On the cross-layer impact of TCP ACK thinning on IEEE 802.11 wireless MAC dynamics

Hyogon Kim*, Heejo Lee*, Sangmin Shin* and Inhye Kang†
*Korea University, †University of Seoul

Abstract— **ACK thinning refers to the technique to discard or reduce TCP acknowledgements (ACKs) for the purpose of diverting scarce bandwidth to TCP data traffic. Delayed ACK and ACK filtering fall into the category. It has been shown that under some circumstances the technique is effective to boost the TCP throughput on wireless links, in particular the IEEE 802.11 wireless LAN (WLAN). In this paper, however, we show that ACK thinning backfires under congestion due to its cross-layer impact on the 802.11 MAC dynamics. With the ACK filtering example, we demonstrate the phenomenon and analyze the cause. Based on the analysis, we show how the IEEE 802.11 contention window size control solves the problem. Although only the ACK filtering and Delayed ACK are considered in this paper, any other techniques to save on TCP ACK bandwidth usage share the same fundamental issues.**

I. Introduction

The cumulative property of the TCP acknowledgement (ACK) scheme is originally designed to equip TCP with the robustness against ACK losses [1]. Even if some ACKs are lost, the acknowledgement information, *i.e.*, the receiver buffer availability ("window" size) and the ACK sequence number, can still be conveyed by subsequent surviving ACKs. From the early days of TCP, however, it has been noticed that this property can be exploited to suppress some ACKs to save scarce bandwidth. The Delayed ACK algorithm that suppresses the transmission of an ACK up to 200ms is now a standard feature of TCP [2]. On the wireless LAN (WLAN) context in particular, delayed ACK has been shown to yield higher throughput by giving a larger share of the wireless bandwidth to TCP data traffic [3]–[5]. Above and beyond, it has also been proposed that TCP ACKs be discarded by network elements, independently of delayed ACK, on constrained [6] or asymmetric link settings [7], [8] for improved TCP throughput. The idea is to drop preceding ACKs if the ACKs from the same connection get to be cumulated in the same interface queue. This technique called *ACK filtering* has been briefly considered for satellite channels [9] in the wireless context.

Collectively, the two classes of techniques introduced above are referred to as TCP *ACK thinning*. In this paper, we explore the impact of ACK thinning on the 802.11 MAC [10] dynamics. We focus on the ACK filtering first, since it is more elegant than the Delayed ACK, which we deal with next. We demonstrate that ACK filtering (or any ACK thinning approach for that matter) backfires in face of congestion on the 802.11 link. It turns out that it is due to the cross-layer dynamics between TCP flow control and 802.11 MAC. When we remove some of the ACKs, the consequence is that each surviving ACK gets to acknowledge more bytes. As a result, more bursty transmission takes place. Larger chunks of data ends up in the MAC queue, raising the level of MAC layer contention. Below, we analyze this cross-layer impact and draw the solution approach from it. The contribution of this paper is that it shows

- ACK filtering improves delay, fairness and stability of TCP connections on 802.11 WLANs,
- why ACK filtering can harm TCP throughput under congestion,
- and the harmful TCP-MAC cross-layer interaction is controllable, and the throughput improvement can be retained through contention window modulation.

This paper is organized as follows. In section II, we briefly discuss the concept of ACK filtering. Since ACK filtering has not been considered for the IEEE 802.11 WLAN environment so far in the literature, we also discuss some advantages of applying it to the wireless interface queue. Aside from expected benefits such as small queuing delay, and higher throughput (which we rebut below), previously unknown properties are also uncovered such as stability and fairness. But the focus is on the throughput problem, and it is discussed in section III. In particular, the throughput is shown to be worse than under no ACK thinning in face of congestion. We analyze the cause to be of cross-layer nature between TCP flow control and 802.11 MAC, and draw a solution approach from the analysis, based on the contention window size modulation. In section IV, we briefly discuss the delayed ACK case. We conclude the paper in section V.

II. ACK filtering

A. Algorithm

```
1: if (pure_TCP_ACK($a_f^{in}$))
      /* TCP pure ACK */
2:     for all $Q[k]$, $1 \leq k \leq sizeof(Q)$ /* search MAC queue */
3:         if ($flow(Q[k])$==$f$ and $N_{ACK}(Q[k]) < N_{ACK}(a_f^{in})$)
               /* replaceable ACK found */
4:             if (pure_TCP_ACK($Q[k]$)) $Q[k] = a_f^{in}$; break;
```

Fig. 1. Pseudocode of the ACK filtering algorithm.

