TCP-MAC Interaction in Multi-hop Ad-hoc Networks

Farzaneh R. Armaghani[1] and Sudhanshu S. Jamuar[2]
[1]*Monash University, GSIT,*
[2]*Department of Electrical Engineering, University of Malaya,*
[1]*Australia*
[2]*Malaysia*

1. Introduction

Recent demands on affordable, portable wireless communication and computation devices have resulted in exponential growth of wireless networks ranging from Wireless Local Area Networks (WLAN) and Wireless Wide Area Networks (WWAN) to Ad-Hoc and Sensor networks. The major goal of wireless communication is to allow users to communicate together and to have access to global network anytime anywhere. This has led to wide acceptance of infrastructure based cellular networks (WWANs) where mobile stations communicate with a centralized controller, often referred as Access Point (AP) that is connected to the wired networks. On the other hand, WLANs have appeared as dominant popular technologies in many venues including a local area such as an academic campus or an airport terminal. These wireless networks mostly rely on IEEE 802.11 Wi-Fi (Wireless Fidelity) technology and its various derived versions (i.e. 802.11a,b,g).

IEEE 802.11 standard supports two operational modes: The infrastructure-based Wireless Local Area Networks (WLANs) and an infrastructure-less Ad-Hoc Networks. A WLAN (Conti, 2003) typically imposes the existence of an AP and normally is connected to the wired networks to provide internet access for mobile devices. Obviously, only one hop link is needed to communicate between mobile devices and AP. In contrast, there is no AP or infrastructure in Ad-Hoc networks. Any two stations can communicate directly when they are in the range of reception of each other. To this end, the stations may use multi-hop routing to deliver their packets to destinations. The ad-hoc protocols (Conti, 2003; Mohapatra & Krishnamurthy, 2005) are self-configured for address and routing in the face of mobility and the network topology may change in each configuration. The multi-hop wireless ad-hoc networks, or multi-hop wireless networks enable wireless networking in the environments where the wired or cellular connections are impossible, inadequate, or cost effective (e.g. battle field, disaster recovery, etc.).

The popularity of internet over the last decades has resulted in rapid advancement of demanding applications. The Transmission Control Protocol/Internet Protocol (TCP/IP) (Stevens, 1994) is a well-known de facto protocol in developing today's internet. Basically, TCP provides a connection-oriented and reliable end-to-end data delivery between two hosts in traditional wired networks. Since TCP is well tuned and due to its wide acceptance in internet, it is desirable to extend and adopt its functionality to wireless networks. On the

other hand, unique characteristics and usage of multi-hop wireless networks require robust, reliable and adaptive designs. This may be achieved by considering the interaction of different layers to meet the increasing demands of these networks.
The reliability in TCP is achieved by retransmitting lost packets and acknowledgment (ACK) confirmation. If the sender does not receive any acknowledgment within a timeout interval or receives duplicate ACKs in the case of out-of-order packets, the packet will be retransmitted. Any packet loss is assumed as congestion in wired networks. When a packet loss is detected, TCP invokes its congestion control mechanism to slow down the sending rate to reduce the congestion. However, packet losses are not mainly due to congestion in wireless networks. It might be due to some wireless specific properties such as high medium access contention, route breakage and high bit error rate in radio channels (Hanbali, Altman, & Nain, 2005; Xiang, Hongqiang, & Jiangfeng, 2005).
The key challenge of TCP protocol is its poor bandwidth utilization and performance when it runs over 802.11 multi-hop wireless networks. The reason can be explained due to the extensive number of medium access carried out by TCP. Basically, TCP sender will be informed of successful transmission by receiving the acknowledgment from the other end host. The MAC overhead can be caused by generating redundant ACK packets that compete in the same route with data packets for the media. Although the TCP-ACK packets are small, they may cause the same overhead as data packets in MAC layer resulting in wastage of wireless resources (Altman & Jimenez, 2003; de Oliveira & Braun, 2007). In fact, the short RTS/CTS control frames to provide the data delivery implemented by 802.11 MAC protocol, cannot eliminate the interference in large topologies (Xu, Gerla, & Bae, 2002). As load increases, the well-known hidden terminal effects caused by interference between ACK and data packets can impact TCP performance dramatically in long paths if TCP acknowledges every incoming data packets. One way to improve the TCP performance over 802.11 in multi-hop ad-hoc networks is to alleviate the medium access contention by reducing the number of generated ACKs, simply called as *delayed ACKs*. This can be done by merging several ACKs in one ACK which is possible due to cumulative ACK scheme used in TCP. Referring to already proposed approaches to reduce the number of induced ACKs, the TCP performance is still affected by a limitation of a method which dynamically selects the number of delayed ACKs based on the channel condition (Altman & Jimenez, 2003; de Oliveira & Braun, 2007). This motivates us to study the performance of TCP-ACKs in interaction with 802.11 over the multi-hop ad-hoc networks and develop a dynamic delayed ACK strategy to adjust TCP to these kinds of networks.

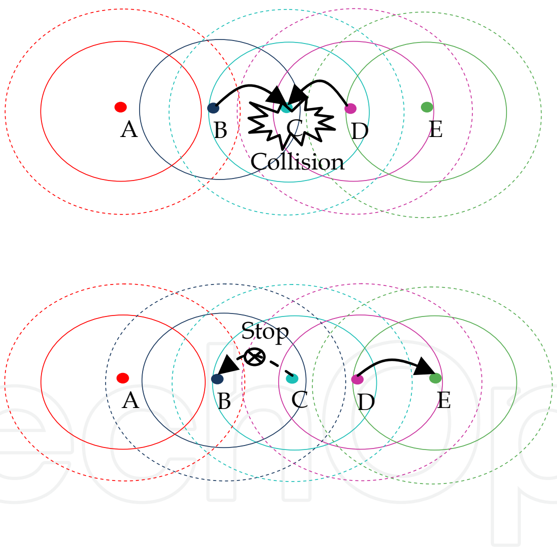
2. IEEE 802.11 challenges

The unique characteristics of ad-hoc designs impose several challenges in comparison with single hop networks such as cellular networks or WLANs, when they run over 802.11 MAC protocol.
The most serious challenge is the RTS/CTS handshaking implemented in 802.11, which is not efficient enough to prevent collisions due to large distribution of mobile nodes and multi-hop function in ad-hoc networks. It has been proved through analytical model and simulation experiments (Fu, et al., 2005; Xu, et al., 2002) that RTS/CTS cannot function well in topologies more than three hops (3 hop scenario) between sender and receiver. For larger number of hops, the RTS/CTS exchange cannot prevent the existence of famous hidden node problem which is discussed later.

Mobile nature of ad-hoc networks, where each node may experience different degree of channel contention and collision, is another problem, (Zhai et al., 2006). The interaction between the MAC and higher layers has a significant effect on the network performance. The interaction between MAC and TCP layer is investigated throughout this chapter.

2.1 Medium contention and spatial reuse

The Hidden and Exposed terminals are defined based on transmission range and sensing range of the nodes (Xu, et al., 2002). Transmission range represents the range within which a packet is successfully received if there is no interference from other radios. Sensing range is the range within which a transmitter triggers carrier sense detection to sense an ongoing signal. Spatial reuse facilitates maximum possible non-conflicting simultaneous transmissions in MAC layer. A hidden terminal is the one that can neither sense the transmission of a transmitter nor correctly receive the reservation packet (i.e. CTS control frame) from its corresponding receiver (Zhai, et al., 2006). In other words, a hidden terminal is a node that is within the transmission range of a receiver but out of the sensing range of an intended transmitter. Therefore, it can interfere with an ongoing transmission at the receiver by transmitting at the same time. Consider the scenario illustrated in Fig. 1a to see the cause of hidden nodes. Here node D is a hidden terminal while B is transmitting to C because it is out of B's sensing range. Therefore, D's transmission collides with RTS reception in C. After seven attempts, both B and D assume that C is unreachable and the packets will be dropped. This leads to bandwidth wastage as both data transmissions are destroyed.



(a) Hidden terminals

(b) Exposed terminals

Fig. 1. Contention and spatial reuse

To achieve high channel utilization, MAC needs to maximize the spatial reuse (Zhai, et al., 2006). Exposed terminal problem is a factor in 802.11 influencing the spatial reuse and is a problem which is caused by a terminal that is within the sensing range of a transmitter and can not interfere with the reception of the receiver; but it would not be able to start a transmission because it senses a busy media. As depicted in Fig. 1b, node C is considered as an exposed terminal when D is transmitting to E. C senses the medium as busy and it has to keep silence, even though it can transmit to B which is out of D's sensing range. In short, the hidden terminals reduce the network capacity due to increase in number of collisions, while exposed terminals affect the spatial reuse due to unnecessary deferring nodes from transmitting.

3. TCP-MAC interaction in multi-hop ad-hoc networks

TCP interaction with lower layers includes the proposals that address the inability of TCP to distinguish between losses due to route failures and network congestion. These proposals involving the network layer suggest notifying the TCP sender about routing failure, when the routing layer detects a route failure (Dongkyun, Toh, & Yanghee, 2000; Holland & Vaidya, 2002; Liu & Singh, 2001; Wang & Zhang, 2002; Yu, 2004). On the other hand, considerable research (Fu, et al., 2005; Gerla, Bagrodia, Lixia, Tang, & Lan, 1999; Gerla, Tang, & Bagrodia, 1999; Khalife & Malouch, 2006; S. Xu & T. Saadawi, 2001) has been carried out to show how TCP performance is significantly affected by MAC protocols in multi-hop ad-hoc networks. We discuss these proposals that address the interaction between TCP and MAC in this chapter.
The fundamental problem in interaction between 802.11 MAC and TCP arises from the impact of hidden and exposed terminals on TCP congestion control mechanism and the impact of TCP transmission rate and ACKs' overhead on existence of hidden and exposed terminals.

