A mobile ad-hoc network is an autonomous system of mobile nodes establishing a network in the absence of any fixed infrastructure. Mobile ad-hoc network due to potentially high mobility have provided new challenges by introducing special consideration differentiating from the unique characteristics of the wireless medium and the dynamic nature of the network topology. It is due to unique art of an ad-hoc network formation that routing has attained a prime focus. Effort has been undergoing to transform TCP so that it could support routing function in an ad-hoc network. In the existing literature a good amount of work has reported focusing TCP functioning within ad-hoc network. However, less attention has paid to analyze these schemes. This work presents a study of previously proposed TCB based routing schemes for mobile ad-hoc network and could be utilized to develop effective routing the solutions for mobile ad-hoc network.

I. INTRODUCTION

Transmission control protocol (TCP) is the most reliable transport layer protocol for the Internet [1]. TCP is responsible end-to-end connection, congestion control, flow control, in
order delivery of packets and reliable transportation of data packets [3, 4]. The contribution of this work is to evaluate previously proposed TCP based solution for mobile ad-hoc network. Rest of this paper has been organized as follows. In section 2 well known TCP based schemes are evaluated and conclusion and future work is presented in section 3.

II. Analysis of TCP based schemes

A Mobile Ad-hoc Network (MANET) is a collection of mobile nodes connected together in the absence of any fixed infrastructure. Nodes in ad hoc networks work both as hosts and routers forwarding data packets for other nodes in the network. This process may involve multiple intermediate nodes, and it may produce the establishing of a multihop connection (multi-hop ad hoc network) between sender and receiver. These networks are appropriate for scenarios where wired networks are not possible such as in a disaster recovery, battlefield, short lived networks as in conference spots, etc. In the last few years MANET are emerged as a flexible and low-cost extension of wired infrastructure networks. MANETs inherit the traditional problems of wireless communication and wireless networking, like high bit error rate, high sensitivity of wireless channel from outside signals, the possibility of path asymmetry, and so on. In addition, the multihop nature of connections, the lack of a fixed infrastructure, and nodes mobility add new problems, such as network partitions, route failures, and the hidden terminal. These new problems pose a number of design constraints that are specific to ad-hoc networking.

2.1. Transmission Control Protocol

Transmission Control Protocol (TCP) is the standard for reliable connection-oriented transport protocols, and is normally used over IP (Internet Protocol) to provide end-to-end reliable communications to Internet applications ([4]). TCP provides a reliable, connection-oriented, and full duplex type of service. In addition, TCP implements both flow control and congestion control mechanisms. The former prevents the TCP receiver's buffer from being overflowed. The second is an end-to-end congestion control mechanism that prevents process to inject into the network an excessive traffic load. Congestion control is concerned with the traffic inside the network. Its purpose is to prevent collapse inside the network when the traffic source (sender) is faster than the network in forwarding data. In a network with shared resources, where multiple senders compete for link bandwidth, it is necessary to adjust the data rate used by each sender in order not to overload the network. Packets that arrive at a router and cannot be forwarded are dropped, consequently an excessive amount of packets arriving at a network bottleneck leads to many packet drops.

These dropped packets might already have travelled a long way in the network and
thus consumed significant resources. Additionally, the lost packets often trigger retransmission, which means that even more packets are sent into the network. Thus network congestion can severely deteriorate network throughput. If no appropriate congestion control is performed this can lead to a congestion collapse of the network, where almost no data is successfully delivered.

2.2. TCP based routing for Mobile Ad-hoc Network

In the light of the conducted research it can be concluded that TCP based routing is an interesting and growing topic with in Ad-hoc network. A good number of solutions concerning TCP in MANET have been reported. However these solutions lack with certain weaknesses, thus require some effective mechanism to support routing operation. In this section an overview of the existing schemes are presented.

2.2.1. TCP-Feedback

TCP-F uses a feedback based approach to avoid congestion in mobile ad-hoc network [4]. It requires the support of a reliable link layer and a routing protocol that can provide feedback to the TCP sender about the path breaks. The routing protocol is expected to repair the broken path within a reasonable time period. When an intermediate node detects a path break, it originates a route failure notification (RFN) packet and sends it to the sender of a TCP session. The intermediate node that originates the RFN packet is called the failure point (FP). Every intermediate node that forwards the RFN packet understand the route failure, updates its routing table accordingly, and avoids forwarding any more packets on that route. If any of the intermediate nodes that receive RFN has an alternate route to the same destination, then it rejects the RFN packet and uses the alternate path for forwarding further data packets.

