Coupling Loss and Delay Differentiation to Enhance TCP Performance within Wireless Multi-hop Ad-hoc Networks

Alaa GHALEB-SEDDIK and Yacine GHAMRI-DOUDANE
Ecole Nationale Superieure d'Informatique pour l'industrie et l'Entreprise (ENSIIE),
1 square de la Résistance, 91025 Evry CEDEX - France
Email: {seddik, ghamri}@ensiie.fr

Sidi Mohammed SENOUCI
University of Bourgogne, ISAT
49 Rue mademoiselle Bourgeois, 58000 Nevers, France
Email: Sidi-Mohammed.Senouci @ u-bourgogne.fr

***Abstract*—Most existing TCP variants cannot distinguish between different packet loss causes within MANETs. TCP was, mainly, developed to deal with network congestion errors. While within MANETs, there are packet loss causes other than congestion. Studying the behaviour of TCP in front of such losses, we notice that TCP doesn't have always the optimum behaviour as it reacts, in most cases, without considering the loss cause. This misbehaviour might cause network performance degradation and resources' waste. To overcome this problem, many LDAs have been designed. However, these LDAs were optimized for data networks where wireless link is only the last hop, meaning that they might be inadequate for MANETs. Also, the proposed LDAs deal only with losses due to wireless channel and/or congestion-induced errors. We show, in this paper, the importance of dealing with a third loss cause that is common in MANETs, which is link failure. We propose a new TCP variant that is called TCP-WELCOME. TCP-WELCOME can: (i) identify the loss cause by coupling loss and delay information, and (ii) trigger the appropriate packet loss recovery according to the identified loss cause. The performance evaluation, through both simulations and experimental tests, shows that TCP-WELCOME optimizes both energy consumption and achievable throughput. TCP-WELCOME does not change the standard and can operate with existing TCP variants.**

***Index Terms*— TCP, MANET, Loss Differentiation Algorithm, Loss Recovery Algorithm, Energy consumption, Throughput, Simulation, Experimental evaluation.**

I. INTRODUCTION[1]

Wireless multi-hop ad-hoc networks differ from traditional wired networks by the multitude of packet loss situations to which they are subjected. This is due to the intrinsic characteristics of wireless channels (e.g. signal fading, interference, obstacles, and environment effects) that might obstruct the proper reception of data packets at the other end. Moreover, in some cases, these vulnerabilities of wireless channel can result in a complete link failure. Although link failure is of low probability in wired networks, since physical cables constitute the data transmission media, it is rather common in MANETs (due to nodes' mobility, battery depletion, or obstacles). The volatility of the communication channel is a typical problem with wireless links, which is not the case with wired cables. TCP is a transport protocol that aims at ensuring high reliability by guaranteeing the reception of data packets. However, TCP was designed primarily to address network congestion, which is the main cause for data packet loss in wired networks. Therefore, other types of data packet loss encountered in MANETs are prone to misinterpretation by TCP, which, in turn, will lead to TCP performance degradation. In order to overcome the performance limitation of TCP when deployed in MANETs, we propose a new TCP variant that we call TCP-WELCOME. TCP-WELCOME optimizes the performance of TCP in both terms of the achievable throughput and the energy consumption of TCP's nodes within the network. This is achieved through its ability to distinguish among, and efficiently deal with, the different data packet loss situations encountered in MANETs. TCP-WELCOME's main idea is based on coupling loss and delay information over the connection in order to classify the cause of packet losses and then reacting properly to recover from them. We mention that, this work extends our previous study conducted in [1]. In this presented work, a complete performance evaluation using a realistic test-bed configuration is conducted to reinforce our previously performed simulation study.

This paper is organized as follows: we start by discussing, in the next section; the main issues that might influence TCP performance in MANETs. After that, we present the related work in section 3, and then we present our proposition to enhance TCP performance within MANETs (TCP-WELCOME) in section 4 describing its main algorithms in detail. In Section 5, we describe the validation process of TCP-WELCOME through

[1]Preliminary version of this paper appeared in the Proceedings of IEEE COMSNETS'2009.

Manuscript received December 12, 2011; revised February 12, 2012; accepted March 28, 2012.

doi:10.4304/jcm.7.12.859-872

simulations and realistic test-bed configuration and discuss the obtained results. Then, we conclude our work in this paper in section 6.

II. TCP within Wireless Ad-hoc Networks

MANETs obviously inherit wireless channel related problems. In addition, ad-hoc networks suffer from other problems related to their specific characteristics, such as transient network partitions, route failures that would result from nodes' mobility, nodes' battery depletion, and the multi-hop nature of the communications. In order to improve TCP performance over MANETs, it must be able to distinguish and to recover from the new data packet loss types that arise within such networks. The new challenges that TCP would confront within such networks can be classified into: wireless lossy channels, multi-path routing, network partitions, network topology and the surrounding environment, link failures, and power constraints. We discuss these issues in the following sections.

A. Wireless Lossy Channel

Wireless channel errors cause packet corruption and result in TCP segments and/or *ACK* loss. When *ACKs* do not arrive at the TCP sender within certain amount of time (Retransmission Time-Out or *RTO*), TCP sender retransmits that segment, exponentially backs off its retransmission timer, reduces its congestion control window threshold (*SSThreshold*), and closes its congestion window (*CWND*) to one segment. Frequent channel errors lead to having small congestion window continuously at the sender side resulting in low connection throughput [2].

B. Network Partition

MANETs may periodically get partitioned for several seconds at a time. If the TCP sender and receiver are in different partitions, all the sent packets will be dropped and TCP sender invokes its congestion control algorithm. If the network remains partitioned for a time relatively high to *RTO*, the situation gets worse because of the "serial timeouts phenomena". Serial timeout happens due to multiple consecutive retransmissions of the same segment while the receiver is disconnected from the sender. Thus, all these retransmissions are lost. Since the retransmission timer at the sender side is doubled with each unsuccessful retransmission attempt (until it reaches 64 sec), several consecutive failures can lead to inactivity that might last even when the sender and receiver get reconnected [2]. The most adequate solution here is to stop data transmission (to avoid flooding the network with packets that cannot be delivered) till the TCP sender/receiver get reconnected.

C. Topology and Environment

The location of nodes and the nature of their surrounding environment determine inter-node reachability and the amount of received interference [3]. If the nodes are located close to each other, there will be a greater chance that data will not have to make as many hops as in a network where the nodes are further apart. Also, networks with a dense concentration of nodes will experience more contention for the available capacity and hence more collisions and interference leading to high TCP packet loss and thus frequent TCP sender congestion control algorithm triggering. On the other hand, walls and objects that hinder radio transmission decrease the effect of high node density.

D. Link Failures

In case of nodes' mobility, each node might move out of the communication range of old neighbors or into the communication range of new ones. This can break the established routes (link failures) and trigger the establishment of new ones within the network. The implemented ad-hoc routing protocol is always in charge of recovering from link failure and allowing to maintain the communication session between the involved end points. Usually, a broken route results in performance degradation, since no data can be exchanged during the time where no new route available. In fact, high mobility is not always a bad thing for ad-hoc networks. Some authors have observed that mobility can increase performance by distributing traffic more evenly over the network [4]. The problem of TCP in the case of link failure is that after resuming the data communication session, TCP sender starts from the Slow-Start phase, with minimum *CWND* over the links. Indeed, during a link failure event, multiple data packets can be lost at once. Thus, TCP sender shrinks its *CWND* to minimum assuming that the loss is due to congestion. However, in case of link failure, the new discovered route might have higher link capacity compared to the lost one. Thus, TCP sender will waste the available bandwidth (which is a scarce network resource) over the connection in this case.

E. Power Constraints

In MANETs, the devices are battery operated. Obviously, in order to ensure good connectivity of the network, the lifetime of network nodes should be maximized. Increasing this lifetime can be done through minimizing the node's energy consumption (i.e. designing network protocols that lead to less energy consumption). In addition, losing a node due to battery depletion leads to broken communication sessions (link failure) even if the node is not the sender or the receiver side of that session. This is because each node within the network forwards data packets when it is involved in multi-hop path.

III. Related Work

In this section, we discuss the main TCP congestion control enhancements proposed in order to improve its performance within wireless and ad-hoc networks. Loss differentiation algorithms can be categorized into two categories [5]: (i) implicit or end-to-end differentiation, and (ii) explicit loss differentiation algorithms. Unlike the implicit ones, explicit algorithms use agents that are deployed on the network's intermediate nodes. End-to-end or implicit solutions could involve the sender side only (e.g. TCP Westwood) [6] [7] or both the sender and

receiver sides (e.g. the 3 Duplicate *ACKs* sent by the receiver to notify the sender of a packet loss).

