{i+1} given by equation (1).

$$W_{i+1} = (W_0 - 2\Delta W) * (\frac{1}{2})^{i+1} + 2\Delta W \qquad (1)$$

Thus, we see that `cwnd` exponentially converges to $2\Delta W$ as the number of CC durations increase.

Next, we compute ΔW. Note $T_{cc} = T_{rto} + T_{ss} + T_{ca}$ where T_{rto}, T_{ss} and T_{ca} are the durations of TCP transmission timeout, slow start, and congestion avoidance, respectively. Since `cwnd` doubles every RTT during slow start and increases by 1 during congestion avoidance, we have $T_{ss} = RTT * log_2 \Delta W$ and $T_{ca} = RTT * \Delta W$. Substituting T_{ss} and T_{ca} into T_{cc}, ΔW is given by equation (2).

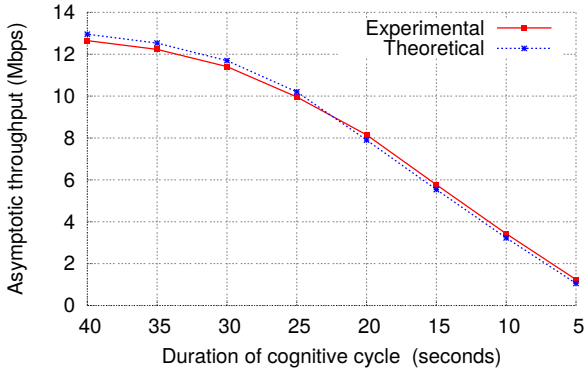


Fig. 3. The asymptotic throughput drops with shorter duration of cognitive cycle ($BW = 15Mbps, RTT = 150ms$)

$$\Delta W + log_2 \Delta W = \frac{T_{cc} - T_{rto}}{RTT} \tag{2}$$

For simplicity, $\mathcal{R}$ denotes $\frac{T_{cc}-T_{rto}}{RTT}$. Since equation (2) has no closed-form solution, we derive a simple approximation formula for ΔW. To do so, notice $\mathcal{R}/2 < \Delta W < \mathcal{R}$ since $\Delta W > log_2 \Delta W$ for $\Delta W > 0$. Taking the log function of each side of the inequality, we get $log_2 \mathcal{R} - 1 < log_2 \Delta W < log_2 \mathcal{R}$. Substituting this back into (2) gives $\mathcal{R} - log_2 \mathcal{R} - 1 < \Delta W < \mathcal{R} - log_2 \mathcal{R}$. Therefore, an approximation of ΔW is given by equation (3).

$$\Delta W = \mathcal{R} - log_2 \mathcal{R} \tag{3}$$

The converged `cwnd` is only decided by the consecutive transmission time ($T_{cc} - T_{rto}$) and RTT, but is unrelated to the channel capacity. This implies that the limiting factor of `cwnd` is the frequent TCP retransmit events, which are caused by CR activities rather than network bandwidth. Next, we study how this influences the flow throughput.

Asymptotic throughput: We calculate the average throughput TP in each CC after `cwnd` converges to $2\Delta W$, which also reflects asymptotic throughput of a single flow.

Let $tp(t)$ and $W(t)$ be the real time throughput and `cwnd` at time t. The network saturates once $W(t)$ reaches one bandwidth product delay (BDP). Assuming the network saturates at time T_{sat} (Fig. 2), then $tp(t) = \frac{W(t)}{RTT}$ when $t < T_{sat}$ and $tp(t) = BW$ when $t > T_{sat}$. Thus, TP can be expressed as:

$$TP = \frac{\frac{1}{RTT}\int_0^{T_{sat}} W(t)dt + BW * (T_{cc} - T_{sat})}{T_{cc}} \tag{4}$$

Depending on the saturation time, three conditions need to be considered to compute TP:

Case 1: $T_{rto} < T_{sat} < T_{rto} + T_{ss}$ The network saturates during slow start stage. In this case, TP can be calculated as:

$$TP = \frac{BDP - 1 + BW * (T_{cc} - T_{sat})}{T_{cc}} \tag{5}$$

where $T_{sat} = T_{rto} + RTT * log_2 BDP$.