3.1 Impact of hidden terminal and exposed terminal problem

The RTS/CTS control frames implemented in 802.11 MAC protocol can not prevent the hidden and exposed terminal problems in the scenarios with more than three numbers of hops.
Normally, when there is an unsuccessful RTS transmission either due to collision or due to an unnecessary deferred transmission; the sender in MAC layer enters a backoff period and it reschedules its RTS transmission when its backoff timer expires. After seven successive unsuccessful attempts, it is assumed that the route has failed and the packet is dropped. The effect of this wrong route failure report on TCP operation is not negligible. In this case, the sender tries to find a new route to destination. If the route takes some time to restore, TCP enters its backoff state and probes for a restored route at increasingly longer time intervals. Hence, the route might be restored for quite some time but TCP remains idle until it retransmits the packet after its time out expiration. Same condition happens when a route failure is reported after four unsuccessful attempts to transmit a data packet in MAC layer.
Therefore, hidden and exposed terminals may cause a lack of ACKs at TCP sender, leading it to retransmit by timeout (de Oliveira & Braun, 2007). As a consequence, TCP invokes its slow start mechanism to slow down its transmission rate to the lowest level instead of fast retransmit. In other words, TCP may not receive three duplicate ACKs due to its small Congestion Window size (*cwnd)* at most of the times. *Cwnd* is defined as maximum number of data packets a TCP sender may inject into the network at anytime without waiting for an ACK from the receiver. Considering this problem, TCP end-to-end throughput may decreases significantly as the number of the hops grow due to considerable delay of waiting for the transmit timer to expire.

3.2 Impact of TCP transmission rate

Another major problem in interaction between TCP and MAC is based on the probability of packet dropping due to increase in link contention as the TCP offered load increases (Fu, et al., 2005). In fact, TCP keeps sending more packets during congestion avoidance phase while a node is trying to access the medium within the MAC retry limits. The increment of sending rate continues until TCP perceives any packet loss indication. As a result, the channel condition may be aggravated because there are more outstanding packets intended to obtain the channel simultaneously. Consequently, TCP experiences an incredible throughput decrease due to the limited spatial reuse imposed by 802.11 MAC protocol.

3.3 TCP redundant ACKs

The issue of spatial contention in multi-hop wireless networks, which is aggravated by hidden and exposed terminals, can be caused by generating redundant TCP acknowledgments. It is noted that although TCP-ACKs are much smaller than TCP data packets, their transmission requires the same signalling overhead of the 802.11 MAC protocol (Altman & Jimenez, 2003). In fact, the receiver must content for the medium using RTS/CTS frames for ACK transmissions exactly as the sender does for data transmissions. The problem can be explained through Fig. 2 in which node D is a hidden terminal when B is transmitting data packets to C.

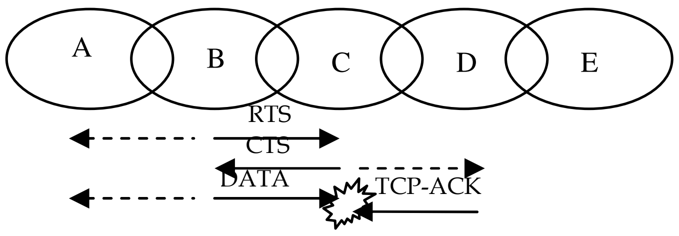


Fig. 2. Collision between DATA and TCP-ACK

After a successful RTS and CTS handshaking, B starts to send the data. Meanwhile, a TCP acknowledgment is flowing back from D to C. As it is shown in the Fig. 2, D's ACK transmission will interfere with C's data reception and TCP acknowledgment will be dropped. When TCP does not receive any ACK in a time interval, it assumes a data packet is lost and invokes its congestion control mechanism. When it comes down to TCP throughput, serious degradation will be observed.

4. TCP modifications over MAC layer in ad-hoc networks

In recent years, there has been an increasing amount of literature available on optimizing TCP performance in interaction with MAC layer over multi-hop wireless networks at both TCP sender and receiver side (De Oliveira & Braun, 2005, 2007; Fu, et al., 2005; Hamadani & Rakocevic, 2005) in which all of them try to minimize the effect of MAC overhead caused by spurious TCP retransmissions, TCP sending rate, and ACK packets. In order to mitigate the spatial contention, the proposal is to apply changes to the TCP *cwnd* size at the sender end. In contrast, proposals at receiver side aim to reduce the MAC overhead and ACK traffic by alternations on TCP acknowledgment strategy or advertised window (*rwin*).

4.1 Limiting TCP's packet output

The effect of 802.11 MAC protocol interferences on TCP is extensively studied (Fu, et al., 2005). They have suggested that for a given specific network topology there is an optimum *cwnd* size, which maximizes TCP throughput by improving the spatial reuse property of wireless networks. It is stated that in a chain topology consisting of h number of hops, the maximum number of simultaneous transmission is upper bounded by $\frac{h}{4}$, at which maximum spatial channel reuse is achieved (Fu, et al., 2005). Therefore, TCP achieves the highest throughput with its *cwnd* size being $\frac{h}{4}$ in an h-hop chain topology to limit the number of outstanding packets over the entire forwarding path.

TCP continuously increases its *cwnd* size until a packet loss is detected. It typically operates at an average window size that is larger than optimal, thereby leading to dropped packets caused by link layer contention. Therefore, two link layer modifications named *Link RED* and *adaptive pacing* is proposed to maintain the optimum *cwnd* size at TCP sender and to reduce the medium contention. The Link RED algorithm aims to reduce the contention on the wireless channel by monitoring the average number of retransmissions. The probability of dropping a packet is computed when this average number becomes greater than a threshold. The goal of adaptive pacing is to improve the spatial channel reuse when the *cwnd* size exceeds the optimum value. This mechanism is enabled within the Link RED and a node increases its backoff timer in MAC layer when it notices that the threshold has reached.

In another major study, Chen et al. (K. Chen, Xue, & Nahrstedt, 2003) have investigated the impact of the TCP's Congestion Window Limit (CWL) on TCP throughput by taking up the Fu et al.'s observations. Based on this claim that the impact of return path is not considered in (Fu, et al., 2005), a dynamic mechanism known as dynamic CWL is proposed in which the CWL depends on the Bandwidth Delay Product (BDP) of the connection. It is discussed that regardless of the MAC protocol, the BDP cannot exceed the Round-Trip Hop-Count (RTHC) in a wireless multi-hop network. In case of 802.11 MAC protocol, authors report that the BDP is less than $\frac{1}{4}$ of the RHTC as only four hops nodes away can transmit concurrently without collisions. As a result, when the *cwnd* exceeds $\frac{RHTC}{5}$, the TCP throughput decreases substantially. However, one major drawback is that the maximum retransmission timeout in TCP is set to 2s as opposed to the 240s which is given in the standard. This might affect the simulation result.

The maximum retransmission timeout in dynamic CWL has improved in (Papanastasiou & Ould-Khaoua, 2004), where it is proposed to throttle the sending rate increase during the congestion avoidance phase to a level below the standard of one segment per RTT. In this way, no upper bound is forced to *cwnd* as it is done in CWL and dynamic CWL and a significant improvement above dynamic CWL has been achieved by realistic setting of the maximum RTO. One can see that limiting the *cwnd* to slow down the transmission rate has been conclusively shown as an effective solution for spatial contention in MAC layer and consequent TCP throughput optimization.

4.2 Managing a shared medium

As discussed earlier, TCP performance suffers from an inefficient spatial channel usage when multiple flows are trying to access the shared radio channel leading to severe unfairness problem (Boggia et al., 2005). It is mentioned that, during the probing phase for available network bandwidth, TCP allows a large number of segments to be outstanding which in turn generates a high collision probability when TCP flows go through an 802.11 ad-hoc networks. A receiver-side approach has been then proposed by Boggia et el. to exploit the *rwin* filed of TCP segments to limit the number of in-flight packets in the network based on the medium condition. In this paper, a cross-layer algorithm which collects the total frame collision probability in MAC layer along the path has been proposed. The measured probability is then communicated to the TCP receiver to properly set the *rwin*. When there is not much collision in MAC layer, the total collision probability value is less than a threshold, *rwin* is increased exponentially by one segment. However, in a high congested media, the increment is much slower. Upon this strategy, a reasonable tradeoff has been achieved between TCP throughput and fairness. But the ratio of retransmitted segments is strongly reduced and the throughput does not improve significantly and stays similar to that obtained using NewReno. These findings suggest that by monitoring the channel condition, we can control the MAC overhead caused by spurious TCP retransmissions and redundant ACKs, resulting in a better TCP performance.

4.3 ACK thinning techniques

The sender's transmission of TCP-DATA segments and the receiver's ACK response contribute to spatial contention while TCP pair are communicating. The mechanisms dealing with the reciprocal ACK response aim to reduce the amount of ACK traffic for an optimized spatial contention. The first optimization of this nature has been introduced through standard TCP with delayed ACK option by delaying ACKs upon receiving two in-order data packets. It has been proved through extensive simulations (Lilakiatsakun & Seneviratne, 2003; S Xu & T Saadawi, 2001) that well-known TCP variants including Reno, New Reno, SACK and Vegas can perform better in case of throughput, bandwidth and energy consumption by employing the delayed ACKs. It has also been shown that in special cases the improvement in TCP throughput is in the range of 15-32% by deploying the optional delayed ACK mechanism (S Xu & T Saadawi, 2001). Such findings motivate the recent studies on more investigation over the degree of improvement that can be achievable with ACK thinning mechanisms.

The study of the behavior of delaying more than two ACKs over 802.11 MAC protocol was first carried out by Altman et al (Altman & Jimenez, 2003) in which the idea of standard delayed ACKs, which is combination of only two consecutive ACKs, has been extended. In the proposed scheme called TCP-LDA (Large Delayed Acknowledgment), an acknowledgment is sent only after a given number *d* of segments or after a certain fixed timeout. The dynamic aspect of TCP-LDA operates with *d* growing with an increasing packet sequence number from one up to $d = 4$. Once this limit is reached, the delay window (*dwin*) size, i.e. the number of the ACKs to be combined, is considered as fixed at 4 data packets even though a timer expires at TCP sender side. This may lead to the shortage of ACK phenomenon which is a condition with a bigger *dwin* size than a *cwnd* size. In this condition, a sender does not receive any ACK in a time interval and is just able to transmit upon a timeout. Moreover, TCP-LDA is not adaptable to different channel conditions and out-of-order packets are not taken into account. This means that when any indication of an out-of-order packet or dropped packet is received, *dwin* is never decremented again and still is fixed at 4 packets leading to poor TCP performance.

