

# CORE: An Accurate Congestion Detection Method in IEEE 802.11 Wireless Mesh Networks

Jae-Yong Yoo and JongWon Kim  
Gwangju Institute of Science and Technology (GIST)  
{jyyoo, jongwon}@nm.gist.ac.kr

**Abstract**—Although IEEE 802.11 wireless mesh networks (WMNs) provide low-cost extension of wireless Internet access coverage, the wide adoption of WMNs is limited due to the bandwidth scarcity. One of the important mechanisms to fully utilize the scarce bandwidth is congestion control. Especially, *congestion detection*, a module of congestion control, plays an essential role since the source rates should be adapted according to the detected network congestion. Due to the various dynamics of IEEE 802.11 WMNs, wireless congestion control mechanisms usually use complicated combination of various metrics, such as channel-idle time, interface queue length and average link-layer retransmission in intermediate nodes. However, this metric combination requires the sophisticated design of congestion control mechanism and eventually leads to the optimization difficulty. Thus, we propose a simple-yet-effective congestion detection metric, CORE (COngestion Residual timE). CORE measures the expected time of congestion-loss occurrence on the interface queue between IP and MAC layers. We obtain CORE by measuring the residual queue length and the slope of the queue length and then refining them with signal processing schemes (e.g., triangular moving average and linear regression). Moreover CORE provides over-generated rate information that is essential to the congestion control mechanisms. By taking a series of experiments, we verify that CORE accurately predicts the time of congestion-loss occurrence as well as the over-generated rate.

## I. INTRODUCTION

IEEE 802.11-based WMNs (Wireless Mesh Networks) are becoming popular since they can be flexibly deployed to improve the Internet access coverage [1]. Despite their benefits, WMNs also require solving several networking challenges including bandwidth scarcity. To efficiently utilize the precious bandwidth resources, many research contributions address congestion control mechanisms that consists of *congestion detection* and congestion reaction modules [2]–[5].

One of them is the natural extension of TCP into WMNs [2]. [2] proposes an improved TCP by detecting “out-of-order packet delivery events” by measuring end-to-end statistics. Recently, congestion control mechanisms use explicit congestion feedback by placing the congestion detection modules in intermediate nodes for more accurate and timely congestion information. [3] proposes a window-based congestion control protocol that the intermediate nodes explicitly notify the congestion to the source node by using a congestion detection metric combined by “channel-idle time, interface queue length and average link-layer retransmission.” This proliferation of various metrics is to consider the complicated dynamics of IEEE 802.11 WMNs. However, this combination of various

metrics requires the sophisticated design of congestion control mechanisms and eventually hinders the optimized operation.

To unravel the complicity on the congestion control in IEEE 802.11 WMNs, we focus on the original purpose of congestion control. The congestion control should adapt the source sending rates within the bound of network capacity, otherwise (i.e., when the rate is over-generated) queues in the network overflow and the precious bandwidth is wasted by the queue overflow packet loss (congestion loss). Since one of the main purposes of congestion control is to efficiently utilize the precious bandwidth we have to prevent this congestion loss. As a way of this, [3] measures channel-idle time and average link-layer retransmission to guess the bound of network capacity. Unlike these complicated metrics, there are schemes that the congestion detection is based on queue length such as RED (Random Early Detection), backpressure scheduling [4], etc. However, these schemes monitor the queue length at a certain moment and do not focus on the queue length evolution over time. Thus those schemes require queue length threshold to determine whether trigger the congestion reaction or not. To efficiently utilize this queue length threshold based detection schemes, the threshold should accurately determined based on the network available bandwidth. However, the network available bandwidth basically depends on the network topology due to the interference of WMNs and very hard to monitor in WMNs. Rather than using those complicated and difficult metrics, we use the metrics below with the follow-up reasons.

- *Congestion-loss occurrence time*: To timely react to the congestion, we need to know the accurate time that the congestion loss will occur. When intermediate nodes are aware of the future congestion-loss occurrence time, they can timely notify the source nodes to decrease their rates.
- *Over-generated rate*: When source nodes decrease the rates from the congestion notification, they have to determine how much to decrease the rates. Too much decrease can possibly leads to the channel idle that is also bandwidth waste. To accurately decrease not to occur the channel idle and congestion loss, the source nodes should decrease the exact portion of over-generated rate.

