TCP with Delayed Ack for Wireless Networks

Jiwei Chen, Yeng Zhong Lee, Mario Gerla, M.Y. Sanadidi
University of California, Los Angeles, CA 90095
Email:{cjw}@ee.ucla.edu, {yenglee,gerla, medy}@cs.ucla.edu

***Abstract*— This paper studies the TCP performance with delayed ack in wireless networks (including ad hoc and WLANs) which use IEEE 802.11 MAC protocol as the underlying medium access control. Our analysis and simulations show that TCP throughput does not always benefit from an unrestricted delay policy. In fact, for a given topology and flow pattern, there exists an optimal delay window size at the receiver that produces best TCP throughput. If the window is set too small, the receiver generates too many acks and causes channel contention; on the other hand, if set the window too high, the bursty transmission at the sender triggered by large cumulative acks will induce interference and packet losses, thus degrading the throughout. In wireless networks, packet losses are also related to the length of TCP path; when traveling through a longer path, a packet is more likely to suffer interference. Therefore, path length is an important factor to consider when choosing appropriate delay window sizes. In this paper, we propose two independent yet compatible adaptive delayed ack mechanisms, based on path hop-count and end-to-end delay respectively. These schemes significantly improve TCP performance in both multihop ad hoc wireless and hybrid wired/wireless networks. The simulated results show that our schemes can effectively improve TCP throughput by up to 30% in static networks, and provide more significant gain in mobile networks. Some simulation results are also validated by real testbed experiments.**

I. Introduction

The Transport Control Protocol (TCP) is the most widely used reliable transport protocol over the Internet. TCP was originally designed for wired links where buffer overflows account for most packet losses. However, in multihop ad hoc wireless networks, several other inherent factors attribute to the TCP performance deterioration, including unpredictable channel errors, medium access contention complicated by hidden/exposed terminal problems, and frequent route breakages caused by node mobility. All these factors pose great challenges to the design of TCP protocols to provide efficient and reliable end-to-end communications. Many researchers have made valiant effort to propose various methods to make TCP survive in such challenging environments. In this paper, we focus on studying the effect of delayed acks on TCP performance. Then based on our analysis, we propose adaptive schemes that address the aforementioned factors effectively.

In standard TCP the receiver generates one ack for each data packet or two in-order data packets with the standard delayed ack option. This mechanism works well in wired networks. In multihop wireless networks, however, this mechanism can be further improved due to:

- Interference issue: since the data and ack packets usually take the same route (or spatially close routes), the interference caused to data packets increases with the number of acks generated.
- Generating acks wastes scarce wireless resources. Though acks are essential to provide reliability, generating more acks than necessary is not desirable in wireless networks. Ideally, the receiver should generate minimal number of acks required for reliable data recovery.

Recently, the delayed ack strategy has been studied to improve TCP performance [1], [2]. However, this field is not fully exploited and many issues remain unsolved. Some important questions include how delayed acks effect TCP performance, and how to choose the optimal delay window (the number of in-order data packets to be waited for before generating an ack) in multihop wireless networks. In this paper we carry out a systematic study to understand the effect of delayed acks on TCP over wireless links. We investigate TCP performance and delayed ack related packet loss characteristics, under various wireless network scenarios and flow patterns. Our objective is to clearly identify the relationship between TCP throughput and delayed ack over multihop wireless links. Through both analysis and simulations, we reveal several interesting findings, which are tremendously beneficial to deeply understand TCP behavior in wireless multihop networks, and to design enhanced TCP protocols.

First, we found that TCP does not always benefit from arbitrary delaying of acks. In fact, for a given network topology and flow pattern, there exists an optimal delay window size at the receiver, at which TCP achieves maximum throughput. Further increasing delay window size induces increased packet losses and degraded TCP throughput. Second, since 802.11 does not guarantee collision free packet transmission, a packet is more likely to be interfered with when going through a long path. Over a long path, a large delay window may cause large bursts of packets in transit. This, in turn, causes severe packet interference with each other. When packet loss rate becomes high, the benefit of delaying acks, via reducing ack packets, disappears. Consequently, TCP performance degrades after reaching the peak with the optimal delay window size.

In order to achieve optimal TCP performance, we analyze the packet loss probability of burst transportation over 802.11 MAC using the worst case scenario. The analysis sheds lights on the effect of delayed ack on TCP performance and provides guidance for optimal delay window selection. It is worth noting that although our analysis is carried out under the worst case scenaio, it still provides a relatively accurate prediction for packet loss in the delay window range we focus on. The correctness of our analysis is validated by simulations.

Armed with the deep understanding of delayed ack impact on TCP performance over multihop wireless links, we propose an adaptive scheme based on the hop count of a

TCP path. The basic idea behind this scheme is, for a short path, we delay the ack as much as possible to maximally improve TCP throughput; while for a long path, we apply an appropriate delay window size restriction to avoid high packet loss and achieve optimal TCP performance. Furthermore, we propose an end-to-end delay based scheme tailored for hybrid wired/wireless networks. These two schemes can be deployed separately, but are also compatible to provide better performance in heterogenous networks. It's also worth noting that our proposed schemes are deployed only in the end hosts, thus no modification on intermediate nodes is needed.

While our work shares the common concept with previous research in that the TCP receiver delays ack up to a certain number of data packets, we take a unique, systematic and optimized approach to understand the delayed ack impact and propose effective solutions. Moreover, to the best of our knowledge, our proposed schemes are the only existing mechanisms designed for both MANET and hybrid wired and wireless networks.

In ad hoc networks, delayed acks may potentially improve TCP throughput regardless of underlying routing protocols. Different routing mechanisms, however, have great impact on TCP performance [3], and to what degree delayed acks can benefit TCP performance. In this paper, we study the performance of TCP-DCA with three popular used ad hoc routing protocols, namely, Ad-Hoc On-Demand Distance Vector Routing (AODV) [4], Dynamic Source Routing (DSR) [5] and Greedy Perimeter Stateless Routing (GPSR) [6]. The reason we include GPSR in our study is that Geo-routing is a recently developed routing scheme promising to be scalable, and robust to node mobility. The significant TCP-DCA performance improvement is shown over the above mentioned ad hoc routing schemes, in both static and mobile ad hoc networks.

The remainder of this paper is organized as follows. Background work is provided in Section II. A thorough study of delayed ack impact on TCP performance is presented in Section III under various network topologies and traffic patterns. We present our worst case packet loss analysis and proposed TCP-DCA (Delayed Cumulative Ack)scheme in Section IV. Section V evaluates TCP-DCA on static and mobile ad hoc network, and hybrid wired/wireless networks as well. The performance comparison with [2] is also presented in this section. Section VI discusses a few issues to further improve TCP with delayed ack. We conclude the paper in Section VII. The impact of different routing protocols on TCP is also considered in proper sections.

II. Related Work

Standard TCP assumes that a packet loss is invariably due to buffer overflow and reduces the congestion window by half when packet loss happens. However, in ad hoc network, packet loss caused by buffer overflow is rare. Instead, it is more likely due to medium contention as shown in [7]. The fundamental problem resides in the limitations of IEEE 802.11 MAC. Since the interference range is usually longer than the transmission range, Request-To-Send (RTS) and Clear-To-Send (CTS) in 802.11 MAC cannot ensure collision free transfer of packets. This causes hidden/exposed terminals leading to packet loss in ad hoc multihop wireless paths [7]. Many research efforts have been made to adapt TCP to the unique characteristic of wireless ad hoc network, e.g. Transport layer "Fixed RTO" in [3], delayed ack in [1], [2], network layer support [8], [9], [10] and even lower-layer assistance, for example, MAC support in [7].

Although a TCP ack packet is small, typically 40 bytes, the transmission of TCP ack packets may require the same overhead as that of data packets in 802.11 MAC depending on the RTS threshold. If interference from TCP acks could be reduced, data packets would suffer less collisions resulting in higher throughput. Several approaches to delay acks have been proposed [1], [2], [11]. TCP-ADA (Adaptive Delayed Ack) [11] proposed to decrease the number of acks to improve TCP performance. They claimed that maximum TCP throughput is achieved when one ack acknowledges the full congestion window. However, the method did not address several important issues, such as packet loss and out-of-order packet reception. In fact, as we show in this paper, TCP does not always benefit from delaying ack as much as possible.