Singh et al (Singh & Kankipati, 2004) have shown enhancement in TCP performance and have derived the relation between throughput and number of data packets covered in one ACK. Based on the analysis, they have proposed TCP-ADA (Adaptive Delayed Acknowledgment) scheme, which tries to decrease the number of ACKs to one per congestion window. However, employing a large *dwin* size equal to *cwnd* size is not an efficient solution in all scenarios resulting in the burstiness of the forwarding packets in long paths. In this case, too many data packets are queued at TCP sender side, waiting for an acknowledgment to be received inducing packet drops in the router's buffer. Also they have not addressed the effect of packet loss event and out-of-order packets in the *dwin* size adjustment.

TCP-DCA (Delayed Cumulative Acknowledgment) (J. Chen et al., 2008) has been proposed to decrease the number of ACKs based on the path length. This study reveals that for a given topology and flow pattern, there exists an optimal delay window size at receiver that produces best TCP throughput. It is shown that path length is an important factor in choosing the *dwin* size because when travelling a longer path, a packet is more likely to suffer interference. For a 3 hop scenario in hidden terminal problem, it may rarely happen that the *dwin* size limit is equal to *cwnd* size. However, for long paths a high *dwin* limit aggravates the channel contention. If data packets arrive in order, the receiver generates one cumulative ACK for every *d* data packets. In case of an out of order packet, the receiver acknowledges immediately without any delay. To get the delay timer period of ACK timeout, the receiver monitors the packet inter-arrival interval and computes a smooth inter-arrival. Moreover, in TCP-DCA the sender reuses the advertised window (*rwin*) field in data packet header for advertising its *cwnd* size to the receiver to prevent the shortage of ACKs phenomenon. The modifications are at receiver side and the mechanism is simulated in scenarios with medium traffics while the higher loaded traffics are not taken into consideration. Table 1 shows these findings on the optimized *dwin* size based on the path length.

Path Length (h)	*dwin* limit
$h \leq 3$	*Cwnd*
$3 < h \leq 9$	5
$h \geq 10$	3

Table 1. Optimized delay window size in different path length

Olivera et al (De Oliveira & Braun, 2005, 2007) draws our attention to some drawbacks and limitations in the previous literature. The authors have improved the fix number of 4 packets handled with a single ACK after the start-up phase and the fixed ACK timeout in Altman and Jimenez’s scheme.

The proposed dynamic delayed acknowledgment scheme called TCP-DAA (Dynamic Adaptive Acknowledgment) of Altman and Jimenez’s applies the concept proposed in RFC 2581 by sending an immediate acknowledgment upon out-of-order packets or packets filling a gap at the receiver. This means that TCP-DAA is adaptive in the term of packet losses in the channel. The receiver maintains a dynamic delaying window (*dwin*) with size ranging from 2 to 4 full sized segments in networks with medium traffics which determines when an ACK will be produced. When a packet loss is observed; the *dwin* breaks down to two ACK packets. When there is no more losses reported; TCP enlarges *dwin* by one up to 4 packets. That means TCP receiver waits for 4 data packets before generating the ACK. To this end, the receiver implements an *ack-count* variable which increases by one until it reaches to the current value of *dwin* whenever a consecutive data packet is received.

To deal with the high delay variations in wireless ad-hoc environments, an effective mechanism has been employed to set the ACK timeout based on the packet inter-arrival times at the receiver. The rational is, in case of a single dropped packet, the next DATA packet will arrive out-of-order, thus triggerring immediate transmission of an ACK. However, if it was only a delay variation and the DATA packet arrives before the expected time for the subsequent packet; no timeout is triggered and the receiver avoids sending an extra and unnecessary ACK packet into the network (De Oliveira & Braun, 2005). The DAA method has been evaluated on string and mesh topologies of varying lengths and different number of flows. The results show an optimization in TCP throughput up to 50% over plain TCP NewReno on string topologies up to 8 hops and 20 flows (De Oliveira & Braun, 2005).

Later in (de Oliveira & Braun, 2007), an extension has been proposed to TCP-DAA concerning the robustness in high traffic environments with considerable packet losses. It has been pointed out that TCP-DAA may be inefficient in such environments. Based on these observations, an enhanced mechanism is suggested which is more adaptable in high traffic channels. In proposed mechanism called TCP-DAAp (TCP-DAA plus), *dwin* enlarges more gradually by a factor between zero and one to its limit of 4 packets to provide enough ACKs to the TCP sender. Additionally, *dwin* is reduced to one packet instead of two as a reaction to packet losses. This behavior is more conservative in comparison with TCP-DAA to prevent from transmitting a burst of data and the shortage of ACK phenomenon due to the small size of *cwnd* in such high loaded lossy environments. The achieved results obtained on a chain topology confirm that TCP-DAAp is as robust as the regular TCP mechanism under heavily constraint environments; however it does not provide the same improvements of TCP-DAA.

The main drawback of this work is that receiver is not dynamically notified to use TCP-DAA or TCP-DAAp in the scenarios mix of moderate and high traffic. Therefore, although the results have shown a significant and robust improvement, respectively in moderate and high traffics, the algorithm is not applicable in the real world where the traffic may change from time to time. To this end, it has been suggested to have an additional monitoring mechanism at the receiver to adjust the TCP-DAA strategy on the basis of channel condition (de Oliveira & Braun, 2007). Thus an adaptive receiver mechanism to switch between DAA and DAAp strategies in scenarios susceptible with high losses; is considered as future work by de Oliver and Braun (de Oliveira & Braun, 2007).

5. Monitoring Delayed Acknowledgment (TCP-MDA)

TCP-MDA dynamically reacts to the existing traffic in the network unlike the TCP-DAA. It can delay more ACKs in low load channels and less in the high traffics. Basically, in a high traffic network, it's more conservative to provide enough ACKs to TCP sender as there are more data packets intended to achieve the channel simultaneously. This is prominent in a long path topology where the spatial reuse property is limited due to hidden terminals and packet loss is more common to happen.

5.1 MAC collision probability measurement

The method implemented in TCP-MDA to measure the collision probability in 802.11 is based on the procedure used in (Boggia, et al., 2005) in which the measured collision probability has been employed to set the advertised window field of TCP to slow down the transmission rate. The same idea has been used in TCP-MDA to properly set the number of delayed ACKs as it is shown in Fig. 3.

A field called as $non\_collision\_prob_i^{tot}$ in the MAC protocol header has been added to meet our requirements. This field is set to 1 in the first hop of the path where no collision is detected. It should be recalled that TCP's functionality is based on the end hosts and it does not need any support from the intermediate nodes. Hence, the first hop is the hop which holds the source of the flow and the last hop is considered as TCP destination node. On the other hand, 802.11 is involved with the intermediate nodes as well as the source and destination nodes due to its responsibility for node-to-node delivery.

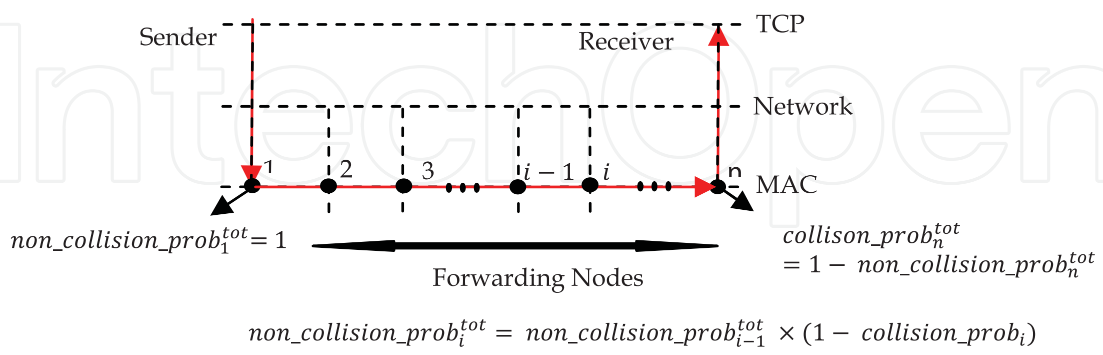

Sender
Receiver
TCP
Network
MAC
1
2
3
i − 1
i
n
$non\_collision\_prob_1^{tot} = 1$
Forwarding Nodes
$collison\_prob_n^{tot} = 1 - non\_collision\_prob_n^{tot}$
$non\_collision\_prob_i^{tot} = non\_collision\_prob_{i-1}^{tot} \times (1 - collision\_prob_i)$

Fig. 3. MAC collision probability measurement

In TCP-MDA, TCP layer sends the SYN segment when a connection initiates. The MAC layer is conscious about flow start by receiving SYN at the first node and sets the $non\_collision\_prob_1^{tot} = 1$. For the same reason, TCP can interact with MAC layer at destination node, allowing the evaluation of the final collision probability and sends it to the TCP layer for the delayed ACK adjustment.

As the packet is traversing the path from sender to receiver in 802.11, the $non\_collision\_prob_i^{tot}$ field is calculated by taking into account the collision probability at each intermediate node (Boggia, et al., 2005).

We define $collision\_prob_i$ as the local collision probability at the ith forwarding node and it is given by the ratio between the number of retransmitted data packets and the total number of transmitted data packets.

