WTP: bandwidth-efficient and reliable cross-layer transport protocol for MANETs

Iurii Voitenko
Royal Institute of Technology (KTH), Sweden
voitenko@kth.se

Abstract—Performance and adaptability of the transport protocol are among the most critical parts of the TCP/IP protocol stack since it is directly related to the user interaction and application experience. In this paper we analyze specific problems of reliable transport protocol performance caused by rapidly changing topology in MANETs in addition to classical transport layer challenges. Then we propose Wireless Transport Protocol (WTP) - a new lightweight, bandwidth-efficient and reliable cross-layer transport protocol capable to adjust its performance to highly dynamic environment as MANETs. Experimental results show that simplifying protocol overhead, its processing and cross-layer optimization enables more precise adjustments to network and connection fluctuations, hence increasing transport protocol efficiency up to 19% in different conditions compared to standard TCP protocol (RFC793, RFC2581).

Keyword—MANET, TCP, transport protocol, cross-layer, reliable data transfer.

I. INTRODUCTION

Over the last decade much ongoing research attention is paid to the development of mobile packet radio networks without any pre-existing infrastructure - mobile ad hoc networks (MANETs). In such networks every node can be used not only for the end-user access but also serves as a router that relays packets to other nodes. It is also assumed that every node is free to move in any direction at any time which makes topology of these wireless networks highly dynamic.

The application scenario of such networks is quite broad. Thus, the MANETs are useful in rescue operations when infrastructure is damaged or destroyed, military operations at the tactical level, crowded areas (e.g. for maintenance conference participants etc.), and where no other telecommunications infrastructure present. In contrast to the hierarchically structured networks with centralized control, mobile ad hoc networks consist of identical nodes where each node has a set of hardware and software tools that allow organizing data transfer from source to destination directly. In addition to that MANETs of the future are based on TCP/IP protocol stack following the “all-IP” trend in modern telecommunications [1].

Transmission control protocol (TCP) is the most widely used protocol for reliable data transfer in today's Internet and, hence, in many research activities it is considered with some modifications to be the appropriate choice for MANETs as well. Residing directly on top of the Internet Protocol (IP) it is supposed to bring seamless integration between wireless MANETs and wired Internet [2] being at the same time directly related to the user interaction and application experience. However, initially defined in the RFC793 and RFC2581 [3, 4], TCP protocol was designed for wired networks where every packet drop is caused by the network congestion.

In this paper we investigate specific properties of TCP along with its modifications proposed for MANETs, design a new lightweight, bandwidth-efficient Wireless Transport Protocol (WTP) with cross-layer functionality and compare its performance with TCP protocol standardized in RFC793. Network simulations are based on several experiments with real hardware and compared to analytical expressions describing protocols behavior.

II. NETWORK AND PROTOCOL CHALLENGES

According to OSI model reliable data delivery and congestion control are carried out by transport layer protocols. Standard TCP treats every packet drop as congestion in the network and decreases the data rate [3, 4]. In MANETs packet drops are caused not only by congestion but primarily by changes in the network topology, medium access collisions, routing protocol properties etc., which leads to unpredicted changes in the performance of available routes. This paper discusses various reasons of deliver data inability in mobile sensor networks and proposes a new transport protocol designed to operate in highly mobile environment.

Wireless networks, not only MANETs, present many challenges, especially when real time application must be supported in terms of providing QoS guarantees. One of the biggest challenges is routing since all nodes can move. Variety of routing protocols has been developed. Some of them, such as AODV and DSR are presented in RFC documents [5, 6]. All of them are designed as independent algorithms which can operate with any transport protocol in accordance with decomposition of OSI layers. However, in the case of severe performance degradation or route failure it can take significant time to find a new route while transport layer will continue to deliver data segments on a network layer. This can lead to unacceptable quality of service on application layer.

Another challenge for MANETs is congestion control on a transport layer. Such classical TCP mechanisms as congestion avoidance, fast retransmit and recovery mechanisms do not
entirely correspond to the wireless environment which this protocol operates in, hence proper adaptation is needed. Thus, the main TCP disadvantages when used in MANETs:

- Segment loss interpreted as network congestion only;
- Decreasing congestion window (CWND) when loss is detected twice the size regardless of loss real reason;
- Detecting maximum CWND size requires rapid increase in amount of sent segments until loss is detected (TCP “saw” behavior);
- Inability to exchange cross-layer information, no access to lower layer protocols for real time network performance evaluation.

