A Cross layer Analysis of TCP Instability in Multihop Ad hoc Networks

Ehsan Hamadani, Veselin Rakocevic
Mobile Network Research Group
School of Engineering and Mathematical Sciences
City University, London EC1V 0HB, UK
[E.hamadani, V.rakocevic]@city .ac.uk

Abstract: *It is well-known that* due *to the nature of some ad hoc network applications (e.g. emergency operation, battlefield communication), TCP instability can have a devastating impact on the Quality of Service requirements. As we will show in this paper, TCP instability is truly a cross layer problem which needs to be addressed by considering the interaction of multiple layers with each other. We first divide the TCP instability problem into intra-flow and inter-flow instability and then propose a set of simple but effective solutions and show through extensive simulations the improvements achieved using the proposed algorithms.*

I) INTRODUCTION

Multihop ad hoc networks are autonomous systems of mobile devices connected by wireless links without the use of any pre-existing network infrastructure or centralized administration. During recent years ad hoc networks have attracted considerable research interest thanks to their easy deployment, maintenance and application variety. To enable seamless integration of ad hoc networks with the Internet (for instance in ubiquitous computing applications), TCP seems to be the natural choice for users of ad hoc networks that want to communicate reliably with each other and with the Internet. On the other hand, one of the prominent random access protocols known for ad-hoc networks is the IEEE 802.11 MAC standard [1] which has been widely used and adopted. However, neither TCP nor 802.11 were primarily designed and optimized to work in multihop ad hoc networks. Not surprisingly, as shown in [2,3], the network exhibits serious performance issues when TCP runs over 802.11 in multihop ad hoc networks. In particular interest to us in this paper is the network instability issue as we believe due to the nature of some of the ad hoc network applications (e.g. emergency operation, battlefield communication) the disconnectivity or starvation of one or more connections for even a short period of time is not acceptable and can have a devastating impact on QoS. More specifically, the ad-hoc network users are more willing to receive a continuous and stable flow of data rather than sending/receiving large bulk of data instantly. This argument holds also true for jitter sensitive applications such as audio or video streaming. As we will show later in the paper, instability is truly a cross layer problem which needs to be addressed by considering the interaction of multiple layers with each other. During recent years, there has been a number of valuable works that have investigated the TCP instability by investigating the interaction of routing protocols and TCP in multihop ad hoc networks [4-6]. In this paper, we take a different approach to address TCP instability by carefully tracking the chain of events occurring between link layer and TCP. In particular, we divide the TCP instability problem into intra-flow and inter-flow instability where the former instability is caused by the interaction of nodes belonging to the same TCP connection while the latter happens when nodes belonging to different connections interact. Based on the findings, we then propose a set of simple but effective solutions and show through extensive simulations the improvements achieved using the proposed algorithms.

The rest of the paper is organized as follows. In section 2, we will give an overview of the IEEE 802.11 MAC and TCP with special emphasis on backoff algorithm and TCP congestion control mechanism, respectively. In section 3, the main causes of inter-flow and intra-flow instability are discussed and explained in fine details. Then, based on the drawn facts, we propose two different schemes in section 4 that are applied to TCP and 802.11. This is followed by the simulation model and the key results obtained by simulating the proposed model against the default TCP and 802.11 MAC protocol in section 5. Finally, in section 6, we conclude the paper with some outlines towards future work.

II) PROTOCOLS OVERVIEW

a) IEEE 802.11 MAC

IEEE 802.11 Distributed Coordination Function (DCF) [1] provides the basic access method of the 802.11 MAC using the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) scheme to access the channel. Each station willing to send data generates a random backoff period according to equation (1) as its deferral time before transmitting, unless the backoff timer already contains a nonzero value, in which case the selection of a random number is not needed and not performed.

$$\textit{Backoff Time} = \textit{Random}() * \textit{SlotTime} \qquad (1)$$

Here, the SlotTime depends on the technology deployed in physical layer and Random() is a random integer number of time slots drawn from a uniform distribution over the interval [0,CW], where CW (Contention Window) itself is an integer within the range of [CW_{min},CW_{max}]. At the beginning of each frame transmission, the contention window takes an initial value of CW_{min}. Then, CW is doubled every time the node is not successful to transmit its frame until the CW reaches the value of CW_{max}. Once it reaches the CW_{max}, the CW remains at the value of CW_{max} until it is reset. Also the CW is reset to CW_{min} after every successful attempt to transmit a frame or the frame is dropped after several tries (specified by Max-Retry-Limit). The above backoff scheme is also known as Binary Exponential Backoff (BEB). After choosing the new backoff timer, if no medium activity is indicated, the backoff timer is decremented by one slot-time. If the medium is determined to be busy at any time during the backoff process, then the backoff timer is suspended. Once the backoff counter reaches zero, the node can transmit the data.