The $non\_collision\_prob_i^{tot}$ is not a local value. It is defined as the product of collision probability at each node from sender to ith node in a given path. Hence, in order to obtain the $non\_collision\_prob_i^{tot}$, it is needed to multiply the local non collision probability of the ith forwarding node which is $(1 - collision\_prob_i)$, with the overall non collision probability written in the header of the packet that is being forwarded from the (i – 1)th node. To this end, when a packet is received at node i, the overall $non\_collision\_prob_{i-1}^{tot}$ is read from the packet header. Then for each transmitted frame, the local non collision probability is estimated and the computed overall $non\_collision\_prob_i^{tot}$ is written back into the packet header. This procedure continues until the packet reaches to the last node and the $non\_collision\_prob_i^{tot}$ is given as:

$$non\_collision\_prob_i^{tot} = non\_collision\_prob_{i-1}^{tot} \times (1 - collision\_prob_i) \qquad (1)$$

Finally, the total collision probability in the path at the last node is computed by using the value of total non collision probability which is read from the packet header. This is given as:

$$collision\_prob_n^{tot} = 1 - non\_collision\_prob_n^{tot} \qquad (2)$$

It should be noted that all the calculated values are considered for the data packets and the transmission of RTC/CTS control frames is not taken into account. The pseudo-code depicted in Fig. 4 describes the whole process at one node.

```
collision_prob = 0;
transmitted_pkts = 0;
retransmitted_pkts = 0;
1.  for (each data packet)
        // Packet is received at a node
2.      if (1st node of the path)
3.          $non\_collision\_prob_1^{tot}=1$;
4.      else
5.          $non\_collision\_prob_{i-1}^{tot}$ is read from the packet header;
6.          $collision\_prob_i = retransmitted\_pkts_i / transmitted\_pkts_i$;
7.          $non\_collision\_prob_i^{tot}=non\_collision\_prob_{i-1}^{tot}*(1-collision\_prob_i)$;
8.      end if
9.      if (last node of the path)
10.         $collision\_prob_n^{tot}=1-non\_collision\_prob_n^{tot}$;
11.     end if
12.     if retransmitted packet
13.         $retransmitted\_pkts_i$ + +;
14.     end if
15.     $transmitted\_pkts_i$ + +;
16. end for
```

Fig. 4. Packet processing at a single node to collect the collision probability

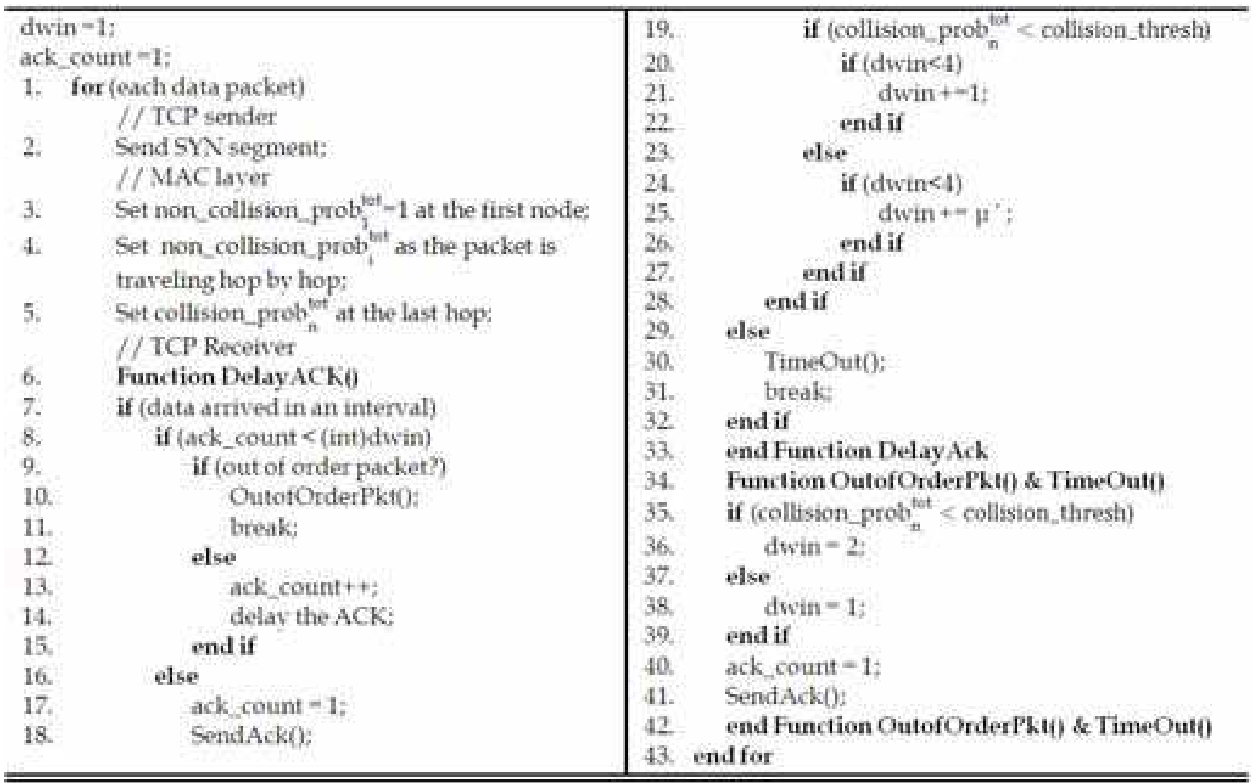
5.2 Delaying window strategy

The ACK processing in TCP-MDA is dependent on the calculated collision probability, *total_collision_prob*, in different channel traffics. Withholding ACK responses is done by maintaining a dynamic delaying window (*dwin*) at TCP receiver to define the number of data packets that would arrive before generating an ACK.

Like TCP-DAA, *dwin* size is initialized to one and it is gradually enlarged to its limit of 4 data packets. When the achieved *total_collision_prob* from MAC layer is less than a threshold (*collision_ thresh*), the channel is considered in the good condition and *dwin* is incremented by one for every received data packets. This means that *dwin* would become 4 faster and the receiver would generate less ACKs. It would be advantageous then to keep *dwin* at 4 as long as the channel is stable. When facing losses, however, *dwin* should be reduced due to the fact that during these periods the channel may have less packets than 4 in flight to trigger the fast retransmit mechanism at the sender. As a result, the channel may timeout if the receiver ACKs are not obtained quickly.

When receiver gets any indication of packet loss or the packet is overly delayed during transit, *dwin* reduces to two packets and again enlarges by one packet into its limit in low traffic channels. The reason to resume *dwin* growth from two instead of one is to go back to a behavior similar to that of the standard delayed acknowledgment (DA) in such situations, which performs better than configurations without it (de Oliveira & Braun, 2007). Figure 5 depicts the pseudo-code of TCP-MDA when a packet arrives at receiver.

To track the number of the delayed ACKs, TCP receiver maintains an *ack_count* variable ranging from one to the current value of *dwin.* Whenever a consecutive data packet is received, *ack_count* variable is increased by one. In this way, when *ack_count* = *dwin,* an ACK response is, immediately produced and *ack_count* is reset to one. It signifies the beginning of the next group of data packets for which the corresponding ACKs will be delayed. In fact, *ack_count* differentiates between each group of data packets.
It is also desirable to produce quick ACK responses so as to allow an increase of sending rate during the slow start phase at the sender. If ACKs are delayed too much during this phase, the sender would not receive enough ACKs to increase its sending rate efficiently due to the ACK requirements of TCP sender to clock out the data. A speeding factor μ, with $0 < \mu < 1$ is considered to enlarge the *dwin* in the startup phase instead of a fixed value of one. Additionally, *maxdwin* is considered as an indicator which turns true when the slow start phase is over and *dwin* reaches its maximum value of 4. Once the *maxdwin* is reached, then this mechanism is not activated again for the same connection. Hence this facility is for short life flows (de Oliveira & Braun, 2007).



```
dwin =1;
ack_count =1;
1.  for (each data packet)
        // TCP sender
2.      Send SYN segment;
        // MAC layer
3.      Set non_collision_prob$_1^{tot}$=1 at the first node;
4.      Set non_collision_prob$_i^{tot}$ as the packet is
        traveling hop by hop;
5.      Set collision_prob$_n^{tot}$ at the last hop;
        // TCP Receiver
6.      Function DelayACK()
7.      if (data arrived in an interval)
8.          if (ack_count < (int)dwin)
9.              if (out of order packet?)
10.                 OutofOrderPkt();
11.                 break;
12.             else
13.                 ack_count++;
14.                 delay the ACK;
15.             end if
16.         else
17.             ack_count = 1;
18.             SendAck();
19.             if (collision_prob$_n^{tot}$ < collision_thresh)
20.                 if (dwin<4)
21.                     dwin +=1;
22.                 end if
23.             else
24.                 if (dwin<4)
25.                     dwin += $\mu'$;
26.                 end if
27.             end if
28.         end if
29.     else
30.         TimeOut();
31.         break;
32.     end if
33.     end Function DelayAck
34.     Function OutofOrderPkt() & TimeOut()
35.     if (collision_prob$_n^{tot}$ < collision_thresh)
36.         dwin = 2;
37.     else
38.         dwin = 1;
39.     end if
40.     ack_count = 1;
41.     SendAck();
42.     end Function OutofOrderPkt() & TimeOut()
43. end for
```

Fig. 5. TCP-MDA pseudo-code

The mechanism described above works well in moderate traffics; however, when the loss rates are considerable, it is desirable to enlarge *dwin* slowly to provide enough ACKs to TCP sender as there are more packets intended to achieve the channel. To meet this design, when *total_collision_prob* exceeds the *collision_thresh*, *dwin* is incremented by a factor μ' between zero and one. This is more aggressive in conditions with considerable losses due to small *cwnd* size in most of the times. In fact, *cwnd* size will be cut when a packet loss is perceived by a TCP sender. Thus, we need to provide enough ACKs to the corresponding sender to prevent from a transmission upon the timeout and to prevent from a bigger *dwin* size than *cwnd* size. For the same reason, it is more appropriate to reduce *dwin* to one instead of two as a reaction to packet loss. The optimized value for the *collision_thresh* is obtained through different simulation results which show the best value among all the indexes in (Armaghani et al., 2008). Figure 5 illustrates the pseudo code of the whole algorithm.

5.3 ACK timeout computation

For every successful delivered data and ACK packets, MDA method allows 4 data packets to produce one ACK response. However, it is desirable to trigger an immediate ACK without waiting an *ack_count* to reach the current *dwin* when a data packet is overly delayed during transmission. The ACK timeout is computed by the means of packets' inter-arrival time (Fig. 6).

That is, an ACK is generated when no data packets arrive within an average inter-arrival time since the last unacknowledged data packet. Therefore, an inter-arrival time gap between each received data packet which an ACK is to be delayed, say $i - 1$, i, $i + 1$, … , and the previous data reception is recorded as δ_{i-1}, δ_i, δ_{i+1}, … .

It should be noted that the inter-arrival time between each data group is not taken into account. These collected inter-arrival time periods are used to calculate a smoothed average to estimate an expected inter-arrival time, $\overline{\delta_i}$ as given by following equation:

$$\overline{\delta_i} = \alpha \times \overline{\delta_{i-1}} + (1-\alpha) \times \delta_i \tag{3}$$

$\overline{\delta_{i-1}}$ is the last calculated average, δ_i is the data packet inter-arrival time sampled and $0 < \alpha < 1$ is an inter-arrival smoothing factor.