Different adjustments to TCP for operating in MANETs have been presented though. To overcome the problem of connectivity loss, some cross-layer techniques have been proposed as well [7]. This way allows collecting the information on radio links, delays and available bandwidth on each hop which can be used for some dynamic adaptation to the network changes. This circumstance is important when applications (usually multimedia) have strict requirements on delay, bandwidth and jitter. In [8, 9] the use of multipath routing protocol along with TCP has been proposed and evaluated. This is useful for decreasing the latency while rerouting because alternative route is immediately available for retransmission of the lost segment. However, all of these proposals address only certain parts of TCP logic leaving the rest of the protocol untouched (TCP Vegas, TCP Reno, Random Early Detection etc.). In addition, TCP employs byte-oriented flow control rather than packet-oriented in spite of packet nature of the MANETs. And, finally, TCP protocol overhead consists of 20 bytes which, together with 40 bytes of IPv6, considered being a lot when typical MTU in wireless ad hoc networks (for example, IEEE 802.15.4) is not more than 128 bytes.

Hence, considering mentioned above, it seems impractical to adapt currently standardized TCP protocol due to enormous amount of changes needed. This leads to the development of a completely new transport layer protocol for MANETs where all peculiarities of these networks will be incorporated into protocol from the very beginning.

III. RELIABLE TRANSPORT PROTOCOL ARCHITECTURE

At the core of every reliable transport protocol there are following key elements: checksum, sequence number, retransmission timer, and positive or negative acknowledgments. Additionally, it must ensure that its receive buffer is uncorrupted and has no gaps or duplications. In order to achieve those properties protocol must contain sequencing and retransmission capabilities. First, consider effect of sequence numbers and acknowledgments.

A. Sequence and acknowledgement numbers

Sequence numbers and positive or negative acknowledgments (ACK) are used by every transport protocol that requires reliability. For example, according TCP specification reliability is achieved by assigning a sequence number to each octet transmitted, and requiring a positive ACK from the receiving TCP. If the ACK has not been received within a timeout interval, the data are being retransmitted [3]. This implies that either every segment must be acknowledged explicitly as shown on Fig.1 or delayed ACK concept must be applied in order to acknowledge many data segments with one ACK packet. The latter can lead to improper round-trip time (RTT) calculation and throughput degradation as receiving host intentionally defers from sending back an ACK waiting for more possible data segments to come. This is shown on Fig.2.

Different adjustments to TCP for operating in MANETs have been presented though. To overcome the problem of connectivity loss, some cross-layer techniques have been proposed as well [7]. This way allows collecting the information on radio links, delays and available bandwidth on each hop which can be used for some dynamic adaptation to the network changes. This circumstance is important when applications (usually multimedia) have strict requirements on delay, bandwidth and jitter. In [8, 9] the use of multipath routing protocol along with TCP has been proposed and evaluated. This is useful for decreasing the latency while rerouting because alternative route is immediately available for retransmission of the lost segment. However, all of these proposals address only certain parts of TCP logic leaving the rest of the protocol untouched (TCP Vegas, TCP Reno, Random Early Detection etc.). In addition, TCP employs byte-oriented flow control rather than packet-oriented in spite of packet nature of the MANETs. And, finally, TCP protocol overhead consists of 20 bytes which, together with 40 bytes of IPv6, considered being a lot when typical MTU in wireless ad hoc networks (for example, IEEE 802.15.4) is not more than 128 bytes.

Hence, considering mentioned above, it seems impractical to adapt currently standardized TCP protocol due to enormous amount of changes needed. This leads to the development of a completely new transport layer protocol for MANETs where all peculiarities of these networks will be incorporated into protocol from the very beginning.

III. RELIABLE TRANSPORT PROTOCOL ARCHITECTURE

At the core of every reliable transport protocol there are following key elements: checksum, sequence number, retransmission timer, and positive or negative acknowledgments. Additionally, it must ensure that its receive buffer is uncorrupted and has no gaps or duplications. In order to achieve those properties protocol must contain sequencing and retransmission capabilities. First, consider effect of sequence numbers and acknowledgments.

A. Sequence and acknowledgement numbers

Sequence numbers and positive or negative acknowledgments (ACK) are used by every transport protocol
example, suppose sender informs receiving host about the amount of data it is going to send (packets or bytes). Then receiving host may reserve appropriate amount of buffer space and wait for certain amount of packets to arrive before sending ACK. This enables also the use of cumulative acknowledgments that decreases total amount of ACKs in the connection improving goodput. We will investigate this effect in details when presenting WTP below.