Allman[1] presented a basic delayed ack scheme which was further improved in [2]. The scheme is called TCP-DAA (Dynamic Adaptive Acknowledgement). In TCP-DAA, the receiver adjusts the delay window according to channel condition (packet loss event). A TCP-DAA receiver delays acking until it receives a certain number of data packets, ranging from 2 to 4 packets. When there is no packet loss, the TCP-DAA receiver waits for more data packets (up to 4) before generating an ack, but reduces the number to 2 in case of out-of-order packet arrival. However, since a TCP sender automatically cuts the congestion window when packet loss occurs, i.e. automatically adapting to the channel state at the sender side, the receiver side adaptation provides little extra improvement. We will show that in our adaptive delayed ack scheme, the receiver does not respond dynamically to packet loss, yet achieves better performance.

In [2], the ack time is set to be one average packet inter-arrival time. That is, an ack is generated when no data packet arrives after one average packet inter-arrival time since last unacknowledged data packet. Since the inter-arrival time is highly variable in the wireless network because of random MAC contention and back-off, it is very difficult to get any accurate enough statistics in a complex large system. Another impact of this timer implementation is that the receiver operates insensitively to the number of acks (2 to 4) to be delayed because any unexpected extra delayed data packet will trigger an ack.

An important aspect of TCP with delayed ack is the delay window size selection. In TCP-DAA [2], the receiver may delay up to four ack packets and this number is limited by the sender congestion window, which is fixed at four packets. The similar delay window size of 3-4 packets is also picked in [1] heuristically. There are issues in this scheme that desires further clarification and improvement. First, although a small congestion window limit helps TCP operation in wireless networks, it is not suitable for hybrid wired and wireless

network where a high bandwidth delay product exists. Second, if the congestion window is not limited, the choice of a delay window size of 4 may no longer be suitable. In this paper, we address the issues above providing effective and general solutions that apply to wired/wireless environments

We also study the impact of delayed acks at a TCP-DCA sender. Since the receiver does not ack as frequently as in standard TCP, the congestion window increase rate for delayed ack is slowed down. Such slower probing rate improves TCP performance in ad hoc networks, as reported in [12]. The conservative window increase, however, hinders efficient transmission in wired network where delay bandwidth product may be large. In this paper, we propose a simple technique to solve this problem and allow TCP-DCA to cope with hybrid wired/wireless networks.

In this paper, we also show the impact of congestion window limit on the proposed TCP-DCA scheme. A small sender congestion window can decrease interference and maximize pipeline effect. [7] revealed that there exists an optimal TCP congestion window size that maximizes spatial channel reuse. Further increasing the window size does not lead to better performance. On the contrary, it results in increased link layer contention and degraded TCP throughput. In [13], the optimal congestion window limit is determined based on the hop count for maximum pipeline effect. In TCP-DCA the sender's congestion window is not limited, and yet, we will show it performs better than the case of limited sender's congestion window defined in [13]. Our results indicate a setting of the congestion window limit more suited to TCP-DCA is needed.

III. TCP Throughput vs. Delay Window

In this section, we investigate the impact of the receiver ack delay window size on TCP performance. The conclusion we draw is that, when the path length is short, TCP achieves better performance with a delay window as large as the entire sender congestion window. When the path length increases, however, a large delay window triggers bursty transmissions, which results in mutual data packet contention and higher losses. In fact, for long paths, there exists an optimal delay window size that achieves maximum TCP throughput. We verify the delay ack impact using various network topologies, including cross and grid topologies with various flow patterns. Real testbed measurement results are also presented to reinforce our conclusions.

In the following, we first study a basic scheme (TCP-DCA) with a receiver delay window limit and evaluate the impact of the delay window size on TCP performance. Here we use a manually configured static routing to investigate delayed ack performance over ad hoc networks ignoring at first any routing impact.

A. TCP with Receiver Delay Window Limit

A TCP-DCA receiver maintains a variable *delay window* "w" representing the number of data packet arrivals before acking at the receiver. This delay window w is bounded by a delay window limit, and also limited by the congestion window size to prevent the delay window from possibly exceeding the congestion window which results in delaying the ack, but for possibly non-existent packets. If data packets arrive in order, the receiver generates one cumulative ack for every w data packets. If an out of order packet is received, or a packet that fills a gap in the sequence space of packets in the receiver buffer (that is, recovery from earlier packet loss) the receiver acks immediately to inform the sender of the packet loss/recovery in a timely manner. The receiver also keeps a fall-back delay ack timer which estimates the expected time to receive w data packets. When the sender receives a cumulative ack acknowledging w data packets, it updates the congestion window and sends out a burst of at least w packets (provided such packets are ready for transmission in the sender buffer).

To get the delay timer period, the receiver monitors the data packet inter-arrival interval and computes a smoothed interval through a low-pass filter. The average inter-arrival interval is used to set the ack delay timer based on the current delay window size. In TCP-DCA, an accurate delay timer is not needed and the timer is solely for fall back purpose. In fact, our inter-arrival time computation is rather loose: the receiver samples the inter-arrival time of *any* consecutively arrived data packets, not necessarily in-order packets. Therefore, the inter-arrival sample is an *inflated* value compared with inter-arrival between in-order packets. A TCP-DCA receiver smoothes such inflated inter-arrival samples through a low-pass filter:

$$I_{avg}^{i+1} = \beta I_{avg}^{i} + (1-\beta) I_s \tag{1}$$

where I_{avg} is the average of *inflated* packet inter-arrival time I_s. I_{avg} is used to set delay ack timer (T_w) which defines the total time for receiving a whole delay window of in-order data packets:

$$T_w = \alpha w I_{avg} \tag{2}$$

where w is the delay window size, α is a parameter to tolerate high dynamic packet delay. Obviously, the delay ack timer is rather inflated and quite robust to high inter-arrival variation. In the paper, we choose β as 0.8 and α as 1.5. The receiver also generates an ack within 500 ms of the arrival of an unacknowledged data packet in accordance with RFC2581 [14].

One apparent drawback to this approach is that the TCP-DCA receiver needs to be informed of sender's congestion window size to prevent delay window larger than congestion window leading to unnecessary delaying at the receiver. In TCP-DCA the sender reuses the advertised window field in data packet header for "advertising" back its congestion window size to the receiver. But our design is not unrealistic due to the following fact that current TCP connection is mostly used for single direction, i.e. there is only data packets from the sender to the receiver and no backward data, and the advertised window field from the sender is usually wasted. We would further discuss this issue in Section VI. Real testbed measurements results with TCP-DCA confirm the feasibility of our delayed ack scheme. Measurements results are provided in Section III-B.2.

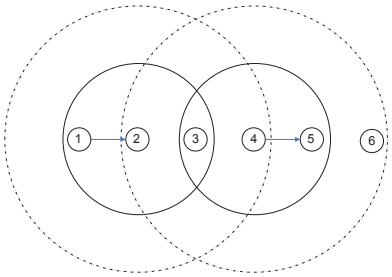


Fig. 1. Chain Topology. The solid-line circle denotes a node's valid transmission range. The dotted-line circle denotes a node's interference range. Node 4's transmission will interfere with node 1's transmissions to node 2.

B. Chain Topology

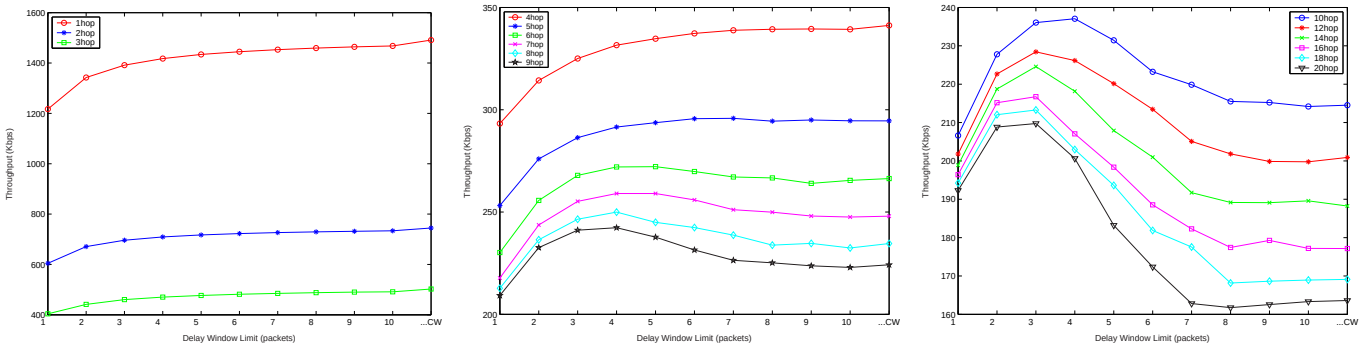
We used the chain topology in Ns2 [15] simulation. An example is shown in Fig.1. The TCP connection is sourced at the first node (node 1) and packets travel over the chain to the end node (node 6). The interference range is 550m and transmission range is 250m. We place the nodes at 200m intervals, so nodes that are 4 hops away can transmit simultaneously without interference. Notice that a node that is 3 hops away is a "hidden node". IEEE 802.11 is the underlying MAC. Data rate on the wireless channel is 2Mbps and one simulation run lasts 300 seconds. Each data point represents an averaged of 5 simulation runs with different random seeds. The packet size is 1000 bytes and TCP NewReno is used.