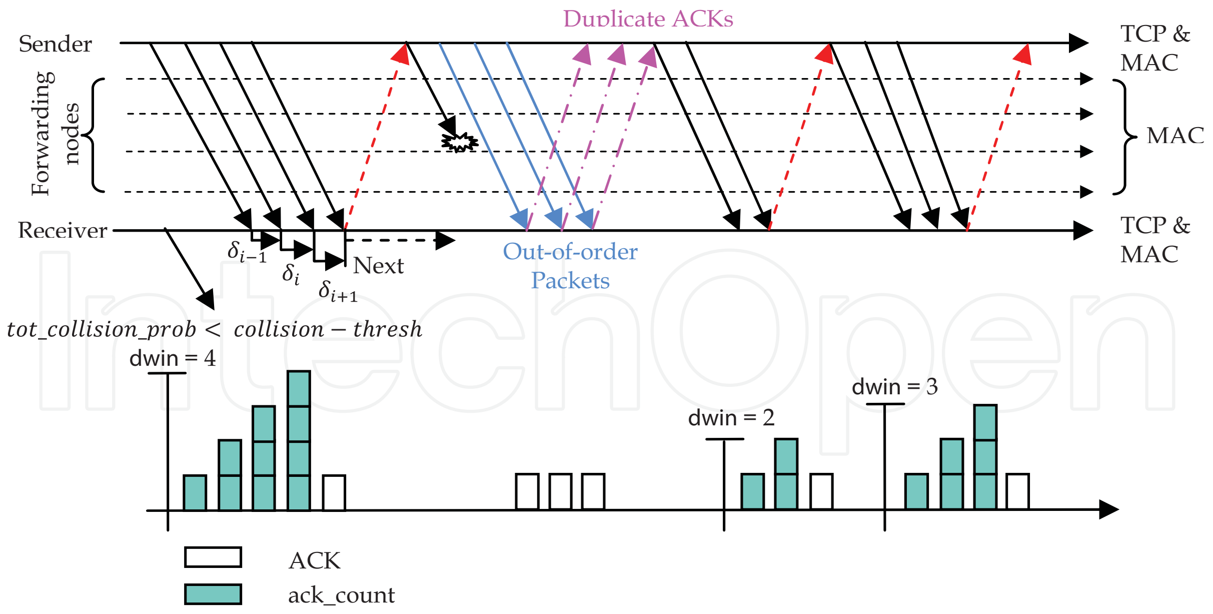


Fig. 6. An example of how TCP-MDA works in the moderate traffic

In case of out-of-order packets, an ACK is immediately prompted; otherwise the receiver waits for the period τ_i before responding. This effective timeout interval is calculated using a timeout tolerance factor k, with k > 0 as given in equation (4).

$$\tau_i = (2+k)\times\overline{\delta}_i \tag{4}$$

The rational here is that due to high delay variations in such environments, it is reasonable to wait for the time the second packet is expected. So, unnecessary timeouts are avoided to be triggered.

5.4 Sender side's modifications

The only requirement at the sender in TCP-MDA is to restrict its *cwnd* size to the maximum of 4 packets. This means that, TCP allows keeping 2, 3 or 4 packets outstanding in the network at any given time. This small size of *cwnd* has been reported in the literature (De Oliveira & Braun, 2005; Fu, et al., 2005) as an efficient size in the short range scenarios and has been thoroughly discussed in Section 4.1. It has been suggested that TCP sender can overcome the spatial contention property by confining the number of the packets in flight in the network (Fu, et al., 2005). So that a limit of $\frac{h}{4}$ for *cwnd* has been reported as an optimal setting in a chain topology; where h is the number of hops between sender and receiver. This setting has been followed in all the methodology's steps described in last sections to confirm the above conclusions and to make TCP-MDA more comparable with TCP-DAA.

5.5 Optimized numbers of delayed ACKs

Different simulations have been run to find the optimal number of in-order data packets to be waited before generating an ACK in different path lengths. In fact, delaying more ACKs in short range scenarios with less than three numbers of hops are found to be more effective as opposed to the upper bounded of four ACKs in scenarios dealing with moderate traffic. However, a large *dwin* over a long path can aggregate the situation by inducing a large burst of data into the network leading to more packet losses (J. Chen, et al., 2008). We take these considerations into account in scenarios, which is mix of low and high traffic/loss rates, An optimal *dwin* size, which acts best in comparison with the other sizes, have been obtained. TCP-MDA with optimal *dwin* size has been compared with TCP-MDA with *cwnd* limit setting and the results are discussed later.

5.6 Performance evaluation

To validate the proposed strategy various simulations representing the derived TCP-MDA scheme under different parameters is presented in this section. The system performance in term of throughput has been studied and the effects of different parameters have been investigated. The evaluation of TCP-MDA has been conducted with the Network Simulator-2 (ns-2) (Fall & Varadhan, 2008).

5.6.1 Simulation area setup

Two scenarios namely chain topology and grid topology as depicted in Fig. 7 have been considered throughout our experiment. The chain topology consist of n nodes with number of nodes *(n)* varying from 2 to 20 and number of concurrent flows varying from 1 to 20 in each simulation. For each simulation, TCP connection is sourced at the first node (node 0) and packets travel hop by hop over the chain to the end node (1 ≤ end node ≤ 19). Simulation has been done for a 5×5 grid topology with three and six TCP flows, respectively. In case of six TCP flows, half of the flows go horizontally and the other half go vertically, spaced evenly.

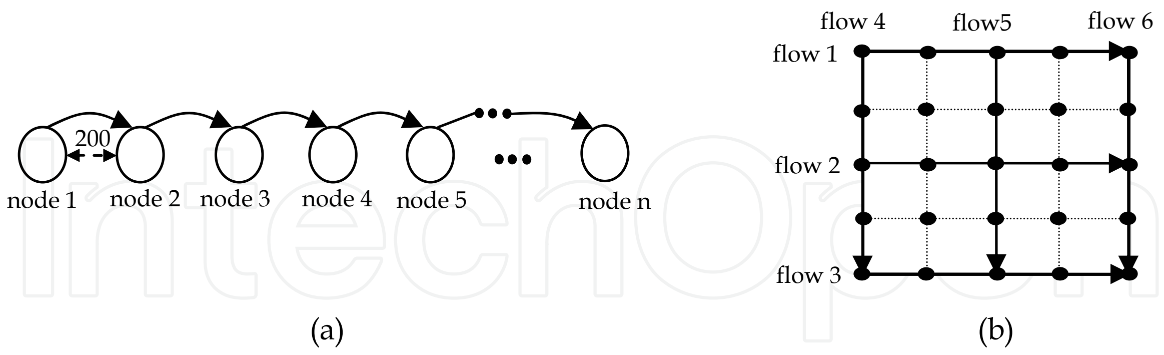


Fig. 7. Simulation scenarios: (a) Chain topology, (b) Grid topology

The nodes are considered as static to minimize the impact of routing dynamics and concentrate on the interaction between TCP and MAC protocol as it is widely followed in the previous researches (K. Chen, et al., 2003; De Oliveira & Braun, 2005; Lilakiatsakun & Seneviratne, 2003; Papanastasiou & Ould-Khaoua, 2004; S Xu & T Saadawi, 2001). In fact, the target is to investigate the dropped packets resulting from channel spatial reuse and contention rather than the dropped packets induced by the route failure which belongs to the mobility fact of network layer. IEEE 802.11 MAC protocol has been considered as widely studied underlying protocol in wireless networks along with Ad-hoc On-Demand Distance Vector Routing (AODV) protocol as a very popular routing protocol in ad-hoc networks. Moreover, nodes access the radio channel at the data rate of 2 Mbps with transmission range set to 250 m and interference range of 550 m.

A TCP-NewReno variant is used which starts transmitting FTP traffic along the chain topology and the packet size is set as 1,460 bytes. Most of the parameters are chosen as given in (de Oliveira & Braun, 2007). These parameters include the value of 0.75 for a as an inter-arrival smoothing factor and 0.2 for k as a tolerance factor tailored to compute the ACK timeout in sending the acknowledgments.We also set the startup parameter μ as 0.3 which provides the best result among the other indexes in (de Oliveira & Braun, 2007). End-to-end TCP throughput has been evaluated and has been defined as total bits transmitted and acknowledged over the simulation time (5).

$$Throughput(kbps) = \frac{\mathrm{Re}ceivedPackets(Packetsize)*8}{STime} \quad (5)$$

Scenarios starting from a moderate traffic/loss rate and ending to a noisy channel with extensive packet losses has been assumed in simulation. A Four State Markov Chain error model has been considered to model this environment as depicted in Fig. 8.

The error rate is changed from 0 in good state to 0.2 in the worst state. There will be more packet losses as the probability of error increases. Here, the packet drops are not only losses due to MAC collisions but also losses induced due to permanent external disturbance. In our proposed strategy, we account the packet losses due to the medium contention and external disturbance is not taken into account. The multi-state error model implements time based error state transitions. Transitions to the next error state occur at the end of the duration of the current state. The next error state is then selected using the transition state matrix (Fall & Varadhan, 2008).

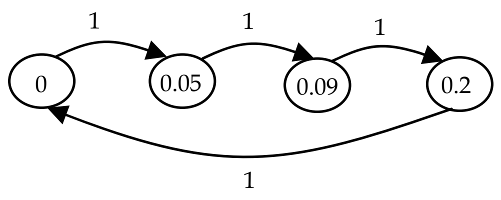


Fig. 8. Four state markov chain error model

The justification for employing this typical error model is its compatibility of introducing different state of collisions through the simulation time which has been achieved by monitoring the simulation trace file. An error rate more than 0.2 might lead to a high collision probability especially in larger ranges more than four hops. This would prevent the proposed strategy to properly show its functionality in low collision probability conditions. All simulation parameters are listed in Table 2. Each data point represents an average of 5 simulation runs with different random seed numbers and each run lasts for 1,000 s. We choose 1,000 s as we target the constrained scenarios ending to high packet loss. So that, it takes a longer time to reach a stable simulation condition.

