IEICE TRANS. COMMUN., VOL.E90–B, NO.2 FEBRUARY 2007

LETTER

On the Cross-Layer Impact of TCP ACK Thinning on IEEE 802.11 Wireless MAC Dynamics*

Hyogon KIM[†a)], ***Nonmember***, **Heejo LEE**[†b)], ***Member***, ***and*** **Sangmin SHIN**[†], ***Nonmember***

SUMMARY ACK thinning refers to the technique to discard or reduce TCP acknowledgements (ACKs) for the purpose of diverting scarce bandwidth to TCP data traffic. 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 letter, 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.

key words: *IEEE 802.11, TCP ACK filtering, ACK thinning, cross-layer interaction*

1. TCP Acknowledgement (ACK) Thinning

TCP ACK filtering [1] and Delayed ACK [2] have been shown to yield higher throughput in wireless links by appropriating some wireless bandwidth for TCP ACKs to use it on the TCP data traffic [3]–[6]. In this letter, we explore the impact of these *ACK thinning* techniques on the 802.11 MAC [7] dynamics. We demonstrate that ACK filtering (or any ACK thinning approach for that matter) backfires in the event of congestion on the 802.11 link. It turns out that it is due to the cross-layer dynamics between the TCP flow control and the 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 end 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, using the ACK filtering example. We investigate the Delayed ACK in a separate study [8].