1) Single TCP Flow: TCP performance over wireless multihop inevitably depends on the path length (hop count). The longer the path is, the lower the throughput would be. We present TCP throughput as a function of delay window limits on chain topologies of variable lengths in Fig.2. Fig.2(a) shows that when a sender and a receiver are within 3 hops, TCP-DCA gets steady performance gain by increasing the delay window size up to the entire congestion window size. For the one hop case, the graph demonstrates a steady throughput increase when the delay window increases from 1 to the whole congestion window at where the delay window limit is set to a very large number. Compared with the throughput of standard TCP (with delay window at 1), the fastest throughput increase can be seen when the delay window is small, say less than 5. The increasing trend becomes slower for delay windows larger than 5 and the throughput approaches the limit when the delay window reaches the congestion window size (CW). Fig.2(a) indicates that delaying acks at the receiver up to the sender window size improves TCP throughput for the one hop case. The same trend is observed when the path length is 2 and 3. The reason for this performance gain is that 802.11 MAC provides collision free packet transmissions in such short path length. Since the interference range is larger than 2 times the transmission range, when the sender and receiver are within 2 hops, every node along the path can sense all other nodes transmissions. In this case, no hidden nodes exist and thus no packet loss occurs. When the hop length is 3, the TCP receiver is the only node hidden from the TCP sender. If the receiver acks only after receiving all packets in a congestion window, no data packet can be interfered. Therefore, there is no data packet loss due to interference on a path less than or equal to 3 hops. TCP gets steady throughput gain by increasing the size of the receiver delay window until the maximal performance gain is achieved when the receiver acks after receiving all packets in a congestion window. We observe that TCP-DCA attains up to 25% throughput gain relative to standard TCP.

Since the interference range is larger than the transmission range, RTS/CTS cannot completely solve the hidden/exposed terminal problem. As the chain becomes longer, packet collision is unavoidable. For example, node 1 and node 4 are interfering nodes in Fig.1 and simultaneous packet transmissions on them will be interfered. Though MAC has a retransmission mechanism to recover lost packets, it cannot recover such lost packet in the presence of severe interference. The hidden terminal potentially results in packet loss and this problem becomes more severe for longer paths because a packet has more chances to be interfered with. Moreover, the sender immediately sends a burst of packets upon receiving a cumulative ack, these packets can interfere with each other. We will show that packet loss probability increases as the size of the packet burst increases in Section IV. And the packet burst size is directly related with the receiver delay window since the size of a burst is at least equal to the size of the delay window. When the packet loss becomes high, the TCP throughput gain from delaying an ack is lost due to the increased transmission burst size and its higher loss probability. Fig.2(b) shows this tradeoff of TCP-DCA performance gain with delay window size for paths larger than 3 hops. When the hop count is 4 or 5, we observe unsuccessful packet transmissions caused by interference in our simulation. However, since a TCP sender is able to recover packet loss rapidly due to the small RTT, TCP-DCA maintains performance gain by delaying ack for more data packets. For paths longer than 5 hops, TCP achieves throughput gain when the delay window size is small, but for large delay window size, delaying ack cannot maintain throughput gain because of excessive burst data packet losses. Further, now that RTT is larger, TCP spends more time detecting packet loss and recovering lost packets by entering fast retransmit/recovery in which only one packet is recovered per RTT, and thus more TCP throughput degradation. Therefore, for long paths with large delay window, large delay window is not preferred.

We also show TCP-DCA performance over a very long path $h \geq 10$ in Fig.2(c). Here, TCP only gets performance gain for small delay window size. For large delay window size, TCP even gets lower throughput than standard TCP.

2) Real Testbed Verification: We investigate the effectiveness of our TCP-DCA in an actual ad-hoc network testbed. Our testbed consists of six Dell Pentium III, 650/500Mhz processor equipped with Orinoco 802.11b PCMICA cards with channel rate of 2Mbps. The laptops run Mandrake Linux distribution 7 with kernel version 2.4.3. Linux PCMCIA package version 3.2.0 and Orinoco wavelan2-cs driver are used for 802.11b devices and the devices are set to ad-hoc mode. The topology of the testbed is a chain and the route is statically configured. There is one source laptop and one selected destination laptop among other 5 laptops depending on the number of hops in



(a) Path Length $h \leq 3$ (b) Path Length $10 > h > 3$ (c) Path Length $20 \geq h \geq 10$

Fig. 2. TCP Throughput vs. Delay Window on Chain Topology

our experiments. We use "Iperf" to generate FTP traffic with packet size of 1000 bytes. From the experiment results shown in Fig.3, we confirm that the TCP throughput vs. receiver delay window follows the same trends as in the simulation results shown in Fig.1.

Fig.4 compares TCP-DCA throughput for 1 hop and 5 hops paths from simulation and actual testbed, when delay window limit is equal to 1 and congestion window respectively. We note that the average difference between the testbed TCP throughput and the simulated results is below 10%. Such difference is mainly caused by parameter setting in the testbed measurements. In our testbed experiments, RTS/CTS control packets are adopted to provide carrier sensing for unicast data packets to overcome the hidden terminal problem if the packet size is above the minimum threshold 256 bytes. However, the size of TCP ack packet (40 bytes) is below the minimum RTS/CTS threshold in linux settings so that no RTS/CTS handshake is performed when transmitting acks in testbed experiments. Compared with simulation results where RTS/CTS is always performed no matter what the packet size is, such "zero" MAC overhead in the measurements has two impacts on the performance results: 1)TCP-DCA throughput in testbed experiments is slightly higher than simulated results (in Fig.2) for small delay window, as shown in Fig.3 when delay window is 1; 2) the throughput gain derived from TCP-DCA in the measurements is lower than the corresponding results in the simulations because of the lack of RTS/CTS reduction for acks in experiments, as shown in Fig.4. The lessons learned from the testbed measurements and simulations have greatly enriched our understanding of the cross layer interdependence and will undoubtedly contribute to more efficient designs in the future.

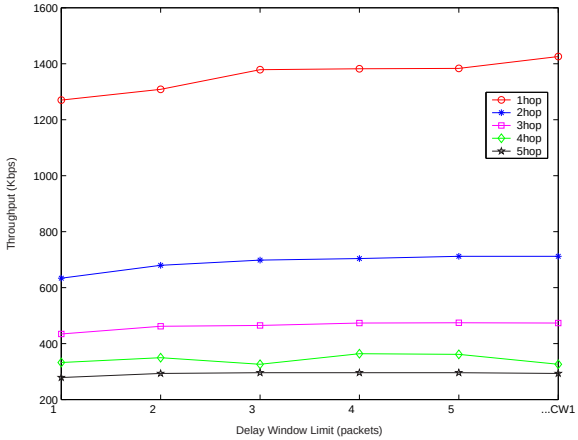
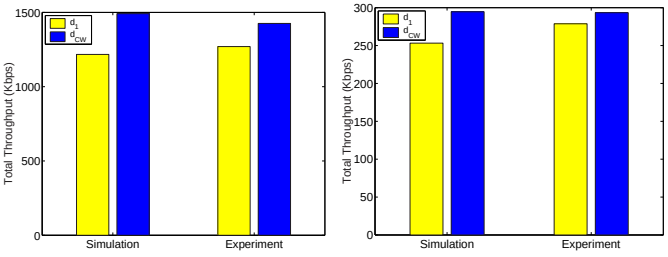


Fig. 3. Real Testbed Measurement



(a) 1 hop (b) 5 hop

Fig. 4. Simulation vs. Experiment

C. More Complex Topologies and Flow Patterns

We expand our study to scenarios of more complex topologies and flow patterns, including cross and grid topologies. We keep the same simulation parameters as before.

For all cases, we observe the similar results to what is described above, i.e. TCP does not always benefit from an unlimited delaying of acks. There exists an optimal receiver delay window at which TCP achieves its best performance. The following provides a short summary of results with more flows and various topologies. We also show our TCP-DCA performance over random topologies with multiple flows in Section V-C.

1) Multiple TCP Flows: Fig.5 exhibits result for 5 concurrent TCP flows on a chain topology with hop count varying from 3 to 7. It shows similar results to Fig.2: TCP throughput gets maximal improvement by delaying acks up to the entire congestion window for short paths. For longer paths, the throughput increases with the increase of delay window until the maximum throughput is reached at a certain delay window size. After the peak point, TCP throughput gracefully degrades with larger delay window sizes.