To meet the above requirements, we propose a simple-yet-effective congestion detection metric, CORE (COngestion Residual timE) that predicts the time of congestion loss at the interface queue between IP and MAC layers. By focusing on a *queue length evolution over time*, CORE is derived by

measuring the residual queue length and the slope of the queue length and then refining them with signal processing schemes such as triangular moving average and linear regression. Also, we derive the over-generated rate from the refined queue slope and the average packet size that enters the queue. We carefully implement CORE monitoring in a Linux-based WMN node using PaPMo (Packet-accurate Protocol Monitor) [6] that provides accurate queue length monitoring. We also verify the accuracy of both congestion-loss occurrence time and over-generated rate in GIST Sigma WMN testbed.

In the remainder of the paper is organized as follows. Section II presents the details of CORE derivation and example case of CORE monitoring. In Section III, we demonstrate the accuracy of CORE in terms of congestion occurrence time and the over-generated rate. Finally, we conclude in Section IV.

## II. CORE: CONGESTION RESIDUAL TIME

We consider IEEE 802.11 WMNs and each node in this network maintains an interface queue between IP and MAC layers. In this environment, we define CORE as the *expected time of congestion loss* at the interface queue and denote  $c(t)$  as CORE value at time  $t$ . We denote  $b(t)$  as the interface queue length at time  $t$  and  $b'(t)$  as the queue slope at time  $t$ . We monitor  $b$  at every  $\tau$  interval. Then, we can derive  $b'(t)$  as follows.

$$b'(t) = \frac{b(t) - b(t - \tau)}{\tau}. \quad (1)$$

The queue slope  $b'(t)$  represents the speed of the queue length over time. We denote  $b_{max}$  as the maximum queue length, then we define  $c(t)$  as follows.

$$c(t) = \begin{cases} \frac{b_{max} - b(t)}{b'(t)} & \text{if } b'(t) > 0, \\ \infty & \text{if } b'(t) \leq 0. \end{cases} \quad (2)$$

The physical meaning of  $c(t)$  is the time when the first congestion loss occurs after time  $t$ . When the queue length is increasing at time  $t$ ,  $b'(t)$  becomes positive and  $c(t)$  represents the time of queue length becoming  $b_{max}$  assuming that the queue slope is not changing until  $c(t)$ <sup>1</sup>. When the queue length is decreasing, then  $b'(t)$  becomes negative that means the congestion loss will not happen in the future and we let  $c(t)$  becomes  $\infty$ .

### A. Technical Challenges for CORE Monitoring

In order to effectively utilize CORE as a part of congestion control mechanisms, we have two technical challenges.

- *Stability issue of CORE monitoring*: Since most interface drivers are implemented as attachable modules<sup>2</sup> the communication between the interface driver and the Linux network kernel is operated by periodic software tasklet (such as kernel threads). This operation benefits on the flexibility of using various interface drivers but results on the short-time burstiness on the queue length.

<sup>1</sup>Note that the metric of CORE is time.

<sup>2</sup>Most modern operating systems uses module-based OS architecture for the flexibility. Thus this issue happens in the most modern operating systems.

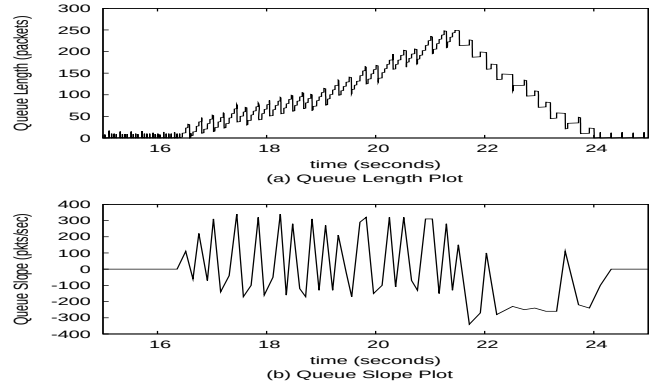


Fig. 1. Behavior of a queue length and queue slope evolution over time in a WMN node. For the queue slope, we use Eq. 1 with applying  $\tau = 0.1$ . We use Atheros chipset with MadWiFi driver on Linux 2.6.26.

Check out the saw-tooth behavior in Fig. 1(a). This saw-tooth behavior also leads to the queue slope fluctuation, as shown in Fig. 1(b). This largely fluctuating queue slope eventually leads to the unstable operation of the congestion control mechanism.