A. Implicit Loss Classification Algorithms

TCP Westwood is an example of implicit loss classification algorithms. TCP Westwood [6] [7] [8] is a sender-side modification of TCP New-Reno [9] that estimates the connection bandwidth based on the rate of the received *ACKs*. TCP Westwood uses the estimated bandwidth to adjust and set its *CWND* and *Slow-Start* threshold parameters. This is in contrast to traditional TCP congestion control implementation, where both *CWND* size and *Slow-Start* threshold are halved whenever a data packet loss is detected within the connection [10]. It had been shown in the literature [8] that this bandwidth estimation algorithm enhances the performance of TCP, in front of random, sporadic data packet losses (wireless channel related errors). In [11] the authors illustrate that, in the link failure case, both TCP New-Reno and TCP Westwood recognize the packet loss with *RTO* expiration. Thus, both react the same way by backing off for a while and entering Slow-Start phase. In the link failure case, the average goodput of TCP Westwood is less than that of TCP New-Reno [9]. This is due to the lost *ACKs*. Indeed, in order to estimate the end-to-end bandwidth and discriminate among loss causes, TCP Westwood relies on the received acknowledgments. In a situation where there are several acknowledgments lost, this may lead to a wrong estimation of the end-to-end bandwidth and consequently to a TCP Westwood misbehavior. We also found that TCP Westwood has higher energy consumption per received bit than TCP New-Reno in most cases [11]. It can also be noticed that TCP Westwood energy consumption gets worse when wireless channel conditions degrades (i.e. increase of the Bit Error Rate). Its dependence on RTT measurements to calculate the estimated bandwidth is also responsible of this effect. Similarly to the link failure case, as the Bit Error Rate (BER) increases over the wireless channels, the returned *ACKs* become prone to loss and corruption. These lost or corrupted *ACKs* can yield to mistaken estimated bandwidth calculations. Another enhancement of TCP's congestion control algorithm is the network congestion avoidance algorithm implemented within TCP Vegas [12]. TCP Vegas relies on measured *RTT* values of sent packets to extend Reno's retransmission mechanisms. According to this measurement, the *RTO* value is updated. When a duplicate acknowledgement is received, Vegas checks if the difference between the current time and the timestamp recorded for the first unacknowledged segment (i.e. its *RTT*) is greater than the timeout value. If so, then it retransmits the segment without having to wait for three duplicate acknowledgements. This change helps TCP Vegas to detect losses much sooner than TCP Reno [12] and other variants. Also, TCP Vegas uses RTT values to calculate the actual *CWND* in the network. Hence, by comparing this value with the expected throughput in the network, TCP Vegas decides how to adapt its *CWND* after loss episodes. TCP Vegas still contains Reno's coarse-grained timeout code as a fallback mechanism. This enhancement improves the performance of TCP in term of throughput as it discovers data packets losses faster than the other variants and in turn recovers from these losses faster, in the case of good estimation or measurement of the *RTT* values over the connection. But, in case of wrong *RTT* values measurement, as when the connection starts and there is already congestion over the network links, the *CWND* calculation will be wrong and might cause a persistent congestion over the connection.

B. Explicit Loss Differentiation Algorithms

Explicit loss identification can be performed through different estimation techniques. In [13], for example, a sender-side method of end-to-end loss differentiation and adaptive segmentation (Robin) is proposed, for enhancing TCP performance in heterogeneous[2] networks. This LDA enables the TCP sender to distinguish congestion from wireless induced losses. Moreover, in order to improve the error recovery phase during a non-congestion period, an adaptive segmentation algorithm is proposed. This algorithm enables the TCP sender, if packet loss is detected, to retransmit smaller packets, having aggregate payload equal to the payload of the lost packet. Decreasing segment size reduces the Packet Error Rate (PER) [14]. In the case of high propagation delays over the network, the evaluation results of this algorithm show that the improvement is negligible. We have to note here that the proposed solution assumes that only the last hop is a wireless link. While in MANETs, all the communication links are wireless channels and the propagation delay may vary significantly compared to the case of a single hop wireless network. In [15] the authors propose a cross-layer solution based on two LDA algorithms in order to classify the loss cause on 802.11-link and react accordingly. The first LDA scheme, acting at the MAC layer, allows differentiating losses due to signal failure caused by displacement or due to noise from other loss types. In this case, it adapts the behavior of MAC layer to avoid a costly end-to-end TCP resolution. The second LDA scheme, which acts at the TCP layer, differentiates losses due to interferences from those resulting from congestion and adapts TCP behavior accordingly. The work done here is considering only single hop (last hop) wireless networks. The Spike Scheme [16], at the receiver side, measures one-way delays. The receiver switches between congestion state and wireless state according to a certain threshold. If the delay exceeds this threshold, it is a congestion state. Otherwise, it is a wireless state. ZigZag Scheme presented in [17], extends the Spike scheme to include both mean and standard deviation values of the measured delays as well as the number of packet losses when computing the delay threshold used. According to this calculation, the higher the number of packet losses, the greater the threshold beyond which a congestion state is assumed. In other words, the wireless state becomes the most likely cause of data packet losses.

From the above, we can see that most of the work done in this domain addresses the problems of TCP within wireless infrastructure networks. All these contributions

[2] Mixed wired/wireless environments.

assume that the wireless channel connection is only the last network's hop. However, in MANETs, all the communication channels are wireless links. In addition, the proposed LDA that addresses the link failure problem is implemented at the MAC layer level and not at the Transport layer. The cross-layer solutions may complicate the deployment and the acceptance of the solution. Furthermore, such solution is also proposed for one-hop wireless networks and its extension to wireless multi-hop ad-hoc networks is not straightforward. We will see in the next section that a link failure case within MANET requires a specific reaction from TCP in order to recover from packet loss. Link failure situations within such networks introduce burst data packet loss over the connection. Although that burst losses could be the result of a network congestion event, the reaction of TCP in front of link failure data losses assuming that it is due to network congestion is an aggressive, inefficient reaction.

IV. Enhancing TCP Performance within MANETs: TCP-WELCOME

MANETs, suffer from the effect of wireless channels (e.g. fading, multi-path routing, interference), the effect of ad-hoc network environments (mainly link failure due to mobility or battery depletion) in addition to the network congestion effects due to buffers overflow. Hence, we have three different reasons (not only two as discussed in previous researches) to lose data packets within MANETs. Therefore, we propose new TCP loss differentiation and loss recovery algorithms that can distinguish among these three loss situations within wireless ad-hoc network environments. This section is organized as follows: we start by presenting the main algorithms of TCP-WELCOME: (i) TCP Loss Differentiation Algorithm that is used to classify the data packet loss cause and (ii) TCP Loss Recovery Algorithm used to recover from each loss type.

A. Loss Differentiation Algorithm (LDA)

With respect to all the concerns and suggestions discussed above, we need an adapted LDA algorithm that enables TCP to correctly classify the cause of data packet losses within MANET environments. This algorithm should be able to differentiate between the most common data packet loss causes in MANETs; taking into consideration and dealing with new loss causes that had not been well investigated before (i.e. link failure). In order to decrease the execution overhead of TCP algorithms and the interaction between the intermediate nodes within the network, our LDA and LRA algorithms are end-to-end sender side modifications to the legacy TCP (i.e. TCP New-Reno). TCP-WELCOME relies on the evolution of RTT samples of sent packets at the sender side in order to take its decisions. We will see later how *RTT* samples evolution can be used to classify different data packet loss causes.

In order to detect the packet loss cause, let $q_{d,i}(t)$ and $P_{d,i}(t)$ be the queuing and processing delays of node *i* at a given time *t* respectively, while $p_{d,l}(t)$ is the propagation delay over the link *l* between two consecutive nodes of the communication path at a given time *t*.

Fig. 1 illustrates the delays experienced by a TCP sent data packet over the connection until receiving the *ACKs* at the sender side. Thus, *RTT* value of a sent packet over a TCP connection, where the route contains *n* hops in the forward path between the source and the destination and *m* hops in the reverse path (from the destination towards the source), at time *t* is calculated as follows:

$$RTT(t) = \sum_{i=1}^{n}[q_{d,i}(t) + p_{d,l}(t) + P_{d,i}(t)] + \sum_{i=1}^{m}[q_{d,i}(t) + p_{d,l}(t) + P_{d,i}(t)] \quad (1)$$

Henceforth, we will consider only the propagation and queuing delays as these values are highly affected by network changes. Whereas, processing delay depends solely on the communication node capabilities and not on network conditions. It is obvious that, when there is a link failure within the network, the propagation delay will change according to the new recovered route. Moreover, with network congestion, the queuing delay will increase. The next section describes the proposed Loss Differentiation Algorithm (LDA) for TCP enhancements. We will later discuss how TCP should adjust its parameters (*RTO*, *CWND*, and *SSThreshold*) according to the identified loss cause.