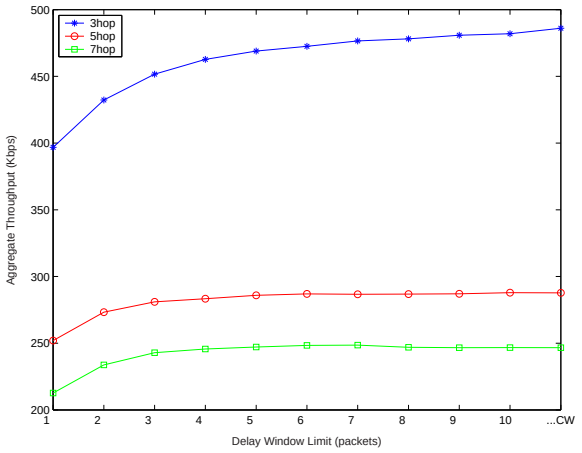
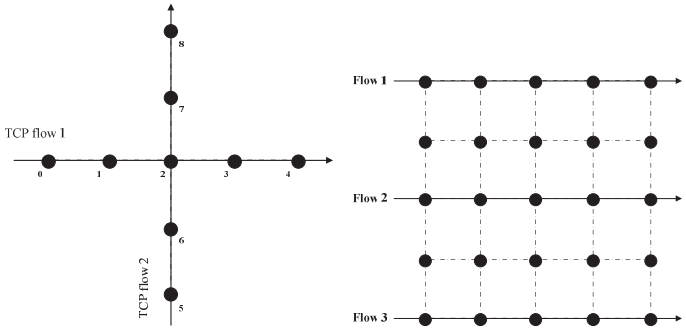


Fig. 5. TCP Throughput vs. Delay Window on Chain Topology (5 flows)

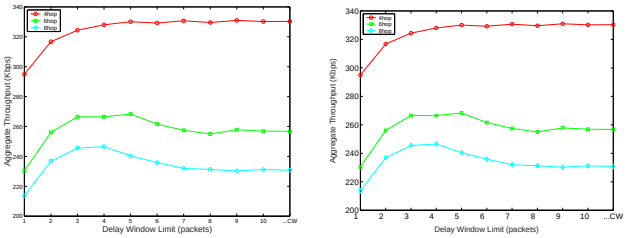


(a) Cross Topology (b) Grid Topology

Fig. 6. More Complex Topologies. Left: cross topology with 9 nodes. 200 meter distance between two adjacent nodes. 2 TCP flows in each direction. Right: 5x5 grid topology, 200 meter distance between horizontal and vertical adjacent nodes.

2) Cross and Grid Topology: Fig.7 shows examples of cross and grid topologies in which each TCP flow traverses a 4 hop path. We actually vary the path length to study its impact. The results are presented in Fig.7. Similar trends to those found in chain topologies are observed. We also ran extensive simulations on cross and grid topologies with multiple overlapped flows instead of one flow. The results, not included here due to space constraints, confirm the same trends discussed above.

To summarize, both simulation and testbed experiments show that TCP with delayed ack can enhance throughput performance. When the path length is short, TCP achieves optimal throughput by delaying acks as much as possible, up to entire sender congestion window. On the other hand, when the path becomes longer, a large delay window hinders effective transmission and deteriorates TCP throughput gradually. We show that there exists a certain delay window size, at which TCP achieves optimal throughput performance. Based on this observation, we analytically study packet losses triggered by delayed acks in a worst case scenario. We will discuss our proposed mechanisms to determine appropriate delay window sizes in the next section.



(a) Cross Topology (b) Grid Topology

Fig. 7. TCP Throughput vs. Delay Window on Complex Topologies

IV. TCP with Adaptive Delayed Cumulative Ack (TCP-DCA)

In this section we analyze the packet loss probabilities associated with bursty traffic triggered by delayed ack and study the optimal delay window size for given topologies. TCP-DCA with adaptive delayed window is proposed according to the underlying path information to optimize TCP performance. The impact from different routing protocols on TCP with delayed ack is also discussed.

A. Packet Burst Transportation over 802.11 MAC

In what follows we study the effectiveness of burst transportation over 802.11 MAC. In what follows we study the effectiveness of burst transportation over 802.11 MAC. In TCP-DCA, the sender emits a burst of packets after receiving a cumulative ack, therefore, the efficiency of such bursts transport has direct impact on TCP performance. For short paths, since the 802.11 MAC can guarantee packet transmission without collision, no packet loss occurs no matter what the burst size is. Therefore, TCP throughput reaches the peak when delay window is at maximum size (congestion window size). However, when paths are longer, collisions occur. In a long path, the packets sent at the beginning of the burst will potentially interfere with the packets at the rear of the burst. Thus, the rear packets have higher loss probability and the loss probability becomes higher with the increase of the burst size and path length. When interference is so severe that 802.11 MAC cannot recover packet losses even after the maximum number of retransmissions, such packets will be discarded. For example, 802.11b typically tries a maximum of 7 RTS retransmissions. In the following we analyze packet loss probabilities for a packet burst and derive the optimal burst size which maximizes the probability of successful delivery of an entire burst. Then, the optimal delay window is set equal to the optimal burst size in order to maximize TCP-DCA throughput gain.

In the following we focus on packet loss due to mutual interference among data packets belonging to the same flow. We derive a worst-case bound on the probability of successful delivery of a whole burst and determine the optimal burst size. Although we only give a worst case analysis, the analytic results are still well matched by simulation results for the window size range of interest . In our analysis we do not take the interference from other flows into account, however, our analysis is a worst-case case study and can tolerate more

packet loss in the presence of interference from other flows. This is verified by simulation. A complete study of burst transportation and packet loss probability on various topology and multiflow is beyond the scope of this paper and is left for future research. For an initial study, please refer to the technical report [16].

Consider packet losses caused by interference among data packets of a burst. For example, in Fig.1, if node 1 is transmitting to node 2 and node 4 is transmitting to node 5 simultaneously, the packet from node 4 is rarely interfered by a transmission from node 1, while the packet from node 1 has high probability of being interfered with. The reason is that node 2 and node 4 can interfere with each other. Packet reception at node 2 will be interfered by node 4 if node 4 is transmitting data packet. Since a data packet is usually much larger than RTS/CTS/ACK in the MAC layer, the packet interference probability at node 2 is significant. On the other hand, the interference at node 4 can happen only when node 4 is receiving CTS/ACK and node 2 is transmitting CTS/ACK, since RTS/CTS/ACK packets are small (at most 20 bytes, i.e. less than 2% of data size). The packet loss probability at node 4 is negligible. With the MAC retransmission mechanism, node 4 can ensure successful packet transmission, but node 2 has high probability of being interfered by transmissions from node 4. Even with MAC retransmissions of collided packets, successful packet transmission probability on node 1 depends on the link-layer queue size at node 4. Because node 4 has little interference, the transmissions from node 4 usually capture the channel while node 1 is in back-off phase induced by heavy interference. In this example, the flow direction is from node 1 to node 6, thus we can safely say that a packet in a burst can only be interfered with by packets previously sent, and not interfered by the packets sent afterwards in the same burst.

We include the above observations in our model. Generally, the interference experienced at a node depends on the queue size at the interfering(hidden) nodes when interfering nodes capture the medium until they empty their queues. If the queue size at interfering nodes is large, the loss probability for an interfered packet waiting to be transmitted is high. In the worst case, an interfered packet needs to wait for all packets at the interfering nodes to leave before successful transmission, i.e. a packet could be interfered with by "all" packets sent previously in a burst and wait for those packets to leave the interfering(hidden) nodes. In terms of a burst, the n-th packet in a burst needs to wait for $n-1$ packet transmission times before successful transmission.

