Issues and Improvements in TCP Performance over Multihop Wireless Networks

Beizhong Chen, Ivan Marsic
Dept. of Electrical & Computer Eng. and CAIP Center
Rutgers University, Piscataway, NJ 08854
bzchen@caip.rutgers.edu, marsic@caip.rutgers.edu

Ray Miller
Bell Labs
Murray Hill, NJ 07974
rbmiller@alcatel-lucent.com

Abstract

TCP performance in contention-based multi-hop wireless networks is shaped by two main factors, which are unlike the wired network case. First, the maximum throughput for a given topology and flow pattern is reached at an optimal congestion window. We provide an improved analysis of this effect. Second, the control traffic uses higher proportion of the channel bandwidth than in the wired case. Our results show that the much smaller TCP ACK packets consume channel resource comparable to the much longer data packets, over high-speed connections. Motivated by this understanding, we reformulate and enhance an existing idea for improving TCP performance by further lowering the number of control packets. Extensive simulations show that our strategy increases the TCP throughput up to 205% compared to the regular TCP, in long-hop wireless networks and 35% in a complex network.

1. Introduction

The TCP (Transport Control Protocol) suffers significant performance deterioration in multi-hop wireless network where the key reason for packet loss is different from that in a wired network. The first one is the link-layer contention [6]. In a contention-based wireless network, a transmitting node prevents its neighbors from transmitting/receiving. Moreover, the well-known hidden/exposed problem causes collisions and degrades the channel utilization. To alleviate this problem, 802.11 adopts RTS/CTS and virtual carrier sense mechanism which solves the hidden/exposed problem when all nodes can hear each other, but not when some nodes are out of the hearing range. This is because the interference range is much larger than the actual transmission range. Note that even without collision and channel errors, packets can still be dropped if the MAC layer retries exceed a given limit. In such a contention-based network, Fu et al. [6] found that there is an optimal value of the congestion window (*cwnd*) which gives the maximum throughput. They showed that the optimal *cwnd* is $n/4$, where n is the total number of hops. Unlike this, our analysis shows that the optimal *cwnd* size should be between $n/3$ and $n/2$, after considering the capture effect.

The second factor is the much higher overhead introduced by lower layers. In this paper, our analysis and simulations show that for high-speed connections, the much smaller TCP ACK packets use channel resource comparable to the much longer data packets, which inspires us to reduce maximally the excessive control packets. Based on this, we propose a new delay ACK algorithm, which can enhance TCP performance significantly, up to 205% improvement over regular TCP in long-hop wireless networks.

We use TCP RenoNew [5] as the regular TCP. In TCP, after receiving a data packet, the receiver replies with an ACK packet which introduced minor overhead in wired network. The regular TCP also provides a delayed ACK option (TCP-DA) which generates a cumulative ACK for every two in-order packets that arrive within a given period [10]. If the receiver does not receive the second packet in this period, denoted as ACK-delay timeout, it sends an ACK without waiting longer. In our proposed approach, the receiver always waits until this timeout event occurs to generate an ACK, no matter how many in-order packets it receives during the timeout period, assuming other conditions introduced in Section 5 are not satisfied.

Similar as in [8], we define *delay window* as the number of data packets that generate one ACK packet. We also define *delay window limit* as the maximum delay window.

We do not address the mobility-related issues in this paper, but this is part of our ongoing project. Besides, we expect the long-hop wireless network to be more likely deployed in static sensor networks.

2. Related work

Fu et al. [6] show that the main factor affecting the TCP performance in multi-hop wireless networks is the link-layer contention, rather than buffer overflow, and there exists an optimal *cwnd* size. K. Chen et al. [3] use the number of hops to determine the optimal *cwnd* window.

Altman et al. [1] found that using different delay-ACK strategies results in TCP performance gains.

Yuki et al. [11] proposed to combine TCP DATA and ACK packets into a single packet. Singh et al. [2] proposed a dynamical method to reduce the number of ACKs. Oliveira and Braun [4] proposed TCP-DAA to combine the idea of restricting *cwnd* to its optimal value while reducing the number of ACK packets. They set the delay window limit as ≤ 4. Chen et al. [8] proposed a method called TCP-DCA to select different delay window limit based on the number of hops.

3. Optimal congestion window

In 802.11, interference range is bigger than its transmission range. This introduces the hidden node problem. For example, in figure 1, the node 4 and node 6 cannot hear each other, but if they both send packets to node 5 simultaneously, the signals will collide. To alleviate this problem, 802.11 introduces the RTS/CTS mechanism. In addition, every node backs off for an exponentially increasing random interval after overhearing other nodes' transmission.