Case 2: $T_{rto} + T_{ss} < T_{sat} < T_{rto} + T_{ss} + T_{ca}$

The network saturates during the congestion avoidance stage (e.g., the scenario shown in Fig. 2). The total packets transmitted in one CC can be calculated as the sum of packets sent during slow start ($P_{ss} = \Delta W - 1$), i.e., from slow start to network saturation ($P_{pre} = (BDP + \Delta W) * (BDP - \Delta W)/2$) and after saturation ($P_{after} = BW * RTT * (2\Delta W - BDP)$). Thus, we have:

$$TP = \frac{\Delta W - 1 - \frac{1}{2} * (BDP^2 - 4 * BDP * \Delta W + \Delta W^2)}{T_{cc}} \tag{6}$$

Case 3: $T_{sat} T_{rto} + T_{ss} + T_{ca}$

The network cannot saturate during CC. The packets transmitted are $\Delta W - 1 + \frac{3}{2} * \Delta W^2$. So, we have:

$$TP = \frac{\Delta W - 1 + \frac{3}{2} * \Delta W^2}{T_{cc}} \tag{7}$$

where ΔW is calculated with (3).

Model Validation: We validate our model through $ns3$ simulation. As shown in Fig.3, the measured asymptotic throughput matches closely with our analytical results. As expected, the throughput drops quickly as the CC duration decreases. The main reason is that the TP is largely decided by the ΔW, which is limited by the slow `cwnd` increase within the traditional AIMD model during one CC.

IV. The Design of C^2

Based on the insight of the key factors that influence the CR network performance, C^2 is designed to increase the flow responsiveness to frequent transmission interruptions and maximizes channel utilization. The key feature of C^2 is that it replaces the conservative operation of the AIMD model by allowing a flow to quickly converge to its optimal share of available bandwidth during slow start (Sec. IV-A). Then, C^2 adjusts its own estimation of available network bandwidth according to varying network loads in the following congestion avoidance stages (Sec. IV-B). In addition, C^2 also incorporates design features to enable fair co-existence with standard TCP variants they may be other protocols deployed in a wide-area CR network (Sec. IV-C).

A. Fast flow convergence in slow start

In standard TCP, the sender increases `cwnd` by one segment during slow start for every newly arrived ACK, until it exceeds the `ssthresh`, or experiences packet loss. The purpose of *slow start* is to quickly probe for the end-to-end network capacity by injecting an increasing number of packets in the network until the bottleneck buffer overflows.However, to ensure the upper bound of network capacity can be attained, the initial `ssthresh` is set arbitrarily high. This can cause two problems: first, it usually takes several RTT-cycles before slow start completes and reduces the convergence speed. Second, it usually results in a bursty loss of packets, and spectrum resource is wasted to retransmit the lost packets.

The above problems are critical in CR network given frequent slow starts triggered by periodic spectrum sensing or channel switching. Thus, the protocol must allow a flow to quickly converge to its optimal sending rate as soon as

possible. To do so, C^2 sets `ssthresh` to $BW * D_p/(N+1)$ directly when entering the slow start stage. Here, BW is the estimated available bandwidth, D_p is the round trip propagation delay, and N is the degree of multiplexing (number of existing flows sharing the bottleneck link). This is the optimal setting of `ssthresh`, since it represents the number of outstanding packets that should be injected into the network for each sender to get its fair share of the network bandwidth.

The next problem to solve is how TCP C^2 should quickly estimate BW, D_p and N.

1) Available Bandwidth BW: C^2 probes the bandwidth using the fact that packets are sent in sequence during the *slow start*. Let the sequence length be L, packet size be S, and the time instances that the source receives the ACK for first packet and last packet of the sequence be T_1 and T_2. Then, the channel bandwidth can be accurately estimated as $(L-1)*S/(T_2-T_1)$ if there is only one flow in the network.