Now let's look at the specific MAC layer back-off mechanism since it determines the time of the eventual successful transmission after recovering from interference. In 802.11b, RTS/CTS help reserve the channel for a packet transmission. If a node successfully receives CTS after RTS transmission, the channel is clearly reserved for data transmission and thus the collision on data transmissions is rare compared with collision on RTS transmissions. In the following, we only consider RTS collisions and the back-off time during RTS retransmissions. If a node cannot transmit a packet after 7 RTS retransmissions, the packet will be lost. We study the

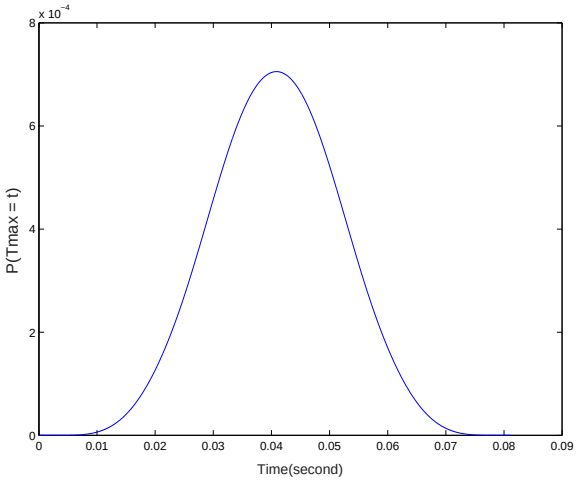


Fig. 8. Maximal Back-off Distribution

802.11b back-off mechanism during retransmissions and use it to determine the optimal burst size. When a node senses the channel busy, it enters the back-off mode by selecting a back-off time uniformly distributed over the range $[0, CW - 1] \times SLOT$, where CW varies between $CW_{min} = 32$ and $CW_{max} = 1024$, the value of SLOT is 20 μs. CW doubles each time when a collision is detected, up to CW_{max}. CW is set back to CW_{min} after a successful transmission.

If a packet is not successfully transmitted after 7 retransmissions, it will be dropped at MAC layer. We proceed to get the total back-off time distribution for 8 transmissions (first transmission plus 7 retransmissions) and derive the success probability for a whole burst. Let i be the number of retransmissions, and let r_i be the discrete random variable uniformly distributed over $[0, CW_i - 1]$, $CW_0 = 32, CW_{i+1} = 2CW_i$ and $CWi \leq 1024$.

The distribution for r_i:

$$P(r_i = T) = \frac{1}{CW_i} \qquad T \in [0, CW_i - 1] \times SLOT \tag{3}$$

Z-transform for r_i:

$$G_i(z) = \sum_{n=0}^{CW_i - 1} \frac{1}{CW_i} z^n \tag{4}$$

The distribution for $P(\sum_{i=0}^{7} r_i = T)$ is the convolution of these 8 random variables and can be derived from their Z-transform product. Let $s = \sum_{i=0}^{7} r_i$, we plot the distribution for s in Fig.8.

We proceed to get the loss probability of a packet in the worst case; that is when a packet is interfered by all previous packets in the same burst. If the burst size is w, in the worst case, the total back-off time of the n-th packet ought to be larger than the transmission time of $n-1$ packets. Here we assume that packets previously sent in the same burst only interfere with this packet once during the packet transmission event on one node. Thus the successful packet transmission probability is at least $P(s > (n-1)T_x)$, where T_x is the packet transmission time. In our simulations, packet size is

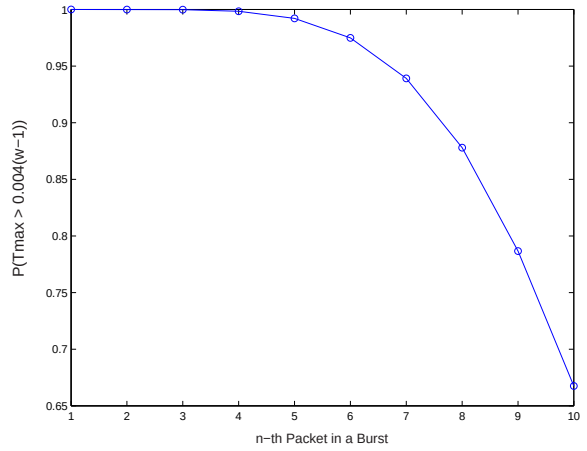


Fig. 9. Successful n-th Packet Transmission Probability in Worst Case

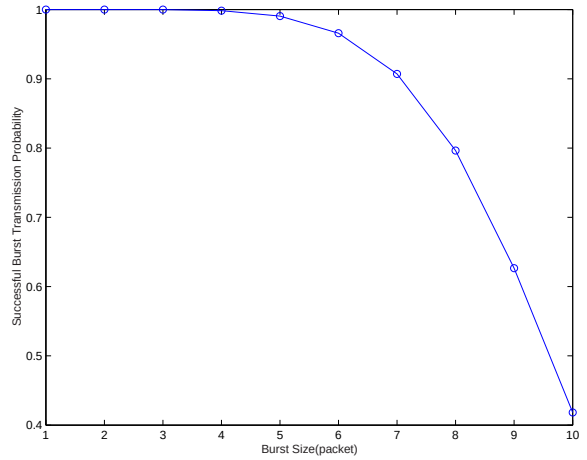


Fig. 10. Successful Burst Transmission Probability in Worst Case

set to 1000 bytes and capacity is 2Mbps, and thus $T_x = 4$ms without considering MAC overhead.

Let $f(n) = P(s > (n-1)T_x)$ represent the successful n-th packet transmission probability in a burst. We plot $f(n)$ vs. n in Fig.9. Intuitively, as n increases, the packet reception success probability decreases, and it decreases in an exponential fashion because of more interference from increased burst size.

If a burst size is w, the successful probability for the entire burst at a node in the worst case is:

$$P(w) = \prod_{n=1}^{w-1} f(n) \tag{5}$$

Fig.10 plots $P(w)$ vs. w. We see that when the burst size is small (≤ 3), the whole burst has 100 percent successful transmission probability. The successful probability begins to decrease exponentially when burst size is beyond 3.

If the path has h hops, the successful delivery probability is the product of successful transmission probability of all possible interfering nodes in the chain topology. The number of all possible interfering nodes in a chain is $h-3$ as depicted in Fig.1. In the worst case scenario, every packet will be interfered with by all the other previously sent packets in the same burst at *every* interfering nodes. Let $P(w,h)$ denote the successful burst delivery probability along a h hop path:

$$P(w,h) = P(w)^{h-3} \tag{6}$$

We plot $P(w,h)$ in Fig.11. Note the same trend as in Fig.10. When burst size is small and less than 4, the whole burst can go through the path without packet loss. The packet loss probability in a burst increases when burst size increases, and successful burst delivery probability decreases exponentially. Moreover, the longer the path, the lower the successful burst delivery probability.

To verify the above analytic results, we run simulations of sending a burst of packets from the sender to the receiver with different path length. For all simulation results, the analytic result did provide a lower bound for the packet loss rate. Interestingly the gap between analytic model and simulation is very small when the burst size is not more than 5, and it becomes large for burst size greater than 5. Fig.12 shows an example of burst success probability on a 5 hop route. In Fig.12, we note that the analytic model can predict packet loss fairly accurately for small delay window. However, the analytic result becomes loose for large delay window. For other path lengths, the results are similar. As we will discuss later, the receiver ack delay window in TCP-DCA never grows beyond 5 for paths longer than 3 hops, and thus the bound obtained is accurate for the range of delay window size of interest.

Now let's look at previous TCP-DCA results in Fig.2(a)-Fig.2(b). A TCP-DCA sender transmits a burst of packets after it gets a cumulative ack. For short paths up to 3 hops, no data packet loss occurs, and TCP-DCA performance gets steady increase with the increase of the delay window up to the entire congestion window. For longer hop count path, from the above analysis, a delay window size of 3 is always a good choice since no packet loss occurs regardless of path length. A larger delay window would cause a larger burst size and make packet loss more likely, thus potentially has a negative effect on TCP performance. On the other hand, a larger delay window would bring more performance gain by generating less ack packets. If TCP could recover fast enough from packet loss, TCP performance would not be significantly affected by larger delay windows. This tradeoff is clearly shown in Fig.2(b) where a delay window slightly larger than 3 still gets performance gain for a moderately long path $h = 4, ..., 9$. For instance, for the 5 hop path, when the delay window is not greater than 6, the successful burst delivery probability is larger than 90% (as shown in Fig.11). TCP-DCA still gets throughput gain from delaying acks. Nevertheless, for delay windows larger than 6, TCP-DCA gets less throughput gain. A large delay window causes large burst and more packet loss. Since TCP-DCA is a TCP clone which employ fast retranmit/recovery to recover lost packets one by one, TCP-DCA cannot efficiently recover packet loss. Therefore, when path is long, small delay window is preferred for best TCP performance. For the very long paths ($h \geq 10$) shown in Fig.2(c), minimum packet loss in a burst is desired. From our analytic model, burst size 3 guarantees burst transport reliability, and this is confirmed in