Figure 1 shows a typical chain topology in which any two adjacent nodes reside 200 meters apart. The transmission range is about 250m and the interference range is about 550m. Without considering capture effect, in a group of 4 subsequent hops, only one can be active at a time. Based on this, previous work [6] claimed that TCP achieves the optimal throughput for *cwnd* equal *n*/4, where *n* is the number of hops. However, capture effect also affects the transmission range. When a receiver receives signals from two different senders, the stronger signal could completely suppress the weaker one and therefore can be correctly decoded by the receiver, if the difference between these two signals exceeds a threshold (typically 10 *dB*). In the two-ray ground reflection model, which gives more accurate prediction at a long distance than the free space model [7], the received power, P_r, at distance d is calculated as

$$P_r(d) = P_t G_t G_r h_t^2 h_r^2 / d^4 L \qquad (1)$$

where P_t is transmit power, G_t and G_t are transmit and receive antennas gain, respectively, h_t and h_r are the heights of the transmit and receive antennas, respectively. L is constant. Assuming P_1 and P_2 are the signal energies received from two nodes, the following formula is used to determine whether the capture effect occurs:

$$10 \times (lg\,(P_1) - lg\,(P_2)) > 10 \qquad (2)$$

We use this formula to calculate the point where the difference of signal energy level is 10 *dB* and have:

$$((400 - x) / (200 + x))^4 = 10$$
$$x = 15.82$$

The above calculation shows that nodes 1 and 2 can communicate at the same time while nodes 4 and 5 are communicating. This means that the optimal *cwnd* size is *n*/3 at least, instead of *n*/4 as in [4][6].

Besides, there is a slight possibility that node 1 can transmit to node 2 while node 4 is transmitting to node 3. For example, nodes 2 and 3 send RTS at the same time to nodes 1 and 4, respectively. Because of the capture effect, the CTS from node 1 to 2 and from node 4 to 3, respectively, can be correctly decoded and the transmission can succeed. But in most cases, the transmission between nodes 1 and 2 would interfere with that between nodes 3 and 4 (for example, when node 2 is transmitting, node 3 cannot receive), which means that the optimal *cwnd* must be less than *n*/2.

Combined with the analysis above, we conclude that the optimal *cwnd* should be between *n*/3 and *n*/2, slightly higher than *n*/3. This accurately explains why optimal *cwnd* is 3 in the 7-hop scenario, as claimed by [4][6], instead of 2, and optimal *cwnd* of 10 hops is 3~4 [4], rather than 2.5.

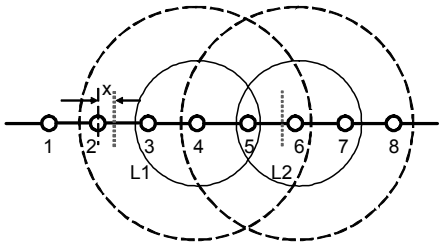

Figure 1. Chain topology. The solid circle is the valid transmission range. The dotted circle denotes the interference range.

4. ACK impact on channel utilization

A typical TCP ACK packet is 40 bytes long. (Note that the TCP ACK is different from the MAC ACK.) Figure 2 shows the timing diagram for transmissions that include RTS/CTS. For simplicity, we ignore other overheads, e.g., PHY header, DIFS, etc., which would support our conclusion even stronger. Assuming the basic rate is 1Mbps (used for RTS/CTS/ACK_{mac}) and the transmission rate for data is 11 Mbps, the overhead transmission time is 3SIFS+RTS+CTS+ACK_{mac} = 414μs. After including this overhead, the transmission times for 40-byte ACK_{tcp} packets and 1040-byte data packets are 443μs and 1170μs, respectively. They are on the same order of magnitude. Moreover, because of the exponential backoff mechanism, the ACK_{tcp} packets have a greater impact than proportional to their transmission time. When the transmission rate becomes higher, the transmission time for data shrinks and therefore the overhead brought by RTS/CTS is greater.

To clarify this conclusion, we compare two cases where the packet sizes are 1040 bytes and 40 bytes

Table 1. Maximum achieved number of transmitted packets vs. packet size, transmission rate 11Mbps

Num of hops	6	8	10	12
Num of 1040 byte pkts	4212	3529	2359	1947
Num of 40 byte pkts	5184	4255	2698	2969
Ratio	1.23	1.21	1.14	1.53

(the size of an ACK_{tcp}). Table 1 shows the simulation result of actual numbers of transmitted packets in a chain network (using the regular TCP). Although the 40-byte packet is only 1/26 of the size of a 1040-byte one, the number of transmitted packets only increases by 53%. This implicitly verifies that channel capacity consumed by ACK_{tcp} is similar to that of data packets.