1.1 Algorithm

The general idea of ACK filtering is simple. When a TCP ACK is about to be queued in the MAC queue, it clears out all preceding ACKs in the same connection that have a smaller ACK number. The removal of TCP ACKs is feasible 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 [9], [10], which is described in Fig. 1. Note that for our scheme, the algorithm runs at the 802.11 access point (AP), dealing with the TCP ACKs in the *downlink* direction. Thus the scheme boosts the data traffic performance in the uplink direction. This is in contrast to the existing TCP boosting methods such as SNOOP [11], that aim at boosting downlink data traffic by conditioning the TCP ACKs in *uplink direction*, towards the server residing in the wired network. With P2P replacing Web as the killer application, wireless stations may well be the "server," driving traffic uplink. Thus it behooves us to consider TCP boosting in uplink direction.

```
1: if (pure_TCP_ACK(a_f^in))
   /* TCP pure ACK */
2:     for all Q[k], 1 ≤ k ≤ 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 running on the AP.

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), `Pure_TCP_ACK` 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)). We do not fix a search method for step (2), but one way to reduce the complexity is to use hashing on the TCP connection identifier (source and destination IP and port number). 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 [12] 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

Manuscript received June 20, 2006.
Manuscript revised September 2, 2006.

[†]The authors are with the Department of Computer Science and Engineering, Korea University, Seoul 136-713, South Korea.

[*]This research is supported by the Ubiquitous Autonomic Computing and Network Project, the Ministry of Information and Communication (MIC) 21st Century Frontier R&D Program in Korea.

a) E-mail: hyogon@korea.ac.kr
b) E-mail: heejo@korea.ac.kr

DOI: 10.1093/ietcom/e90–b.2.412

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 the retransmission timer from expiring on the sender side [12].

1.2 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` simulator [13] a 802.11b Basic Service Set (BSS) where n stations contend for bandwidth using Distributed Coordination Function (DCF) [7]. The stations are backlogged, and use TCP Reno to upload the data through the AP. In reality, a station could concurrently run a few flows, but for simplicity, we assume that each station originates a single connection. We assume that all stations are within mutual sensing range, and are equi-distant from the AP, thus there is no hidden station problem. The wireless channel is assumed to be in good condition, so that we can focus on the impact of the TCP flow control on the 802.11 MAC protocol. The stations have 50 or 250 slots in the MAC queue. In particular, the 250 slot setting roughly maps to 250 ms of queuing under 11 Mbps bottleneck speed, which is a rule-of-thumb guideline for Internet router configuration [14].