Parameter	**Value**
Channel bandwidth	2 Mbps
Channel delay	25 µs
Transmission range	250 m
Interference range	550 m
Packet size	1460 bytes
Window limit	4 packets
Regular TCP	NewReno
Routing protocol	AODV
Traffic type	FTP
α	0.75
κ	0.2
μ	0.3
μ'	0.28
collision_thresh	0.3

Table 2. Simulation parameters

5.6.2 Throughput in the chain topology

As discussed earlier, we know that when the channel is in good condition, *dwin* is incremented by one to its limit of 4 to generate less ACKs. However, when the loss rates are considerable, it is more proper to enlarge *dwin* slowly by a factor µ′ which has value between zero and one to provide enough ACKs to TCP sender. Monitoring the channel condition is done by comparing the achieved *total_collision_prob* from MAC layer with the *collision_ thresh.*

The optimized value for µ′ has been obtained by the analytical evaluation given in (de Oliveira & Braun, 2007). It has been proved that following a very conservative procedure, *dwin* should be increased by about 0.28 for each in-order data packet received, and is same as simulation results in (de Oliveira & Braun, 2007). The same value of µ′ has been used here in all the simulations based on the observations in (de Oliveira & Braun, 2007). In addition to μ′, the optimized value of 0.3 for the *collision_thresh* has been obtained through different simulation results which show the best value among all the indexes. All the simulation results are presented in (Armaghani, et al. 2008) for different values of *collision_thresh.*

To evaluate the effectiveness of the TCP-MDA strategy against TCP-DAA and TCP-DAAp (De Oliveira & Braun, 2005, 2007), further simulations have been conducted for 2, 4, 9 and 16 hop scenarios. The rest of simulation parameters and experimental setup were identical to ones selected in Section 5.6.1. It can be concluded that in a 4 hop scenario with up to 5 concurrent flows, TCP-MDA performs similar to TCP-DAA (Fig. 9a). However, for concurrent flows more than 5, results show the improvement of TCP-MDA over the other protocols.

This behavior of TCP-MDA could be due to the setting of *collision_thresh* in our experiments. In fact, the total collision probability (*total_collision_prob)* measured in the MAC layer has been observed to be less than 0.3 in the scenarios up to 5 numbers of flows. Therefore, TCP-MDA throughput is same as TCP-DAA by enlarging *dwin* by one to its limit of 4 packets.

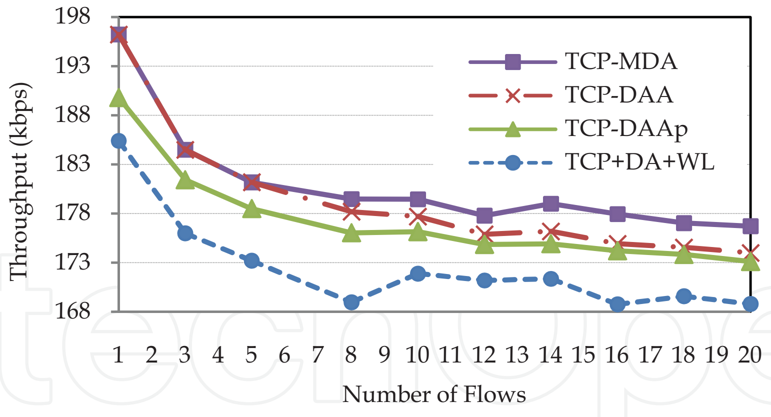
As the number of flows increases, the *total_collision_prob* has a value more than 0.3. Thus, TCP-MDA reacts under this condition by enlarging *dwin* more gradually in order to avoid timeout at the receiver and a bigger *dwin* size than *cwnd* size, i.e. shortage of ACK phenomenon. These results also show that the efficiency of TCP-DAA goes down to the level of TCP-DAAp when we have several concurrent flows running in the network. This behavior closely confirms the rationale of enlarging *dwin* more gradually in a higher loaded channel.

For 2 hop scenario simulation results do not show considerable improvements over the TCP-DAA. A possible explanation for this might be due to the limited spatial reuse property imposed by MAC layer. Spatial contention is negligible in a small network with a short path and we still have a steady state condition where less packet loss may occur with increased load. So that, the calculated collision probability in MAC layer is less than the assumed threshold in these scenarios and *dwin* enlarges by one to meet the need of combining 4 ACKs in one ACK.

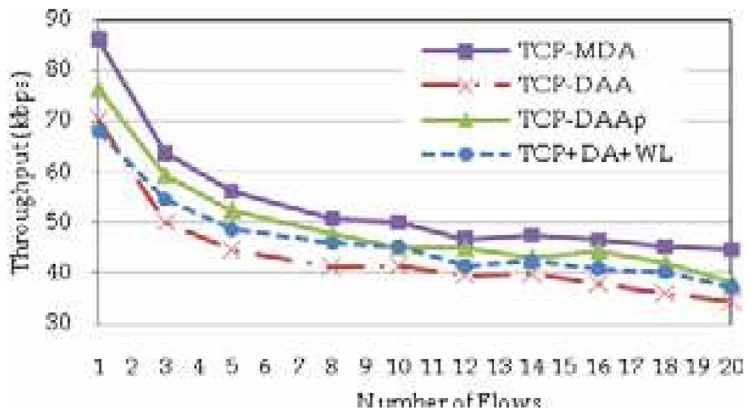
The results of the evaluation with 9 hops are shown in Fig. 9b where we observed that TCP-MDA strategy again proves superior to all other protocols. The improvement ranges between 4 to 13% over TCP-DAAp and 10 to 30% over TCP-NewReno+DA+WL and even higher over TCP-DAA.

The throughput results for a 16 hop scenario depicted in Fig. 9c is not very encouraging. Although, TCP-MDA still seems to slightly outperform others, but TCP instability problem is observed in this experiment. This instability may be explained due to the fact that there are more hidden and exposed terminals that cannot sense each other for transmissions in longer path. In fact, there would be more timeout reports and retransmission efforts in the MAC layer. After several unsuccessful retransmission efforts, the MAC layer would report a link breakage and a route discovery would be triggered immediately after the route failure has been reported. In this way the source would have to wait for the duration until the new route has been established. This is likely to affect the throughput.

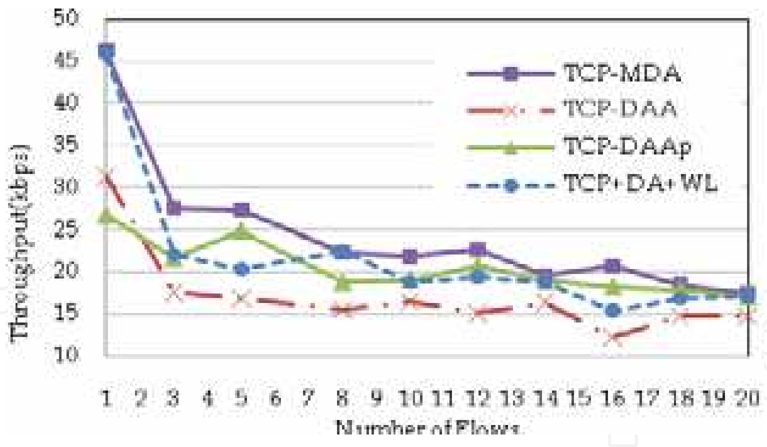
Another reason that could be attributed may be due to the consequence of high interference on TCP sender RTT estimation. This implies that longer end-to-end connection would result in higher amount of contention among nodes because all of them try to access the channel at



(a) 4 hop



(b) 9 hop



(c) 16 hop

Fig. 9. TCP throughput vs. number of flows in a 4 hop chain topology

the same time, and time for the TCP sender to detect lost packets would be longer. Figure 10 depicts a simple scenario where all nodes have at least one packet to send in the forward direction. We assume that node B and D initially have the channel access and they start to transmit at the same time. Soon after the transmission, there would be collision in packet from B to C with the packet from D to E. Meanwhile, A has been waiting to start transmitting several packets to B before releasing the channel.

However, B would be still unable to access the channel and buffers the new packets in addition to packet(s) already in its buffer and would start building up its queue. Therefore, a bottleneck may occur at node B of the path resulting to an artificial increase of the RTT delay measured by the sender. As a result, TCP would overestimate the available bandwidth and enlarges its *cwnd* size leading to the network overload in the next RTT. This procedure would continue until a packet drop would be reported within a MAC retry limits specified by 802.11 MAC standard.

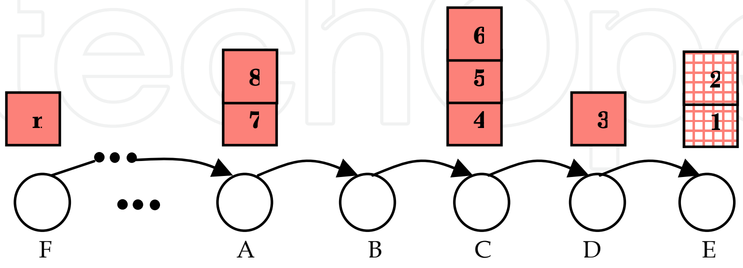


Fig. 10. Network overload scenario

It has been observed that TCP-MDA shows improvements in comparison with both TCP-DAA and TCP-DAAp in short range networks (up to 10 hops). This is basically because TCP-DAA and TCP-DAAp have not been designed for the scenarios facing the tradeoff between moderate and high loss rates, so they are more adaptable to the environment when they come together as TCP-MDA with a channel monitoring mechanism.
The drawback of the proposed strategy is that, TCP-MDA does not estimate the internal network state. However, in a channel with high loss rate, packet drops are not only due to the MAC collision. Packet loss might be due to the high medium induced errors and external disturbance. Since TCP-MDA is not tailored to monitor the channel state, so it is unable to demonstrate the level of medium errors.

5.6.3 Throughput in grid topology

Grid topology is a more complex scenario with various interactions among the nodes. Extensive channel contention exists and so more packet drops are expected as a consequence.
Grid topology is commonly used in literature to evaluate the effect of multiple interfering flows on TCP performance (Boggia, et al., 2005; J. Chen, et al., 2008; De Oliveira & Braun, 2005). Figure 11 compares the performance of TCP-MDA, TCP-DAA, TP-DAAp as well as standard TCP with DA extension.