The main idea of ACK filtering is simple. When a TCP ACK is about to be queued in the MAC queue, it clears

This work was supported by grant No. R01-2006-000-10510-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

out all preceding ACKs in the same connection that have a smaller ACK number. The removal of TCP ACKs is doable since, in TCP a following ACK always carries more up-to-date acknowledgement information than its predecessor, if not the same. Implementations can vary, but we will follow the schemes of [7], [8], which is described in Fig. 1. In the algorithm, a_f^{in} denotes the incoming packet for flow f, Q is the MAC queue, and $N_{ACK}(\cdot)$ is a function that returns the ACK number. In step (1), the ACK filtering process is triggered by an incoming ACK. `Pure_TCP_ACK` is a test that checks the protocol number field of IP header, the `A` flag of TCP header, and the length of the payload. If the ACK is a pure ACK (carrying no data), the MAC queue is searched for a replaceable old ACK for the same connection (step (2)). If a match is found (step (3)), it is checked if the match is also a pure ACK. If so, the replacement is made in step (4). Notice that if every incoming ACK causes this operation, at most one preceding ACK with a smaller ACK number can be found in the queue, so we break in step (4) as soon as we replace an old ACK. Also notice that the strict inequality in step (3) is important. In case an incoming ACK finds a preceding ACK with the same ACK number in the queue, neither is removed so that TCP's Fast Retransmit algorithm [1] is not affected. If we decided to drop duplicate ACKs, it could prevent the Fast Retransmit from firing, not being able to gather the required 3 duplicate ACKs. Also notice that no additional queuing delay is caused by the replacement since the newly incoming ACK simply replaces the preceding ACK at its found position, *i.e.*, step (4), which prevents itself from affecting the Retransmission Time Out (RTO) calculation on the sender side [1].

Fig. 2 exemplifies the ACK filtering operation with a 7-slot queue. Connections are identified by an alphabet, and subscripts represent TCP ACK numbers. For simplicity we assume all packets in the queue are ACKs. In this example, events occur in the following order (the transmission events are independent of the arrivals):

1) ACK c_2 arrives and replaces the preceding ACK c_1; a_1 is transmitted.
2) ACK e_1 arrives and is queued at the tail since there is no preceding ACK for connection e in the queue; f_1 is transmitted.
3) ACK b_2 arrives and replaces b_1; g_1 is transmitted.
4) ACK b_3 arrives and replaces b_2; c_2 is transmitted.

The ACK filtering algorithm is more elegant than delayed ACK or its enhanced variants that eliminate ACKs more aggressively. This is because the timing and the number of ACK deletions in ACK filtering are automatically determined by the congestion level on the wireless link itself. This is because raised [lowered] congestion level will see more [less] ACKs incoming to the interface queue before the preceding ACK in the same connection is transmitted. As a consequence, we do not need an additional control parameter to determine the intensity of ACK elimination.

B. Advantages of TCP ACK filtering on 802.11

In this section, we explore the performance impacts of ACK filtering *other than* throughput, the main topic we deal with in the next section. For experiments, we simulate in `ns-2` [11] a 802.11b Basic Service Set (BSS) where n terminals contend for bandwidth using Distributed Coordination Function (DCF) [10]. Fig. 3 shows the simulation topology used in the experiments. We assume that all terminals are within mutual sensing range, are equi-distant from the AP, and have 50 (small queue size scenario) or 250 slots (large queue size scenario) in the MAC queue. In practice vendor implementations can wildly vary in queue size, but the 250 slot setting roughly maps to 250ms of queuing under 11Mbps bottleneck speed, which is a rule-of-thumb guideline for Internet router configuration [13].

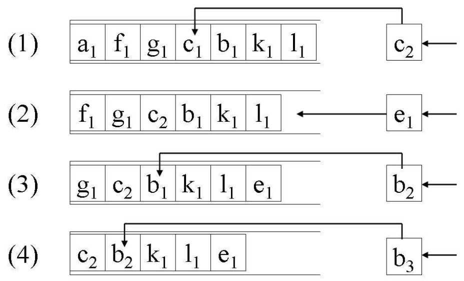

(1)
a_1 f_1 g_1 c_1 b_1 k_1 l_1 c_2
(2)
f_1 g_1 c_2 b_1 k_1 l_1 e_1
(3)
g_1 c_2 b_1 k_1 l_1 e_1 b_2
(4)
c_2 b_2 k_1 l_1 e_1 b_3

Fig. 2. ACK filtering execution example.