b) TCP Congestion Control

TCP Congestion Control was added to TCP in 1987 and was standardized in RFC2001 [7] and then updated in RFC2581 [8]. In a broad sense, the goal of the congestion control mechanism is to prevent congestion in intermediate router's buffer by dynamically limiting the amount of data sent into the network by each connection. To estimate the number of packets that can be in transit without causing congestion, TCP maintains a congestion window (cwnd) that is calculated by the sender as follows: when a connection starts or a timeout occurs, slow start is performed where at the start of this phase, the cwnd is set to one MSS (Maximum Segment Size). Then the cwnd is doubled after each window worth of data is acknowledged. Once cwnd reaches a certain threshold, the connection moves into the congestion avoidance phase where TCP gently probes the available bandwidth by increasing the cwnd by one packet in every round trip time (Additive Increase). During this time if the TCP detects packet loss through duplicate acknowledgments, it retransmit the packet (fast retransmit) and decreases the cwnd by a factor of two (Multiplicative Decrease) or it goes to slow start according to the TCP version used. Alternatively, if the sender does not receive the acknowledgment within retransmission time out (RTO), it goes to slow start and drops its window to one MSS.
After calculating the current value of *cwnd*, the effective limit on outstanding data (i.e. flight size), known as '*send window*' *(swnd)*, is set as the minimum of the *cwnd* and available receiver window *(rwnd)*. The *rwnd* is the amount of available buffer size in the receiver side and is taken into account in order to avoid buffer overflow at the receiver by a fast sender (flow control). Therefore:

$$swnd = \min\{rwnd, cwnd\} \quad (2)$$

III) PROBLEM DESCRIPTION

a) Intra-flow Instability

To understand the main cause of TCP instability in a single TCP flow (intra-flow instability), let us recall from section 2 that the performance of TCP directly depends on the *swnd* which its optimal value should be proportional to bandwidth-delay product of the entire path of the data flow. It is important to note that the excess of this threshold does not bring any additional performance enhancement, but only leads to increased buffer size in intermediate nodes along the connection. As shown in [2,9], the bandwidth-delay product of a TCP connection over multihop 802.11 networks tends to be very small. This is mainly because in 802.11, the number of packets in flight is limited by the per-hop acknowledgements at the MAC layer. Such property is clearly quite different from wireline networks, where multiple packets can be pushed into a pipe back-to-back without waiting for the first packet to reach the other end of the link. Therefore, as compared with that of wired networks, ad hoc networks running on top of 802.11 MAC, have much smaller bandwidth-delay product. However, as shown in [3], TCP grows its congestion window far beyond its optimal value and overestimates the available bandwidth-delay product. To get a better understanding of TCP overestimation of available bandwidth-delay product in ad hoc networks, consider a simple scenario in fig.1 where all nodes can only access their direct neighbors. Here a TCP connection is running from node A to E and all nodes have at least one packet to send in the forward direction.

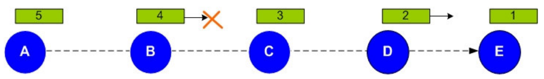

Fig. 1. 4 hop chain topology

Let us assume nodes B and D initially win the channel access and start to transmit their data into the network at the same time. Soon after both stations start transmitting their data, the packet from B to C is collided with the interference caused by D→E transmission. Following this case, node A is very likely to win the access to the channel and starts transmitting several consecutive packets towards B before releasing the channel [10]. Meanwhile, since B is unable to access the channel it buffers the new packets in addition to packet(s) already in its buffer and starts building up its queue (figure 2).

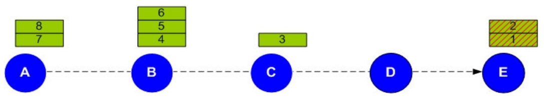

Fig. 2. Queue build up in a 4 hop chain topology

This results in an artificial increase of the RTT delay measured by the sender as node B now becomes the bottleneck of the path. Such situation leads to an overestimate of the length of available data pipe and therefore an increase of the TCP congestion window and hence more network overload in the next RTT. Fig.3 summarizes the chain of actions that occur following a network overload and lead to TCP intra-flow instability. Initially, increasing the network overload causes more contention among nodes as all of them try to access the channel (stage 2). On the other hand, when the level of contention goes up, more packets need to be retransmitted as the probability of collision increases with the increasing level of contention (stage 3). This in turn introduces extra network overload and therefore closing the inner part of the cycle (stage 1→stage2→sage3→stage1).
This cycle is continued until one or more nodes cannot reach its adjacent node within a limited number of tries (specified by the MAC_Retry_Limit

in 802.11 MAC standard) and drop the packet (packet contention loss). This packet loss is then recovered by the TCP sender either through TCP fast retransmit or through TCP timeout (stage 4). In both cases, TCP drops its congestion window resulting in a sharp drop in number of newly injected packets to the network (stage 5) and therefore giving the network the opportunity to recover. However, soon after TCP restarts, it creates network overload again by overestimating the available bandwidth-delay product of the path, and the cycle repeats.