When a TCP sender receives an RFN packet, it goes into a snooze state. In the snooze state, a sender stops sending any more packets to the destination, cancels all the timers, freezes its congestion window, freezes the retransmission timer, and sets up a route failure timer. This route failure timer is dependent on the routing protocol and network size. When the break path rejoins or another path is detected then a route re-establishment notification (RRN) is send to the sender and the sender changes from the snooze state to the connected state.

In TCP-F the RRN packet is generated when the intermediate node detects re-establishment of broken path and it depends on information from routing protocol. TCP-F has an additional state compared to the traditional TCP state machine, and hence its implementation requires modifications to the existing TCP libraries. Another disadvantage of TCP-F is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP-F.
2.2.2. TCP with Explicit Link Failure Notification (TCP-ELFN)

TCP-ELFN uses explicit link failure notification for improving TCP performance in mobile ad-hoc network [6]. This is similar to TCP-F, except for the handling of explicit link failure notification (ELFN) and the use of TCP probe packets for detecting the route reestablishment. The ELFN is originated by the node detecting a path break upon detection of a link failure to the TCP sender. There are different ways in which the ELFN message can be implemented e.g. by sending an ICMP destination unreachable message to the sender. Once the TCP sender receives the ELFN packet, it disables its retransmission timers and enters into a standby state. In this state, it periodically originates probe packets to see if a new route is re-established. Upon reception of an ACK by the TCP receiver for the probe packets, it leaves the standby state, restores the retransmission timers, and continues to function as normal.

In TCP-ELFN when the network is temporarily partitioned, the path failure may last longer; this can lead to the origination of periodic probe packets consuming bandwidth and power. Another disadvantage is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP receiver.

2.2.3. Ad-hoc Transmission Control Protocol (ATCP)

ATCP is implemented as a thin layer residing between the IP and TCP protocols and doesn’t need changes in the existing TCP protocol [5]. The ATCP layer essentially makes use of the explicit congestion notification (ECN) for maintenance of the states. This layer is active only at the TCP sender. The major function of the ATCP layer is to monitor the packets sent and received by the TCP sender, the state of the TCP sender, and the state of the network. There are four states in the ATCP normal, CONGESTED, LOSS, and DISCONN.

When a TCP connection is established, the ATCP sender is in NORMAL state. In this state, ATCP don’t interfere with the operation of TCP and it remains invisible. When packets are lost or arrived out-of-order at the destination, it generates duplicate ACKs. ATCP sender counts the number of duplicate ACKs received and if it reaches three, instead of forwarding the duplicate ACKs to TCP, it puts TCP in persist state and ATCP in the LOSS state. In the LOSS state, ATCP retransmits the unacknowledged segments from the TCP buffer. When a new ACK comes from the TCP receiver, it is forwarded to TCP and the TCP sender is removed from persist state and then the ATCP sender changes to the NORMAL state.

When the network gets congested, the ECN flag is set in the data and the ACK packets. When the ATCP sender receives this ECN message in the normal state, it changes to the CONGESTED state and just remains
invisible, permitting TCP to invoke normal congestion control mechanism. When a route failure or network partition occurs in the network, ATCP expects the network layer to detect these and inform the ATCP sender through an ICMP destination unreachable message. Upon reception of the destination unreachable message, ATCP puts the TCP sender into the persist state and enters into the DISCONN state. It remains in the DISCONN state until it is connected and receives any data or duplicate ACKs. The connected status of the path can be detected by the acknowledgments for the periodic probe packets generated by the TCP sender. When ATCP puts TCP into the persist state, it sets the congestion window to one segment in order to make TCP probe for the new congestion window when the new route is available.

ATCP depend on the network layer protocol to detect the route changes and partitions, which not all routing protocols may implement. Addition of a thin ATCP layer to the TCP/IP protocol stack requires changes in the interface functions.

2.2.4. Split_TCP

Split-TCP splits transport layer objectives into congestion control and end-to-end reliability [3]. Split-TCP splits a long TCP connection into a set of short concatenated TCP connections called segments or zones, with a number of selected intermediate nodes known as proxy nodes. A proxy node receives the TCP packets, reads its contents, stores it in its local buffer, and sends an acknowledgment to the source (or the previous proxy) called local acknowledgment (LACK).