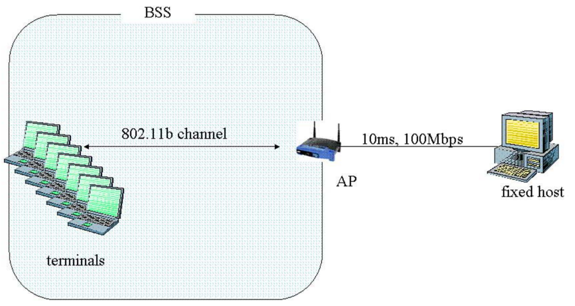
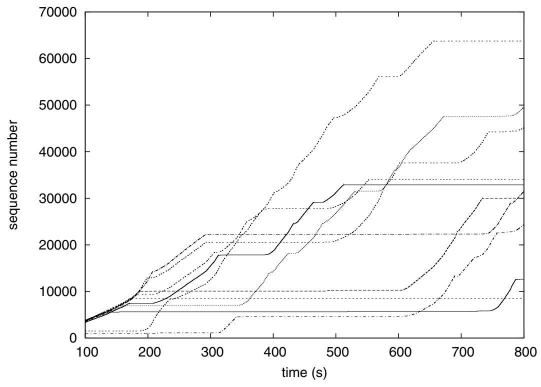


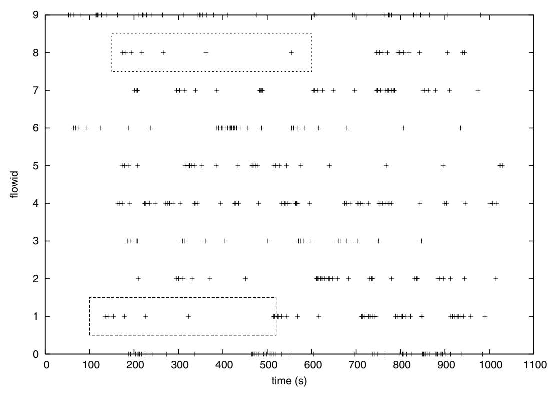
Fig. 3. Simulation topology.

The algorithm exemplified in Fig. 1 and 2 implies that the queueing delay should be smaller with ACK filtering. It is because at most one ACK per connection can be queued in the AP queue, barring the case of duplicate ACKs. Fig. 4 confirms the intuition. We notice that the queue length without ACK filtering converges to the physical size quickly, while it is bounded by the number of TCP connections with ACK filtering for both 50 and 250 slots. As a consequence, the queue length under ACK filtering becomes not only smaller but also independent of the physical queue size configuration.

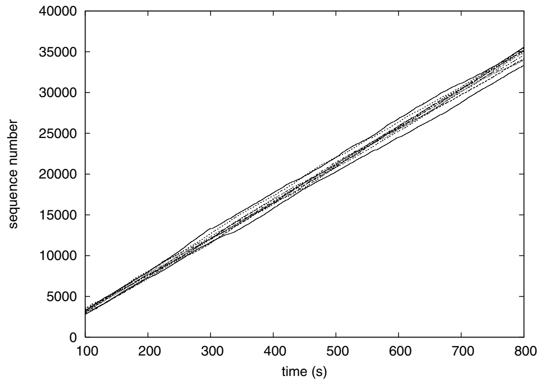
Perhaps a more interesting aspect of ACK filtering is its superior stability and fairness. It turns out that the full queue size possible in ordinary AP queue as shown in Fig. 4 has an unexpected implication in terms of the stability and fairness of the TCP upload traffic. Fig. 5 compares the TCP packet sequence number progression for 10 terminals with and without ACK filtering. Fig. 5(c) is the ACK filtering case, in which all connections share the uplink bandwidth equally. With all other conditions (*i.e.*, RTT, MSS, maximum advertised window size) being equal in the simulation setting,