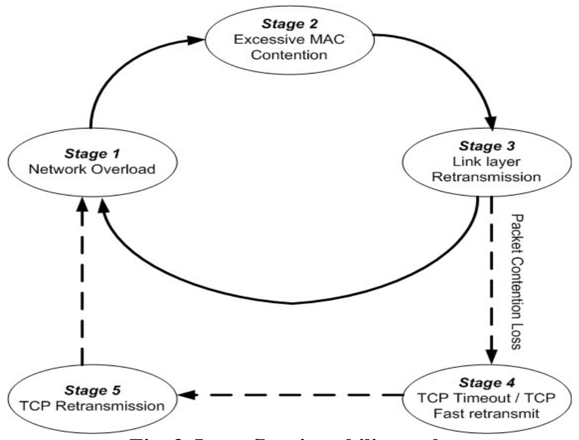

Fig. 3. Intra-flow instability cycle

Fig.4 shows the change of *cwnd* and the instances of TCP retransmission (caused by packet contention losses) in a 4 hop chain topology shown in figure 1 using 802.11 MAC. The result fully supports the above argument and confirms that TCP behavior towards overloading the network causes extensive packet contention drops in the link layer. These packet drops are wrongly perceived as congestion by the TCP and result into false trigger of TCP congestion control algorithm, frequent TCP packet retransmissions and finally TCP instability.

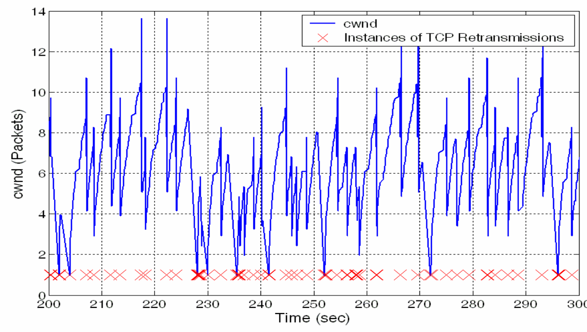

Fig. 4. Change of *cwnd* and the instances of TCP retransmission in a 4 hop chain topology

b) Inter-flow Instability

Unlike the intra-flow instability that is caused by the interaction of nodes belonging to the same connection, inter-flow instability is observed when multiple flows compete to access the channel. In particular, inter-flow instability mainly happens when one connection is able to monopolize the channel resources at the expense of other contending connections. Therefore, as we will see in this section, the inter-flow instability is closely linked to well-know unfairness problem reported in [10]. To investigate the cause of inter-flow instability, we have conducted number of simulations using different scenarios. Our main observation was that regardless of the transport and routing protocol used in the network, inter-flow instability is a serious issue that mainly (but not exclusively) lies on the binary exponential backoff (BEB) algorithm adopted in 802.11 MAC. To see how the BEB can cause instability, consider a static cross topology shown in fig.5 where connection 1 runs from node A to node G and connection 2 runs from nodes H to node M.