LACK does not guarantee end-to-end delivery. The responsibility of further delivery of packets is assigned to the proxy node. A proxy node clears a buffered packet once it receives LACK from the immediate successor proxy node for that packet. The source node clears the buffered packets only after receiving the end-to-end acknowledgment for those packets. Transmission control window at the TCP sender is also split into two windows, i.e. the congestion window and the end-to-end window. The congestion window changes according to the rate of arrival of LACKs from the next proxy node and the end-to-end window is updated based on the arrival of end-to-end ACKs.

Split-TCP requires modifications to TCP protocol. The end-to-end connection handling of traditional TCP is violated. The failure of proxy nodes or frequent path breaks, affects the performance of split-TCP.

2.2.5. Cross-Layer Approach

Cross-layer design is the interaction among the layers in the protocol stack [2]. For compatibility with the Internet, existing standard protocol stacks would be deployed in the new networks and mobile devices. However, these protocol stacks which are architected and implemented in a layered manner do-not function efficiently in mobile wireless environments. The system performance of future networks will be
enhanced by cross-layer design between PHY, MAC and higher layer protocols [1]. Following are some of the key information available at different layers that can be exchanged among each other for cross-layer design.

2.2.6. Slow Start and Congestion Avoidance

A TCP sender must use the slow start and congestion avoidance algorithms to control the amount of outstanding data injected into the network. To implement these algorithms, two state variables are added to the TCP per-connection state. The congestion window is a sender limitation on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK). The receiver’s advertised window is a receiver limitation on the amount of outstanding data.

2.2.7. Fast Retransmit and Fast Recovery

When the TCP receiver receives an out-of-order segment, it sends an immediate duplicate acknowledgment to the TCP sender. Duplicate acknowledgment happens when the TCP receiver receives an out-of-order segment and since it did not receive the segment(s) before this out-of-order segment, it cannot acknowledge the reception of this segment. Keeping in mind this fact, the TCP receiver responds with an ACK that has the sequence number of the expected packet, which is the same ACK it used to acknowledge the last in-order segment it received. This duplicate acknowledgment informs the TCP sender that the TCP receiver received an out of order packet and the sequence number of the expected packet. From the TCP sender’s perspective duplicate acknowledgment can be caused by the following:

- Dropped segments.
- The re-ordering of data segments by the network.
- Replication of ACK or data segments by the network.

2.2.8. Network Feedback Approaches

In these approaches, network implements a monitoring mechanism that generates a notification message when it detects an abnormal event so that TCP may react. TCP-F is proposed to overcome the TCP false reaction towards route failures in MANET. As soon as the network layer at any node detects the disruption of a route, it explicitly sends a Route Failure Notification packet to the source. Consequently, the TCP sender stops sending packets and freeze all its variables (such as timers and congestion window size). When one of the intermediate nodes learns about a new route to the destination, it sends a Route Re-establishment Notification packet to the source. The TCP sender leaves the snooze state, restarts the timers from their frozen values and resumes the transmission based on the stored sender window and timeout values. Similarly, the approach is uses an Explicit Link failure Notification to inform the TCP sender about the route failure. The only difference from TCP-F is that the sender in the snooze state periodically probes the network and when an
ACK is received, it considers it as an indication of route reestablishment.

TCP-RC recomputed the congestion window size and the slow start threshold for the TCP connection after the route is reconstructed instead of using the frozen values. An obvious limitation of this approach is that these techniques need to be deployed at every node. ATCP, proposed in [6], deals with the problems of high BER, route failures, network partitioning and multipath routing. A thin layer called ATCP is inserted between TCP and IP layers. The ATCP layer monitors TCP state and the state of the network (based on ECN and ICMP message) and takes appropriate action. The ATCP’s four possible states are: Normal, Congested, Loss and disconnected.

When ATCP sees that three duplicate ACKs have been received, it considers it as channel loss and only transmits the unacknowledged segments. Congestion is detected by ECN message. In case of temporary network partitioning, the ATCP receives an ICMP “Destination Unreachable” message. Hence, it puts the TCP sender in the persist state, sets TCP’s congestion window (cwnd) into one and enters itself in the disconnected state. ATCP periodically generates probe packets until it start receives their ACKs. This removes TCP from persist mode and moves ATCP back into normal state.

TCP-BuS (TCP Buffering capability and Sequence information) is another approach used to detect route failures. When a node detects a route failure, it sends an Explicit Route Disconnection Notification to the source containing the sequence number of the TCP segment pending in the head of the node’s transmit queue. All the intermediate nodes will buffer the packets in their queues. When a route is discovered, the receiver sends to the sender the last sequence number it has successfully received. The sender only transmits the lost packets and the intermediate nodes starts sending the buffered packets.