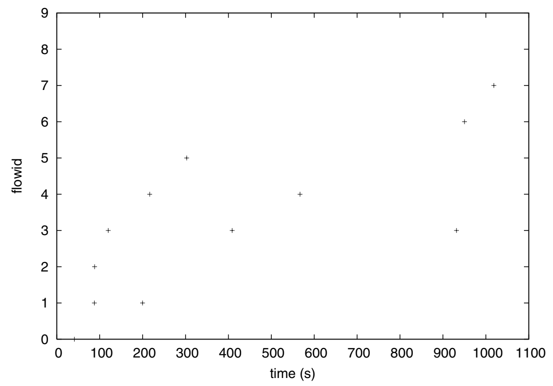
(a) TCP transmission with unmodified queue



(b) TCP timeouts with unmodified queue



(c) TCP transmission with ACK filtering queue



(d) TCP timeouts with ACK filtering queue

Fig. 5. Sequence number progression with and without ACK filtering, 10 connections.

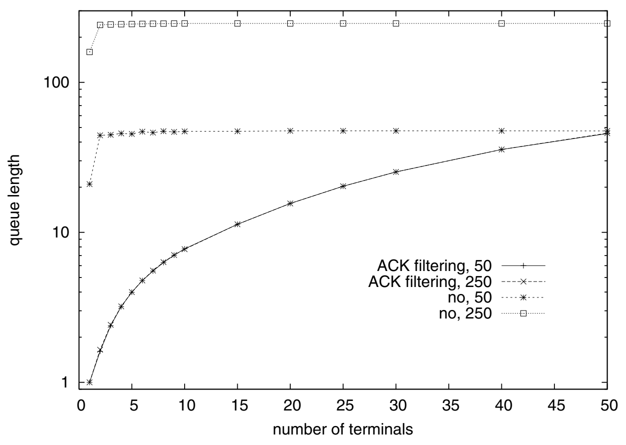


Fig. 4. Queue length with and without ACK filtering, queue size $q = 50$, 250.

the results is only natural since the 802.11 MAC guarantees asymptotically equal channel access probability to all contending parties [12]. Nevertheless, the transmissions are extraordinarily stable. This is because of few TCP timeouts (5(d)) and the controlled growth of the congestion window linked to the wireless channel traffic condition. It must be emphasized again that ACK filtering takes effect only in the situation where ACKs can be accumulated in the AP queue, whose occurrence is related with packet losses and timeouts [14]. The ACK filtering-controlled congestion window size growth is adequate in such a situation, and contributes to the stability that we see in Fig. 5(c). In contrast, Fig. 5(a) shows that TCP connections without ACK control are unstable, mid- to long-term unfair, and involve many long instances of stalled transmission.

The contrast between Fig. 5(a) and (c) is beyond our expectation. In particular, we notice from Fig. 5(a) that once a connection times out, it can take a long time until it resumes transmission. Since the AP queue is almost always fully occupied in the ordinary MAC queue, an arriving ACK to such a full queue is subject to a high drop probability. Although ACK drops do not immediately lead to TCP timeouts and retransmissions (thanks to cumulative ACK), the probability that the ACK drop will lead to the timeout becomes higher for a smaller number of outstanding ACKs for a connection. Therefore, a connection that recently timed out and is retransmitting (with a single data packet from Slow Start) becomes the most vulnerable. It is because for the resulting ACK, there are no ACKs that back up its rear. This is evident from the trace of TCP timeouts in Fig. 5(b). It is visible even at this timescale that the flow 1 and 8 are suffering from RTO exponential backoffs (in boxes), which is only possible when packet drops occur back-to-back [1]. Also the highly concentrated timeouts observed from all flows strongly suggest that once a connection falls into a timeout with unmodified AP queue, it is likely that its subsequent retransmission will be also unsuccessful. In contrast, we confirm that ACK filtering has far fewer timeouts in Fig. 5(d).

III. Cross-layer impact on throughput dynamics

We have seen that ACK filtering has benefits both in queueing delay and stability/fairness. But does it improve throughput as consistently as in wired environment(*i.e.* [7], [8])? In this section, we argue that the answer can be negative, and that it has a cross-layer explanation. First, we show the throughput performance of ACK filtering as compared with that of unmodified MAC queue, in Fig. 6.