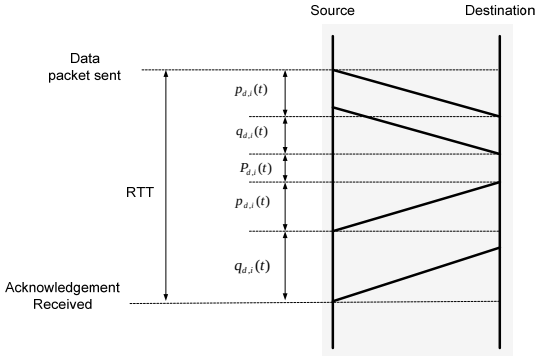

Figure 1 Round Trip Time (RTT) of a TCP sent data packet

1) Coupling Loss and Delay for Loss Differentiation: In this section, we identify the basic concept of our proposed LDA in order to classify the different data packet loss causes over a TCP connection. The main idea is based on observing the history of *RTT* samples evolution within the network and the way in which TCP identifies the data packet loss (Fig. 2). Next, we will discuss how TCP can use *RTT* values as an indication of each type of data packet loss.

a) Network Congestion Event: When the network suffers from a congestion situation, the queuing delay increases as the nodes' buffers are filled with time. So, when a packet loss occurs and the evolution of *RTT* samples at the sender side is seen to be increasing gradually, the loss is more likely to be due to network congestion. This remains true regardless of how TCP recognizes data losses; 3 Duplicate *ACKs* or *RTO* expiration. This only gives information about the importance of the congestion and thus the actions to be undertaken in order to recover from it. Fig. 3 gives an example of *RTT* evolution in the case of network congestion. It is clear from the figure, that before losing data packets due to network congestion,

the evolution of *RTT* values over the TCP connection increases gradually. Such gradual evolution is the indication that we consider for an imminent network congestion episode.

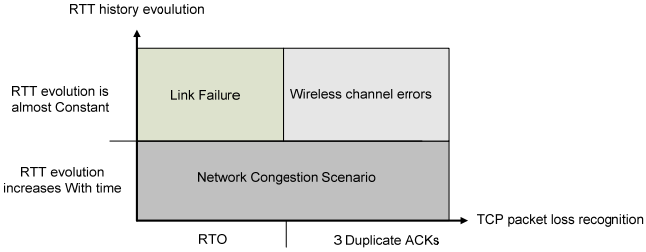

Figure 2. TCP Proposed Loss Differentiation Algorithm

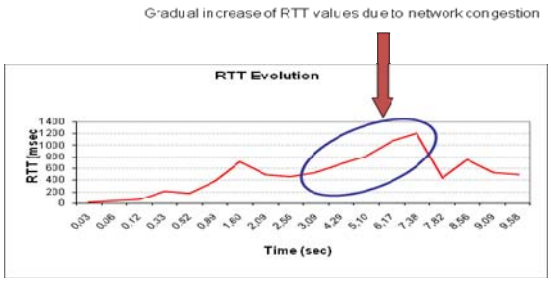

Figure 3. RTT Variations within wireless ad-hoc network (Network Congestion)

b) Wireless Channel Related Losses: If the evolution of *RTT* samples over the connection is not highly fluctuating (i.e. staying around an average value) and the data packet loss is identified through three Duplicate *ACKs*, thus the data packet loss is due to wireless channel inefficiency on one of the links over the communication path. While in wireless channel related losses, both queuing and propagation delays are almost constant, and *RTT* samples over the connection should not experience high fluctuations with time. Additionally, when there is a valid route between the source and the destination, despite the presence of link errors, the source can always receive *ACKs* from the destination. The corruption of *ACKs* is of a lesser probability since the *ACK* packet size is relatively small. Thus, in the case of wireless channel induced losses, *RTT* samples will stay around an average value (± RTT_THRESHOLD):

$$RTT(t) = \sum_{i=1}^{n}[q_{d,i}(t) + p_{d,i}(t)] + \sum_{i=1}^{m}[q_{d,i}(t) + p_{d,i}(t)] \approx Const. \qquad (2)$$

Hence, if a packet loss is identified by the reception of three Duplicate *ACKs* and at the same time we notice that RTT values stays almost constant, this means that data packet losses over the connection are more likely to be due to wireless channel errors.

c) Link Failure Loss Event: As stated previously, in case of link failure, TCP may experience packet losses in a burst. At the same time, we can notice that before such event there is no reason to have an increase in the *RTT* value (as no congestion is foreseen). From this we can state that, if the evolution of *RTT* samples over the TCP connection is relatively constant and TCP recognizes data packet losses through *RTO* expiration, then data packet loss is more likely to be due to a link failure situation along the route towards the destination. We should also note that in the case of link failure, after link loss recovery, the following observations could be noticed: both Propagation and Queuing delays change suddenly since the new discovered route might (i) not be having the same length (i.e. number of hops) as the lost route, or (ii) be more/less loaded than the lost one. In the above simulated link failure situation (Fig. 4), we can see that RTT evolution, after a certain time of the simulation's onset (during which the ad-hoc routing protocol finds a route towards the destination), the connection enters in a steady state phase where the *RTT* evolution stays almost constant. The Figure shows that before the link failure event, *RTT* fluctuation is not high and can be considered within an average value. Actually, in the event of link failure, two situations may occur depending on the time required by the ad-hoc routing protocol to recover the failed link or to find an alternative one: (i) If this time is shorter than TCP's *RTO*. TCP identifies packet losses through duplicate *ACKs*. When the TCP sender checks the evolution of *RTT* samples and finds them relatively constant, it will classify the loss as wireless-channel related. TCP, then, will verify the *CWND* and modify it to be relevant to the Slow-Start threshold (as will be seen below). This action is less aggressive compared to the traditional TCP assumption that losses are due to network congestion. (ii) If this time is longer than TCP's *RTO*. TCP sender identifies data packet losses through *RTO*. When the TCP sender checks *RTT* samples' evolution and finds them almost constant, TCP will classify the packet loss as link failure related and reacts accordingly.

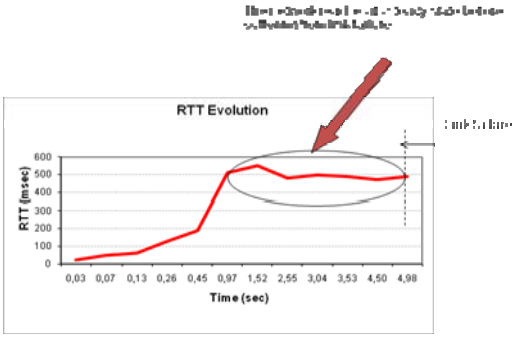

Figure 4. RTT Variations within wireless ad-hoc network (link failure)

B. Loss Recovery Algorithm (LRA)

Upon identifying the packet loss cause through the proposed LDA, TCP-WELCOME should react with the most appropriate action accordingly (Fig. 5).
TCP reaction should be as follows:

- When wireless errors increase over the network channels, it is unnecessary to stop data transmission or to decrease TCP's transmission data rate after a loss event.

- In case of link failure within the network (i.e. a broken route between the communicating end points), it will be sufficient to stop data transmission till an alternative route towards the destination is found. Route re-establishment/recovery is the responsibility of the implemented routing protocol. The transmission rate here will be adjusted according to the available bandwidth of the new route. It is obvious that, the length and the load of the communication path impact the Round Trip delay Time (*RTT*) between the end points. Hence, it would be necessary, in this case, to recalculate both the TCP's *CWND* and *RTO* values according to the characteristics (length and load) of the new route. We notice here, that this reaction of TCP-WELCOME in front of data link failure, avoids the effect of the previously presented "serial time out phenomena". Serial time out phenomena could be due to the unawareness of the TCP source about data link failure event within the path, and thus it keeps sending data over the lost link.

- When there is a congestion situation in the network, TCP should keep its traditional behavior. It reacts according to how the congestion had been detected (3 Duplicate *ACKs* or *RTO*). In all congestion cases that are detected through retransmission time out, TCP stops data transmission during a certain period of time and resumes it afterwards with a reduced data transmission rate.

In the following sections, we explain how the TCP connection parameters (*CWND* and *RTO*) are adjusted after each data packet loss event over the connection. Before that, let us explain our interest in these two parameters. Indeed, the way in which TCP adapts its *CWND* has a direct impact on its performance in terms of throughput and energy consumption. More transmitted and less retransmitted data packets over the connection leads to better exploitation of the available bandwidth.