The algorithm exemplified in Fig. 1 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. Figure 2 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. 2 has an unexpected repercussion in terms of the stability and fairness of the TCP upload traffic. Figure 3 compares the TCP packet sequence number progression for 10 stations with and without ACK filtering. Figure 3(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, the results is only natural since the 802.11 MAC guarantees asymptotically equal channel access probability to all contending parties. Nevertheless, the transmissions are extraordinarily stable. This is because of few TCP timeouts (Fig. 3(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 [15]. In contrast, Fig. 3(a) shows that TCP connections without ACK control are unstable, mid- to long-term unfair, and involve many long instances of stalled transmission.

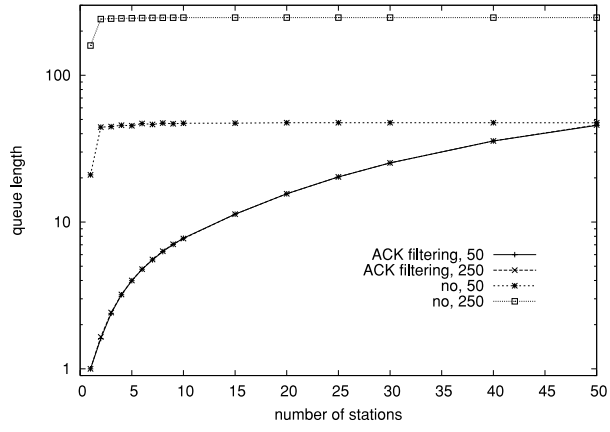


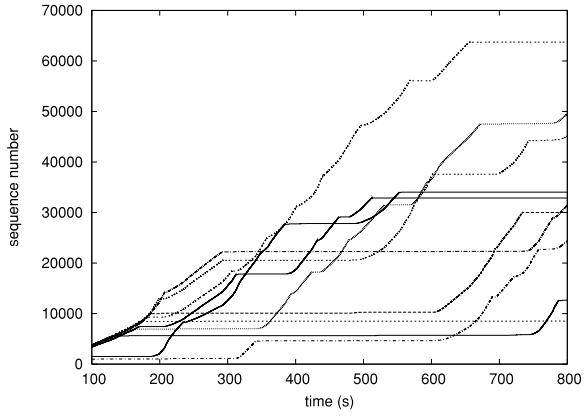
Fig. 2 Queue length with and without ACK filtering, when queue size q = 50 or 250.

The contrast between Fig. 3(a) and (c) is beyond our expectation. In particular, we notice from Fig. 3(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 the 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. 3(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 [12]. Also the highly concentrated timeouts observed from all flows strongly suggest that once a connection falls into a timeout with an 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. 3(d).

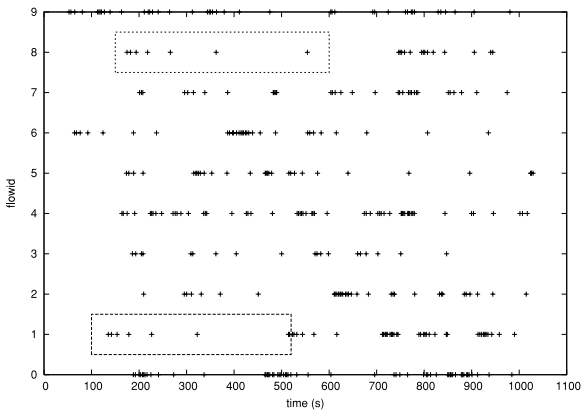
2. Cross-Layer Impact on Throughput Dynamics

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.*[9], [10])? 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. 4.

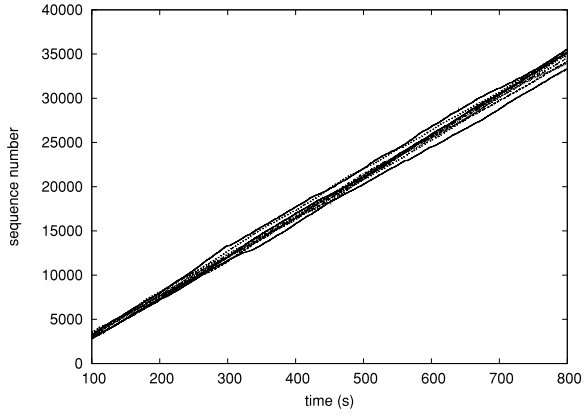
We observe that the throughput of ACK filtering starts higher for small number of contending stations, as we would expect. But as the number of stations increases, the perfor-



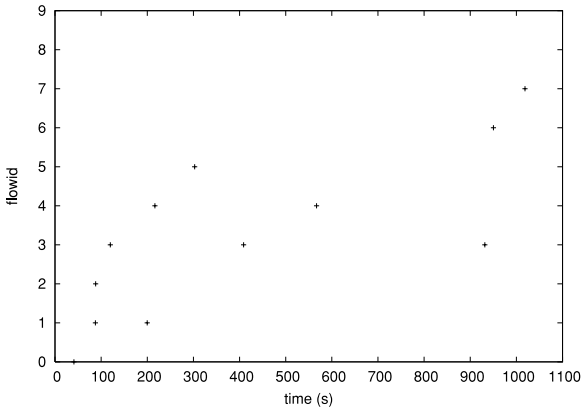
(a) TCP transmission with unmodified queue.



(b) TCP timeouts with unmodified queue.



(c) TCP transmission with ACK filtering queue.



(d) TCP timeouts with ACK filtering queue.

Fig. 3 Sequence number progression with and without ACK filtering, 10 connections.

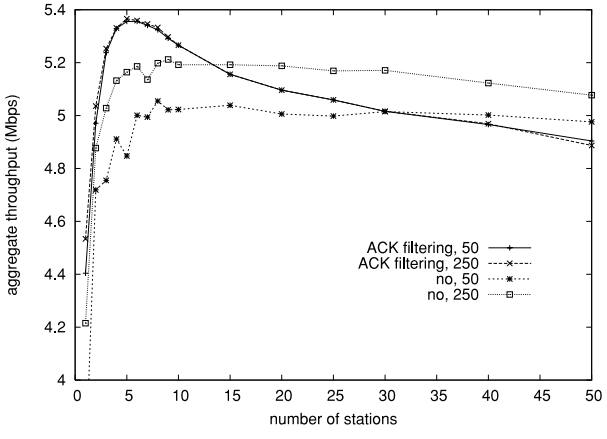


Fig. 4 Throughput with and without ACK filtering.

mance is overturned. We remark that the wireline works [9], [10] do not have this problem. The rather unexpected result in Fig. 4 suggests that there should be a complex cross-layer interaction between TCP flow control and the 802.11 MAC dynamics.

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 station, leading to higher 802.11 MAC contention level. Figure 5 validates this claim, where ACK filtering results in multiple times the collisions that the unmodified interface queue generates for $q = 50$ ($q = 250$ exhibits the same qualitative behavior).

Since excessive contention is apparently the throughput inhibiting factor for the ACK filtering queue, if we relax the MAC-level competition by using larger minimum contention window size (the CW_{min} parameter of the IEEE 802.11 MAC), the throughput gain from redundant ACK removals should be more fully and consistently manifested. Figure 6 proves 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 can we determine the optimal CW_{min} value for ACK filtering? For one, adaptive methods such as [16] could be brought in to accomplish the objective. For instance, Fig. 7 plots the throughput in each case where the

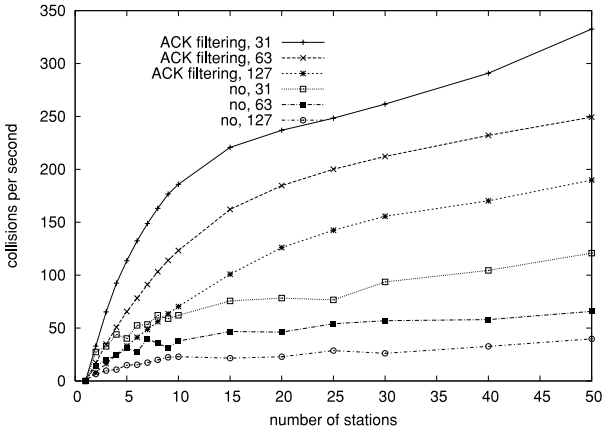