B. Retransmission timeout

Retransmission timeout (RTO) is another key element that enables reliability at transport layer. Suppose, the sender has sent a data segment and either that segment or its ACK has been lost. However, the sender does not know whether the data or ACK were lost or simply delayed somewhere in the network. In either case, the sender will retransmit the last segment again after waiting certain amount of time introducing the possibility of data duplication at the receiver side. Hence, the calculation of the appropriate RTO is a crucial and requires a countdown timer together with supplementary statistics such as average RTT.

Thus, to ensure reliability of the connection, sender sets a retransmission timeout for every data packet it sends. Its value is based on simple statistical information such as round-trip time (RTT). According to fast retransmission mechanism employed in latest versions of TCP reception of 3 duplicate ACK packets (DupACKs) trigger immediate retransmission of lost segment. However, if sender did not send enough packets to generate 3 duplicate ACKs, then unacknowledged data are retransmitted on timeout.

To calculate appropriate retransmission timeout following equations are used [10]:

\[
SRTT = (1 - \alpha) \cdot SRTT_{OLD} + (\alpha \cdot RTT_{LAST}),
\]

where \( SRTT \) is “smoothed” round-trip time; \( 0 < \alpha < 1 \). Usually \( \alpha = 1/8 \), then (2) can be expressed as:

\[
SRTT = \frac{7}{8} SRTT_{OLD} + \frac{1}{8} RTT_{LAST}.
\]

In RFC 1122 [11], the following retransmission timeout calculation is recommended:

\[
RTO = SRTT + 2 \cdot SDEV,
\]

where \( SDEV \) is “smoothed” deviation, to calculate which the absolute deviation \( DEV \) is found:

\[
DEV = |RTT_{LAST} - SRTT_{OLD}|
\]

Then SDEV can be calculated as:

\[
SDEV = \frac{3}{4} SDEV_{OLD} + \frac{1}{4} DEV
\]

Current TCP implementations do not support calculation of RTT for each particular route used as no cross-layer information flow defined, i.e. TCP doesn’t know over which route each segment is sent. Equations (2)–(6) do not support multi-route capabilities as well since they imply the calculation of one timeout value regardless of number of routes available. We will present a way of calculation of multiple RTOS later in this paper.

C. Congestion control

Yet another important issue of transport layer is efficient congestion control and connection maintenance. Again, we address TCP protocol in this aspect as most popular even though some other reliable transport protocols exist. Many papers investigate in details TCP congestion control and therefore we will cover only MANET specific problems of congestion control in this work. In mobile ad hoc networks, resolving this issue is much more complicated due to inability to define a real cause of packet loss: network congestion, topology change, wrong checksum or any other reason. Using cross layer information, the exchange may help to define some reasons such as low signal-to-noise ratio on the radio link or wrong checksum on network layer but not all of them. Moreover, in mobile ad hoc network there is a multihop environment where the packet can be discarded at any intermediate node with no ability to predict it. Some protocols propose to use link layer ACK in order to notify about topology change. However, this may lead to unacceptable delays and, hence, lower QoS at application layer.

Ignoring the initial slow-start period when a connection begins and assuming that losses are indicated by triple duplicate ACK or timeouts, TCP congestion control consists of linear (additive) increase in congestion window (cwnd) of 1 MSS per RTT and multiplicative decrease of cwnd (halving the size) on triple duplicate ACK event or timeout. Hence, whenever new ACK arrives data sender increases congestion window by 1 MSS in slow start state, and by MSS/cwnd bytes in congestion avoidance state. TCP congestion control is also byte-oriented as cwnd operates with bytes and, hence, average TCP throughput of the connection including retransmitted data can be expressed by the following equation:

\[
S = \frac{CWND}{RTT}
\]

where, \( S \) – estimated TCP throughput, b/s; cwnd – congestion window, b; \( RTT \) – average round trip time, ms. Note, in equation (6) retransmissions and respective loss rate is assumed to be within one cwnd round-trip time.

However, as we will show below using packet-oriented stream at transport layer along with information on lost segments can successfully address congestion problems in MANETs.
IV. WIRELESS TRANSPORT PROTOCOL

As we discussed earlier, standard TCP has no means to distinguish routes it sends segments over as neither cross-layer information exchange nor its processing specified. It has also relatively big overhead compared to network MTU (128 bytes, for example) and pretty complicated internal structure with great footprint on the code size making its implementation a non-trivial task. This results in switching most of the embedded systems for wireless networking towards UDP-like protocols. However, UDP does not provide any kind of reliability which is sometimes required by applications therefore a special reliability add-on (or protocol) is required on top of it.

So, here we introduce Wireless Transport Protocol (WTP) - a new transport protocol on top of UDP that brings reliability with small overhead (only 4 extra bytes) to embedded systems in MANETs. Header format of WTP is shown on Fig.3.