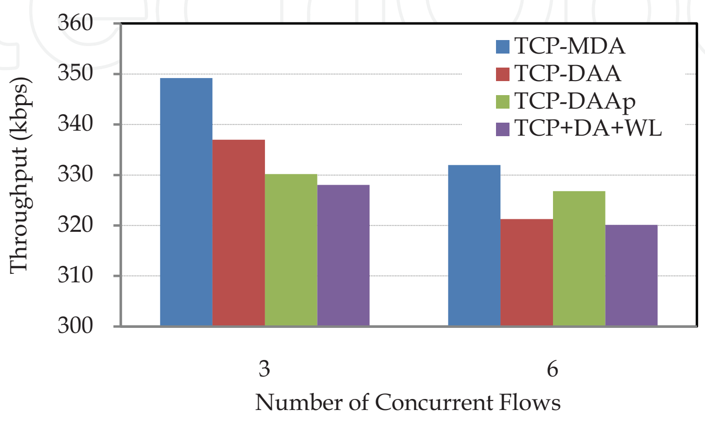


Fig. 11. TCP throughput over a 5×5 grid topology

Here, the flows do not share the same path but still interfere due to the hidden terminals and interference between nodes' transmission ranges. The results depicted in Fig. 11 again mirror the optimized throughput of TCP-MDA over the other protocols in scenarios with dynamic traffic. There are fewer contentions in the case of three flows and so TCP-MDA maintains the traffic by enlarging *dwin* rapidly up to four delayed ACKs. As the level of contention upsurges, TCP-MDA turns to perform more moderately by a gradual *dwin* enlargement, i.e. in the case of six cross flows.

TCP-DAAp provides a better throughput over TCP-DAA and TCP+DA+WL in the case of six flows which again prove the need of providing more ACKs in high traffic channels. In general, the same observation as chain topology holds true for grid topology. It can be deducted that in chain and grid scenarios, TCP-MDA benefits by delaying more ACKs in low traffic and less in high traffic channels.

5.6.4 Impact of congestion window limit

It is reported in earlier studies that limiting *cwnd* size improves TCP performance by maximizing the spatial reuse. So, in this study a limit of up to 4 packets has been considered for *cwnd* in scenarios with not more than 19 hops to make our work more comparable with the ones presented in (de Oliveira & Braun, 2007).

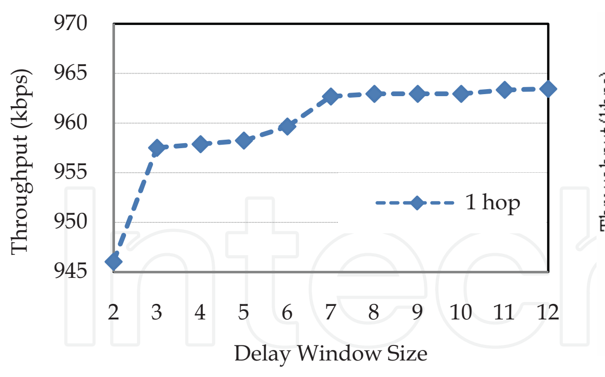
It would be noted that TCP-MDA may not provide the same improvement in some scenarios and the performance may degrade to the level of standard TCP that uses DA and window limit (WL). This behavior can be explained as following: first, limiting *cwnd* by itself would decrease the channel interference and maximize the spatial reuse. On the other hand, delaying ACKs helps TCP sender to slowdown its transmission rate by triggering the *cwnd* growth to its limit in a longer interval. In this way, the total number of induced data packets in the network might be affected by a slow transmission rate and the receiver delaying window adaption provides little extra improvement.

The above discussion has motivated to do more investigation on the impact of *cwnd* limit along with the *dwin* limit. To this end, we have run different simulations in which the *cwnd* has been unbounded and *dwin* size has been varied with different values. All the simulation parameters are same as in earlier simulations. Our objective has been to identify the relationship between TCP throughput and optimized *dwin* size in different path lengths. The results are presented in Fig. 12.

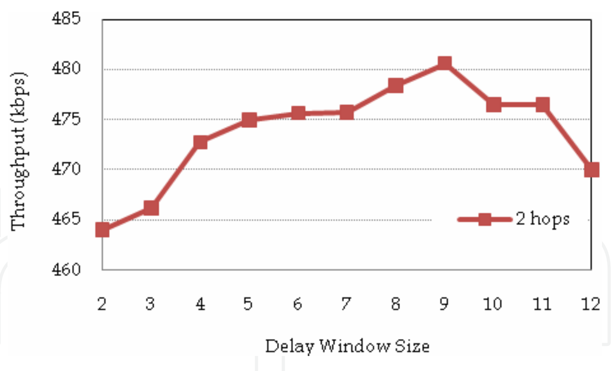
The above observations determine that *dwin* size in TCP-MDA is based on the path length of a TCP connection. We have observed that for a short path (hops ≤ 3); the ACK can be delayed up to a large value. The reason lies on the 802.11 capability to transmit the packets without collision in short ranges no matter what the burst size is.

However, employing a large *dwin* size is not an efficient solution in all scenarios resulting in the burstiness of the forwarding packets in long paths. In this case, too many data packets are queued at the TCP sender side, waiting for an acknowledgment to be received inducing packet drops in the router's buffer. Since there are more interfering nodes, there might be more packet losses because the packet has more chances to be interfered in a long path. The proper values for TCP-MDA *dwin* size according to our observations in different path length are listed in Table 3.

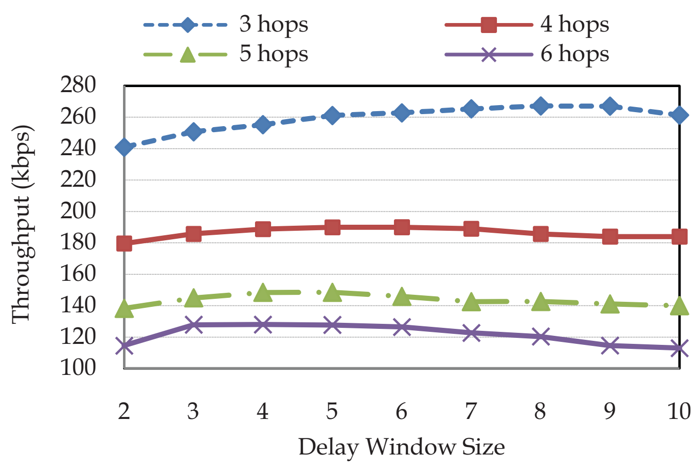
Although, there are more unsuccessful packet transmissions caused by interference in the chains between 4 and 6 hop counts, TCP-MDA still could maintain performance gain by delaying ACK for more data packets since a TCP sender is able to recover packet loss rather rapidly due to the small RTT.



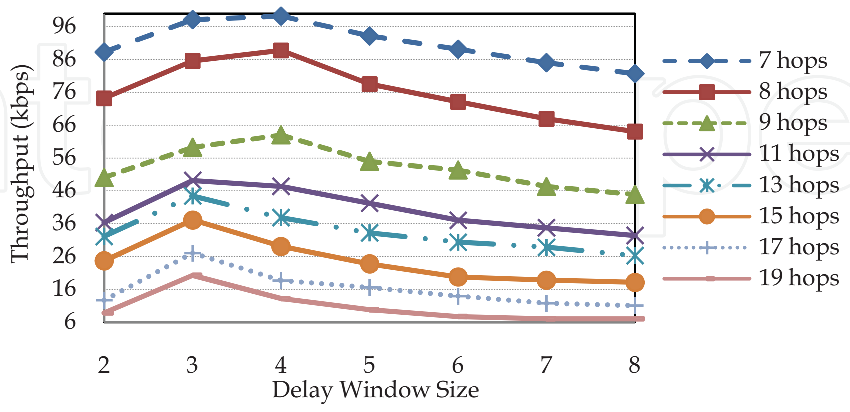
(a) hop count = 1



(b) hop count = 2



(c) 2 < hop count ≤ 6



(d) 6< hop count ≤ 19

Fig. 12. TCP throughput vs. delay window size in chain topology

Path length (No. of hops)	*dwin* Limit
h ≤ 3	9
3 < h ≤ 5	5
5 < h ≤ 9	4
9 < h ≤ 19	3

Table 3. Optimized *dwin* size in different path lengths

Similar trend also exists for paths longer than 6 hops, where TCP-MDA achieves throughput gain only when the delay window size is equal to 4. For larger topologies than 10 hops, large delay window size may not maintain throughput gain due to excessive data packet losses. Further, TCP-MDA spends more time detecting packet loss due to the larger RTT. Therefore, for long paths, large delay window is not preferred.

Next experiment is tailored to evaluate the performance of TCP-MDA – WL over TCP-MDA. Figure 13 shows the performance of proposed strategy with and without *cwnd* limit. It has been shown in previous simulations that TCP-MDA outperforms the other protocols in a short chain of hops. Here, the impact of *cwnd* limit has been investigated in our assumed scenario with medium and high loss rates. The number of the acknowledgments to be delayed in TCP-MDA – WL has been accordingly based on observation in each path length as listed in Table 3.

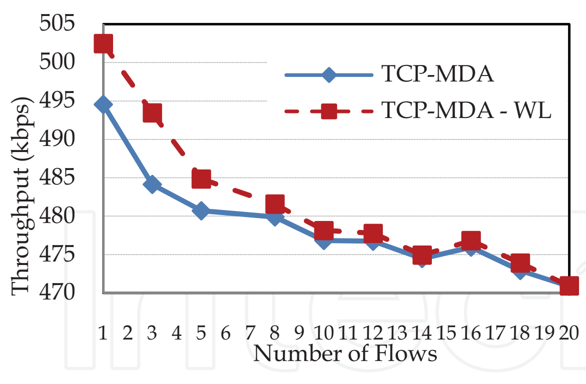
In this experiment, the two graphs in Fig. 13a and Fig. 13b show that *cwnd* limit on TCP-MDA does not bring considerable benefit on a short path with less number of flows. In fact, bounding the *cwnd* along with *dwin* may restrict the TCP performance by confining the total number of packets in flight in the network in small topologies, i.e. small burst size. Therefore, a large *dwin* solely might be enough effective on throughput improvement in these kinds of scenarios. For a longer path in Fig. 13c *cwnd* limit provides more throughput gain which is prominent in less number of flows.