This fact points to a promising direction to increase the TCP performance over multi-hop network by lowering the number of transmitted ACKs even more. Motivated by this finding, we propose a new algorithm.

5. New way to improve TCP throughput

Two main factors affect the TCP performance in multi-hop wireless network: the injected traffic load and the control traffic. Because setting a fixed *cwnd* would restrict its applicability, we focus on improving TCP performance by minimizing the number of ACK packets. The first strategy is to let the receiver always waits until the ACK-delay timeout to generate an ACK if arrived packets are in-order.

Most existing methods use a small delay window limit. When a delay window limit is small, it will be generally reached sooner than the ACK-delay timeout, which means that an ACK is mostly generated before the timer expires. To reduce further the number of ACK packets, our first strategy makes the number of ACK generated by ACK-delay timeout dominates that by reaching delay window. Therefore, we set up a large delay window (in this paper it is set at 25). Chen et al. [8] point out that a higher delay window limit does not necessarily result in higher throughput. However, because in our method ACK-delay timeout generates many more ACKs than reaching the delay window limit, we can achieve better performance with large delay window limits. In this paper, we set the ACK-delay timer at 200 ms, based on extensive experimentation and some consideration of application scenarios, e.g. the effect of mobility.

When the congestion window is small, the receiver waits for ACK-delay timeout to send ACK even if no packet is arriving. This results in channel under-utilization. To avoid this problem, we adopt a solution similar as in [8]. The sender puts the current value of *cwnd* into the option field of the TCP header to inform the receiver of the current *cwnd*. When the number of received-but-unacknowledged packets equals *cwnd*, the receiver knows that the sender is waiting for an ACK and sends one immediately. The difference is that we do not need to count the path length, as required in [8]. For lost- or gap-filling packets, we use the same mechanism as in the regular TCP. That is, when such a packet is detected, the receiver generates the ACK immediately.

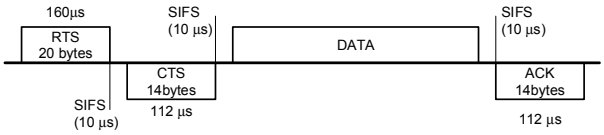

Figure 2. Timing diagram for a packet transmission with RTS/CTS

Extensive simulation results show that our simple strategy improves TCP performance significantly by reducing more unnecessary ACK packets.

6. Performance evaluation

In our evaluation, we mainly focus on comparing against TCP-DCA, TCP-DA and the regular TCP. We do not include TCP-DAA because it is designed for short chains; also, TCP-DCA is reported to work better than TCP-DAA [8]. Based on results reported in [4], we do not select SACK and Vegas. For the sake of clarity, we name our new algorithm TCP-TDA because we use the ACK-delay timeout as a trigger in a different way from the existing work.

6.1. Simulation scenario

We used the ns2 simulator [7] in our evaluations of two typical topologies where each node resides 200 meters apart from its neighbors. Most of parameters use the NS2 default value. We have chosen FTP as the traffic generator, AODV as the routing protocol, simulation time is 100s, and channel bandwidth is 11 Mbps. The capture effect is also taken into account. All data points are obtained by averaging the result of 4 or 5 simulations with different random seed.

6.2. Throughput in chain topology

Figure 1 shows the topology. The TCP sender resides on the first node and the receiver locates on the last one. Due to the lack of space, we only show results for cases of 1 and 6 flows.

Figure 3-left shows the throughput for one flow and hop counts 1 to 3. We can see that TCP-TDA and TCP-DCA have almost the same performance. This is expected because these two methods basically have the same factor, reaching the *cwnd*, which affects the ACK generating in such scenarios. Both algorithms outperform TCP/TCP-DA significantly. The

