

Adaptive Transmission Scheme for TCP in Wireless Multi-hop Network

Joon Yeop Lee, Hyunsoon Kim, Woonghee Lee, and Hwangnam Kim

School of Electrical Engineering, Korea University

Seoul, Korea, Republic of

Email: {charon7, gustns2010, tgorevenge, hnkim}@korea.ac.kr

Abstract—Wireless multi-hop networks are scalable in that devices can easily connect to the network. Such scalability is also required in IoT. However, wireless multi-hop network mostly suffers from high probability of transmission failure due to interference, and nodes connected to the network often experience connection loss and subsequent segment loss which node mobility and routing update incur. Due to the fact that the congestion is not a sole reason for segment loss in wireless multi-hop network, the congestion control of TCP should consider momentary link instability, which unnecessarily reduces transmission speed. For this reason, this paper suggests adjustment of slow-start threshold (sssthresh) value which is used for congestion control algorithm. The adjustment algorithm is called adaptive sssthresh decision (ASD) algorithm that is to reduce unnecessary decrease of transmission speed in wireless multi-hop network.

Index Terms—Transmission Control Protocol, Slow-Start Threshold, Congestion Control, Ad-hoc

I. INTRODUCTION

Network architectures in IoT should be different with traditional network architectures. Generally, multi-hop network can be used to provide network connectivity for plenty of nodes in IoT. Such the network environment is quite dynamic in terms of network scale and topology change. Considering this kind of tendency, wireless multi-hop ad-hoc networking is required in IoT to let it have flexibility on connectivity and routing. However, wireless multi-hop network often suffers from data loss [1]. If there are many transmitting nodes causing signal interference, the nodes will suffer from low quality signal, which sometimes results in TCP segment loss and bit errors. Also, power drained nodes often disappear from the network topology, which causes routing change that may trigger TCP retransmission timeout. Moreover, if there are a few nodes forming a topology with sparse density, small movement of a node may cause connection loss, resulting in time delay to find an alternative path to handle packet delivery. The delay will cause the end nodes to understand the phenomenon as a buffer overflow due to excessive amount of traffic and will make the sender reduce burst traffic. Indeed, we experienced such the phenomenon frequently from numerous experiments on ad-hoc networks we designed using the Odroid devices [2]. However, both connection loss and momentary link interference should not be reasons to reduce the congestion window (CWND) size because a path that recovered from the above momentary problems has enough capacity to handle original congestion window size.

Segment loss in wireless multi-hop network are not caused by buffer overflow in many cases. Since buffer overflow is not the only reason for packet dropping, reducing the congestion window size has been unnecessary because the link has enough capacity to handle transmission. Under the condition of wireless transmission having performance problem compared to wired [3], rapid recovery of transmission speed is necessary. However, current TCP end-to-end controls cannot verify whether the reason of segment loss is excessive amount of traffic or momentary link instability [4]. TCP algorithm should have additional solutions for congestion due to reasons other than buffer overflow. Therefore, this paper suggests Adaptive Ssthresh Decision (ASD) algorithm that is designed to reduce the penalty of segment loss without buffer overflow, which is suitable for wireless multi-hop network. The aim of ASD algorithm is listed as follows:

- The proposed algorithm should determine whether the congestion is due to excessive amount of traffic or momentary link instability.
- For momentary link instability, TCP should quickly recover window size to properly utilize the available capacity of network.
- On the other hand, the algorithm should reduce transmission speed to ease excessive amount of traffic.

ASD algorithm adjusts the slow-start threshold (sssthresh) value for adaptive transmission of wireless multi-hop network. When the segment loss happens, TCP decides between congestion avoidance and slow-start phase according to the relation between CWND size and sssthresh value. If the reason is momentary link instability, ASD algorithm will recover CWND size using the slow-start algorithm with the most recent sssthresh value before congestion notification. On the other hand, the ASD algorithm will activate ordinary TCP congestion avoidance mechanism to reduce traffic.