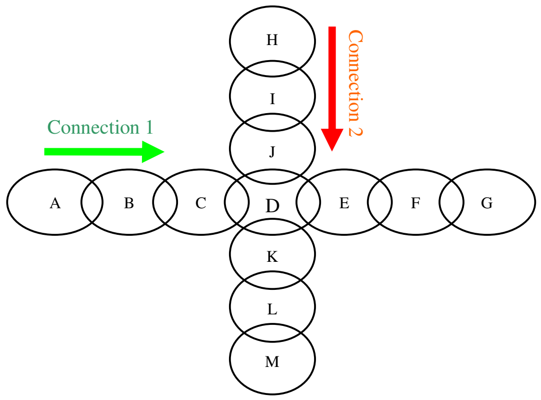

Fig. 5. A cross topology

Let us consider the case where nodes C and J are competing in their first try to access node D (which is shared between 2 connections). As the *CW*s at both stations are very small (e.g. less than 31) the transmission of RTSs of nodes C and J (that are hidden from each other) may very likely overlap partially, and as a result there will be a collision. The collision may occur several times until the *CWs* are large enough to allow either node to get control of the medium [11]. In particular, one of the two nodes (let us say, node C) may select a small back-off time from its *CW*, while the other node (i.e., J) selects a large value resulting in letting the C→D RTS/CTS handshake to be successfully completed. Once the data transfer is completed, node C resets its CW and backs-off before initiating another handshake. However, the remaining back-off timer at node J may be large compared to the back-off timer at node C, which is drawn from the range [0, CW_{min}]. In that case, nodes C and D may exchange several more frames (belonging to connection 1) before node J's back-off timer reduces to zero. Whenever the back-off timer at node J reduces to zero, node J starts transmitting to node D. However, as the *CW* at node C is equal to CW_{min} (because of previous successful transmission) the contention is most likely to result in a collision. After the collision, node C doubles its *CW* from CW_{min} whereas node J doubles its *CW* from a larger value ($CW >> CW_{min}$) as it could not succeed in its previous transmission. Therefore, the *CW* at node J is very likely greater than that at C and node C is more likely to get control of the medium again! This obviously brings connection 2 into TCP instability situation as connection 2 has been starved during this period and is unable to send its data to the destination. Even worse, this process (i.e., several packet transmissions from connection 1) may repeat several times if the J→D RTS/CTS handshake starts around the same time as B→C or E→F RTS/CTS handshakes as well; as in all cases node D will still be reserved by connection 1.

Figure 6 shows the achieved throughput by connection 1 and 2 using UDP and static topology in a cross topology. The results fully supports the above argument and confirms that the *inter-flow instability experienced by one or more connections in multihop ad hoc networks is mainly (but not exclusively) due to the BEB algorithm used in 802.11 MAC.* In particular, as we have showed in [12], the inter-flow instability problem becomes more severe when TCP is used over 802.11. This is because medium contention results to considerable number of link layer packet drops and therefore false trigger of TCP congestion control algorithm which can result to even more instability experienced by one or more connection(s) as described earlier.

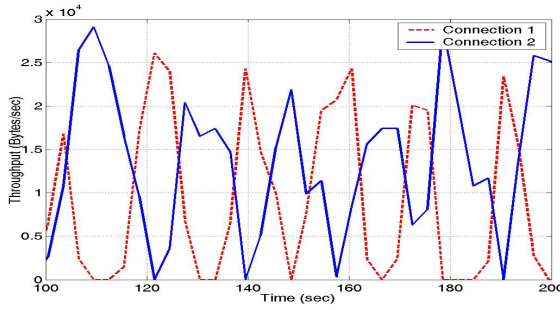

Fig. 6. Inter-flow instability

IV) PROPOSED SOLUTIONS

a) TCP Contention Control

To tackle the TCP intra-flow instability by controlling the amount of network overload as discussed in section 3, we use a novel cross layer algorithm called TCP Contention Control (TCC) which is implemented by the TCP receiver. The basic idea behind TCC algorithm is quite simple. In each RTT, TCC monitors the effect of changing the number of outstanding packets in the network on the achieved throughput and the level of contention delay experienced by each packet (we will shortly explain how the contention delay is measured by TCC). Then, based on these observations, the TCC estimates the amount of traffic that can be sent by the sender to get a balance between the maximum throughput and the minimum contention delay by each connection. To achieve this, TCC defines a new variable called TCP_Contention which its value is determined based on the pseudo code in fig.7.

```
if ($\Delta_{Throughput} \geq 1$)
  {
   if ($\Delta_{Contention} > 1$)
       $TCP\_Contention = TCP\_Contention - \frac{MSS * MSS}{TCP\_Contention}$
   else
       $TCP\_Contention = TCP\_Contention + MSS$
  }
else
  {
    if ($\Delta_{Contention} > 1$)
        $TCP\_Contention = 2 * MSS$
    else
        $TCP\_Contention = TCP\_Contention + \frac{MSS * MSS}{TCP\_Contention}$
  }
if ($TCP\_Contention < 2 * MSS$)
  $TCP\_Contention = 2 * MSS$
```

Fig. 7. Pseudo code of calculating TCP_Contention

As it can be seen in the code, the calculation of TCP_Contention depends on the value of two parameters named *DeltaThroughput* and *DeltaContention.* DeltaThroughput which is calculated as in formula 3, simply compares the amount of throughput received by the receiver in current RTT (RTT_new) and the last RTT (RTT_old)

$$\Delta_{Throughput} = \frac{(data\ received)_{RTT\_new} * (RTT\_old)}{(data\ received)_{RTT\_old} * (RTT\_new)} \quad (3)$$