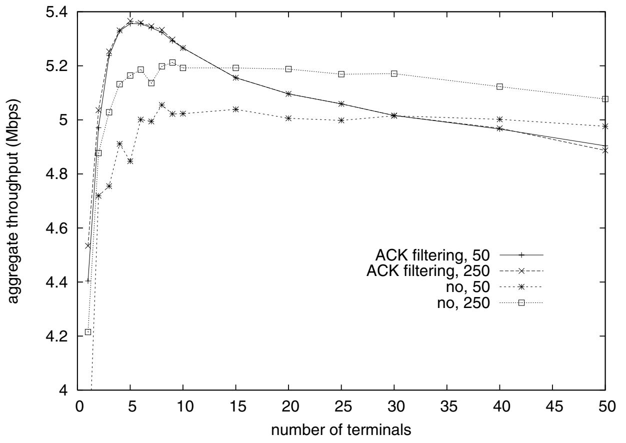


Fig. 6. Throughput with and without ACK filtering.

We observe that the throughput of ACK filtering starts higher for small number of contending terminals, as we would expect. But as the number of terminals increases, the margin tapers off, and eventually it is overturned. For instance, $n = 13$ is the critical point for ACK filtering under the MAC queue size $q = 250$. To tell the conclusion first, this rather unexpected result is due to the fact that the wireline works [7], [8] did not have the issues of 802.11 MAC dynamics and the complex cross-layer interaction between TCP and the MAC.

When we drop redundant ACKs in ACK filtering, each surviving ACK gets to acknowledge more bytes. The sliding window at the TCP sender proceeds accordingly, resulting in a larger burst of data flushed down to the MAC layer. It does not lead to the overall increase of the transmission rate of the TCP connection, but it does imply that *a significant part of the throttle functionality moves down to 802.11 MAC from TCP flow control*. Namely, packets get to be held relatively longer in the interface queue than in the TCP socket queue than before. As a consequence, the average interface queue occupancy increases and so does the MAC transmission attempt probability from a wireless terminal, leading to higher 802.11 MAC contention level. Fig. 7 validates this claim, where ACK filtering results in multiple times the collisions that the unmodified interface queue generates (For clarity we only show $q = 50$ case, since $q = 250$ exhibits the same qualitative behavior.)

Since excessive contention is apparently the throughput inhibiting factor for the ACK filtering queue, if we relax

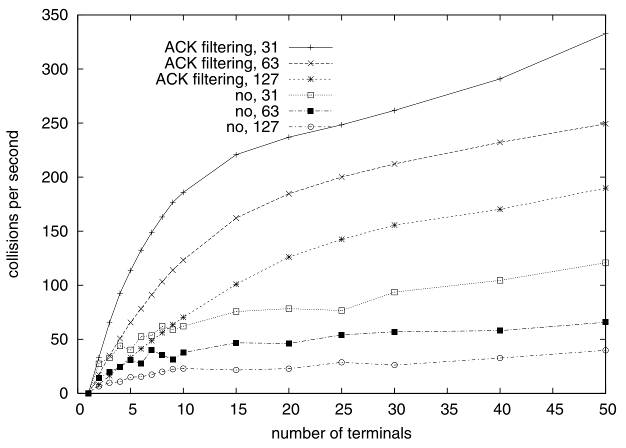


Fig. 7. Number 802.11 collisions, with and without ACK filtering.

the MAC-level competition by using larger contention window size CW_{min}, the throughput gain from redundant ACK removals should be more fully and consistently manifested. Fig. 8 bears out the expectation. The figure shows that with the relented contention level, ACK filtering now outperforms the unmodified MAC queue by a visible margin. But how

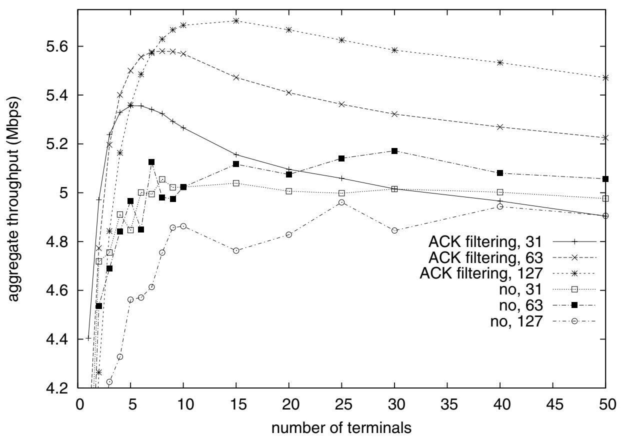


Fig. 8. Larger CW_{min} impact.