This paper is organized as follows. Details of ASD are described in Section II. To prove the performance of ASD, experimental setup and results are shown in Section III. At the end, Section IV concludes the paper and lists several future works to be done.

II. ASD ALGORITHM EXPLANATION AND IMPLEMENTATION

To improve the communication quality for wireless multi-hop network with TCP, we suggest the ASD algorithm. Sec-

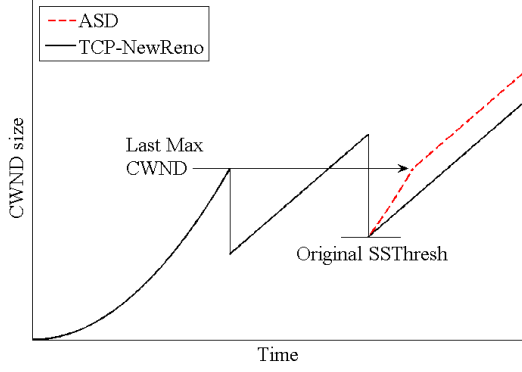


Fig. 1: CWND fluctuation in ASD activated scenario

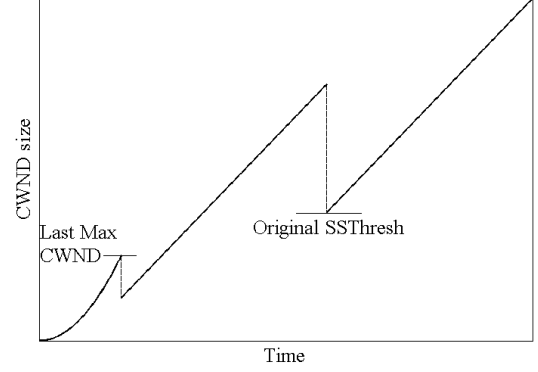


Fig. 2: CWND fluctuation in ASD deactivated scenario

tion II-A and II-B describe details of the ASD algorithm. Then, Section II-C describes details of implementation in Android smartphone which is used for real testbed configuration.

A. ASD algorithm

The main concept of ASD algorithm is adjusting ssthresh to decide CWND increase rate according to the main reason of congestion. If the congestion is due to burst traffic, CWND should increase slowly not to incur frequent congestion later on. On the other hand, CWND should increase fast in order to fully utilize the wireless link which was momentarily instable. For this purpose, ASD algorithm adjusts ssthresh value as shown in Algorithm 1.

Algorithm 1 New ssthresh value decision

```

1: if segment_loss_detected then
2:   function recalculate_ssthresh(current_max_cwnd)
3:     original_ssthresh = current_max_cwnd/2
4:     final = max(original_ssthresh, last_max_cwnd)
5:     last_max_cwnd = current_max_cwnd
6:   return final
7: end function
8: end if

```

When duplicated ACK arrives, ordinary TCP (like NewReno) reduces the CWND size by half. At the same time, the ordinary TCP updates ssthresh value to make the congestion control enter congestion avoidance phase. Since original TCP mechanism changes ssthresh value to the size of half-reduced CWND, congestion avoidance is performed automatically. On the contrary, ASD algorithm changes ssthresh value in a way to make TCP congestion control move to slow-start phase if the congestion is caused by momentary link instability. To choose appropriate ssthresh value, ASD algorithm uses the latest CWND size prior to the previous segment loss event, named *last_max_cwnd*. Because *last_max_cwnd* is the CWND size of successful transmission until the segment loss occurs, ASD algorithm is based on the presumption that *last_max_cwnd* reflects the available

capacity of the network for transmission. When a segment loss occurs, *last_max_cwnd* is compared to the originally calculated ssthresh value, *original_ssthresh*. The bigger value will be the new ssthresh. After that, *last_max_cwnd* will be updated to the latest CWND size prior to the current segment loss for ssthresh decision of future segment loss.