A new approach called Split TCP to improve the performance of TCP in terms of fairness and throughput. This approach depends on splitting long TCP connections into shorter localized segments. The interfacing node between two localized segments is called proxy. The proxy intercepts TCP packets, buffers them and acknowledges their receipt to the source (or previous proxy) by sending a local acknowledgment (LACK). Upon the receipt of a LACK from the next proxy (or the final destination), a proxy will purge the packet from its buffer. The source keeps transmitting according to the rate of arrival of LACKs from the next proxy, but purges a packet from its buffer only upon receipt of an end-to-end ACK for that packet from the destination. This keeps the end-to-end reliability of TCP.

2.2.9. End-to-End Approaches

End-to-end approaches require no network support. The end nodes (sender or receiver) can detect the network state by measuring appropriate traffic parameters. For example, high volume of out of order delivery
signifies route change. A heuristic is employed to distinguish between route failures and congestion without relying on feedback from other network nodes. When timeouts occur consecutively, this is taken to be evidence of a route loss. The unacknowledged packet is retransmitted again but the RTO remains fixed until the route is reestablished and the retransmitted packet is acknowledged.

TCP-DOOR (Detection of Out-Of-Order and Response) is another pure end-to-end approach to improve TCP performance by detecting and responding to out-of-order packet delivery events, which are interpreted as an indication of route failure. The non-decreasing property of ACK sequence numbers makes it simple for the sender to detect out-of-order delivery of non-duplicate ACK packets. To detect out-of-order delivery of duplicate ACK packets, they use one-byte TCP option which is incremented with each duplicate ACK packet.

Comparing the two approaches, we find that end-to-end approaches are easier to implement and provide more flexibility, while feedback approaches are more accurate as the information is coming directly from the network. Furthermore, it is clear that each approach deals only with one or subset of the factors causing the bad performance of TCP in MANETs. However, most commonly, these solutions deal with route failures. Actually, this is reasonable because in such a dynamic environment the frequency of route failures is very high due to node mobility. We also find that most of the presented approaches take reactive actions. In these approaches TCP takes different actions rather than invoking congestion control when a non-congestion loss occurs. Some approaches are preventive (e.g., Split TCP). The target of this kind of approaches is to reduce the probability of other losses that may lead to false notification and unnecessary congestion control reaction.

2.2.10. TCP Variants

This section presents the main TCP variants that have been investigated in the literature. Each variant has its own features tailored to a specific problem faced by TCP congestion control, and in most cases each new variant represents an evolution of the previous one. Slight refinements to these implementations have been described in the RFCs 2581, 2582, and 3782, but the general concepts remain unchanged.

2.2.10a. TCP Tahoe

Tahoe represents the basic TCP version that was specified by Jacobson. It was the first TCP designed to solve the congestion collapse affecting the Internet. Modern TCP implementations still use most of the mechanisms developed for Tahoe, as it will be shown below. In addition to the retransmit timeout mechanism, which was already implemented in early TCP-like transport protocols, TCP Tahoe counts on the three key mechanisms: Fast Retransmit, Slow Start, and Congestion Avoidance. Although Tahoe solved the congestion collapse problem, it rapidly
proved to be too conservative by always resetting its cwnd to one upon a lost packet.

2.2.10b. TCP Reno

TCP Reno conserved the three essential mechanisms of the basic TCP Tahoe, namely Slow Start, Congestion Avoidance and Fast Retransmit. The novelty introduced into TCP Reno is the Fast Recovery mechanism. This mechanism prevents the communication path from going empty after Fast Retransmit, thereby avoiding the need to Slow Start to refill it after a single packet loss.

Fast Recovery is generally invoked when a TCP sender receives a predefined threshold of duplicate ACKs, just after the Fast Retransmit mechanism. This threshold, usually known as tcprextmthresh, is generally set to three. Once the threshold of dup ACKs is received, the sender retransmits the packet that seems to have been dropped and reduces its congestion window (cwnd) by one half. Unlike TCP Tahoe, TCP Reno does not invoke Slow Start, but uses the additional incoming duplicate ACKs to clock out subsequent outgoing data packets.

Fast Recovery assumes that each dup ACK received represents a single packet having left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data. Specifically, during Fast Recovery the usable TCP window is defined as min (rwin, cwnd+ ndup), where rwin refers to the receiver advertised window and ndup tracks the number of duplicate ACKs received. By using the ndup variable, the sender may estimate the amount of packets in flight. After entering Fast Recovery and retransmitting a single packet, the sender effectively waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data (called a “recovery ACK”), the sender exits Fast Recovery by setting ndup to 0.