Fig. 5 Number 802.11 collisions, with and without ACK filtering.

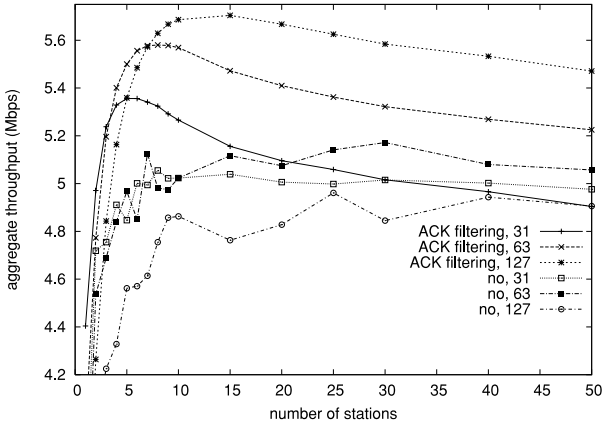


Fig. 6 Larger CW_{min} impact on throughput.

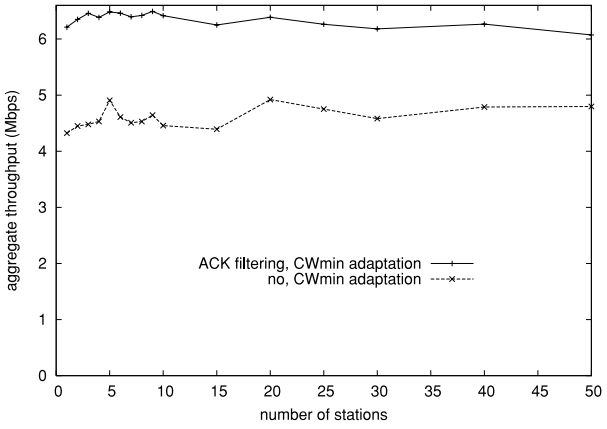


Fig. 7 Throughput comparison under MIMD CW_{min} adaptation.

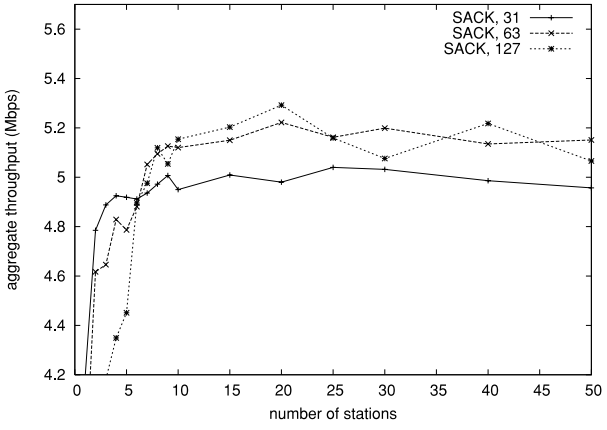


Fig. 8 Throughput TCP SACK under CW_{min} adaptation.

CW_{min} size is conditioned upon transmission failure/success [16]. As implied by Fig. 6, 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. 4). But the ACK filtering throughput is always over 6 Mbps, outperforming the unmodified system by more than 1 Mbps 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 letter, Fig. 7 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, we expect the CW_{min} modulation to come in handy for ACK filtering-enabled APs.

Finally, we remark that the use of TCP Selective Acknowledgement (SACK) option cannot be an alternative to ACK filtering, because it hardly reduces the absolute number of TCP ACK packets. As a matter of fact, Fig. 8 shows that the improvement through the use of TCP SACK is only marginal over the throughput TCP Reno without the ACK filtering that we saw above, although the CW_{min} increase does not as severely degrades throughput.

References

[1] P. Karn, "Dropping TCP ACKs," Mail to the end-to-end mailing list, Feb. 1996.
[2] R. Braden, "Requirements for Internet hosts," RFC 1122, Oct. 1989.
[3] D. Miorandi and E. Altman, "On the effect of feedback traffic in IEEE 802.11b WLANs," Research Report 4908, INRIA, Aug. 2003.
[4] A. Kherani and R. Shorey, "Performance improvement of TCP with delayed ACKs in IEEE 802.11 wireless LANs," Proc. IEEE WCNC, pp.1703–1708, 2004.
[5] E. Altman and T. Jimenez, "Novel delayed ACK techniques for improving TCP performance in multihop wireless networks," IFIP-TC6 8th International Conference on Personal Wireless Communications, pp.237–250, 2003.
[6] V.N. Padmanabhan, H. Balakrishnan, K. Sklower, E. Amir, and R. Katz, "Networking using direct broadcast satellite," Proc. Workshop on Satellite-Based Information Systems, Nov. 1996.
[7] ANSI/IEEE, "802.11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," 1997.
[8] H. Kim, S. Shin, and I. Kang, "On the performance and cross-layer dynamics of TCP ACK Filtering over IEEE 802.11 links," Korea Univ. Techreport, available at http://widen.korea.ac.kr/ACKfiltering.pdf
[9] I. Tam, D. Jinsong, and W. Wang, "Improving TCP performance over asymmetric networks," ACM Comput. Commun. Rev., vol.30, pp.45–54, July 2000.
[10] H. Balakrishnan, V.P. Padmanabhan, and R. Katz, "The effects of

asymmetry on TCP performance," Proc. ACM Mobicom, pp.77–89, 1997.
[11] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," Proc. ACM SIGCOMM, pp.756–769, 1996.
[12] R. Stevens, TCP/IP Illustrated V.1, Addison-Wesley, 1994.
[13] The Network Simulator NS-2, available at http://www.isi.edu/nsnam/ns
[14] R. Bush and D. Meyer, "Some Internet architectural guidelines and philosophy," RFC 3439, 2002.
[15] C.E. Koksal, H. Kassab, and H. Balakrishnan, "An analysis of short-term fairness in wireless media access protocols," Proc. ACM SIGMETRICS, pp.118–119, 2000.
[16] Q. Pang, S.C. Liu, J. Lee, and S.H. Chan, "A TCP-like adaptive contention window scheme for WLAN," Proc. IEEE ICC, pp.3723–3727, 2004.