The effectiveness of ASD algorithm is shown in Fig. 1. The figure shows different CWND size fluctuation between original TCP-NewReno and ASD algorithm. The initial rise of CWND is the result of TCP slow-start. And then, it takes the first segment loss which results in halving CWND value. At this moment, original TCP-NewReno updates its current ssthresh, *original_ssthresh*, as the halved CWND in order to enter congestion avoidance phase. Interestingly, ASD algorithm also enters congestion avoidance phase because *last_max_cwnd* is not updated before. Assuming *last_max_cwnd* is 0 at this moment, maximum ssthresh value becomes *original_ssthresh*, resulting in congestion avoidance phase after all. Then *last_max_cwnd* is updated to CWND size before halved, which will be used for the next segment loss event. At the second segment loss event, *last_max_cwnd* becomes higher than *original_ssthresh* value. Because the value of CWND is less than the new ssthresh, the *final* value in Algorithm 1, TCP flow quickly recovers CWND by using the slow-start algorithm rather than congestion avoidance. After the second segment loss, CWND rising curve of ASD algorithm seems like a linear curve. However, it is a quadratic curve that is increased by slow-start. Slow-start algorithm increases CWND per ACK received. Therefore, if there exists large enough CWND size which produces many ACKs, slow-start algorithm raises CWND rapidly with plenty of received ACKs. Since slow-start phase increases CWND size quadratically, the transmission speed of ASD algorithm recovers rapidly, resulting in better performance than original TCP-NewReno.

B. Treating buffer overflow

If CWND always rises rapidly with slow-start algorithm, wireless link can frequently experience buffer overflow due to

non-decreasing multiple data streams. This is why *ASD* algorithm uses the *last_max_cwnd*, instead of the latest CWND prior to the current segment loss event, *current_cwnd*. If *ASD* algorithm uses *current_cwnd*, TCP will always increase CWND using the slow-start algorithm after segment loss event even in the case of congestion due to buffer overflow. In order to prevent such the case, *ASD* algorithm uses the aforementioned *last_max_cwnd*. If *last_max_cwnd* is larger than currently reduced CWND size after congestion, proposed *ASD* algorithm will start slow-start to recover CWND size close to the *last_max_cwnd*. If not, it executes congestion avoidance phase as usual TCP congestion control does. Fig. 2 shows such the scenario that causes *ASD* algorithm to perform congestion avoidance. In Fig. 2, on the second CWND drop, the saved *last_max_cwnd* of the previous segment loss is smaller than *original_ssthresh*. In this case, the value of the *original_ssthresh* will be the final ssthresh value. Because ssthresh is not larger than reduced CWND size, congestion avoidance algorithm is executed.

C. Implementation in android smartphone environment

To prove both applicability and performance of *ASD* algorithm, we applied aforementioned changes on Android mobile phone named Galaxy Nexus. From the process of implementing algorithm on the device, we found that Galaxy Nexus used TCP-CUBIC algorithm as default, which is one of the TCP algorithms implemented in GNU/Linux [5]. Android mobile platform is a Linux based system, and Galaxy Nexus is based on a version of Linux OS that uses TCP-CUBIC as default. TCP-CUBIC is developed in consideration of the wireless communication environment with both high bandwidth and high latency [6]. The congestion avoidance of TCP-CUBIC increases CWND size as cubic [7]. Because *ASD* algorithm adjusts ssthresh value, *ASD* algorithm can be implemented in most of TCP algorithms that use ssthresh value, including TCP-CUBIC.

III. PERFORMANCE EVALUATION

We conducted a set of experiments in a form of multi-hop topology, using various mobile devices with ad-hoc network. Section III-A describes details of the experimental setup to compose the aforementioned topology. In order to emphasize the strength of the proposed algorithm on any condition, we conducted experiments for both momentary link instability case and heavy traffic case. The details are described in Section III-B respectively.