can we determine the optimal CW_{min} value for ACK filtering? For one, adaptive methods such as [15], [16] could be brought in to accomplish the objective. For instance, Fig. 9 plots the throughput in each case where the CW_{min} size is doubled upon transmission failure (*i.e.*, collision in our setting) and halved upon success. This method is called the multiplicative-increase multiplicative-decrease (MIMD) mode in the literature [15]. As implied by Fig. 8, the adaptation is more advantageous for ACK filtering than for the unmodified queue. For the unmodified queue, the adaptation even yields slightly worse throughput for large population (Compare with the unmodified system throughput with $q = 250$ in Fig. 6). But the ACK filtering throughput is always over 6Mbps, outperforming the unmodified system by more than 1Mbps in all population regimes, *i.e.*, in excess of 20% improvement. Although designing the optimal CW_{min} adaptation algorithm is beyond the scope of this paper, Fig. 9 is enough to demonstrate the potential of ACK filtering when it is supported by the CW_{min} adaptation. With the advent of the new 802.11e standard [17], we expect the CW_{min} modulation to come in handy for ACK filtering-enabled APs. We remark that ACK filtering can be directly applied to higher-speed 802.11 links, as well as the 802.11b exemplified in this paper.

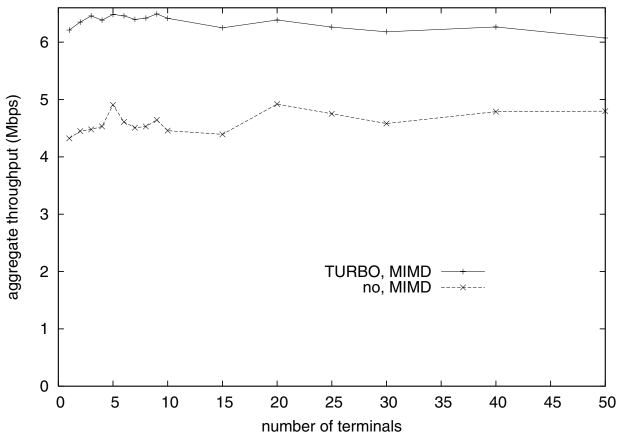


Fig. 9. Throughput comparison under MIMD CW_{min} modulation.

IV. The case of delayed ACK

As delayed ACK techniques also remove ACKs, they should also cause the contention level to rise on the 802.11 MAC layer. Although not as pronounced as in ACK filtering, the effect of relaxing CW_{min} from 31 to 63 in Fig. 10 ("no+DA") is more visible and consistent than in Fig. 8 ("no"). It implies that there is increased contention under delayed ACK, so the contention window modulation has more visible effect.

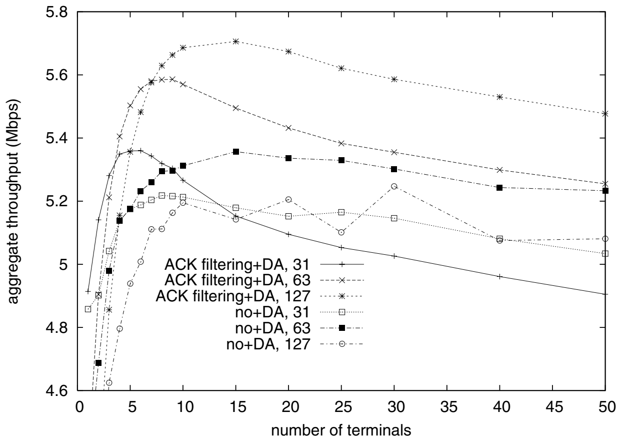


Fig. 10. Effects of CW_{min} on delayed ACK and ACK filtering.

Another observation from the figure is that delayed ACK combined with ACK filtering yields almost identical throughput as ACK filtering alone. It means that among the two, ACK filtering drops ACKs more aggressively especially for large n values, and the delayed ACK drops less than optimal amount of ACKs. And delayed ACK or not, relaxing CW_{min} helps boost ACK filtering, but delayed ACK alone does not benefit as much as ACK filtering. For delayed ACK, $CW_{min} = 63$ is better than $CW_{min} = 31$. However, with $CW_{min} = 127$, the throughput becomes even less than with $CW_{min} = 31$. So Fig. 10 suggests that for delayed ACK there is not enough contention to justify $CW_{min} = 127$. This is contrast to ACK filtering, where the comparison of the peak throughput in Fig. 10 and 9 suggests that even larger contention window size is necessary for ACK filtering.