- *Time-lag issue of CORE monitoring*: CORE is basically derived from the queue length monitoring. This monitored queue length results are time-lagged to the actual queue length especially when applying signal processing techniques to mitigate the stability issue presented in Section II-B. This time-lag information should be measured to provide accurate time of congestion indication.

We present solutions for these two challenges in the following subsections.

### B. Stable CORE Monitoring

For the stability challenge of CORE monitoring, we perform a series of signal processing schemes on  $b(t)$ , named as PTL (peak sampling, triangular moving average, and linear regression) to derive non-fluctuating queue slope.

- *Peak Sampling*: We first sample  $b(t)$  to efficiently decrease the number of queue length points by sampling the local high and low peaks. Since the micro-time-scale behavior of the queue length is already diminished by the software tasklet, it is better to ignore the micro-time-scale behavior and only focuses on the macro-time-scale behavior, in this case, peaks. The result after peak sampling is discrete and thus we denote the queue length after peak sampling as  $b_{PS}(i)$  where  $i$  is the  $i$ -th peak.
- *Triangular Moving Average*: We perform triangular moving average on the peak sampled results to get smoothed queue length. Triangular moving average is a class of moving average in that it shows the mean values over a specified number of previous samples. However, the triangular moving average differs from other moving averages in that it is double smoothed (i.e., it is averaged twice). Thus it can filter out the high-frequency part of the samples efficiently. Note that since triangular moving

average uses window-based average, there is a time-lag,  $C_{lag}$ , between the actual and smoothed queue lengths.

- **Linear Regression:** We derive the slope of the queue length by using linear regression with ordinary least-square estimator. The estimate  $\hat{b}'$  we use is as follows.

$$\hat{b}'(i) = \frac{n \sum_{j=i-n}^i b_T(j)t(j) - \sum_{j=i-n}^i b_T(j) \sum_{j=i-n}^i t(j)}{n \sum_{j=i-n}^i b_T(j)^2 - (\sum_{j=i-n}^i b_T(j))^2}, \quad (3)$$

where  $b_T$  is the triangular moving averaged sample,  $n$  is the number of samples and  $t(i)$  is the time of  $i$ -th peak from peak sampling.

By substituting  $b'$  of Eq. (2) to  $\hat{b}$ , we can acquire smoothed CORE. This smoothed CORE can be utilized further by congestion control mechanisms to stabilize them.

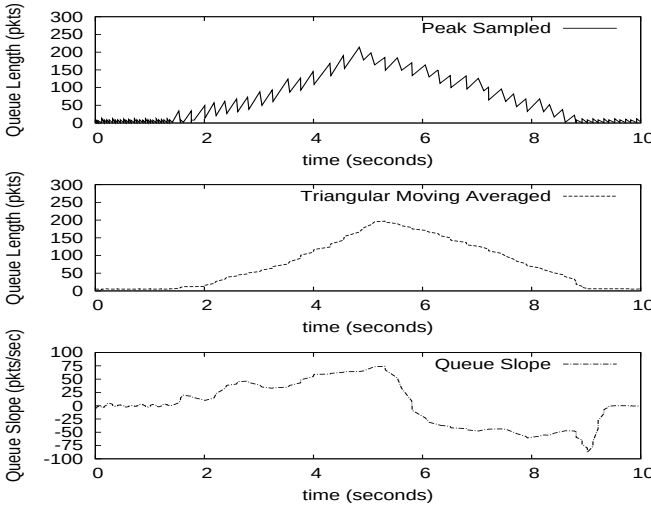


Fig. 2. Queue length results after performing peak sampling. Observe that only local and high peaks are sampled compared to Fig. 1(a).

We now present an example PTL processing results on the queue length of Fig. 1(a). We plot processing results of peak sampling, triangular moving average, and linear regression in Fig. 2 (from top to bottom). In this example, we generate 2.5 Mbps traffic in the 4-hop path of 2 Mbps maximum throughput. The packet size is 1500 bytes that is the maximum MTU (Maximum Transmission Unit). We also set the window size of triangular moving average to 9. We observe that PTL smooths the queue slope compared to the one in Fig. 1(b).