A. Experimental setup

The topology is configured as shown in Fig. 3. Smartphone 1 and 2 are the senders of the TCP transmission, each of which uses different congestion control: TCP-CUBIC and *ASD* algorithm. Server is a receiver of TCP data transmitted by the smartphones. To form a wireless multi-hop network, two Odroid devices acted like wireless intermediary nodes which relay data from smartphones to Server. Imagining that smartphones give feedback to Server through an access

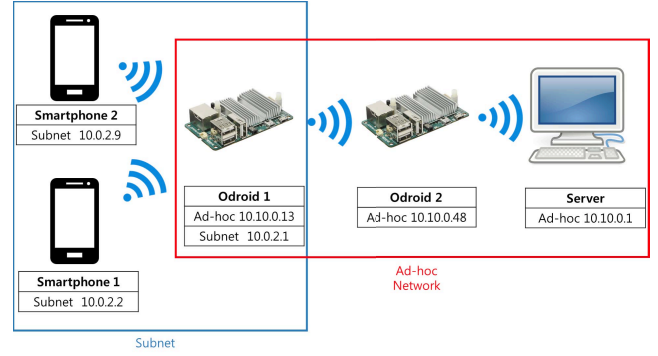


Fig. 3: Experiment topology

network, Odroid 1 acted like an access point (AP) to create a subnet which enables smartphones to access the network through Odroid 1. This 3-hop topology was created to imitate a multi-hop network environment where user-held smartphones provide their status to surrounding transmission nodes which relay the information to a server. Galaxy Nexus devices are used as the aforementioned smartphones. Therefore, all TCP communications in this experimentation are made based on TCP-CUBIC. Since TCP related implementation is held on the device kernel, *ASD* was implemented by modifying TCP-CUBIC congestion control algorithm of Android kernel.

Details of wireless multi-hop network setup are described as follows. The aforementioned Odroid devices were single board computers with heterogeneous multi-processing, and support Ubuntu OS. The hardware version we used is Odroid u3 plus and its specification is almost the same as Galaxy S3. The Odroid uses a Network Interface Controller (NIC) that supports 5GHz bandwidth, resulting that the Server and Odroid devices communicate through less interfered Wi-Fi environment, compared to 2.4GHz. Server is a PC with Linux OS which acted like a gateway of ad-hoc network and assigned IP addresses to Odroid devices to form a topology. After establishing an ad-hoc network composed of one PC and two Odroid devices, connections between smartphones and Odroid 1 are done by adding extra NIC to Odroid 1. The extra NIC was used as AP mode. Since Odroid 1 generated subnet through the extra NIC, Odroid 1 acted as a router and two smartphones accessed to Odroid 1 to act as tethered stations. To complete the proposed wireless multi-hop environment, a script that changes IP table between two NICs was implemented, which delivers packets from AP mode NIC to ad-hoc mode NIC (or vice versa for ACKs) for wireless multi-hop delivery to Server.

With this multi-hop environment, two android smartphones are connected to Odroid 1 with Wi-Fi. Smartphone 2 uses original kernel, and Smartphone 1 uses modified kernel which implemented *ASD*. After establishing connection between smartphones and Odroid 1, smartphones simultaneously transmit TCP data to Server by using iperf which is an application for data transmission.

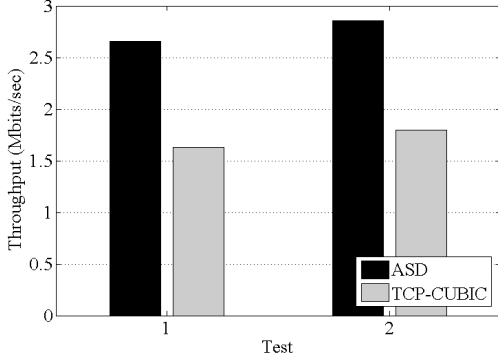
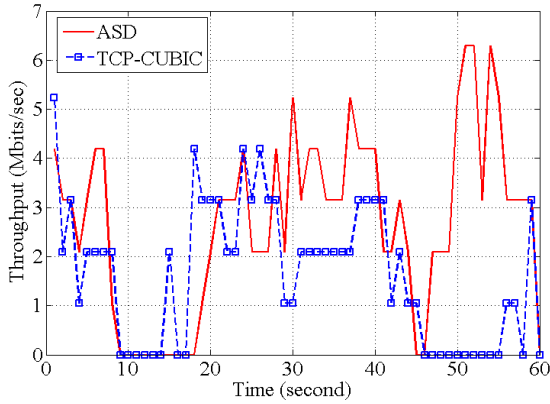
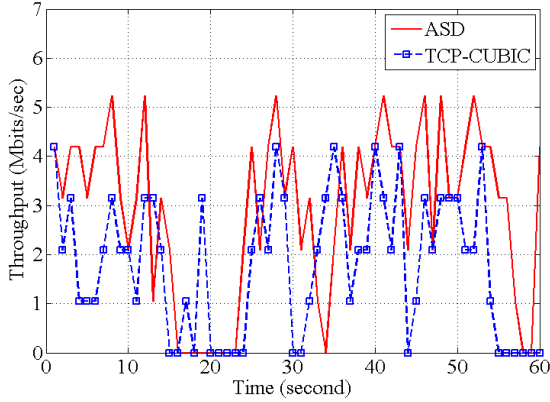