6. Conclusion

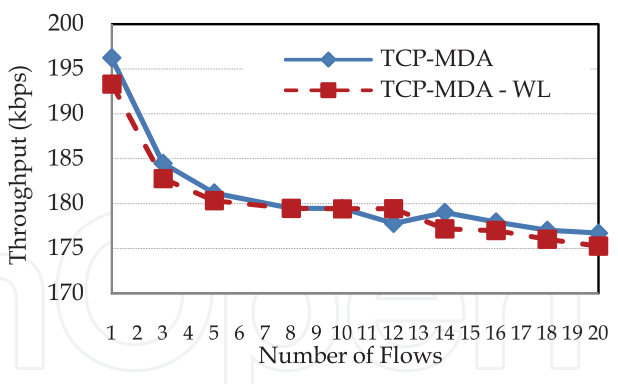
In this chapter poor bandwidth utilization and performance of TCP, when it runs over 802.11 MAC protocol in multi-hop ad-hoc networks, has been addressed. This problem can be due to the extensive number of medium access carried out by TCP by generating redundant ACK packets that compete in the same route with data packets for the media. First, the reasons of TCP performance degradation in ad-hoc networks have been studied. Then the impact of delay acknowledgments has been studied which helps to improve TCP performance by reducing the number of generated ACK. Taking into account the importance of delay ACKs on TCP performance enhancement, a dynamic TCP-MAC interaction strategy has been proposed to reduce ACK induced overhead and consequently collisions.

The results have shown that the proposed dynamic TCP-MAC interaction approach reduces the number of ACKs transmitted by a TCP receiver by monitoring the medium collision probability and reacting to packet losses. The results comparison has shown improvement in short path lengths between 4 to 9 hops in a chain topology.

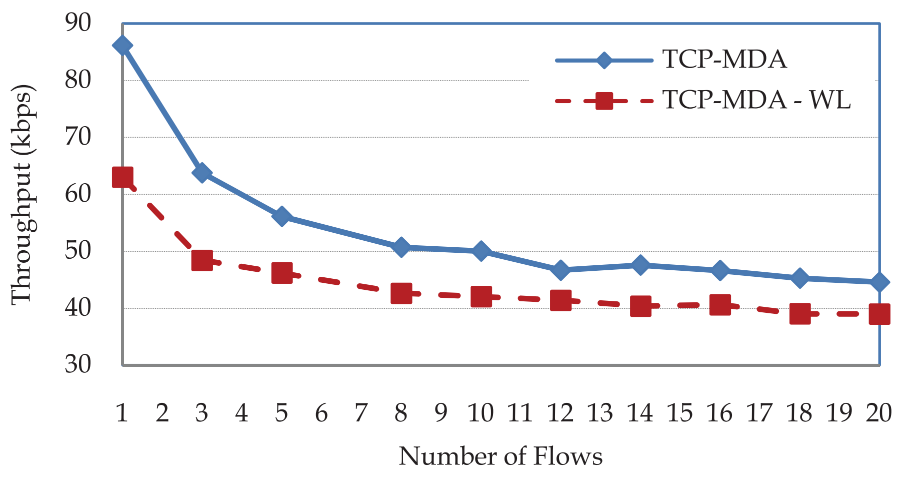
The impact of TCP *cwnd* size along with delayed ACK on MAC spatial reuse have also been studied. The findings show that with an unbounded *cwnd*, for each topology there exists an



(a) 2 hops



(b) 4 hops



(c) 9 hops

Fig. 13. Comparison of TCP-MDA with and without *cwnd* limit

optimal delay window size in which TCP throughput is maximized. Armed with the optimized numbers of delayed ACKs, the proposed strategy has been evaluated with two different adjustments: with a limited *cwnd* and maximum delayed ACKs of 4 packets in all the topologies; and an unbounded *cwnd* along with a various *dwin* based on the path length of the topology. The related results draw to the conclusion that limiting *cwnd* is not beneficial in all the scenarios and a large *dwin* may solely help to alleviate the spatial reuse contention in short range topologies with less number of flows.

7. Directions for future work

Based on the achieved results, following problems may be subject for further study:

- Error detection mechanism: TCP-MDA is basically based on the consideration that having a dynamic loss rate, the packets suffer from the channel interference and MAC collision and accordingly the channel collision probability is taken into account. Hence, TCP-MDA does not detect the exact internal state of the network. However, multi-hop networks are prone to much higher bit error rates in a lossy channel leading to very complex conditions. As a consequence, TCP-MDA lacks robustness in detecting what is exactly going on within the network so that it can't take proper action upon error based losses. So, it would be an interesting prospect to develop an end-to-end basis error detection mechanism to inform TCP about the actual cause of any packet loss so the TCP recovery mechanism can take the most appropriate action. This error model can be designed using more heuristic methods and fuzzy logic to consider more realistic transitions among the various states of the network.

- TCP sender adoption: TCP-MDA focuses more on the TCP receiver side and the only investigation on the sender side is over the impact of congestion window limitation. It is a basic TCP functionality that the sender relies on ACKs for computing its timeout interval and transmits new data packets. Moreover, TCP RTT computation can be affected by high/low delay variance. Hence, TCP performance can be disturbed by unnecessary delaying ACKs. As a result, a comprehensive study might reveal issues which are not captured in present research.

8. References

Altman, E., & Jimenez, T. (2003). Novel delayed ACK techniques for improving TCP performance in multihop wireless networks. *Lecture Notes in Computer Science, 2775*, 237-250.

Armaghani, F., Jamuar, S., Khatun, S., & Rasid, M. (2008). Performance Analysis of TCP with Delayed Acknowledgments in Multi-hop Ad-hoc Networks. To appear in *Wireless Personal Communications (on line available)*.

Boggia, G., Camarda, P., Grieco, L. A., Mastrocristino, T., & Tesoriere (2005, 7-7 Sept. 2005). *A Cross-layer Approach to Enhance TCP Fairness in Wireless Ad-hoc Networks.* Paper presented at the Wireless Communication Systems, 2005. 2nd International Symposium on.

Chen, J., Gerla, M., Lee, Y., & Sanadidi, M. (2008). TCP with delayed ack for wireless networks. *Ad Hoc Networks, 6*(7), 1098-1116.

Chen, K., Xue, Y., & Nahrstedt, K. (2003). On setting TCP's congestion window limit in mobile ad hoc networks. *Communications, 201*, 03.

Conti, M. (2003). Body, personal, and local ad hoc wireless networks *The Handbook of Ad Hoc Wireless Networks* (pp. 3-24). Florida: CRC Press, Inc.

De Oliveira, R., & Braun, T. (2005). *A dynamic adaptive acknowledgment strategy for TCP over multihop wireless networks.* Proc. IEEE INFOCOM'05, March 2005, 1863 - 1874, Miami (USA)

de Oliveira, R., & Braun, T. (2007). A smart TCP acknowledgment approach for multihop wireless networks. *IEEE Transactions on Mobile Computing, 6*(2), 192-205.

Dongkyun, K., Toh, C. K., & Yanghee, C. (2000, 2000). *TCP-BuS: improving TCP performance in wireless ad hoc networks.* Paper presented at the Communications, 2000. ICC 2000. 2000 IEEE International Conference on.

Fall, K., & Varadhan, K. (2008). The ns manual. from The VINT Project: http://www.isi.edu/nsnam/ns/ns-documentation.html

Fu, Z., Luo, H., Zerfos, P., Lu, S., Zhang, L., & Gerla, M. (2005). The impact of multihop wireless channel on TCP performance. *IEEE Transactions on Mobile Computing, 4*(2), 209-221.

Gerla, M., Bagrodia, R., Lixia, Z., Tang, K., & Lan, W. (1999, 1999). *TCP over wireless multi-hop protocols: simulation and experiments.* Paper presented at the Communications, 1999. ICC '99. 1999 IEEE International Conference on.

Gerla, M., Tang, K., & Bagrodia, R. (1999, 25-26 Feb 1999). *TCP performance in wireless multi-hop networks.* Paper presented at the Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on.

Hamadani, E., & Rakocevic, V. (2005). *Evaluating and Improving TCP Performance Against Contention Losses in Multihop Ad Hoc Networks*.

Hanbali, A., Altman, E., & Nain, P. (2005). A survey of TCP over ad hoc networks. *IEEE Communications Surveys & Tutorials, 7*(3), 22-36.

Holland, G., & Vaidya, N. (2002). Analysis of TCP performance over mobile ad hoc networks. *Wirel. Netw., 8*(2/3), 275-288.

Khalife, H., & Malouch, N. (2006, June 2006). *Interaction Between Hidden Node Collisions and Congestions in Multihop Wireless Ad-hoc Networks.* Paper presented at the Communications, 2006. ICC '06. IEEE International Conference on.

Lilakiatsakun, W., & Seneviratne, A. (2003). *TCP performances over wireless link deploying delayed ACK*. Proc. VTC '03, Vol. 3, April 2003, 1715 - 1719.

Liu, J., & Singh, S. (2001). ATCP: TCP for mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on, 19*(7), 1300-1315.

Mohapatra, P., & Krishnamurthy, S. (2005). *AD HOC NETWORKS: technologies and protocols*: Springer.

Papanastasiou, S., & Ould-Khaoua, M. (2004). TCP congestion window evolution and spatial reuse in MANETs. *Wireless Communications and Mobile Computing, 4*(6).

Singh, A., & Kankipati, K. (2004). *TCP-ADA: TCP with adaptive delayed acknowledgement for mobile ad hoc networks*. Proc. Wireless Communication and Networking Conference, March 2004, 1685 - 1690.

Stevens, W. (1994). *TCP/IP illustrated, vol. 1*: Addison-Wesley.

Wang, F., & Zhang, Y. (2002). *Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response*. Paper presented at the Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking \&amp; computing.

Xiang, C., Hongqiang, Z., & Jiangfeng, W. (2005). A survey on improving TCP performance over wireless networks: Network Theory and Applications, voll6.[S. 1.]: Springer.

Xu, K., Gerla, M., & Bae, S. (2002). *How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks*. Proc. IEEE GLOBECOM Vol. 1, Nov. 2002, 71 - 76.

Xu, S., & Saadawi, T. (2001). Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *Communications Magazine, IEEE, 39*(6), 130-137.

Xu, S., & Saadawi, T. (2001). *Evaluation for TCP with Delayed ACK Option in Wireless multi-hop Networks*. Proc. IEEE VTC'01, Fall 2001, 267 - 271.

Yu, X. (2004). *Improving TCP performance over mobile ad hoc networks by exploiting cross-layer information awareness*. Paper presented at the Proceedings of the 10th annual international conference on Mobile computing and networking.

Zhai, H., Wang, J., Chen, X., & Fang, Y. (2006). Medium access control in mobile ad hoc networks: challenges and solutions. *Wireless Communications and Mobile Computing, 6*(2), 151-170.