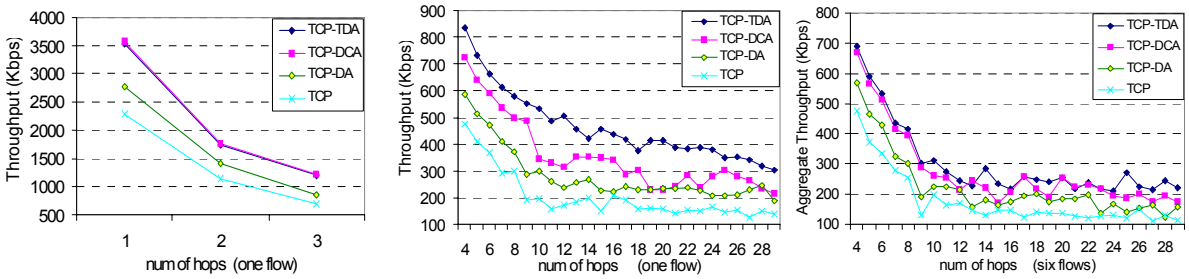

Figure 3. Throughput for the chain topology

Table 2. Number of ACKs and data packets for a 13-hop chain and 1 flow

	Num of ACKs sent	Num of packets sent	Ratio	Throughput (Kbps)
TCP-TDA	693	5644	0.12	478
TCP-DCA	1509	4321	0.35	368
TCP-DA	1542	2887	0.53	242

improvement is 26%, 24%, 41% over the TCP-DA and 55%, 53%, 72% over the regular TCP, for 1 to 3 hops. Similar results hold for other number of flows.

Figure 3-center shows the throughputs for the hop counts from 4 to 29. The performance gain of TCP-TDA over TCP-DCA is in the range of 12% ~ 80%, on average 35%. In most cases, the performance gain is more than 60% over TCP-DA, up to 110%, on average 68%, and 100% over TCP, up to 205%, on average 139%. We notice that the performance gain is higher for most cases in larger hop counts. One reason for this is that the TCP sender emits a burst of packets after receiving a cumulative and incurs heavy contention in the first few hops. When the chain is longer and the packets spread out the chain, this burst effect is alleviated. After considering capture effect, the burst effect does not impair performance greatly as expected earlier [8].

For other numbers of flows, the performance gains are also significant. In the 2-flow case, the average gains are 36%, 77%, 87% over TCP-DCA, TCP-DA, TCP, respectively. In 6-flows case, the gain is 11%, 38%, 76%, respectively. But we noticed that with the increasing of number of flows, although the performance gain over TCP-DA/TCP is still significant, that over TCP-DCA decreased significantly. In the 6-flow case, TCP-TDA works only slightly better (Figure 3-right). In a few cases, the TCP-TDA works even worse than TCP-DCA. The explanation is given below. The fact that all four algorithms work worse for larger number of flows suggests that the number of flows should be kept low, especially with large number of hops. This sheds light on application design for such scenarios.

Table 2 shows a typical example of number of ACKs and data packets sent. TCP-TDA generates much less ACKs than other algorithms. In the 6-flow case, we observed that, in many scenarios, the ratio for TCP-TDA increases to more than 0.3 and accordingly, the ratio for TCP-DCA increases to more than 0.4, which means the number of generated ACK increases significantly and therefore the performance gain deteriorates. This confirms that reduction in the number of ACKs does help improve the performance. Because other factors also affect the TCP behavior, throughput does not increase linearly with the number of reduced ACKs. Moreover, in the 6-flow case, we observed in some scenarios that although TCP-TDA sends 10% fewer ACKs than TCP-DCA, the performance gain is small, or even worse. This implies that our algorithm can be improved further.

TCP-TDA significantly reduces excessive ACK packets but it may result in TCP-sender timeout if numerous ACK packets lost due to channel error. To solve this problem, we modify the TCP-TDA to setup another timer to resend the last ACK packet if receiver does not receive new packets in a given time after an ACK is sent. Moreover, receiver can be set to send ACK a little sooner than the number of arrived packets reaches the *cwnd*. We are currently investigating further improvements for these strategies and their performance in complex scenarios. The results will be reported in the future.

We can also see an interesting phenomenon in Figure 3. The throughput does not decrease monotonously with the increasing hop count, for all four algorithms. The reason is that errors generate duplicate ACKs, which results in *cwnd* shrinking sooner. This, in turn, brings the injected traffic closer to the optimal value and improves the throughput.

6.3. Cross topology

We also evaluate the performance in a more complicated grid topology shown in Figure 5. The filled circles depict the sending nodes, the triangles indicate the receivers, and the arrow lines point the

transmission direction. Due to the lack of space, we only report the results for 6 cross flows, averaged over 5 runs with different seeds. In this critical scenario, the interference and contention has greater impact than in the chain topology. Therefore, the performance gain of our algorithm is degraded significantly, and shows approximately 8%, 25%, 35% improvement over the TCP-DCA, TCP-DA and the regular TCP, respectively. We believe TCP-TDA will be further improved if combined with other techniques.