However, if there were multiple flows sharing the bottleneck link, the above estimation is not accurate because packet sequences from different flows may intersect with another. To solve this problem, C^2 sets `cwnd` to 1 first and then estimates the bandwidth for the next sequence of packets when `cwnd` is 2. To demonstrate why this would greatly reduce the probability of mixing within sequences, consider the network scenario as shown in Fig. 1, where the bandwidth of the wired side of the network is BW_{fast} and the CR channel bandwidth is BW_{slow}. We assume L senders send the first packet simultaneously. Since the CR wireless link needs S/BW_{slow} to transmit one packet, there would be a time gap ΔT between a receiver's receipt of the two packets and the transmission of the ACKs.

Clearly, ΔT is also the time gap between any two senders who choose to begin transmitting the next sequence of packets. As long as ΔT is long enough for the wired link to transmit the second sequence of packets, no mixing of packet sequences would occur. We calculate the probability of estimation error caused by sequence intersection as $P(Error) = P(L * \frac{S}{BW_{fast}} > \frac{S}{BW_{slow}}) = P(L > \frac{BW_{fast}}{BW_{slow}})$

In summary, the estimation is accurate as long as the wired link bandwidth is at least L times faster as wireless link bandwidth. This is the common case where the network is bottlenecked by the slow last-mile wireless access link, while the uplink to the Internet (Ethernet) offers much higher bandwidth [5].

2) Propagation delay D_p: C^2 takes the smallest RTT observed during transmission as the estimation of propagation delay D_p, which is usually the first RTT sample taken during slow start. However, D_p may be an over-estimate due to the accumulated packets in the bottleneck link buffer. However, this is not a critical issue since C^2 will deplete the queued packets by decreasing `cwnd`, as described in Sec. IV-B.

3) Number of concurrent flow N: C^2 requires the destination to piggyback the number of total flows in the SYN-ACK packet header to the sender during the initial 3-way handshake. After that each sender updates the value of N adaptively, as explained in Sec. IV-B.

B. Adaptive rate adjusting in congestion avoidance

C^2 needs to update the estimation of active flow number N so that each sender can adaptively adjust its sending rate and adapt to the varying network load during congestion avoidance stage. C^2 uses a delay-based approach to keep track of network states and separates the operation into three stages: *saturation*, *starvation*, and *ideal* according to RTT and propagation delay D_p:

- $RTT \geq D_p * (1+\alpha)$: *saturation* stage, the network may be congested and each sender will check whether it has underestimated the flow number N.
- $RTT \leq D_p * (1+\beta)$: *starvation* stage suggests possible underutilization of bandwidth. It may happens when one or more flows leave the network.
- $RTT \in D_p * (1+\alpha, 1+\beta)$: *ideal* stage, indicates that the network has converged to the optimal point where each flow gets its own fair share, and the queuing delay is low.

Note α and β are parameters controlling the tradeoff between throughput and delay. C^2 will try to keep the network in *ideal* stage to maintain high throughput and minimum queuing delay. Next, we describe the operations taken by C^2 during *saturation* and *starvation* stages to achieve this goal.

Saturation stage: This state indicates packets accumulating in the queue, which may occur when a new flow joins in. Hence, C^2 re-estimates the active flow number N using Algorithm 1.

Algorithm 1: Flow number estimation

Input: TP: measured throughput ; BW: estimated total link bandwidth
Output: N: estimated flow number

```
N = 1  r = TP/BW
while True do
    if r > 1/N * 1/(1+1/(N+1)) then
        break
    N = N + 1
return N
```

The idea of Algorithm 1 is that if the throughput of the current flow quickly drops to a new lower value TP, then C^2 considers that new flows may have joined in. Therefore, we can use the ratio between TP and link bandwidth as an indication of the change in flow number. Next, we describe how to estimate TP.

Assume N flows are equally sharing the bottleneck bandwidth. For simplicity, we assume that these flows saturate the bottleneck link and there are no packets in the queue. Thus, the `cwnd` of each sender is $BW * D_p/N$.