To measure the DeltaContention, we assume the presence of a new field, known as ContentionDelay in the MAC Protocol Data Unit (MPDU) that keeps the value of Contention Delay (CD). CD is calculated to be equal to the time from the moment the packet is placed at the beginning of buffer utill it leaves the buffer for actual transmission on the link layer. Therefore, the CD does not record the queuing delay experienced by each packet. This is an important feature of contention delay as it helps the TCP to distinguish between network congestion losses and network contention losses and therefore react properly as we explain later in this section. Then each packet alongside the connection records the CD experienced in each node and add the new CD to the ContentionDelay field. In this manner, the total contention delay experienced by each packet along the path are collected at the MAC layer and are delivered to the TCP receiver. The TCP receiver then calculates the Contention Delay per Hop (CDH) by dividing the CD by total number of hops traversed by that specific packet. Finally the receiver derives the Average Contention Delay per Hop (ACDH) by calculating the mean value of CDH received during one RTT. Having the value of ACDH, the DeltaContention is calculated as the value of ACDH in current RTT ($ACDH_{RTT\_new}$) divided by the ACDH measured in last RTT ($ACDH_{RTT\_old}$)

$$\Delta_{Contention} = \frac{ACDH_{RTT\_new}}{ACDH_{RTT\_old}} \quad (4)$$

We should also note that because of TCP Delayed ACK algorithm which generates an ACK every other received segment, we set the minimum TCP_Contention to 2*MSS to make sure at least 2 segments are in the network and can trigger the transmission of TCP ACK at the receiver without waiting for maximum ACK delay timer to expire.

Having calculated the TCP_Contention by TCC, the important question that needs to be answered now is how we propagate the value of TCP_Contention (which is calculated by the receiver) back to the sender. To do that, let us recall from section 2 in which we mentioned the TCP sender cannot have a number of outstanding segments larger than the rcwnd which is advertised by its own receiver. By default, the TCP receiver advertises its available receiving buffer size, in order to avoid saturation by a fast connection (flow control). We propose to extend the use of rcwnd to accommodate the value of TCP_Contention in order to allow the receiver to limit the transmission rate of the TCP sender also when the path used by the connection exhibits a high contention and frame collision probability. Therefore, when TCC is used, the new value of rcwnd becomes the minimum of TCP_Contention and the available buffer size in the receiver (available_receiver_buffer).

$$rwnd = \min \{ available\_receiver\_buffer , TCP\_Contention \} \quad (5)$$

It is important to note that the value of TCP_Contention in every other RTT. In between of each change, the TCP_Contention remains fixed to make sure the packets received by the receiver are sent into the network after the sender has applied the changes imposed by the receiver in the last RTT.

b) Fair Backoff Algorithm

To address the inter-flow instability we propose a new scheme called *Fair Backoff Algorithm (FBA)* that aims to tackle the instability encountered by multiple flows sharing the channel while avoiding any control message exchange between competing nodes. This adds great advantage to make the algorithm practical and as simple as possible. Since the contention window is the main parameter used in each node to access the channel, we have defined three different stages in each node using FBA (figure 8).

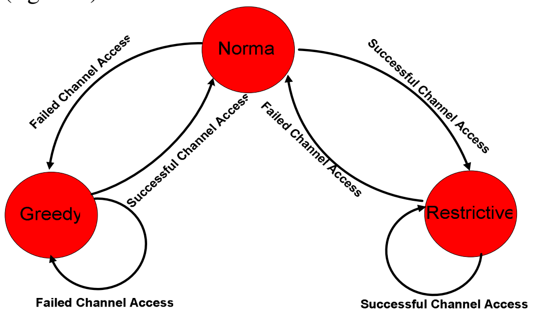

Fig. 8. Different node stages in a FBA scheme

Normal: this stage is entered when the station firstly has data to send in its buffer and secondly it has either recovered back from a failed channel access or it has failed to gain the channel after a successful transmission. The main purpose of this stage is two-fold. It first improves short-term fairness and thus instability as we will explain later in this section. Secondly, it decreases the number of collisions between contending nodes by assigning relatively large *CW* in this stage.

Restrictive: following a successful channel access in the Normal stage, the node enters a Restrictive stage where the probability of node's channel access decreases with respect to the number of consecutive channel access events and increases with its buffer size. This stage has been designed to force the successful nodes to release the channel in favor of others while giving higher priority to the congested node (such as node D in figure 1) compared to non-congested ones.

Greedy: if the node is unsuccessful after choosing its initial back off at Normal stage, it will enter the Greedy stage where the station takes high priority to access the channel by choosing relatively smaller contention window compared to successful stations. This stage is included to prevent nodes from getting starved while avoiding experience further collisions by the station.