TCP Reno is optimized for the case when a single packet is dropped from a window of data. In such cases, the TCP sender can retransmit at most one dropped packet per Round-trip Time (RTT). TCP Reno is more efficient than its predecessor (Tahoe) but does not work so well when more than one packet is dropped from a window of data. The problem is that TCP Reno may reduce the cwnd multiple times for recovering the lost packets, leading the connection to experience poor performance.

2.2.10c. TCP NewReno

NewReno improves the Reno implementation with regard to the Fast Recovery mechanism. The objective of TCP NewReno is to prevent a TCP sender from reducing its congestion window multiple times in case several packets are dropped from a single window of data. NewReno can also avoid retransmission by timeout in scenarios where the involved congestion window is small preventing enough ACK packets from reaching the sender. In TCP Reno, when the sender receives a partial ACK packet it exits Fast
Recovery. The term partial ACKs refers to ACK packets that acknowledges some but not all of the data packets that were outstanding when the Fast Recovery was started. Upon receipt of a partial ACK, the Reno sender brings the usable window back to the congestion window size, and so exits Fast Recovery. If there are sufficient outstanding packets, the sender may receive enough duplicate ACKs to retransmit the next lost packet (or packets) until all dropped packets are retransmitted by the Fast Recovery mechanism.

At every invocation of the Fast Recovery, cwnd is halved. If there are not enough packets outstanding due to a low window size, then the sender needs to wait for the expiration of the retransmission timer. In this case the cwnd is reset to one, inducing bandwidth wastage. Differently from Reno, the NewReno do not exit Fast Recovery when it receives partial ACKs. Instead, TCP NewReno treats partial ACKs received during Fast Recovery as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, TCP NewReno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted. TCP NewReno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated has been acknowledged. In this way, TCP NewReno avoids multiple reductions in the cwnd or unnecessary retransmit timeout with Slow Start invocation, thereby improving the end-to-end performance.

2.2.10d. TCP Sack

TCP Sack (Selective Acknowledgment) preserves the basic principles of TCP Reno. In fact, it uses the same algorithms of Reno for increasing and decreasing its congestion window. The novelty in TCP Sack lies in its behavior when multiple packets are dropped from one window of data, similarly to TCP NewReno. In Sack, the receiver uses the option fields of TCP header (Sack option) for notifying the status of data received and queued by the receiver.

The SACK option field contains a number of SACK blocks, where each SACK block reports the received and queued bytes of data that are contiguous and isolated (there are gaps in the data stream). The first block in a SACK option is required to report the most recently received segment, and the additional SACK blocks repeat the most recently reported SACK blocks. The sender keeps a data structure called scoreboard to keep track of the Sack options (blocks) received so far. In this way, the sender can infer whether there are missing packets at the receiver. If so, and if its congestion window permits, the sender retransmits the next packet from its list of missing packets.

In case there are no such packets at the receiver and the congestion window allows, the sender simply transmits a new packet. Like
TCP Reno, the Sack implementation also enters Fast Recovery upon receipt of generally three duplicate acknowledgments. Then, its sender retransmits a packet and halves the congestion window. During Fast Recovery, SACK monitors the estimated number of packets outstanding in the path (transmitted but not yet acknowledged) by maintaining a variable called “pipe”. This variable determines if the sender may send a new packet or retransmit an old one.

The sender may only transmit if pipe is smaller than the congestion window. At every transmission or retransmission, pipe is incremented by one, and it is decremented by one when the sender receives a duplicate ACK packet containing a SACK option informing it that a new data packet has been received by the receiver. The Fast Recovery terminates when the sender receives an ACK acknowledging all data that were outstanding when Fast Recovery was entered. If the sender receives a partial ACK, i.e., an ACK that acknowledges some but not all outstanding data, it does not exit Fast Recovery.

For partial ACKs, the sender reduces pipe by two packets instead of one, which guarantees that a SACK sender never recovers more slowly than it would do if a Slow Start had been invoked. If it happens that a retransmitted packet is dropped, the SACK implementation reacts exactly as the Reno implementation. In such cases, the sender times out, retransmits and enters Slow Start. SACK incorporates all the advantages found in NewReno and may recover multiple lost packets in a window of data in just one single RTT. A SACK implementation requires changes at both sender and receiver, though.