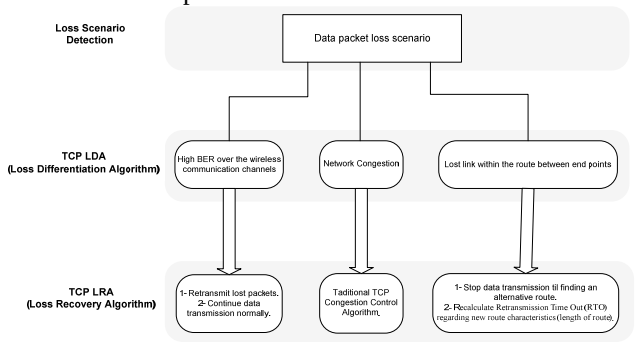

Figure 2. TCP Proposed LDA and LRA Algorithms

Also, unnecessary packet retransmissions lead to higher nodes' energy consumption. Also, the time that TCP waits; after the loss occurrence and before resuming the communication session (*RTO*) has a severe impact on its performance. For example, if the *RTO* is less than *RTT* value over the connection, unnecessary data packet retransmissions will lead to TCP performance degradation in terms of both achievable throughput and energy consumption. On the other hand, a too long *RTO* value will in turn lead to resource waste as the available bandwidth may not be well utilized. Hence, the efficient adaptation of these values over the connection is crucial to improve TCP performance.

a) RTO Adaptation Algorithm: The choice of the ad-hoc routing protocol algorithm is important from two points of view: (i) its robustness and promptness to recover from a link failure, (ii) the overhead and frequency of its routing information update messages which might result in a congestion or traffic interference over the network links. For example, if the time needed by the implemented ad hoc routing protocol to recover from link failures is longer than the TCP's *RTO*, TCP triggers its congestion control algorithm, and backs off for a certain period of time, then enters Slow-Start phase. Also, it might happen that the routing protocol recovers from the link failure but TCP stays in the idle state, since TCP does not know about the link recovery. On the other hand, if the time taken by the ad-hoc routing protocol is lower than TCP's *RTO*, TCP may recover from data packet loss without entering Slow-Start phase and decreasing its *CWND* to minimum. Moreover, the overhead of ad-hoc routing update messages could aggravate the congestion situation over the TCP connection. This leads to more congestion control actions triggered to recover from the packet losses. Thus, it is important to inform TCP of the route re-establishment or at least to give it the ability to discover this information as soon as possible to help TCP recover faster after a link loss recovery without waiting unnecessarily and wasting the network resources.

RTO estimation differs from *RTT* estimation in three ways. First, the goal is not to accurately estimate the truly maximal possible *RTT*, but rather a good compromise that balances avoiding unnecessary retransmission timeouts due to low *RTO* value, versus being slow to detect that a retransmission is necessary when the *RTO* value is high. Second, the TCP sender needs to estimate the *RTT* of data packets, the time taken from the sender to the receiver plus the time required at the receiver side to generate an *ACK*. For example, a receiver employing the delayed *ACK* algorithm may wait up to 500msec before transmitting an *ACK*. Thus, estimating a good value for RTO timer not only involves estimating a property of the network path, but also a property of the remote connection peer. Third, if loss is due to congestion, it might be necessary that the sender waits longer than the maximum *RTT* time, in order to give the congestion more time to diffuse from the network. If the sender retransmits as soon as the *RTT* time elapses, the retransmission could also be lost, whereas sending it later would be successful [18]. It has long been recognized that the setting of *RTO* timer cannot be fixed but needs to reflect the network path in use, and it generally requires dynamic adaptation due to the great extent to which *RTTs* could vary over the connection [19][20]. Thus, we propose that TCP-WELCOME adjust its *RTO* value according to the loss cause identified after each loss episode within the network. In the case of wireless channel related errors, no *RTO* estimation will be done. Whereas, when there is a

link failure case within the network, the *RTO* value have to be modified based on the characteristics (length and load) of the new route recovered by the routing protocol. The *RTT* value and its evolution after the loss episode are the best network performance parameters to depict those characteristics. So, we propose to make the *RTO* adaptation algorithm depending on the new *RTT* value over the new recovered route. It is obvious that the number of hops within the route between the source and the destination as well as the load of each link/node along this route affect the *RTT* value over the connection. Thus, with different routes that have different number of hops, we will have different end-to-end network delays. That's due to the fact that queuing delay is a variable component of the overall network/connection delay. We note that queuing delay is greatly influenced by the network/connection loads, and is difficult to analyze directly. In [21], the author shows the relation between the link utilization and the queuing delay and states that before that the link utilization reaches 70%, the queuing delay has the tendency to increase quietly slow. While after 70%, it increases sharply. Hence, Calculating the *RTO* value in such a way reflecting the characteristics of the new recovered route might be a good proposition.

Let RTT_{old} be the delay over the lost old route, and RTT_{new} be the delay over the new recovered route. Then, the new *RTO* could be calculated as follows:

$$RTO_{new} = \left(\frac{1}{a}\right) RTO_{old} \quad \text{and,} \quad a = \left(\frac{RTT_{old}}{RTT_{new}}\right) \tag{3}$$

Where *a*, is the performance parameter modification factor. The number of the *RTT* samples needed by TCP in order to calculate its performance parameters *(RTO_NEW_RTT_SAMPLES)* is determined through simulations. This modification of *RTO* value is made only in the case of link failure induced loss. Finally, let us precise that if network congestion is recognized to be the cause of the packet loss within the network, the *RTO* evolution stays the same as in traditional TCP New-Reno.

b) TCP Data Transmission Rate Adaptation: Bandwidth estimation algorithm is needed by TCP in order to well adjust its data transmission rate. Determining the available bandwidth of a new connection is a big issue in TCP. Clearly, if a transport protocol sender knows the available bandwidth, it would like to immediately begin sending data at that rate. But in the absence of knowledge about the available bandwidth, TCP must estimate it. In legacy TCP, this estimation is currently made by exponentially increasing the sending rate until experiencing packet losses. The loss is taken as an implicit signal that the rate had grown too big, so the rate is effectively halved and the connection continues in a more conservative way [18]. TCP bandwidth estimation algorithm can be a sender-side or a receiver-side estimation algorithm [18]. Aiming to make all our TCP enhancements in the same side of the communication end points (for ease of deployment), our proposed solution will be based on a sender-side bandwidth estimation algorithm. The implemented bandwidth estimation in TCP-WELCOME is that proposed by Antonio Capone and Fabio Martignon in [22]. TCP-WELCOME bandwidth estimation is based on the idea of received acknowledgements in order to estimate the number of data packets over the network. This has the advantage of eliminating the impact of interaction between intermediate nodes over the connection on TCP performance. We also apply a double filtering model to avoid errors that may lead to wrong bandwidth estimation. Estimating TCP data transmission rate is dependent on the networks' links capacity and the queuing or buffering conditions within the network's nodes. In the following, we will explain how the proposed TCP-WELCOME Loss Recovery Algorithm (LRA) adjusts its data transmission rate according to the data loss event (identified by LDA) within the network.

In the case of wireless channel related losses, there will be no modification of the data transmission rate in TCP-WELCOME. However, in the case of link failure along the route between the source and the destination, TCP-WELCOME should adjust its data transmission rate according to the characteristics of the recovered link. In this case, three propositions can be followed to decide how TCP may adjust its data transmission rate:

1. TCP-WELCOME can keep its actual *CWND* (before losses). Then, TCP-WELCOME adjusts it according to its congestion control algorithm, if necessary.
2. TCP-WELCOME might decrease its *CWND* automatically after data loss episode. We may propose to half the actual *CWND* before loss. This could be considered as a conservative action of TCP-WELCOME. In this way, we minimize the risk of having a congestion event over the links (i.e. in case the new route is more loaded than the lost one). In this case, the *CWND* will be calculated as follows:

$$CWND_{new} = \frac{CWND_{old}}{2} \tag{4}$$

Where $CWND_{new}$, is the adjusted TCP-WELCOME data transmission rate after data loss episode, and $CWND_{old}$ is the actual data transmission rate (i.e. before the loss episode). At the same time the new Slow-Start threshold of TCP will be calculated as follows:

$$SSThreshold = Bw_{estimated} * RTT_{\min} \tag{5}$$

Where $Bw_{estimated}$, is the estimated available bandwidth that is calculated by TCP.

3- TCP-WELCOME can adjust its *CWND* according to the proportion of the new *RTT* value over the new recovered route to that over the lost one. Here again, we follow a conservative mechanism in order to prevent a congestion episode over the network links. We take, here, the realistic assumption that the new discovered route contains other competing data traffic. With this conservative algorithm we try to help enhancing TCP-WELCOME fairness within the network. The new values of *CWND* and *SSThreshold* will then be calculated as follows; let RTT_{old} be the delay over the lost old route, and RTT_{new} be the delay over the new recovered route:

$$CWND_{new} = a.CWND_{old} \quad \text{and,} \quad a = \left(\frac{RTT_{old}}{RTT_{new}} \right) \tag{6}$$

$$SSThreshold = Bw_{estimated} * RTT_{\min} \tag{7}$$

Thus *a*, is the performance parameter modification factor.