When a new flow arrives, it will quickly probe the bandwidth BW, propagation delay D_p, and existing flow number N and set `cwnd` to be $BW * D_p/(N+1)$ as we described in Sec. IV-A. Thus, the total number of outstanding packets in network is $BW * D_p * (1 + 1/(N+1))$. Note the fraction of throughput over link bandwidth should be equal to the fraction

of cwnd over total number of outstanding packets. Thus, the original sender's throughput TP converges to:

$$TP/BW = \frac{1}{N} * \frac{1}{1 + \frac{1}{N+1}} \quad (8)$$

The right side of (8) is the ratio used to estimate the number of flows in Algorithm 1 (Line 3). After N is estimated, C^2 checks if there has actually been an increase in flow number. If so, C^2 updates the sending rate by setting $cwnd = BW * D_p/N$. Otherwise, C^2 reduces cwnd by 1 over each RTT to deplete the accumulated packets in the bottleneck link buffer.

Starvation stage: C^2 regards $RTT < D_p * (1 + \beta)$ as a sign that the channel may not be fully utilized. In this case, C^2 will increase cwnd aggressively according to Alg. 2 to quickly saturate the network.

Algorithm 2: Adjust cwnd in *starve* state

Input: N: estimated flow number ; T_w: time window

```
1 k = N
2 T_epoh = now_time
3 while in starve stage do
4     if now_time − T_epoh > T_w then
5         k = k * 2
6         T_epoh = nowtime
7     cwnd = cwnd + k/cwnd
```

In the first time epoch, C^2 increases cwnd k times faster than classical AIMD of standard TCP, where k is initialized to the number of current estimated flows. Then, if the network is still in *starve* stage in next time window T_w, we double k to speed up cwnd, an increase by a factor of two. The idea is to start increasing the cwnd cautiously. If C^2 observes that this action does not result in extra delay, it means that the bottleneck is not saturated and it should increase cwnd more aggressively.

A key insight here is that the increasing speed of cwnd is proportional to its estimated flow number N. The advantage is that it can still lead to fairness allocation of bandwidth even though some flows have inaccurate estimation of N. We use an example to illustrate this. Assume flow 1 and flow 2 are competing for the same bottleneck link. However, flow 2 mistakenly estimates that there are three flows sharing the link and has lower throughput. The network is underutilized as a result, and both flows enters into aggressive stage. Since the increasing speed of cwnd is proportional to the estimated flow number N, flow 2 will increase its cwnd at least 1.5X faster than flow 1. This ensures that the two flows will quickly converge to fair share of the bottleneck bandwidth, which will result in flow 1 reaching the correct estimation of N.

C. Fairness with loss-based TCP

Fundamentally, C^2 is a delay-based approach as it uses RTT as congestion indication. A key limitation faced by delay based TCP is that it behaves conservatively when competing with loss-based TCP, since it starts to back off whenever queuing delay increases. Meanwhile, loss-based TCP will continue injecting packets into the network, which further increases delay and forces the competing delay-based TCP flavors to slow down.

To enable C^2 to compete fairly with loss based TCP, we adopt a *tit-for-tat* approach. The idea is that TCP C^2 monitors the aggression level of other flows. It behaves as described earlier if the competing flows take corrective actions with increasing delay. Otherwise, if aggressive action is detected, it enters into a *competition mode* during which cwnd is increased quickly to compete with co-existing flows. Next, we describe how C^2 detects aggressive flows.

C^2 requires every sender to keep RTT stable within $D_p * (1+\alpha, 1+\beta)$. Therefore C^2 keeps checking the *slope* of RTT and assumes that there are some aggressive senders if the slope is larger than a threshold σ for three consequential time windows. We choose the $\sigma = 1/(N*RTT)$ as it approximates the maximum increasing speed of RTT if every flow follows C^2. In this case C^2 will increase cwnd by N times faster than classical AIMD. Note N is inversely proportional to the flow's throughput. This implies that the more aggressive the competing flow, the quicker C^2 will increase its cwnd.