### C. Time-lag Measurement

We now present a method to measure  $C_{lag}$ . This time-lag  $C_{lag}$  is the time distance between the actual queue length and the triangular-moving-averaged queue length. This time distance depends on two factors: the window size of triangular moving average and the average distance between two peaks from the peak sampled results. However, the average distance between the two peaks depends on the system configuration

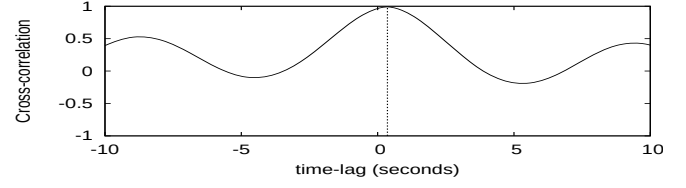


Fig. 3. Cross-correlation between the smoothed queue length and the original queue length. By observing the highest peak near at time 0.36 second, we can point out the time-lag between the two queue lengths.

of handling software tasklets and differs over various systems. Thus, rather than mathematically deriving  $C_{lag}$ , and we use cross-correlation function and measure  $C_{lag}$  offline. This is because once the system has been configured, the  $C_{lag}$  barely changes.

In Fig. 3, we plot the cross-correlation of the triangular-moving-averaged queue length and the original queue length. From this, we can figure out  $C_{lag}$  is about 0.36 seconds. This  $C_{lag}$  should be subtracted from the CORE to predict congestion-loss occurrence time since CORE is derived from the time-lagged queue length.

### D. Remarks on CORE

Based on the CORE definition, we can remark about the congestion due to the queue overflow.

- **Congestion-induced packet loss time prediction:** As mentioned earlier, CORE expects the occurrence time of congestion in future from the queue length history. This information can be fed to the congestion control mechanisms in order to decrease the rate on time by jointly considering the congestion feedback delay.
- **Over-generated rate estimation:** While computing CORE, we use  $b'(t)$ , the slope of the queue length. This slope information basically conveys the difference between the incoming and outgoing rates on the corresponding queue. By measuring packet size of individual packets passing the queue, CORE provides the over-generated rate. This information can be also fed to the congestion control mechanisms so that it can accurately control the rate.

From the above two remarks, CORE benefits the design of congestion control mechanisms to react to the congestion loss on time. Since CORE gives the expected time of congestion, intermediate nodes can determine the exact time to feedback the source node when to notify congestion. Also, rather than measuring available bandwidth (i.e., essential to the congestion control mechanisms), over-generated rate can be fed as an alternate of the available bandwidth information to the congestion control mechanisms. It is because accurately and quickly measuring the available bandwidth of WMNs is absolutely difficult [7].

## III. EVALUATIONS

We evaluate the accuracy of congestion-loss occurrence time prediction and over-generate rate estimation from CORE in the following environments. We setup three nodes: A source

node, an intermediate node and a destination node. We use a static route from the source node to the destination node via the intermediate nodes. We also use the most stable 6 Mbps link modulation rate to exclude the effects on the queue other than congestion (due to the traffic over-generation). As an operating system, we use Linux 2.6.32 and we use MadWiFi to operate the Atheros WiFi interface. To accurately monitor the queue length behavior, we port PaPMo [6] into the nodes. To accurately validate CORE, we use the following metrics.

- **Congestion-loss Time Accuracy:** It presents the time of actual congestion loss and the CORE time. These values tell how CORE can accurately predict congestion on time.
- **Over-generated Rate Accuracy:** We measure the actual over-generated and estimated over-generated rates from CORE. To derive the actual over-generated rate, we generate the traffic into the path that the available bandwidth is previously measured.

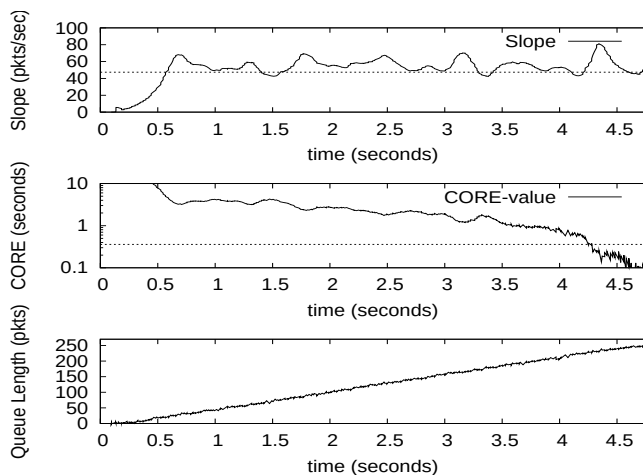


Fig. 4. Overall evaluation of CORE. Three plots are depicted: Queue slope, CORE and original queue length from top. In the queue slope plot, the dotted straight line presents the average queue slope between time 0 to 4.5 seconds. In the CORE plot, the dotted straight line presents  $C_{lag}$  0.36 seconds.