Again here, the number of the *RTT* samples needed by TCP in order to calculate its performance parameters *(RTO_NEW_RTT_SAMPLES)* is determined through simulations. We must note here that, the *CWND* adaptation algorithm tends to calculate and re-adjust the initial *CWND* value over the connection after link loss situation. Thus, for example, if the bandwidth of the new discovered route is much larger than the previous lost one, while queuing and propagation delays are almost the same, TCP-WELCOME starts the communication after route recovery from this calculated *CWND* value rather than starting from minimum (*1 segment*), and then increases its *CWND* normally. This *CWND* adaptation algorithm tends to avoid wasting both available bandwidth and energy consumption. Finally, let us remind that TCP-WELCOME keeps its default congestion control algorithm as in TCP New-Reno, in case network congestion is foreseen to be the loss cause over the links.

c) RTT Estimation Algorithm: Since our proposed solution is based on RTT samples' evolution history, it is important to be sure that the RTT samples over the connection are accurately measured. There are many proposed algorithms in the literature describing different mechanisms for measuring RTT samples [23]. Among them: (i) Measuring from the first transmission, (ii) Measuring from the most recent transmission, (iii) Ignoring round-trip times for packets that have been retransmitted, and (iv) Karn's algorithm.

Karn's algorithm accepts only good samples and uses the retransmission back-off strategy to ensure that good samples will eventually be available even if round-trip times increase dramatically [23]. The main idea of Karn's algorithm is to use *RTO* in order to obtain accurate *RTT* measurements that are not affected by retransmission ambiguity. This algorithm does not take into consideration the acknowledgements of retransmitted data packets. Only the data packets that are acknowledged without retransmissions will be considered in *RTO* calculations. This action ensures that only accurate RTT measurements will be taken and used. Since Karn's algorithm is recognized to be the best performing option [23], we decide to implement it in our proposed solution. The *RTT* measurements have high-frequency characteristics that are desirable to detect. To be able to follow step changes in the *RTT* mean value due to increased network load, new competing traffic flows, or sudden path changes, more advanced algorithms for *RTT* estimation are needed. Currently most TCP versions implement the first-order linear filter. In mobile ad hoc networks, network parameters estimation is difficult because network observations are noisy. Current *RTT* estimator in TCP uses only one exponentially-weighted moving average (EWMA) static filter [10]. When a new observation is available, the EWMA filter produces a new estimate using linear combination of the old estimate plus the new observation, each given some weight. In traditional EWMA filters, the gain that determines the proportional weight assigned to the new observation and the old estimate is fixed. When old estimates are given more weight, the filter provides good stability; it resists noise in individual observations. However, when new observations are given more weight, the filter provides good agility; it is able to detect performance changes quickly. These filters are either able to detect true changes quickly or to mask observed noise and transients, but cannot do both at the same time. Ideally, one would like to have a filter that is agile when possible but stable when necessary, depending on current circumstances. Therefore, filters should be adaptive. In [24], an adaptive Flip Flop filter is proposed. The Flip Flop filter uses two EWMA filters, one is agile with a gain of 0.1, and the other is stable with a gain of 0.9. A controller selects between the two. The underlying principle of this controller is to employ the agile filter when possible, but to fall back to the stable filter when observations are noisy (*RTT* samples vary drastically and become noisy).

As previously discussed, our proposed LDA is based on the history of *RTT* variation over a TCP connection. If the network experiences congestion, the variation of *RTT* samples will be noticeable. Otherwise, with wireless channel errors the variation of *RTT* samples will be relatively constant. Using a Flip Flop filter, we define an upper control limit, η (*RTT_G_THRESHOLD)* excess value. Then, *RTT* samples that exceed that defined limit (*RTT_G_COUNT_THRESHOLD)* are used as an indication of a network congestion event. In [5], the authors consider that much delayed packets (whose *RTT* exceeds the control limit) as "outliers" η. Fig. 6 shows the modified pseudo code of our proposed solution using the Flip Flop filter. In our proposed algorithm, we keep η at a fixed value[3]. Furthermore, it is proved that Flip Flop filter fairness is competitive to regular TCP and its overhead is lower than that of TCP Westwood. Lowering overhead is an important issue for battery-operated devices.

During TCP connection, TCP-WELCOME keeps measuring *RTT* values and builds "*RTT history*". First, TCP-WELCOME should be able to decide either the increase of *RTT* values happens occasionally or in a consecutive manner. TCP-WELCOME considers the connection is entering a congestion situation when the number of increasing *RTT* values reaches a certain threshold (*RTT_G_COUNT_THRESHOLD*). This threshold is defined through simulations and is fixed in such a way to avoid considering congestions early and unnecessarily triggering the congestion algorithm, and in the same time, to avoid late recognition of the congestion situation over the connection. The variation of *RTT* values over the connection should also be considered to better identify when to consider that *RTT* values are noticeably increasing or it is just within the considered

[3] This value will be defined empirically through simulations to find the most appropriate parameters.

average *RTT* values over the connection. Thus, another threshold is defined and fixed (through simulations) that corresponds to a certain deviation of the calculated average *RTT* value over the connection (*RTT_G_THRESHOLD*). If the calculated deviation or is greater than the defined threshold, this is considered as a considerable increase of *RTT* values over the connection. TCP-WELCOME then triggers its counter to check if it is an imminent congestion situation or no. In order to re-initialize the TCP connection after link loss episode, we define and fix (through simulations) a new variable that represents the number of *RTT* samples required to calculate and re-adjust the new connection's parameters (*RTO_NEW_RTT_SAMPLES*).

Finally, we mention that TCP-WELCOME's average *RTT* values are calculated using an identical filter to that used in standard TCP implementation. Fig. 7 illustrates the pseudo code of the proposed LDA and LRA algorithms to enhance TCP behavior in MANETs.

```
if (loss detected by 3 duplicate acks) then
   if ( I ≤ η )
       do wireless loss recovery algorithm  // classifioed as wireless channel error loss
   else
       do congestion control algorithm // calssified as network congestion loss
   endif
elseif (loss detected by RTO) then
   if ( I ≤ η )
       do link failure loss recovery algorithm  // classified as link failure loss
   else
       do congestion control algorithm // calssified as network congestion loss
   endif
endif
```

Figure 6. pseudo code of TCP-WELCOME proposed LDA

```
if (loss detected by 3 duplicate acks) then
   if (no. of bits set in vector ≤ η)  // wireless channel error loss
        ssthresh = Bwe * RTT_min
        if (cwnd > ssthreshold)
           cwnd = threshold
        endif
   else                                  // calssified as network congestion loss
        ssthresh = Bwe * RTT_min
        cwnd = 1
   endif
elseif (loss detected by RTO) then
   if (no. of bits set in vector ≤ η)  // classified as link failure loss
        ssthresh = Bwe * RTT_min
        cwnd_n = a * cwnd_(n-1)
        if (cwnd_n > ssthreshold)
           cwnd_n = threshold
        endif
        RTO_n = (1/a) * RTO_(n-1)
   else                                  // calssified as network congestion loss
        ssthresh = Bwe * RTT_min
        cwnd = 1
   endif
endif
```

Figure 7. Pseudo Code of TCP-WELCOME LDA and LRA Algorithms

V. TCP-WELCOME VALIDATION

We implemented TCP-WELCOME in the Linux kernel 2.6 since it supports pluggable congestion avoidance modules [25]. Pluggable congestion avoidance modules facilitate the introduction of new TCP congestion control mechanisms within Linux. Then, we used "A Linux TCP implementation for NS2" patch [24] in order to import our Linux implementation code of TCP-WELCOME into the network simulator NS-2 [27]. In this way, we had the ability to test and validate our TCP-WELCOME implementation code through NS-2 simulations and realistic test-bed configuration measurements. In our simulations, all nodes communicate through identical wireless radio settings using the standard MAC 802.11 having a bandwidth of 2 Mbps and a radio propagation range of 250 meters. FTP traffic is used with a 512 bytes data packets size. The results of TCP-WELCOME through both approaches are detailed in the following sections. Table 1 defines the variables and default values used in our TCP-WELCOME implementation and validation tests. We note that these values have been modified and tested several times in the implementation and validation processes in order to get the best performance parameters and calculations variables.

A. Test-Bed TCP-WELCOME Measurements

We evaluate TCP-WELCOME's Linux kernel implementation through a realistic test-bed configuration to study its computational energy consumption. The congestion control algorithm implemented in TCP involves running a number of complex operations to calculate the values of the different timers and other performance parameters. These are CPU-intensive operations that, in turn, lead to an increase in energy consumption at the TCP Node's CPU. Thus, in order to understand the effect of TCP congestion control algorithm in the case of different data packet loss cases, we need to study the TCP computational energy cost in the event of each data packet loss situation. In this section, we show and analyze the results of TCP–WELCOME evaluation tests comparing them with four of the most common TCP variants (*TCP New-Reno, TCP SACK, TCP Vegas, and TCP Westwood*). The results are discussed according to different data packet loss situations within MANET environments (*congestion, interference, link loss, and signal loss*).

TABLE I.