Each node then chooses an appropriate CW following the rule given in equation (6) in each stage.

$$CW=\begin{cases} CW_{min} * Tradeoff\_co & Normal \\ CW_{min} * [\ 1+Success*(1-Buffer\_co)]\ & Restrictive \\ CW_{min} * Failed & Greedy \end{cases} \quad (6)$$

Here *Success* and *Failed* are the number of *consecutive successful and consecutive failed channel access tries* (and not necessarily failed transmission try) seen by each node, respectively. We have also defined two other variables called "Tradeoff Coefficient" (*Tradeoff_co*) and "Buffer Coefficient" (*Buffer_co*) which are both critical to the performance of the algorithm.

Tradeoff_co is an integer number between 1 (i.e. CW= CW_{min}) and 33 (i.e. CW=CW_{max}). It addresses the high probability of collisions between stations that are in Normal stage by assigning relatively large contention windows to such nodes. This will firstly result in smaller number of packet collisions and secondly will improve short-term fairness as it gives immediate equal opportunity to nodes coming from different stages regardless of their prior stage. On the other hand, a large value of Tradeoff_co will result to greater number of idle slots in the channel which obviously degrades the achieved throughput. Therefore, the value of Tradeoff_co can be seen as the tradeoff between the achieved fairness and the throughput. As we concluded in [13], the Tradeoff_co value of 10 shows to satisfy the objectives of introducing Tradeoff_co in the system.

The next parameter is Buffer_co which is a number between 0 and 1 and gives higher priority to nodes who are situated in a more congested area of the network. To understand the idea behind Buffer_co in the restrictive stage, consider again figure 5 where ideally the probability of accessing channel by node D should be higher than other relaying nodes as D is routing packets from 2 connections. We define Buffer_co as below:

$$Buffer\_co=\begin{cases} 0 & Buffer<=Thresh_{min} \\ \dfrac{Buffer - Thresh_{min}}{Thresh_{max} - Thresh} & Thresh_{min} < Buffer < Thresh_{max} \\ 1 & Buffer>=Thresh_{max} \end{cases} \quad (7)$$

where *Buffer* is the current number of packets inside the MAC buffer; $Thresh_{max}$ and $Thresh_{min}$ are chosen to be 20% and 80% of the maximum buffer size, respectively[1]. It should be noted that to measure the contention around each node we use Buffer_co. This is because to a large extent, the amount of contention around each node can be reflected in its buffer occupancy ratio.

V) RESULTS

The simulations were performed using OPNET simulator [15].The transmission range in each node is

[1] - These parameters were chosen following the RED [14] algorithm

set to 100m. The data rate is set to 2Mbps and the channel uses free-space with no external noise. Each node has a 20 packet MAC layer buffer pool and in all scenarios, the application operates in asymptotic condition (i.e., it always has packets ready for transmission). Nodes use DSR as the routing protocol. In transport layer, TCP NewReno flavor is deployed and the TCP advertised window is set its maximum value of 64KB so the receiver buffer size does not affect the TCP congestion window size. TCP MSS size is assumed to be fixed at 1460B. RTS/CTS message exchange is used for packets larger than 256B (therefore no RTS/CTS is done for TCP-ACK packets). The number of retransmission at MAC layer is set to 4 for packets greater than 256B (Long_Retry_Limit) and 7 for other packets (Short_ Retry_Limit) as has been specified in IEEE 802.11 MAC standard. All scenarios consist of nodes with no mobility.

a) Cross Topology

To verify the performance of the proposed algorithms (TCC+FBA), we first use a cross topology showed in figure 5 with the change that while connection 1 runs from the beginning of the simulation, connection 2 start time is set to 300 seconds. Our main goal of conducting this simulation is to show the effect of the intra-flow instability (from start time to time 300) and the inter-flow instability (from time 300 seconds to the end of simulation) and show how TCC and FBA can alleviate each problem, respectively. Fig.9 depicts the throughput seen by connection 1 over the simulation time. It can be easily seen that in default TCP and 802.11, until time 300 seconds, the inter-flow instability causes frequent TCP throughput drop to zero. Also it is obvious the instability becomes more severe, as soon as connections 2 starts at time 300 where due to intra-flow instability connection 1 rarely gain access to the channel. On contrary, using TCC + FBA eliminates both intra-flow and inter-flow instability to a great extent and result into more smooth and stable TCP throughput during simulation time.