As an example case of computing the two performance metrics, we plot the queue length, the CORE value, and the queue slope over time. We set the UDP traffic with the source rate of 3.1 Mbps and the packet size of 1500 bytes. The end-to-end maximum throughput is around 2.6 Mbps in Fig. 4. From this setting, the over-generated rate becomes 0.5 Mbps. First, we compute the estimated over-generated rate from the averaged queue slope (represented as dotted straight line). Since the average slope is 47 (i.e., 47 packets are over-generated in a second), the estimated over-generated rate is 0.56 Mbps which is close to 0.5 Mbps. Second, we acquire the time of congestion-loss occurrence as 4.65 seconds (see the bottom plot of Fig. 4). Last, the middle plot of Fig. 4 shows that CORE touches  $C_{lag}$  at 4.3 seconds and it indicates that congestion might happen in around 0.3 seconds.

To see the overall behavior over the various over-generated rate, we increase the sending rate with 0.5 Mbps and measure the estimated over-generated rate. Also, we plot the time of

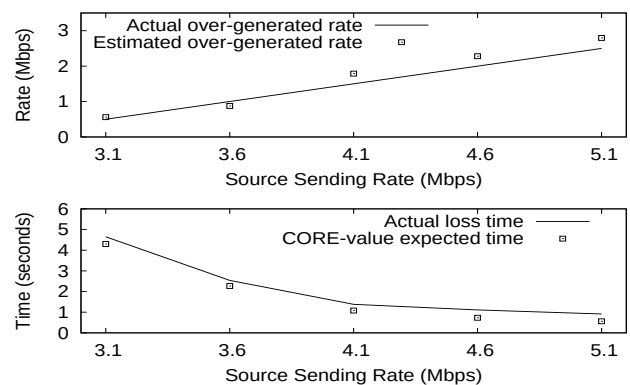


Fig. 5. CORE evaluation by varying the source sending rate from 3.1 Mbps to 5.1 Mbps. Two plots are depicted: Over-generated rate accuracy and congestion loss-occurrence time accuracy from the top.

CORE prediction and the actual congestion-loss occurrence. From Fig. 5, we observe that CORE expects over-generated rate with fair accuracy as well as the time of congestion-loss occurrence. This accurate time prediction of congestion-loss occurrence tells the intermediate nodes when to notify the source nodes for congestion. Also, the notification of congestion is not just a binary information but also the over-generate rate which basically conveys how much the network is overloaded. These two predictions can be usefully exploited by congestion control mechanisms in IEEE 802.11 WMNs.

#### IV. CONCLUSION

In this paper, we propose a new metric, CORE, that reflects the time of congestion-loss occurrence by measuring the queue slope. CORE can inform various congestion control mechanisms about the expected congestion so that we can utilize it as crucial information while determining the rate.

#### V. ACKNOWLEDGEMENT

This work was supported by the GIST Basic Research Project through a grant provided by GIST in 2010 and by Mid-carrier Researcher Program through NRF grant funded by the MEST (No. 2009-0086222).

#### REFERENCES

- [1] W. W. I. F. Akyildiz, X. Wang, "Wireless mesh networks: A survey," 2005, vol. 47, no. 4, pp. 445–487, Jan.
- [2] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response," in *Proc. of ACM MobiHoc*, Sep. 2002.
- [3] Y. Su and T. Gross, "WXCP: Explicit congestion control for wireless multi-hop networks," in *Proc. of IWQoS*, Jun. 2005.
- [4] A. Warrior, S. Janakiraman, S. Ha, and I. Rhee, "DiffQ: Practical differential backlog congestion control for wireless networks," in *Proc. of IEEE INFOCOM*, Apr. 2009.
- [5] Z. Fu, B. Greenstein, X. Meng, and S. Lu, "Design and implementation of a TCP-friendly transport protocol for ad hoc wireless networks," in *Proc. of IEEE ICNP*, Nov. 2002.
- [6] J. Y. Yoo, T. Huehn, and J. Kim, "Active capture of wireless traces: Overcome the lack in protocol analysis," in *Proc. of WinTECH Workshop at Mobicom*, Sep. 2009, pp. 41–48.
- [7] G. Y. M. Y. S. M. G. L. J. Chen, T. Sun, "Adhoc probe: Path capacity probing in wireless ad hoc networks," in *Proc. of WICON*, Jul. 2005.