TCP-WELCOME Implementation Variables and Values

Variable	Definition	Value
a ; g	Performance parameter modification factor	2/3
RTT_SAMPLE_ COUNT	RTT sample count to take into account	10
RTT_TRESHOLD	RTT value excess threshold (%)	10
RTT_G_ THRESHOLD	RTT growth threshold (%) beyond which values are considered as growing	5
RTT_G_COUNT_ THRESHOLD	Number of consecutive growth needed to trigger real congestion scenario	5
DUPACK_ THRESHOLD	Dupack threshold (usually 3)	3
RTO_NEW_RTT_ SAMPLES	We want a few RTT samples before adjusting CWND and new RTO	4

B. Test-Bed Configuration

Our test-bed configuration (Fig. 8) is composed of a laptop playing the role of the sender side while the receiver side is a Personal Computer (PC). Between the communicating nodes we implement SEDLANE (*MANET emulation that will be described in the next section*), to get the effect of a MANET environment between the sender and receiver sides. The laptop communicates with the PC over a wireless link channel. In order to calculate TCP energy consumption in the CPU unit, we measure both (i) the total energy consumption

within the laptop, (ii) the energy consumed within the wireless card for packet transmission/reception and (iii) the idle energy consumed by the laptop. The computational energy cost is the total energy consumption after subtracting both the wireless energy and the idle energy consumptions. Obviously, the measurements are taken at the TCP sender side. Synchronization is ensured between the communicating end points and the PC where the measurements were taken. In order to match the computational energy consumption to the TCP operations, we use a minimal Linux distribution in which we turn off the display, the power manager and the x-server in order to minimize the effect of any other running applications on the measured current. The reason for turning off power management as described in [28] is to better determine the current draw that corresponds to TCP code execution. Additionally, all the processes/daemons that are not necessary to TCP operations are simply removed from the Linux distribution making it minimal. By taking all these precautions, we ensured that the remaining energy consumption is due to TCP congestion control algorithms execution and timer adjustments. Energy consumption is determined by measuring the input voltage and current draw using two Agilent 34401A digital multi-meters that have a resolution of one millisecond. We do not use the laptop's battery to allow for a more steady voltage to be supplied to the device [29]. In order to directly measure the current and voltage draw of the wireless 802.11b PCMCIA card, the card was attached to a Sycard PCCextend 140A CardBus Extender [30] that in turn attaches to the PCMCIA slot in the laptop. This way, we can separately but simultaneously measure the current draw of the laptop and the current draw of the wireless 802.11b PCMCIA card[4].

C. SEDLANE: Simple Emulation of Delays and Losses for Ad-hoc Network Environment

Unlike other evaluation studies, our objective here is to apply realistic delay and loss rates to TCP connections (i.e. *delay and loss rates that correspond to MANET characteristics*). This is obtained with our MANET emulation tool called SEDLANE [31]. As depicted in Fig. 9, the main idea is to use a hybrid evaluation approach that takes benefit from simulation results in order to enhance real test-bed experiments. It allows configuring Dummynet [32] pipes (i.e. defining packet loss and delay rules) through NS-2 (*Network Simulator-2*) [27] trace files. More precisely, SEDLANE uses NS-2 TCP trace file to identify the classes of packets by gathering the packets that have similar *RTT* values. Then, SEDLANE dedicates one pipe or communication channel for each group of packets. Hence, delay values (i.e. *RTT/2* on each way) and loss rates are distributed among classes. Then, SEDLANE dynamically generates the Dummynet rules to be applied on the packets. This way, we control the different ad-hoc network parameters using simulation approach in order to make our experiments more realistic compared to those previously used. For more details on SEDLANE, the reader can refer to [31].

a) Validation Scenarios: In order to have a wide range of results that help to better understand TCP behavior in front of different data loss situations, we define different data loss situations that represent the most common loss scenarios in MANET environments. Our predefined loss scenarios are: (i) network congestion, (ii) interference, (iii) link losses and (iv) signal losses. We note that, we implement Ad-hoc On-Demand Distance Vector (AODV) [31] as routing protocol in our simulations. These scenarios are implemented in NS-2 as explained below.

1) Creating Network Congestion: In this packet-loss model, we create a congested node at the middle of a five node topology. This is done by generating three TCP data traffic flows that must pass by this intermediate node to reach the other communicating end. Fig. 10 illustrates this simulation scenario. Different levels of data congestion can be generated by controlling the number of TCP data flows crossing this node at a certain time.

2) Interference between Neighboring Nodes: Fig. 11 illustrates this scenario in which two TCP connections are on-going in parallel. The main TCP connection (*TCP data flow 1*) is disturbed by the interference generated through the second TCP connection (*TCP data flow 2*). Indeed, the node acting as forwarder for the main TCP connection is placed within the interference range of the second TCP connection sender. This situation creates interference and thus data packet drop.

3) Link Failure and Communication Route Changes: In this model we force TCP to change its communication path by shutting down the intermediate node between the communicating end points. In addition, we imply routes with different number of hops (Fig. 12). Thus, once TCP changes the communication route, the characteristics of the path between the communicating nodes change. It is obvious that the choice and the establishment delay of the new route will be dependent on the implemented ad-hoc routing protocol. Obviously, in this model, packet losses and delay changes are implied through both link loss and the characteristics of the recovered route.

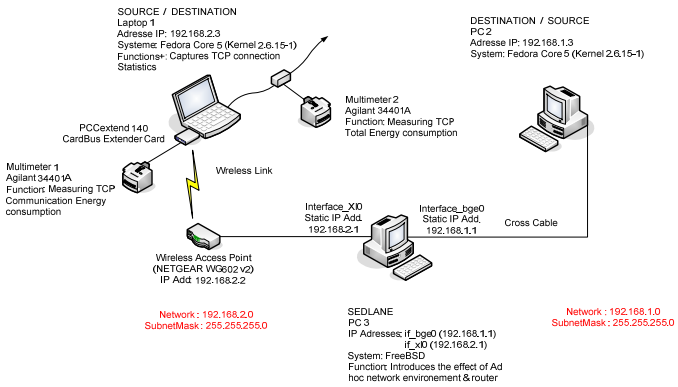

Figure 8. TCP Energy Consumption Measurements Test-bed

[4] Sycard PCCextend 140 CardBus extender card is a debug tool for development and test of PC cards and hosts.

Figure 9. The principle of SEDLANE operation

4) Wireless Signal Loss: This scenario illustrates the instability of wireless signals. The communicating nodes loose the connection due to signal loss then they resume the communication when the signal comes back. As shown in Fig. 13, Signal losses are generated by moving an intermediate node out of the radio range of its connection neighbor for a while and then moving it back.

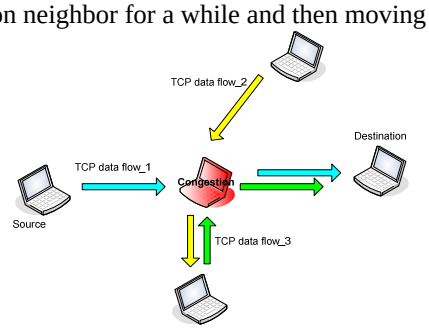

Figure 10. Network Congestion Scenario

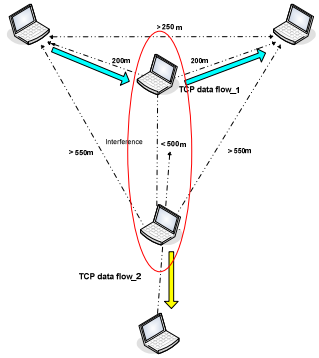

Figure 11. Interference Scenario

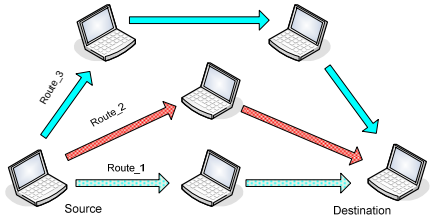

Figure 12. Link Failure and communication route changes

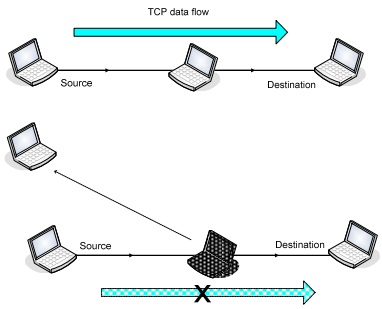

Figure 13. Wireless Signal loss scenario

Using simple network scenarios that define precise and deterministic data loss situations is done explicitly in order to study the exact reaction of TCP-WELCOME faced with each data loss situation separately. In fact, we chose simple scenarios rather than having an "all-in-one" scenario that may complicate the explanations process.

b) *Test-Bed Results:* In this section, we analyze the results of TCP–WELCOME evaluation tests. The results are discussed according to the different data packet loss situations presented above and compared to other existing TCP variants' results.

1) Network Congestion Scenario: We can see from Fig. 14 that TCP-WELCOME computational energy cost is slightly higher than that of TCP New-Reno in the event of network congestion. This is due to the fact that TCP-WELCOME verifies the cause of data packet losses (LDA) before triggering its data loss recovery action (LRA). While in TCP New-Reno, it stops data transmission without searching the cause behind data packet losses. This loss classification process in TCP-WELCOME implies more CPU calculations which lead to more computational energy cost. Also, we notice from the same figure that TCP-WELCOME still outperforms both TCP SACK and TCP Vegas in terms of computational energy cost. Indeed, these two variants use more complex algorithms without necessarily getting better results in terms of throughput. Similarly, recall that TCP-WELCOME sends more data than both TCP New-Reno and TCP Westwood.