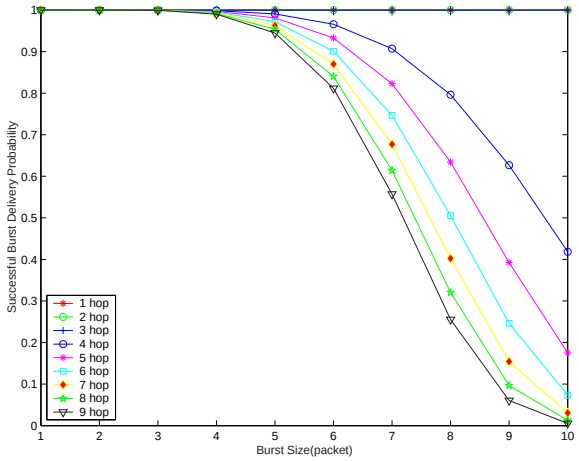


Fig. 11. Successful Burst Delivery Probability in Worst Case

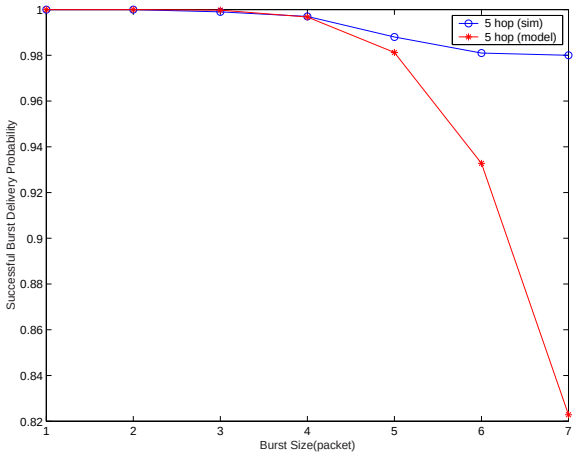
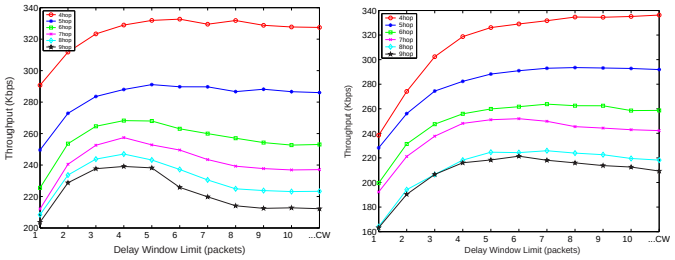


Fig. 12. Successful Burst Delivery Probability in Worst Case (5 hop)

Fig.2(c) where TCP-DCA gets optimal throughput for delay window of 3 over such long paths.

B. Routing Impact on Delayed Acks

In ad hoc networks, routing schemes have great impact on TCP performance [3]. In what follows we study the performance of TCP with delayed ack running over AODV and GPSR and assuming a chain topology. For short paths with 3 hops, no data packet collision occurs, so there is no routing impact on TCP-DCA performance. TCP-DCA results on these two routings are the same as those on static routing in this case. For paths longer than 3 hops, we show TCP-DCA results in Fig.13. These results are similar to results on static routing, however, some minor differences exist and primarily come from different routing implementations. In GPSR, when packet transmission failure occurs, GPSR drops the packet since it does not rediscover the old path (only one path is available in the chain topoloy). Thus TCP-DCA performance on GPSR is similar to that on static routing. AODV gets better performance when the delay window is slightly larger than that in static routing. The reason is that AODV implements a local repair scheme in which a packet to be lost at MAC layer can be salvaged by another route discovery at intermediate nodes.



(a) TCP-DCA over GPSR (b) TCP-DCA over AODV

Fig. 13. Routing Impact on TCP-DCA

TABLE I
DELAY WINDOW AT TCP-DCA RECEIVER

Path Length (h)	Delay Window Limit
$h \leq 3$	Congestion Window
$3 < h \leq 9$	5
$h \geq 10$	3

With local repair, the reliability of burst transport in AODV is improved. However, the performance gain with the delay window size greater than those defined in Table I is marginal.

Here we have studied the routing impact on TCP with delayed ack when nodes are static, in Section V-C we further discuss the case of mobile nodes.

C. TCP-DCA with Adaptive Delayed Ack

Up to now we have presented the relationship between packet burst size and delay window size, and the tradeoff between TCP throughput and delay window size on different path lengths. When delay window is not large, TCP keeps performance gain by generating fewer acks. When the delay window increases, the reliability of burst transportation deteriorates on long paths. When packet loss becomes so high that TCP-DCA cannot recover lost packets fast enough, TCP-DCA throughput is degraded. Inspired by this observation, we dynamically select the delay window size in TCP-DCA based on the hop count of a TCP connection. For a short path ($h \leq 3$), TCP-DCA could delay ack for a whole congestion window to get best performance; for a long path ($h \geq 10$), a small delay window, say 3, is preferred; and for path lengths falling between these two extremes, delay window slightly larger than 3, say 4 or 5 is a good choice. TCP-DCA receiver could get the path length information from the TTL field in TCP packet header or from the routing layer. Table I lists the delay window size applied in TCP-DCA according to the path length.

D. Ack Loss

In standard TCP, more acks are generated than TCP-DCA. Due to the cumulative property of the ack, one ack successfully received at the TCP sender can makes up for all the preceding lost acks. Thus, the ack loss in standard TCP is not serious. However, the number of acks in TCP-DCA is much less than standard TCP, and thus an ack loss has more negative impact on TCP-DCA performance. In order to be robust to ack loss,

a retransmission timer at the receiver is adopted to retransmit a cumulative ack if necessary.

A challenge here is to estimate RTT at the receiver which is needed for setting the ack retransmission timeouts. If TCP includes two-way data, the receiver can get the accurate RTT measurement since the sender acks the data immediately. If TCP only has one-way data, RTT measurement at the receiver may not be straightforward. If the TCP timestamp option is used, the receiver could use it to estimate RTT, though such an estimate may be inflated if the sender does not send data packets immediately after receiving an ack [17]. However, in TCP-DCA, the ack retransmission timer is only a coarse timer for predicting when to retransmit acks, an accurate RTT measurement is not required and thus an inflated RTT can be tolerated.

The receiver computes the ack retransmission time based on this RTT measurement. We use the following formula to calculate the retransmission timer period:

$$\begin{aligned} SRTT^r_{k+1} &= \frac{7}{8}SRTT^r_k + \frac{1}{8}RTT^r_{k+1} \\ RTT^{var}_{k+1} &= \frac{3}{4}RTT^{var}_k + \frac{1}{4}|RTT^r_{k+1} - SRTT^r_{k+1}| \\ RTO^r_{k+1} &= SRTO^r_{k+1} + 4 \times RTT^{var}_{k+1} \end{aligned}$$

Where RTT^r is the RTT estimation at the receiver, $SRTT^r$ is the smoothed RTT. RTT_{var} and RTO^r are RTT variance and retransmission timer period at the receiver. The minimum value of the retransmission timer period is set to 1 second. This ack retransmission timer is started after sending an ack to act as a fallback timer for the ack loss event, and is canceled after receiving an data packet to let delay timer act as a fall back timer.

Note that all previous results were not obtained with ack retransmission . This is due to two facts: first, the results with and without ack retransmission are almost the same for static networks. Mostly because the ack loss is rare and the ack retransmission timer period is large, i.e. at least 1 second. Second, without the ack retransmission timer, it is easy to see the primary impact of the delay window on TCP performance. In the following simulations, the ack retransmission timer is enabled.

V. Performance Evaluation

This section presents the TCP-DCA performance over MANET and hybrid networks. First, we show TCP-DCA performance on static multihop wireless networks and compare it with TCP-DAA in [2]. Second, we demonstrate TCP-DCA performance on mobile ad hoc network, and discuss the different routing scheme impact on TCP-DCA performance. Last, we propose an end-to-end delay based scheme to further improve TCP-DCA in hybrid wired/wireless networks.

A. Comparison with TCP-DAA

In this section, we compare our TCP-DCA with TCP-DAA in [2] which is an interesting extension of [1]. The major differences between TCP-DAA and TCP-DCA are shown in Table II.

TABLE II
Design Difference

	TCP-DAA	TCP-DCA
Sender	Duplicate acks threshold to trigger retransmission is 2; CW upper limit is 4; Retransmission timer is increased fivefold	Puts CW into advertised window field in packet header
Receiver	Delay window is adaptive from 2 and 4 based on loss event	Delay window is adaptive on the path length

We compare the performance of TCP-DCA with that of TCP-DAA [2]. The simulations include chain and grid topology as in [2]. The delay window in TCP-DCA is configured in Table I. Each result is the average of 5 simulation runs.