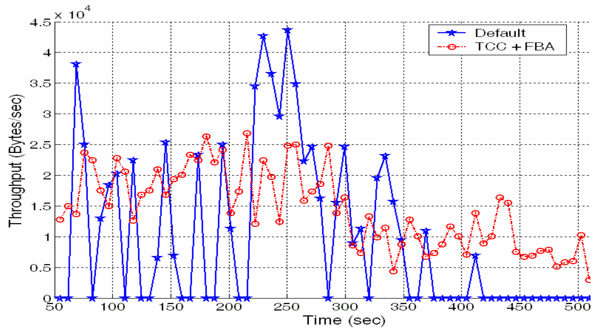

Fig. 9. TCP throughput in a cross topology

To further illustrate the effectiveness of TCC and FBA in addressing the problems explained in section 3, fig.10 depicts the smooth average number of packets buffered in all nodes and fig.11 shows the Jain's fairness index[1] respectively. This is because as explained in section 3, the former graph is linked to the intra-flow instability and can be used to assess the effectiveness of TCC while the latter can be interpreted as the inter-flow instability and hence be used to evaluate the FBA. It can be seen from fig.10 that while in default operation of TCP and 802.11, on average 2.4 packets are queued in each node, the number declines to 1.3 when TCC and FBA are introduced. On the other hand, in default settings, the average queue size experiences a sharp increase when connection 2 starts at time 300, while the number of buffered packets almost is unchanged in the new algorithm. This confirms the TCC is efficiently controlling the amount of outstanding data in the network regardless of number of contending connections. Similarly, fig.11 suggests the introduction of FBA has magnificently improved both short term (small sliding window) and long term (large sliding window) fairness by giving equal chances to both connections to access the channel and therefore guarantees smooth and stable TCP throughput for both connections.

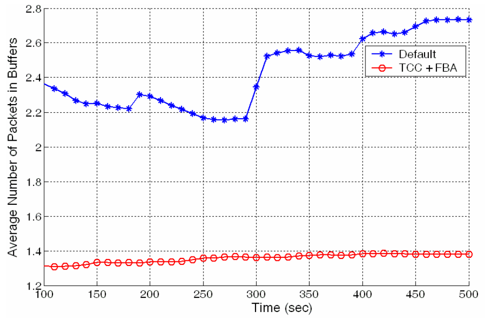

Fig. 10. Average number of packets buffered in all nodes in cross topology

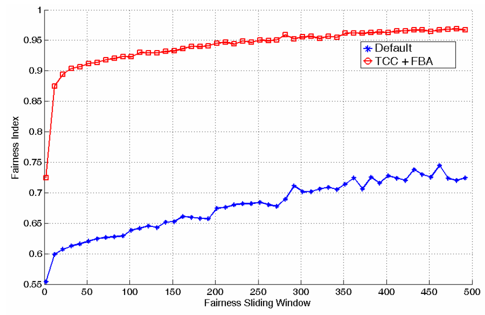

Fig. 11. Fairness Index in a cross topology

b) Grid Topology

To further verify the performance of the TCC and FBA algorithms, we used a grid topology in fig.12

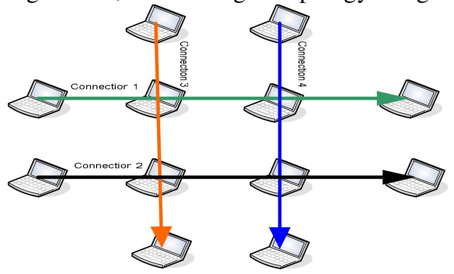

Fig. 12. 4x4 Grid topology

Fig.13 and Fig. 14 show the average number of packets buffered in all nodes and the fairness index measured between all four connections, respectively.

[1] - The Jain's fairness index graphs in this paper are calculated using the fairness sliding window technique described in [16]

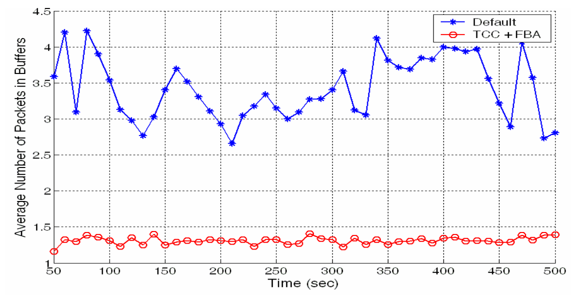

Fig. 13. Average number of packets buffered in all nodes in a grid topology

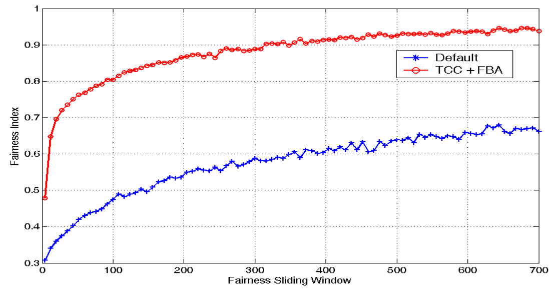

Fig.14 Fairness Index between 4 connections in grid topology

The results show that in scenarios with higher number of interacting TCP flows and therefore higher contention, the TCC + FBA still outperforms the default operation of TCP and 802.11.