V. Performance Evaluation

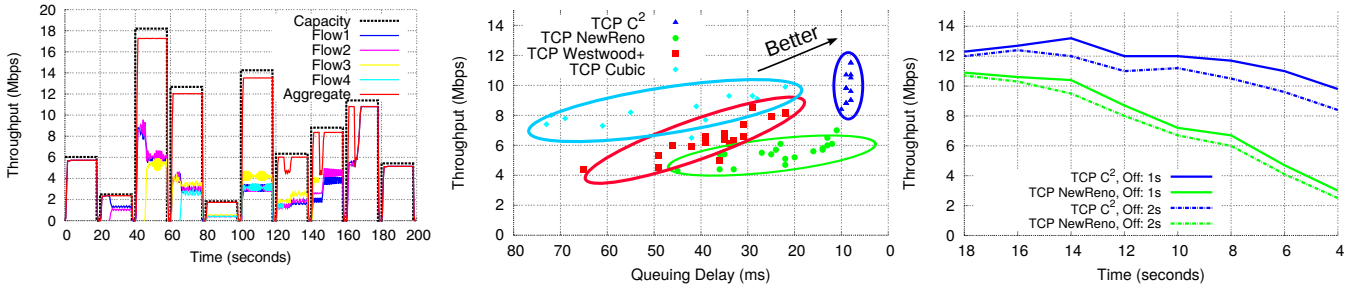
In this section, we present evaluation results of C^2 using packet level *ns3* simulations. We mainly focus on showing how C^2 adapts to the varying network conditions (bandwidth, frequent interruptions caused by PU activity/sensing) and its fairness properties. We retain the network model shown in Fig. 1 with multiple sources and CR mobile devices. We model the consecutive transmission *On* period and the disconnection caused by spectrum sensing/switching as *Off* period. We empirically choose $\alpha = 0.2, \beta = 0.05$ for stable performance in all evaluations.

A. Adapting to varying channel condition

We show that C^2 is able to rapidly respond to varying bandwidth after channel switching. We fix the "On" period to be 8s and the "Off" period to be 2s. The varying range of channel capacity is chosen between 1 and 20Mbps. Thus, the bandwidth availability changes by uniformly picking a value between 1 and 20Mbps every 20 seconds. As shown in Fig. 4(a), the aggregated throughput (red line) of four flows is very close to the total channel capacity (black line), which indicates a high link utilization. Also, when new flows arrive at 25s, 45s, and 65s, C^2 quickly reacts and converges to a level where each flow equally shares the available bandwidth. The reason for such fast convergence is that C^2 estimates the flow number according to measured throughput and available bandwidth, and jumps to its own optimal share directly. Besides, we note that the throughput quickly increases whenever flows leave (generally less than 1s), owing to our more aggressive increase of cwnd in the *starvation* stage.

B. Comparison with alternate protocols

We compare the performance of C^2 with other end-to-end TCPs: NewReno, Westwood+, and Cubic. In Fig. 4(b), the top-right area on the plot represents better performance (higher



(a) (b) (c)

Fig. 4. (a) Convergence of four C^2 flows, (b) Throughput - delay of TCP variants and (c) Aggregated throughput of two flows with varying *On* - *Off* duration. Bottleneck bandwidth is uniformly distributed between [1,20] Mbps.

average throughput, shorter delay), and C^2's performance points are distributed in this region. As the average throughput is around 10Mbps and the queuing delay is 10ms, its operating point is stable and very close to the optimal. In comparison, NewReno and Westwood+ have much lower throughput and longer delay. While Cubic has higher throughput than NewReno and Westwood+, its performance shows more deviation than C^2. Cubic's aggressive strategy to increase `cwnd` also brings in much longer queuing delay than C^2.