2) Interference Scenario: Fig. 15 shows that TCP-WELCOME has almost the same performance in terms of computational energy cost as TCP Westwood. In the case of data interference loss event, both of them have the ability to correctly classify data loss cause as due to wireless link problems. While, other TCP variants will misinterpret data packet loss and consider it as if it was due to congestion. Also, the high computational energy cost of TCP Vegas is due to its high computational processing cost at the reception of each ACK.

3) Link Failure and Signal Loss Scenarios: For both Link failure and Signal loss situations, Fig. 16 and Fig. 17, we notice the high computational energy cost of TCP-WELCOME compared to other TCP variants. Actually, as none of the other variants has the ability to classify and recognize the data packet loss cause over the connection, they all react by stopping data transmission and enters the slow-start phase. On the other hand, TCP-WELCOME classifies the data packet loss cause and then reacts by calculating and adapting its performance parameters (*RTO*, and *CWND*), which leads to better performance over the TCP connection.

c) Simulation Results: NS-2 energy model does not include the node's TCP computational energy cost. It applies only the communication energy cost. That is why we thought of incorporating the results obtained through the earlier described test-bed measurements into NS-2. By including the computational energy cost of TCP's algorithms (*Slow-Start, Fast Retransmit/ Fast Recovery, and Congestion Avoidance*), we can get the total energy consumption at network nodes using NS-2 simulations.

We have integrated the obtained results into NS-2's energy consumption module and modified the code in

such a way that each time TCP enters into a specified TCP function or algorithm it measures the time passed in this function and then calculates the computational energy cost to subtract it from the total available energy for each node. This is done with all TCP algorithms.

This section provides the global TCP-WELCOME validation results through our modified version of NS-2 that incorporates TCP computational energy cost. We compare TCP-WELCOME's performance with other TCP variants (*New-Reno, SACK, Vegas, and Westwood*) in terms of both total energy cost and average throughput.

1) Network Congestion Scenario: The results show that, in network congestion, TCP-WELCOME has almost the same performance compared to the other variants, in terms of total energy consumption (Fig. 18). This is expected as TCP-WELCOME reacts to congestion similar to TCP New-Reno. Regarding the average throughput, Fig. 19 shows that TCP-WELCOME has also a comparable performance with most studied variants (*TCP New-Reno, TCP SACK, and TCP Westwood*). This result confirms the ability of TCP-WELCOME to correctly classify as well as to recover from network congestion induced-losses.

2) Interference Scenario: Fig. 20 and Fig. 21, show clearly that in front of interference, TCP-WELCOME outperforms all the other variants in terms of both average throughput and total energy consumption. The ability of TCP-WELCOME to classify the cause of packet loss, as wireless signal related, and consequently not decreasing its *CWND*, as most variants do, improves its performance. We notice also that TCP-WELCOME outperforms TCP Westwood, which was developed for wireless networks, in both terms of throughput and energy consumption. The ability of TCP-WELCOME to classify correctly the wireless-related losses and not decreasing its *CWND* or modify it in this case allows it to outperform TCP Westwood.

3) Link Failure Scenario: In MANETs, it is obvious that the communication paths between the communicating end points can break (due to mobility or nodes' batteries depletion). Unlike other TCP variants, TCP-WELCOME takes this situation into account. Fig. 22 and Fig. 23, show that its average throughput and its energy consumption are improved significantly compared to those of other TCP variants. The ability of TCP-WELCOME to detect that the packet losses are due to link failure and to react appropriately leads to a much better performance compared to all other TCP variants which react assuming that losses are due to congestions and decrease data transmission rate to minimum, thus, leading to low throughput. In TCP-WELCOME, adjusting data transmission rate according to the new discovered route characteristics allows maximizing the average throughput which also helps efficiently conserving node's energy.

4) Signal Loss Scenario: Losing the radio signal is another reason to get disconnected from the other communicating end. In link loss, the sender and the receiver would search for another route to complete the session. While in signal loss, we consider that such alternate route is not available. After signal loss recovery, most TCP variants' will start the communication session again, starting from the *Slow-Start* phase. This will be the case, each time the communicating nodes get disconnected due to signal loss. TCP-WELCOME, however, outperforms them all in both terms of total energy cost (Fig. 24) and average throughput (Fig. 25). TCP-WELCOME does not decrease its *CWND* after data packet loss due to an identified signal loss (as in other TCP variants) leading to the observed throughput gain and to an optimum usage of wireless channel bandwidth.

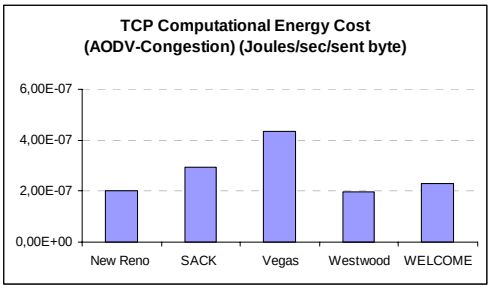

Figure 14. TCP Computational Energy Consumption

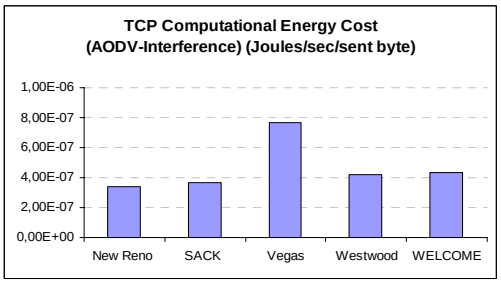

Figure 15. TCP Computational Energy Consumption

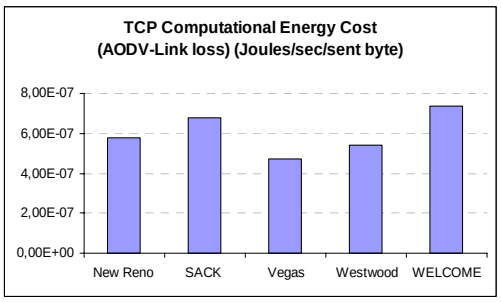

Figure 16. TCP Computational Energy Cost

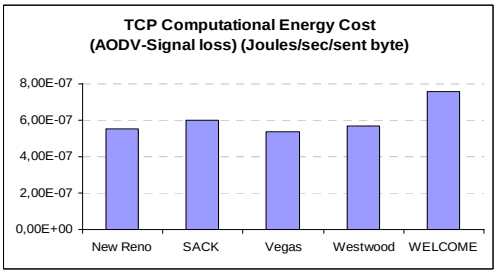

Figure 17. TCP Computational Energy Cost

VI. Conclusion

TCP suffers from drastic performance degradation when deployed within MANETs. This is due to the fact that TCP cannot differentiate between different data

packet loss situations over the connection. Misinterpreting the data packet loss cause and reacting as if it is due to congestion leads to waste of network and nodes resources (such as bandwidth and energy consumption). Hence, the ability of TCP to classify correctly the packet loss cause over the connection helps to improve its performance within MANETs. In this paper we introduced TCP-WELCOME, a new TCP variant that is suitable for MANET environments. Unlike other TCP variants, it uses a Loss Differentiation Algorithm (LDA) that is able to identify accurately the three common data packet loss causes within such network: network congestion, wireless channel related errors, and link failures. This is made by coupling delay and loss information. Moreover, TCP-WELCOME adopts a new Loss Recovery Algorithm (LRA) that reacts efficiently to each identified data packet loss cause with the most appropriate action. In order to show the performance improvement of TCP-WELCOME we implemented it into both Linux Kernel and the Network Simulator version-2 (NS-2). We compared its performances to different TCP variants under different data packet loss scenarios (congestion, interference, link failure, and signal loss). This comparative study showed that both TCP average throughput and total energy consumption have been significantly improved. We also showed that TCP-WELCOME outperforms other TCP variants in most cases thanks to its ability to identify correctly the data packet loss cause through its LDA and to take the most appropriate action to recover from data losses thanks to its LRA.