![WTP header format](image)

WTP has header of 4 bytes, where SEQ_NO – 8-bit sequence number; A – acknowledgment needed bit; ACK_NO – 15-bit acknowledgement number; WINDOW (WND) – 4-bit sent or receive buffer size in MSS; FLAGS – 2-bit flags bitmap: 00 – PSH, 01 – SYN, 10 – ACK, 11 – FIN; CNT – 2-bit retransmission upcounter. It is packet-oriented protocol where sequencing and acknowledgments are based on segments of the predefined in connection establishment phase size.

A. Connection establishment phase

WTP is a connection-oriented protocol therefore hosts must establish a logical connection before sending data. Proposed protocol uses almost the same three-way handshake as TCP but identifies segment propagation routes, changes the flags and their processing during those 3 steps. WTP also takes advantage from cross-layer information exchange, specifically from network layer on routing and stability of the radio links as well as requires multipath routing to be used. Fig. 4 depicts how WTP connection is established in MANET between hosts A and G.

The initial sequencing starts always with seq_no=0, ack_no=0, A bit=0, flags=SYN and WND=receive buffer space in MSS, CNT=0. The SYN segment, sent by host A at step 1, also carries MSS_A – a maximum segment size at A (2 extra bytes appended). The destination host G at step 2 always replies to all SYN with SYN ACK where seq_no=1, ack_no=1, A bit=1, flags=SYN, WND=receive buffer size in MSS appending its own MSS_G size; CNT remains unchanged as it helps distinguish retransmissions. Note, each SYN ACK is sent back via the same route as SYN came from (the same applied to step 3) but, assuming IPv6 used on the layer below, assigns to each route a unique flow label.

Thus, upon receiving SYN ACK from remote host with unique flow label for each route, the sender A is able to estimate RTT per route and calculates a common MSS for the connection as:

\[ MSS = \min(MSS_A, MSS_G) \]  

This is needed for both parties to agree on max segment size used in this connection and, thus, reserve appropriate amount of buffer space. Finally, at step 3, sender A replies to each segment from step 2 with ACK carrying seq_no=2, ack_no=1, A bit=0, WND=MSS, flags=ACK; CNT remains unchanged, common MSS calculated with (7) and attached, flow labels mark routes how the segments must be forwarded to the destination.
example, RREQ and RREP packets if DSR routing used (steps 1, 2). Alternatively, any multipath routing protocol providing explicit packet forwarding may be used. Additionally, step 3 of the three-way handshake ensures that links on the routes used in previous steps are symmetric. Thus, after connection is established hosts have exchanged their receive buffer space available, agreed on common MSS size, determined the routes the packets will be forwarded over and initial RTT.

WTP does not support multiplexing/demultiplexing of the streams among many applications occupying the same UDP port. Hence, an ambiguity present when hosts establish another connection using the same source and destination ports. As a solution parallel connections can run by encrypting each connection with unique session key. This way each successfully decrypted packet will be linked to a particular socket, though it requires a special key agreement protocol analogous to, for example, IKE or SSL.

B. Data transfer phase

After WTP connection is established hosts can send data. WTP is a Go-Back-N (GBN) kind of protocol with selective repeat and cumulative acknowledgments but keeps sequential numbering in the stream. Therefore first data segment must be carried data sets flag PSH and must not oversize the MSS agreed. Consider typical WTP dialog between sender A and receiver G and no data loss.

On Fig.5 event \( t1 \) on the receiver side denotes reception of data segment with flag PSH and A bit set. Every data segment with flag PSH set carries also a send buffer size in WND field. So, each group of segments sent by sender explicitly informs a receiver how many segments it should expect. It means the segment with seq_no=3 is the only one in the group expecting ACK since WND=1. Thus, when this data segment reaches receiver it immediately lifts the data up to application process and sends back ACK replacing WND with its own receive buffer, t.i. amount of MSS segments it is ready to receive next. ACK segment seq_no is calculated according to (8) and is known in advance at the sender side:

\[
SEQ.NO_{ACK} = SEQ.NO_{DATA} + WND_{sender} \quad (8)
\]

Note, ACK is generated only if A bit is set, otherwise data segment is sent unreliably with no retransmission supported. This feature might be useful for resource constrained embedded systems mixing reliable and unreliable traffic such as VoIP services with SIP/RTP dialogs.

Field ACK is a bitmap where each correctly received segment sets a bit in 15-bit bitmap based on its seq_no at position according to (9):

\[
i = (SEQ.NO_{rt} + 255) - (SEQ.NO_{ls} + 255) - 1, \quad (9)
\]

where \( i \) – segment position in receive buffer; \( SEQ.NO_{ls} \) - last received sequence number; \( SEQ.NO_{rt} \) - last sent sequence number. Note, maximum amount of segments to be acknowledged by one ACK is 15 since max WND size is 15 also (4-bit field).