C. Stability under varying spectrum availability

We show that C^2 keeps performance stable despite interruptions caused by spectrum sensing and PU activity. We set up two flows and plot the the aggregated throughput vs. different length of the *On-Off* periods. We observe that NewReno's throughput drops quickly with decreasing *On* duration. As shown in Sec. III, the reason for this is that conservative AIMD cannot utilize bandwidth efficiently with frequent interruptions. In contrast, C^2 significantly increases the robustness against these interruptions by allowing fast flow convergence to the optimal share of channel bandwidth. Also, the longer *Off* duration only slightly influences the throughput, indicating that the key to performance is not length of path disconnection, but how fast a flow can recover from it.

D. Fairness with loss-based TCP

In this experiment we show the fairness property of C^2 when competing with loss-based TCP. We set up two flows, one using TCP C^2 and one that uses Reno. We vary the RTT of the channel (100ms, 140ms, 180ms) and compute the average throughput of each flow during the transmission with random *On-Off* duration and channel bandwidth. The results are summarized in Table I.

TABLE I
FAIRNESS BEHAVIOR OF COMPETING FLOWS

Protocol \ RTT	100ms	140ms	180ms
TCP C^2	4.9 Mbps	6.2 Mbps	6.8 Mbps
NewReno	4.8 Mbps	2.8 Mbps	1.8 Mbps

We see that C^2 is able to achieve the fair share with NewReno when RTT is 100ms. Though C^2 appears to behave too aggressively, analyzing the variation in throughput reveals that C^2 does not capture the bandwidth from NewReno. Instead, it only utilizes the spare bandwidth. It is only when NewReno is unable to utilize the full bandwidth, usually when the RTT is higher, that C^2 grabs a larger share of bandwidth.

VI. Conclusion

In this paper we (1) mathematically demonstrated that standard AIMD model cannot utilize channel efficiently given frequent connection disruptions in CR cellular networks, and (2) proposed an end-to-end transport protocol C^2 that is able to quickly react to spectrum sensing and channel switching. C^2 increases the channel utilization by allowing a flow to converge directly to its optimal rate based on the estimation of available bandwidth and flow numbers. Simulation results show that C^2 significantly outperforms classical end-to-end TCPs like NewReno, Westwood+ and Cubic, while also remaining fair when co-existing with loss-based TCP variants.

Acknowledgment

This work was supported in part by the U.S. Office of Naval Research under grant number N00014-16-1-2651.

References

[1] Huang, et al. An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *ACM SIGCOMM*, 2013
[2] S. Floyd and T. Henderson. RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm. 1999.
[3] Mascolo, Saverio, et al. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *ACM Mobicom*, 2001
[4] Ha, Sangtae, et al. CUBIC: a new TCP-friendly high-speed TCP variant. In *ACM SIGOPS Operating Systems Review*, vol.42, no.5, 2008.
[5] Sundaresan, Srikanth, et al. Home network or access link? locating last-mile downstream throughput bottlenecks. In *ACM PAM*, 2016.
[6] Wei, David X., et al. FAST TCP: motivation, architecture, algorithms, performance. In *IEEE/ACM Trans. on Networking*, vol.14, no.6, 2006.
[7] Tan, Kun, et al. A Compound TCP Approach for High-speed and Long Distance Networks. In *Proceedings of IEEE INFOCOM*, 2005.
[8] Brakmo, Lawrence S., et al. TCP Vegas: End to end congestion avoidance on a global Internet. In *IEEE J. Sel. Areas Commun*, vol.13, no.8, 1995
[9] Winstein, et al. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *USENIX NSDI*, 2013.
[10] K.R.Chowdhury, M.Di Felice, et al. TCP CRAHN: A Transport Control Protocol for Cognitive Radio Ad Hoc Networks. In *IEEE Trans. on Mobile Computing*, vol. 12, no. 4, pp. 790-803 Apr. 2013.
[11] Al-Ali, et al. TFRC-CR: An equation-based transport protocol for cognitive radio networks. In *Ad Hoc Networks*, 1836-1847,2013
[12] Luo, Changqing, et al. Cross-layer design for TCP performance improvement in cognitive radio networks. In *IEEE Trans. on Vehicular Technology* , 2485-2495, 2010