Although not shown for space, it should be noted that there is also a qualitative difference between ACK filtering and delayed ACK: unlike ACK filtering, delayed ACK does not solve the instability problem [18]. In essence, it exhibits the same phenomenon that we saw in Fig. 5(a). The qualitative difference between delayed ACK and ACK filtering does not stem from the difference in the aggressiveness in ACK dropping. Rather it results from how ACK filtering controls the drops. ACK filtering reacts to the wireless link condition, and automatically regulates the dropping rate. In contrast, (enhanced) delayed ACK is arbitrary, simply removing every k^{th} ACK. It cannot, and is not designed to, adapt to the wireless link condition. So it always harbors the potential to manifest the instability problem.

V. Conclusion

This paper is the first to show that TCP ACK thinning induces a complex cross-layer dynamics, which makes the throughput improvement nontrivial on 802.11 WLANs. The contention level on MAC layer rises, so with increased population, the throughput improvement is offset. This issue can be resolved by modulating the 802.11 contention window size parameter, which is possible in the new 802.11e standard [17]. The modification enables ACK thinning queue to consistently outperform the unmodified queue by a significant margin. The analytical modeling of the cross-layer dynamics between the TCP flow control and 802.11 MAC is currently under way.

References

[1] R. Stevens, *TCP/IP Illustrated V.1*,Addison-Wesley, 1994.
[2] R. Braden, "Requirements for Internet Hosts," RFC 1122, October 1989.
[3] D. Miorandi, E. Altman, "On the Effect of Feedback Traffic in IEEE 802.11b WLANs," *Research Report 4908*, INRIA, August 2003.
[4] A. Kherani, R. Shorey, "Performance Improvement of TCP with delayed ACKs in IEEE 802.11 Wireless LANs," in proceedings of IEEE WCNC, 2004.
[5] E. Altman, T. Jimenez, "Novel delayed ACK techniques for improving TCP performance in multihop wireless networks," IFIP-TC6 8th International Conference on Personal Wireless Communications, 2003.
[6] P. Karn, "Dropping TCP ACKs," Mail to the end-to-end mailing list, Feb. 1996.
[7] I. Tam, D. Jinsong, and W. Wang, "Improving TCP Performance Over Asymmetric Networks," *ACM Computer Communication Review*, July 2000.
[8] H. Balakrishnan, V. P. Padmanabhan, and R. Katz, "The Effects of Asymmetry on TCP Performance," in proceedings of ACM Mobicom, 1997.
[9] V. N. Padmanabhan, H. Balakrishnan, K. Sklower, E. Amir, and R. Katz, "Networking Using Direct Broadcast Satellite," in proceedings of Workshop on Satellite-Based Information Systems, November 1996.
[10] ANSI/IEEE, "802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 1997.
[11] The Network Simulator NS-2, available at http://www.isi.edu/nsnam/ns/.
[12] H. Kim, S. Yun, and I. Kang, "Resovling 802.11 performance anomaly through service differentiation," *IEEE Communications Letters*, July 2005.
[13] R. Bush and D. Meyer, "Some Internet Architectural Guidelines and Philosophy," *RFC 3439*.

[14] C. E. Koksal, H. Kassab, and H. Balakrishnan, “An Analysis of Short-Term Fairness in Wireless Media Access Protocols,” in proceedings of ACM SIGMETRICS, 2000.
[15] Q. Pang, S. C. Liu, J. Lee, S. H. Chan, “A TCP-like Adaptive Contention Window Scheme for WLAN,” in proceedings of IEEE ICC 2004.
[16] Z. Haas, and J. Deng, “On Optimizing the Backoff Interval for Random Access Schemes,” *IEEE Transactions on Communications*, 51(12), Dec. 2003.
[17] ANSI/IEEE, “802.11e: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Enhancements for Quality of Serivce (QoS),” Nov. 2002.
[18] H. Kim, S. Shin, and I. Kang, “On the performance and cross-layer dynamics of TCP ACK Filtering over IEEE 802.11 links,” Korea University Techreport, available at http://widen.korea.ac.kr/ACKfiltering.pdf.