Fig. 4: Average throughput comparison of experiment in momentary link instability environment



(a) Transmission 1



(b) Transmission 2

Fig. 5: Throughput comparison of simultaneous experiment in momentary link instability environment

B. Experimental results

Each smartphone sends TCP data to Server via Odroid 1. Iperf application was used to create TCP flows and data log for throughput measurement. Simultaneous transmissions of two smartphones were done for two times, 60 seconds long

for each trial. We evaluated TCP-CUBIC and ASD at the same time to obtain throughput under fair environment. Fig. 4 shows the average throughput of the experiment. As shown at the figure, the result of ASD algorithm shows higher throughput compared to TCP-CUBIC. Fig. 5 shows the throughput fluctuation of two simultaneous transmissions. The fluctuation of throughput indicates the instability of wireless multi-hop network. Most of experiment time, ASD algorithm shows the higher throughput compared to original TCP-CUBIC. This is because ASD algorithm recovers CWND more quickly most of time by using the slow-start algorithm.

IV. CONCLUSION

This paper presents the ASD algorithm which controls ssthresh value for efficient communications of wireless multi-hop network. This algorithm quickly restores unnecessarily decreased transmission speed and re-saturates throughput quickly. The purpose of our algorithm fits the aim of TCP on utilizing network resources fully. Also, ASD copes with heavy traffic that causes buffer overflow. Although wireless multi-hop network gives more challenging features compared to wired network, ASD overcomes with the difficulties and shows better TCP performance than other ordinary TCP congestion controls.

ASD algorithm uses the previous segment loss event to adjust ssthresh value. However, the value is too static to take account of variety of traffic patterns we can expect from IoT world. Therefore, we will improve ASD by statistically analyzing segment loss events through communication history and surrounding environment of the device. By adaptively changing ssthresh value, ASD algorithm will be more suitable for the personalized service which is the pursuit of IoT environment.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2015R1D1A1A01059151)

REFERENCES

- [1] K. R. Chowdhury, M. Di Felice, and I. F. Akyildiz, "Tcp crahn: A transport control protocol for cognitive radio ad hoc networks," *Mobile Computing, IEEE Transactions on*, vol. 12, no. 4, pp. 790–803, 2013.
- [2] M. an Odroidian and L. Gaming, "Odroid."
- [3] G. Xylomenos, G. C. Polyzos, P. Mähönen, and M. Saaranen, "Tcp performance issues over wireless links," *Communications Magazine, IEEE*, vol. 39, no. 4, pp. 52–58, 2001.
- [4] X. Chen, H. Zhai, J. Wang, and Y. Fang, "Tcp performance over mobile ad hoc networks," *Electrical and Computer Engineering, Canadian Journal of*, vol. 29, no. 1/2, pp. 129–134, 2004.
- [5] S. Arianfar, "Tcps congestion control implementation in linux kernel," in *Proceedings of Seminar on Network Protocols in Operating Systems*, p. 16.
- [6] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.
- [7] L. A. Marrone, A. Barbieri, and M. Robles, "Tcp performance," *Journal of Computer Science & Technology*, vol. 13, 2013.