7. Discussion and future work

Our method is motivated by the fact that a short ACK packet consumes channel capacity comparable to the much longer data packet, over high-speed connections. For higher transmission rates, lower layer overhead would be greater and the ACK packets will consume more channel capacity. This means that our method would perform even better. Because RTS/CTS introduces significant overhead, it would be interesting to investigate how to alleviate this negative effect.

Our algorithm can work with methods that use ELFN-like technique [9], which could be used to inform the sender to lower its packet injection rate, e.g. by reducing the *cwnd*, rather than keeping the *cwnd* from shrinking. Modifying existing ELFN-like techniques to work with our algorithm is part of our future work.

The performance of all existing algorithms deteriorates for increasing numbers of flows. We are currently investigating new methods to improve performance in this case.

Long-hop wireless network is present in scenarios such as sensor networks. Different sensor network implementation may use different contention-based MAC design. But our idea still holds and could be applied with different implementations.

8. Conclusion

In multi-hop wireless networks, unlike previous work, we showed that the optimal congestion window size is between $n/3$ and $n/2$, instead of $n/4$, where n is the total number of hops. This conclusion matches the simulation results more accurately than the prior work. We also showed that the small ACK packets consume comparable channel capacity as the much longer data packets, in high-speed network. Motivated by this, we proposed an improved delay-ACK algorithm. Extensive simulations show that our method lowers the excessive ACKs significantly and therefore improves TCP performance significantly, achieving up to 205% gain in long-hop networks and 35% gain in a complex grid network, compared to the regular TCP. Unlike existing work, which usually focuses on short-hop networks, our evaluation includes long-hop networks, which is particularly valuable for large sensor networks. We also provide some possible improvements to the proposed TCP-TDA algorithm. Our method could be improved further by combining it with complementary methods.

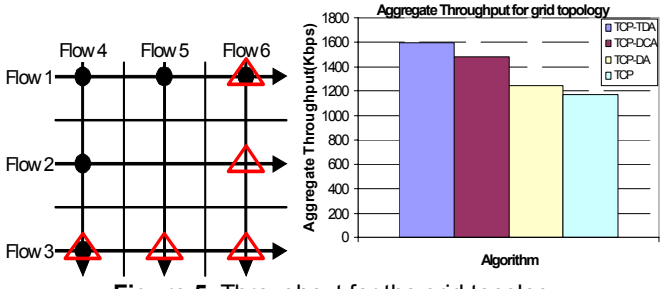

Figure 5. Throughput for the grid topology

9. Acknowledgements

The authors are thankful to Dr. Zhibin Wu, from Qualcomm, Inc., for valuable discussion and help.

10. References

[1] E. Altman and T. Jimenez, "Novel delayed ACK techniques for improving TCP performance in multihop wireless networks," *IEEE Pers Wireless Comms*, Sep 2003.

[2] A. K. Singh and K. Kankipati, "TCP-ada: TCP with adaptive delayed acknowledgement for mobile ad hoc networks," *Proc. IEEE WCNC*, 2004.

[3] K. Chen, Y. Xue, and K. Nahrstedt, "On setting TCP's congestion window limit in mobile ad hoc networks," *Proc. IEEE ICC*, 2003.

[4] R. de Oliveira and T. Braun, "A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks," *Proc. IEEE INFOCOM '05*, 2005.

[5] S. Floyd and T. Henderson, RFC 2582, http://www.ietf.org/rfc/rfc2582.txt

[6] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," *Proc. IEEE INFOCOM '03*, Mar 2003.

[7] NS2 network simulator, http://www.isi.edu/nsnam/ns/

[8] J. Chen, Y. Z. Lee, M. Gerla, and M. Y. Sanadidi, "TCP with delayed ACK for wireless networks," *Proc. IEEE BROADNETS*, 2006.

[9] G. Holland and N. H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Proc. ACM MobiCom '99*, Seattle, WA, Aug 1999.

[10] R. Braden. "Requirements for internet hosts – communication layers", RFC-1122, IETF Network Working Group, Oct 1989.

[11] T. Yuki, T. Yamamoto, M. Sugano, M. Murata, H. Miyahara, and T. Hatauchi, "Performance improvement of TCP over an ad hoc network by combining of data and ACK packets," *IEICE Transactions on Communications*, 2004.