2.2.10e. TCP Vegas

Differently from the above examined TCP versions, TCP Vegas is not an ACK-clocked congestion control. Specifically, TCP Vegas does not increase its congestion as a function of the number of ACKs received. Yet, while the previous TCP variants detect network congestion by lost packets, TCP Vegas does so by monitoring the changes in the RTTs associated to the packets that it has sent previously through the connection. If the observed RTTs increase, the Vegas sender infers incipient network congestion and so it reduces the congestion window (cwnd) by one. Otherwise, if the observed RTTs decrease, the sender interprets that as an indication that the network is free of congestion, and so it rises the cwnd by one. There is a RTT range in which the cwnd remains unchanged. TCP Vegas keeps the window size unchanged. This is to prevent losses inside the network. The key idea here is to use the actually available network bandwidth without causing excessive traffic within the network.

TCP Vegas has another feature in its congestion control algorithm, during Slow-Start, TCP Vegas increments its cwnd slower than TCP Tahoe. Specifically, to be able to detect and avoid congestion during Slow Start, Vegas allows exponential growth only every other RTT. In between, the congestion window
stays fixed so a valid comparison of the expected and actual rates can be made.

2.2.11. Explicit Congestion Notification (ECN)

The ECN scheme specified in RFC 3168 proposes to use network feedback to assist a TCP connection in reacting to congestion effects. By using this mechanism, TCP does not need to await a dropped packet due to buffer overflow to detect congestion and properly slow down. Rather, it is informed by the intermediate nodes (routers) when incipient congestion starts. ECN can prevent time wastage at the sender that, without ECN, always has to wait for either three duplicate acknowledgments or timeout timer expiration. The implementation of ECN requires specific flags in both IP and TCP headers. Two bits are used in each header for proper signaling among sender, routers and receiver. The active queue management (AQM) inside the routers marks packets when congestion reaches a given threshold. The receiver simply echoes back the congestion indication into the ACKs to the sender which reduces its sending rate to prevent severe congestion. ECN is appealing to be used in the Internet since it does not render any overhead regarding the current IP flows. Its drawback lies in the fact that to be effective, it requires changes to every network element.

2.2.12. Delayed Acknowledgments (DA)

When data arrives at the receiver, the protocol requires that the receiver sends back an acknowledgment for reliability reasons. The data packets are sequentially numbered so the receiver can acknowledge data by sending to the sender the sequence number of the highest data packet it has in its buffer. The acknowledgment scheme is cumulative, which means that by receiving the highest sequence number, the sender infers that all prior data were successful received. Thus a TCP receiver does not necessarily have to transmit an acknowledgment for every incoming data packet.

RFC 813 introduces a new mechanism that optimizes transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. This RFC shows that reducing the number of ACKs provides two benefits: lower processing overhead at the sender and robustness against the well-known Silly Window Syndrome (SWS). Measurements of TCP implementations, in particular on large operating systems, suggest that most of the overhead involved in a packet handling is not in the TCP or IP layer processing.

In fact, the most significant processing occurs in the scheduling of the handler that must deal with the packet at the sender. The delay ACK mechanism optimizes transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. However, if the network is facing constraints, additional mechanisms are needed to make sure that the receiver does not lead the sender to miss ACKs. Hence, RFC 813 recommends the use of a timer at the receiver to trigger ACK transmissions for data packets.
that do not arrive at the receiver in due time. This timer should be reset at every new income data packet and its duration could be either a fixed interval on the basis of the channel characteristics such as typical RTT or be adaptive to the channel conditions. Although RFC 813 establishes the foundation for the delayed ACK mechanism, it does not specify clearly the actions to be taken by the receiver under a constrained channel.

2.2.12. Node Feedback Based TCP Scheme

Node feedback TCP based mechanism introduces a new flavor of TCP for mobile ad-hoc network. It follows an intermediate approach in between some of the existing mechanisms of TCP based schemes for mobile ad-hoc network. We have addressed TCP slow start mechanism in the context of mobile ad-hoc network and introduce measures through whom TCP can differentiate between real congestion and congestion assumed by TCP due to packet lost or route failure in mobile ad-hoc network. In addition our proposed mechanisms also deal with out-of-order delivery problem of TCP in mobile ad-hoc network. It is important to mention that NFBTCP not only address TCP related issues but also provides a number of different operations to assists in the smooth running of an ad-hoc network.

III. Conclusion and Future Work

In this paper we have presented a detailed analysis of TCP based schemes of mobile ad-hoc network. The findings of this work could contribute towards the development of effective TCP based routing solutions for mobile ad-hoc network.

REFERENCES