To show the TCP stability improvements using the proposed algorithms in a grid topology, Table 1 compares the aggregated throughput, total number of TCP retransmissions, average RTT and average of RTT standard deviations for all connections.

Table 1- TCP measurements in a grid topology

	Default	TCC+FBA
Aggregated TCP Throughput (Bytes/sec)	42347	51385
Total number of TCP retransmissions	1535	345
Average RTT (sec)	0.2425	0.1021
Average of RTT standard deviations	0.2524	0.0826

It is obvious that the introduction of TCC+FBA has resulted into higher and smoother throughput and end-to-end delay compared to default TCP and 802.11. In particular, it is interesting to note that the RTT has been almost halved thanks to TCC that tries to minimize the unnecessary network load and therefore decreasing the contention and queuing delay experienced by individual packets in the network.

VI) CONCLUSION AND FUTURE WORK

In this paper, we addressed the TCP instability by carefully tracking the chain of events occurring between link layer and TCP. We divided the TCP instability problem into intra-flow and inter-flow instability where the former instability is caused by the interaction of nodes belonging to the same TCP connection while the latter happens when nodes belonging to different connections interact. Based on that, we proposed two separate solutions named TCC and FBA to tackle the intra-flow and inter-flow instability, respectively. The main characteristic of TCC was to control the amount of outstanding data in the network and hence alleviate intra-flow instability by monitoring the level of contention in the link layer and achieved throughput in the transport layer. On the other hand, FBA addressed the inter-flow instability by improving the channel access fairness between contending TCP flows. As the initial results obtained from a cross and a small grid topology were promising, in future we plan to extend our simulations to a more general scenarios with larger number of TCP flows in a medium to large scale random topology.

References

[1] "IEEE Standards for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY),Part 11:Technical Specifications" .1999

[2] Z. Fu, X. Meng, and S. Lu, "How Bad TCP Can Perform in Mobile Ad Hoc Networks", IEEE Symposium on Computers and Communications, 2002

[3] Z. Fu and others, "The Impact of Multihop Wireless Channel on TCP Performance," IEEE Transactions on Mobile Computing, vol. 4, no. 2, pp. 209-221, 2005.

[4] K. Nahm, A. Helmy, and C.-C. J. Kuo, "TCP Over Multihop 802.11 Networks: Issues and Performance Enhancement", MOBIHOC, 2005

[5] X. Yu, "Improving TCP Performance Over Mobile Ad Hoc Networks by Exploiting Cross-Layer Information Awareness", MobiCom, 2004

[6] P. C. Ng and S. C. Liew, "Re-Routing Instability in IEEE 802.11 Multi-Hop Ad-Hoc Networks", 29th Annual IEEE International Conference on Local Computer Networks, 2004

[7] W.Stevens, "RFC 2001 - TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms,".Technical Report, Jan.1997.

[8] M.Allman, V.Paxson, and W.Stevens, "RFC 2581 - TCP Congestion Control,".Technical Report, Apr.1999.

[9] S.Xu and T.Saadawi, "Does the IEEE 802.11MAC protocol work well in multihop wireless ad hoc networks", 39 ed pp. 130-137.2001

[10] C. Ware and others, "Unfairness and Capture Behaviour in 802.11 Adhoc Networks", IEEE International Conference on Communications, 2000

[11] Zhifei Li, Sukumar Nandi, and Anil K.Gupta, "Modeling the Short-Term Unfairness of IEEE 802.11 in Presence of Hidden Terminals," Accepted to Appear in Elsevier Performance Evaluation, 2006.

[12] E Hamadani and V Rakocevic, "Evaluating and Improving TCP Performance Against Contention Losses in Multihop Ad Hoc Networks", Marrakech, Morocco, 2005

[13] E Hamadani and V Rakocevic, "TCP Contention Control: A Cross Layer Approach to Improve TCP Performance in Multihop Ad Hoc Networks", 5th International Conference on Wired / Wireless Internet Communications, 2007

[14] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Transactions on Networking, vol. 1, no. 4, pp. 397-413, 1993.

[15] OPNET simulator, http://www.opnet.com

[16] C. E. Koksal, H. Kassab, and H. Balakrishnan, "An Analysis of Short-Term Fairness in Wireless Media Access Protocols", Proceedings ACM SIGMETRICS, 2000