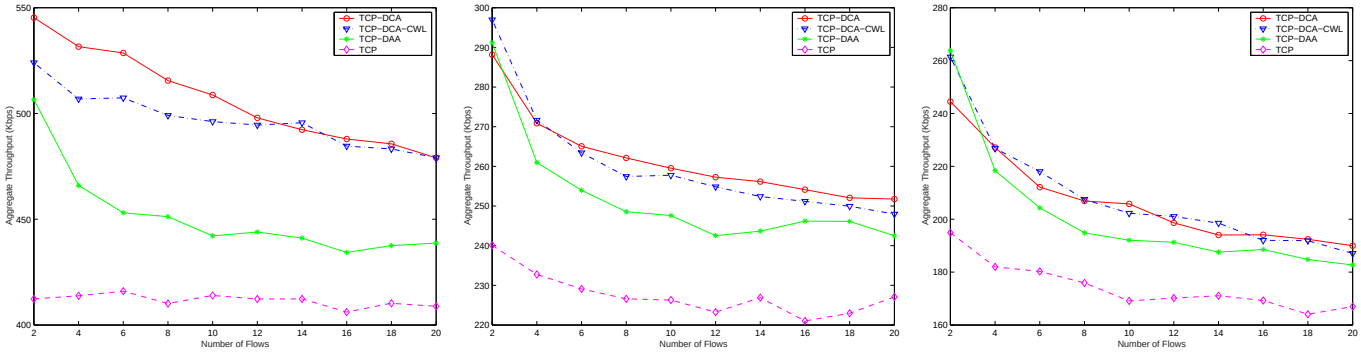
In Fig.14 we compare TCP-DAA to TCP-DCA in the chain topology with different hop count and number of concurrent flows. Although TCP-DCA is designed without congestion window limit, we show TCP-DCA with congestion window limit at 4, named TCP-DCA-CWL, for fair comparison with TCP-DAA. Fig.14 shows that TCP-DCA provides better performance than TCP-DAA in such chain topologies, except for the cases of 2 connections on 5 and 7 hop path. Overall, TCP-DCA can get improvement up to 15% over TCP-DAA, and 30% over standard TCP. Interestingly, the TCP-DCA does not perform worse than TCP-DCA-CWL, and even shows better performance in Fig.14(a)-Fig.14(b). It indicates that congestion window limit of 4 is not a good choice for TCP-DCA. The optimal congestion window limit needs further investigation for TCP with delayed ack.

We also give results for the grid topology in [2] to compare TCP-DCA with TCP-DAA. This grid topology is shown in Fig.15(a) and we only show the case with 6 flows running on the grid. The case with 3 flows has similar result, but is not provided here due to space limitation. The performance of TCP-DCA and TCP-DAA is presented in Fig.15(b). We compare TCP-DAA to TCP-DCA with and without congestion window limit equal to 4. When the congestion window size is limited by 4, the performance of TCP-DCA is similar to that of TCP-DAA, and neither scheme provides significant improvement over standard TCP. When the congestion window is not limited, TCP-DCA achieves better performance than TCP-DAA and standard TCP.

From simulations in Fig.14 and Fig.15(b), we show that TCP-DCA outperforms in most cases. Furthermore, we will demonstrate that TCP-DCA can achieve much better performance than TCP-DAA in wired/wireless networks.

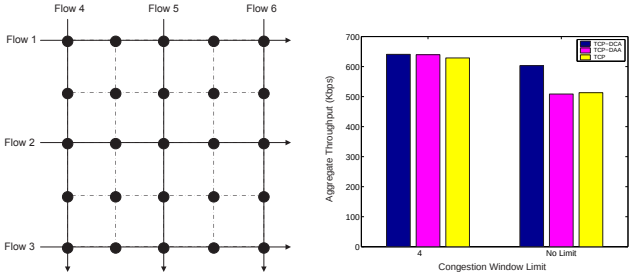
B. Hybrid (Wired and Wireless) Network Performance

As wireless networks emerge, it becomes common to communicate across wireless networks to an end node in the wired network. For example, in the real life, one needs to connect a mobile device (e.g. laptop) to the Internet to download files from a remote server, and may also need to upload files from mobiles to the Internet. Improving TCP performance over such network scenarios is important for the efficiency of applications using TCP.



(a) 3 Hop Path (b) 5 Hop Path (c) 7 Hop Path

Fig. 14. Aggregate Throughput for Chain Topology



(a) Grid Topology (b) Aggregate Throughput for Grid Topology

Fig. 15. Performance Comparison on Grid Topology

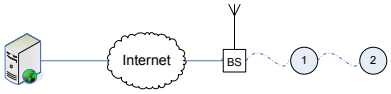


Fig. 16. Hybrid Wired/Wireless Topology

In this section we study TCP-DCA in the wired and wireless ad hoc environment. Since wired network has abundant bandwidth, it demands a much larger congestion window than the pure ad hoc network for effective TCP performance. In current TCP implementation, TCP sender increases the congestion window based only on the number of acks received while not taking into account the acknowledged data covered in the acks. Therefore, using delayed ack could potentially cause slow congestion window increase and thus hurt TCP performance. The pure adaptive delay window based on the hop count appears not suitable for this scenario because the congestion window increase rate is too slow. Although slow increase for congestion window is helpful for improving TCP performance in ad hoc networks as shown in our previous results and reported in [12], it is not desirable for wired network. In RFC3465 [18] Mark proposed to increase the congestion window based on the number of bytes acknowledged by the arriving acks rather than based on the number of acknowledgments that arrive at the sender in RFC2581 [14]. However, this approach also causes large bursts if the delay window is large. In a nutshell, an appropriate delay window limit needs to be investigated in this scenario.

Because TCP-DCA delays ack packets more than standard TCP, the congestion window increase in TCP-DCA is slower than standard TCP, particularly when large propagation delay is encountered in the wired part. To combat the inefficiency of TCP-DCA in this scenario, we propose the following approach: if minimum RTT is small, the congestion window can increase rapidly even with delayed ack, thus the receiver can adjust the delay window based on the hop count as before; otherwise, the receiver limits the delay window to a small value. Recalling that large end-to-end delay also prevents TCP from fast packet loss detection/recovery, therefore this scheme also helps in TCP recovery from packet loss. In our simulation, TCP-DCA receiver sets the delay window limit to 5 when minimum RTT is beyond 80ms. Although the RTT measurement at the receiver is potentially inflated as discussed in Section IV-D, such inflation does not cause any significant problem since only the minimum RTT is needed, and generally can be estimated accurately at the receiver.

Note that this end-to-end delay based scheme is compatible with hop based scheme, and it can also work independently. The minimum RTT on a 10 hop wireless path is about 80ms. A TCP receiver could set the delay window purely according to the end-to-end delay: when end-to-end delay is large, a small delay window limit is used, otherwise a large delay window limit is used.

In Fig.16 we show a TCP connection from a wired network to a wireless network with up to 2 wireless hops. The one-way propagation delay on the wired link is 50ms. The performance of TCP-DCA, TCP-DAA and standard TCP are shown in Fig.17. Since TCP-DAA limits the congestion window, the performance of TCP-DAA is much inferior to that of standard TCP or TCP-DCA. TCP-DCA provides better performance, about 20% throughput gain over standard TCP for both cases of 1 and 2 hop counts.

C. Mobile Ad Hoc Network

We have demonstrated that delayed ack improves TCP performance in a static network, now we show that it can improve TCP performance in a mobile network as well.

In Fig.18 we show the performance of standard TCP with

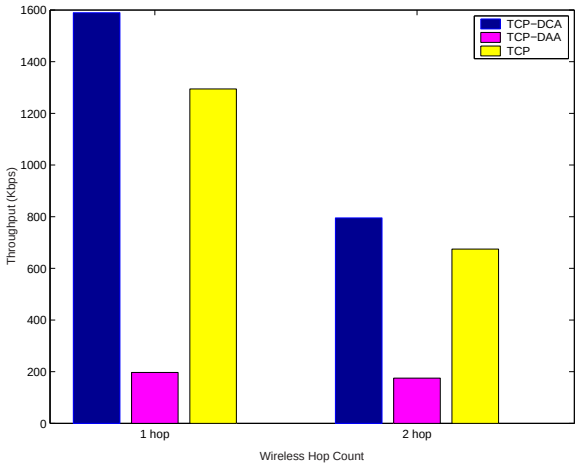


Fig. 17. Wired and Wireless Performance

and without delayed ack running over the routing schemes: AODV, DSR, and GPSR. The experiment consists of 40 nodes randomly placed within a 1000m $\times$ 1000m area and nodes move within this area. Each node moves with the same speed without pause and the speed ranges from 0m/s to 20m/s to study low to very high mobility. The Random Waypoint model is used. Notice that setting the speed to 0m/s also represents a random static topology. All results displayed are calculated as the average of five runs with the same traffic models, but with different randomly generated mobility scenarios. Since the mobility scenarios are different at different speed, we only compare TCP-DCA with standard TCP at each individual speed where they run on the same mobility scenarios. For fairness, identical mobility and traffic scenarios are used across protocols.