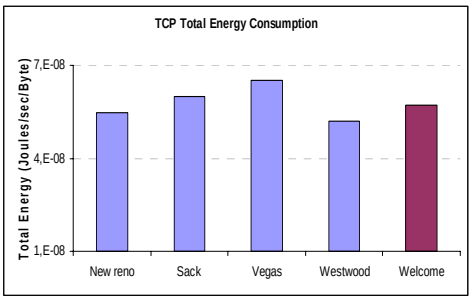

Figure 18. TCP Total Energy Consumption

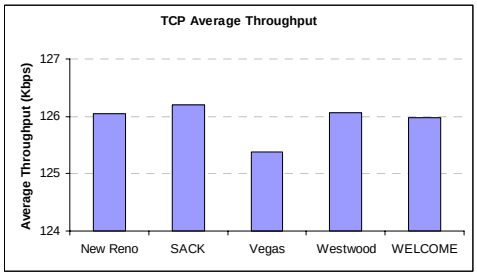

Figure 19. TCP Average Throughput

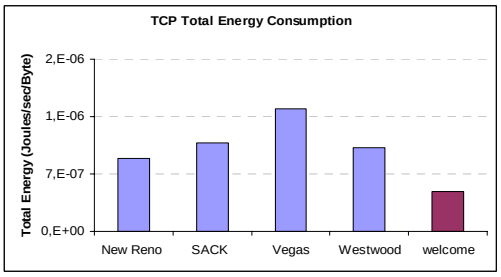

Figure 20. TCP Total Energy Consumption

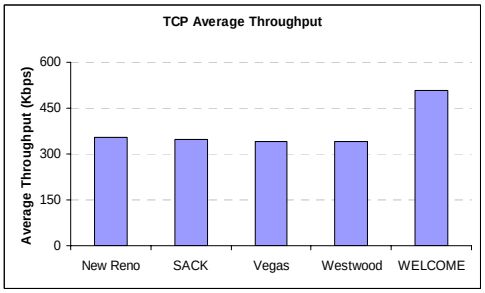

Figure 21. TCP Average Throughput

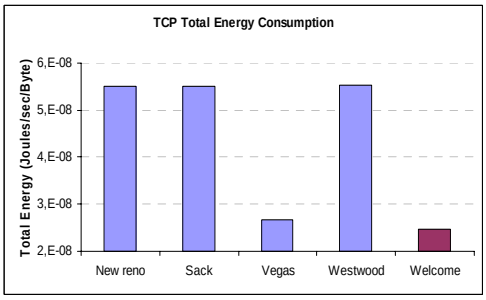

Figure 22. TCP Total Energy Consumption

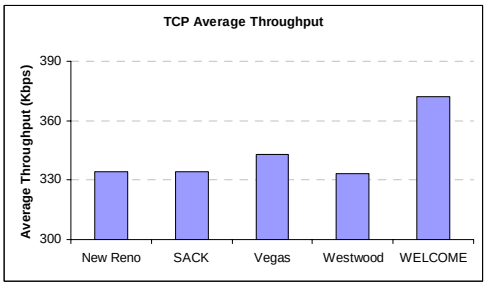

Figure 23. TCP Average Throughput

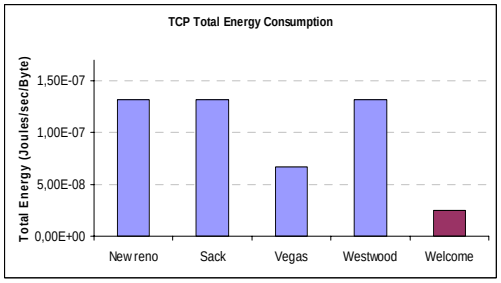

Figure 24. TCP Total Energy Consumption

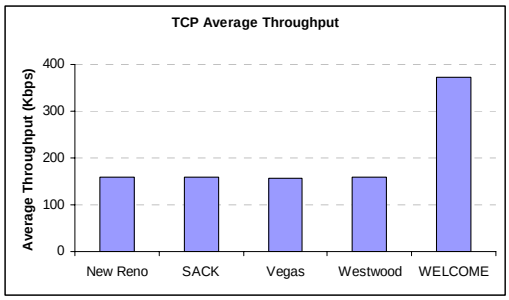

Figure 25. TCP Average Throughput

REFERENCES

[1] A. Seddik-Ghaleb, Y. M. Ghamri Doudane, and S.-M. Senouci, "TCP WELCOME: TCP variant for Wireless Environment, Link losses, and COngestion packet loss ModEls", In the 1st International Conference on COMmunication Systems and NETworkS (COMSNETS), COMSNETS'09 , January, 2009.

[2] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," IEEE Journal on Selected Areas in Communications, vol. 10, no. 7, July 2001.

[3] Ola Westin, "TCP Performance in Wireless Mobile Multi-hop Ad Hoc Networks," SICS Technical Report T2003:24, Swedish Institute of Computer Science ISSN 1100-3154, 2003.

[4] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario-based Performance Analysis of Routing Protocols for Mobile Adhoc networks," In Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking, p. 195–206, August 1999.

[5] D. Barman and I. Matta, "Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks," Boston University, Technical Report, 2002.

[6] L. A. Grieco and S. Mascolo, "TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks," In Seventh International Workshop on Protocols For High-Speed Networks (PfHSN'2002), April 2002.

[7] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," In 7th Annual International Conference on Mobile Computing and Networking, ICMCN'01, July 2001.

[8] R. Wang, M. Valla, M. Y. Sanadidi, B. K. F. Ng, and M. Gerla, "Efficiency/Friendliness Tradeoffs in TCP Westwood," In ISCC 2002: Seventh IEEE Symposium on Computers and Communications, July 2002.

[9] S. Floyd, T. Henderson, Gurtov, and A., "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782, IETF, April 2004.

[10] V. Jacobson, "Congestion Avoidance and Control", In ACM SIGCOMM Conference Stanford, CA, pp. 314-329, August 1988.

[11] A. Seddik-Ghaleb, Y. M. Ghamri Doudane, and S.-M. Senouci, "A Performance Study of TCP variants in terms of Energy Consumption and Average Goodput within a Static Ad Hoc Environment," July 2006.

[12] L. S. Brakmo, S. W. O'Malley, and Larry L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," ACM SIGCOMM'94, pp. 24-35, August 1994.

[13] M. Mancuso, "A Novel Scheme of Loss Differentiation and Adaptive Segmentation to Enhance TCP Performance over Wireless Networks," Politecnico di Milano, Dept. of Electronics and Information.

[14] G. Xylomenos, G.C. Polyzos, Mahonen P., and M. Saaranen, "TCP Performance Issues over Wireless Links," IEEE Communications Magazine, vol. 39, no. 4, pp. 52-58, April 2001.

[15] L. Stéphane, Y. Ghamri Doudane, and G. Pujolle, "Cross-Layer Loss Differentiation Algorithms to improve TCP Performances in WLANs," in the 11th IFIP International Conference on Personal Wireless Communications (PWC'06), September 2006.

[16] Y. Tobe, H. Aida, Y. X Aida, and H. Tokuda, "Detection of Congestion Signals from Relative One-Way Delay," IPSJ Journal, vol. 42, no. 12, 2001.

[17] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-End Differentiation of Congestion and Wireless Losses," In MMCN2002: SPIE Multimedia Computing and Networking, vol. 4673, pp. 1-15, January 2002.

[18] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," ACM SIGCOMM, vol. 29, no. 4, pp. 263-274, October 1999.

[19] J. Nagle, "Congestion Control in IP/TCP Internetworks," RFC 896, IETF, January 1984.

[20] D. Dykeman, M. Kaiserswerth, B.W. Meister, H. Rudin, and R. Williamson W. Doeringer, "A Survey of Light-Weight Transport Protocols for High-Speed Networks," vol. 38, no. 11, p. 2025–2039, November 1990.

[21] W. Stallings "High-Speed Network and Internets; Performance and Quality of Service", 2nd Edition, Prentice Hall, 2002, pp.148-156.

[22] A. Capone, F. Martignon "Bandwidth Estimates in the TCP Congestion Control Scheme", Proceedings of Tyrrhenian IWDC 2001.

[23] P. Karn and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," In Proceedings of ACM SIGCOMM '87, August 1987.

[24] M . Kim and B. Noble, "Mobile Network Estimation," In Mobicom 2001: The Seventh Annual International Conference on Mobile Computing and Networking, July 2001.

[25]Pluggable congestion avoidance modules. http://lwn.net/Articles/128681/.

[26]A Linux TCP implementation for NS2 Linux. http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/.

[27] NS-2. http://www.isi.edu/nsnam/ns/.

[28] B. Wang and S. Singh, "Computational energy cost of TCP," In Proceedings of IEEE INFOCOM'04, March 2004.

[29] P. Gauthier, D. Harada, and M. Stemm, "Reducing power consumption for the next generation of pdas: It's in the network interface," In Proceedings of MoMuC'96, Septembre 1996.

[30] Sycard technologies. (1996, July) Sycard technologies, pccextend 140 cardbus extender. [http://www.sycard.com].

[31] A. Seddik-Ghaleb, Y. M Ghamri Doudane, and S.-M. Senouci, "Emulating End-to-End Losses and Delays for Ad Hoc Networks," In Proceedings of IEEE International Conference on Communications, ICC'07, June 2007.

[32] Dummynet. http://info.iet.unipi/it/luigi/ip_dummynet/.

[33] C. E. Perkins and E. M. Royer, "Ad-hoc On-Demand Distance Vector Routing," In proceedings of 2nd IEEE Wksp. Mobile Comp. Sys. And Apps, WMCSA'99, February 1999.