From Fig.18 we make two observations. First, the ack retransmission timer discussed in Section IV-D does not have much impact on TCP-DCA running over AODV and GPSR, while it has more impact on TCP-DCA running over DSR. The retransmitted acks help DSR in kicking out stale route information and improving TCP performance. The second observation is that the delayed cumulative ack contributes to remarkable performance gain over standard TCP, regardless of what routing is used. For one TCP flow in Fig.18(a), TCP-DCA can provide at least 20% performance gain across all routing protocols. For five TCP flows in Fig.18(b), TCP-DCA provides similar noticeable throughput improvement except TCP-DCA over AODV which produces a smaller improvement.

Another interesting result is that TCP over GPSR performs generally better than TCP over AODV and DSR. When node moves fast, DSR cannot adapt to the fast changing routes because of its aggressive use of route cache, thus standard TCP over DSR has very low throughput [19]. AODV shows better performance in high mobility, but the performance is not as good as GPSR. GPSR has advantages in mobile network because nodes only keep geo-locations for their neighbors, and supports end-to-end communication pattern without explicit route establishment. Therefore TCP performance on GPSR is not much affected by mobility. The disadvantage of Geo-routing is that it needs GPS and Geo-location service. For more details, please refer to [6], [20], [21]. An ad hoc routing which does not require Geo-location service, but maintains Geo-routing property is highly desired, and it is one of our future research goals.

VI. Future Work and Discussion

Delayed ack inevitably triggers burst transportation at the sender. The burstiness increases the packet loss and potentially hurts TCP performance. To reduce the burstiness, strategies to limit burstiness could be applied. For example, in [22], congestion window increase is limited by at most 2 packets for each delayed ack. However, the problem is more tricky in the wireless network because of low wireless bandwidth and pipeline effect. How to find a good limited burst size is a task for future investigation. Another possible approach to decrease burstiness is to use rate control at the sender. Certainly rate control could relax medium contention and decrease packet loss [23].

In general, the advertised window field in TCP packet header is not used by the sender since TCP connections are predominantly half duplex. We propose in TCP-DCA to let the sender reuse the advertised window field for "advertising back" its congestion window size to the receiver. An alternative solution is to imbed the congestion window size in an option field of the packet header.

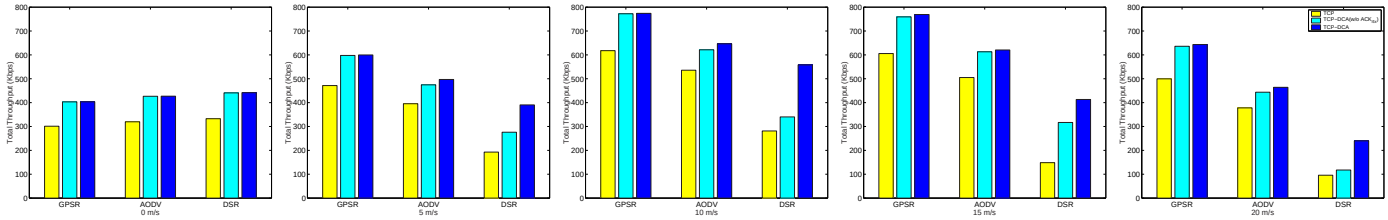
In this paper we have focused on how to reduce MAC layer interference between data and ack packets, via optimal delayed ack policy. We do not, however, address packet losses due to lossy physical channel. In the future, we plan to further investigate the lossy channel issue. We believe that TCP performance can be further improved by combining TCP-DCA with schemes that effectively combat packet losses due to error-prone physical channel, such as TCP-Westwood [24] and ELFN (Explicit Link Failure Notification) [8].

We have studied on TCP performance with delayed ack over 802.11b MAC with fixed channel rate and fixed data packet size. It is possible to extend our work to other MAC protocols, and with varying rates and packet sizes. The TCP receiver could dynamically choose a suitable delay window according to underlying MAC protocol, data rate and packet size to maximally improve TCP performance. This would be our future work on the cross-layer design in wireless networks.

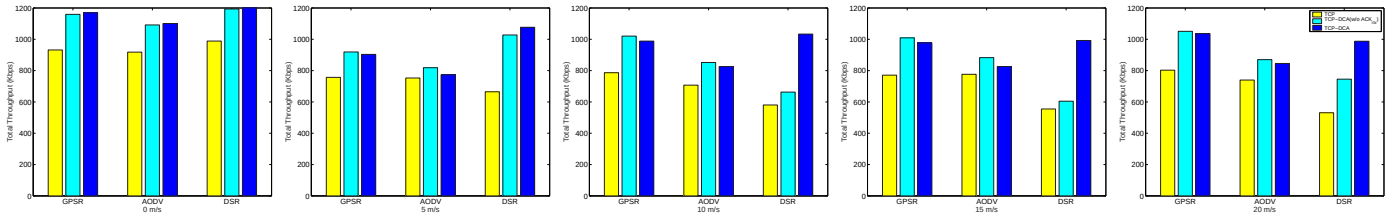
Furthermore, TCP-DCA is mainly a receiver-side modification, it can be combined with sender side modifications to achieve better performance, such as a setting of new congestion window limit suitable for TCP-DCA. It is also achievable to be integrated with other mechanisms [7], [8], [9], [10] to improve TCP performance in wireless networks.

VII. Conclusion

TCP, a dominant reliable transport protocol in wired networks, is a highly competive candidate for providing reliable data transport in wireless and wired/wireless networks. This paper systematically examines the relationship of TCP performance to delayed ack via analysis, and simulation and



(a) 1 TCP Flow



(b) 5 TCP Flow

Fig. 18. TCP performance over Mobile Network

testbed experiments. We find that TCP does not always get throughput gain by delaying unlimited acks. The maximal TCP throughput is achieved at a certain delay window that balances decreasing ack flow and burst loss. Thus, We introduced two novel adaptive delayed ack mechanisms compatible with TCP called TCP-DCA, based on the path hop length and/or end-to-end delay, to maximize the TCP performance for MANET and hybrid networks. Our proposed schemes are shown to have superior performance, achieving up to 30% gain over standard TCP in static networks (wireless or hybrid wired/wireless networks). We also demonstrated better TCP performance with our proposed mechanisms over different routing protocols in mobile ad hoc networks.

REFERENCES

[1] E. Altman and T. Jimenez, "Novel delayed ack techniques for improving tcp performance in multihop wireless networks," *Personal Wireless Communications*, September 2003.

[2] R. de Oliveira and T. Braun, "A dynamic adaptive acknowledgment strategy for tcp over multihop wireless networks," *Infocom*, 2005.

[3] T. D. Dyer and R. V. Boppana, "A comparison of tcp performance over three routing protocols for mobile ad hoc networks," *Mobihoc*, 2001.

[4] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," *Proceedings of IEEE WMCSA'99*, Feb. 1999.

[5] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," *Mobile Computing, edited by T. Imielinski and H. Korth, Chapter 5*.

[6] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," *Proceedings of ACM MobiCom'00*, Aug. 2000.

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

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

[9] K. Xu, M. Gerla, L. Qi, and Y. Shu, "Enhancing tcp fairness in ad hoc wireless networks using neighborhood red," *Mobicom*, 2003.

[10] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A feedback-based scheme for improving TCP performance in ad hoc wireless neworks," *IEEE Personal Communications Magazine*, vol. 8, no. 1, 2001.

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

[12] K. Nahm, A. Helmy, and C. J. Kuo, "Tcp over multihop 802.11 networks: Issues and performance enhancement," *Mobihoc*, 2005.

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

[14] M. Allman, "Tcp congestion control," *RFC 2581*, 1999.

[15] "The network simulator - ns2," *Available at http://www.isi.edu/nsnam/ns/*.

[16] A. Persson, C. Marcondes, and J. Chen, "Wireless chain topology: A closed network queueing approach," UCLA Computer Science, Tech. Rep., March 2005.

[17] V. Jacobson, "Tcp extensions for high performance," *RFC 1323*, 1992.

[18] M. Allman, "Tcp congestion control with appropriate byte counting (abc)," *RFC 3465*, 2003.

[19] S. R. Das, C. E. Perkins, and E. M. Royer, "Performance comparison of two on-demand routing protocols," *Proceedings of IEEE INFOCOM'00*, Mar. 2000.

[20] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," *Mobicom*, 2003.

[21] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," *Mobicom*, 2000.

[22] M. Allman, "On the generation and use of tcp acknowledgments," *ACM Computer Communication Review*, 1998.

[23] S. M. ElRakabawy, A. Klemm, and C. Lindemann, "Tcp with adaptive pacing for multihop wireless networks," *MobiHoc*, 2005.

[24] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," *Proceedings of ACM MobiCom'01*, July 2001.