Event at time \( t2 \) indicates that a sender has received ACK from receiver with expected sequence number according to (8). Then, assuming it has more data to send, it extracts receive buffer size from WND and calculates amount of segments which can be sent immediately by (10):

\[
p = \min(WND_{sender}, WND_{receiver}) \quad (10)
\]

Where \( p \) – amount of segments to send in the next group; \( WND_{sender} \) – amount of data ready to send (send buffer size), MSS; \( WND_{receiver} \) – receive buffer size. When first segment reaches receiver it identifies the total number of segments in the next group and calculates next \( SEQ.NO \) for its ACK. WTP uses cumulative acknowledgments therefore after last segment reached receiver a bitmap is filled and ACK sent back immediately.

Sender expects ACK with \( SEQ.NO=9 \), thus having received such packet it starts removing sent segments from its send buffer processing ACK bitmap using following \( ack\_process \) algorithm in pseudocode below. The algorithm returns the size send buffer after all acknowledged segments have been removed. So, a sender can easily detect undelivered segments and retransmit them instantly with no timeouts or triple duplicate ACK. WTP supports half duplex data exchange. This implies a receiver may also respond to data segment with another data (flag PHS) instead of just ACK with no data. A

---

**Fig.5. WTP data transfer dialog**
sender then is able to simultaneously process data and ACK bitmap. An ACK is sent back to receiver if bit A is set, thus maintaining a reliable two-way dialog.

1: Algorithm ack_process( ack_bitmap ):

2:     i=0, j=0
3:     while i < sendbuf.size() do
4:         if ( ack_bitmap & (1 << j) == 1 ) then
5:             sendbuf.removeAt( i )
6:         else
7:             i=i+1
8:         endif
9:     j=j+1
10:    endwhile
11:    return sendbuf.size()

TCP supports full duplex data exchange since sequencing is maintained by each party individually. In practice this feature is redundant because application processes running on hosts must assemble the whole data block before issuing new data in response. For example, in SSL handshake protocol it would be reasonable to exchange client and server certificates at the same time. But this approach violates the processing workflow since protocol is unable to predict the certificate content which makes full duplex data flow irrelevant.

Another important property of WTP is inbuilt support for pipelined data flow over predefined routes. This feature is illustrated on Fig. 6.

![Fig.6. Parallel data pipelining in WTP](image)

Using cross-layer information WTP hosts can forward data and ACK segments to different, preferably independent, routes if any, which allows to more efficiently spreading the load in the network. For this each host maintains respective cross-layer table which contains information from transport and network layers such as total number of routes found, flow labels, hops, RTT, losses and WND size. Example related to network on Fig.6 is illustrated in Table 1.

<table>
<thead>
<tr>
<th>#</th>
<th>Route</th>
<th>Flow label</th>
<th>Hops</th>
<th>RTO</th>
<th>RTT</th>
<th>Loss rate</th>
<th>WND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABEG</td>
<td>1234</td>
<td>3</td>
<td>195</td>
<td>22</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>ACDFG</td>
<td>5678</td>
<td>4</td>
<td>250</td>
<td>31</td>
<td>0.0</td>
<td>3</td>
</tr>
</tbody>
</table>

This might be useful in switching data flow to another route if current loss rate or delay (RTT) exceeds the QoS limits stated in the user application.

However, as data can be lost in any network, consider more realistic scenario of WTP dialog containing packet losses. Fig.7 and Fig.8 compare WTP and TCP dialogs with one of the packets lost.

![Fig.7. WTP dialog with lost segment](image)

![Fig.8. TCP dialog with lost segment](image)
First, analyze TCP flow control shown on Fig.8. At time $t_1$ when segment with seq_no=31 reaches destination receiver immediately replies with dupACK since a gap in data stream is detected. TCP receiver at this time has no means to determine whether the segment with seq_no=21 is lost or delayed in the network, therefore it simply reacknowledges last data segment according to TCP receiver’s ACK generation policy specified in RFC1122 and RFC2581. However, event at time $t_2$ when this dupACK reaches sender triggers no retransmission because there must be three dupACK for fast retransmit. It imposes significant limitations on further behavior of the sender. It can either continue to transmit new data but in this case lower the rate or stop sending while waiting for timeout event for lost segment. TCP specification does not regulate this circumstance leaving implementation up to developers. For instance, devices with limited resources running popular uIP stack [12] in this situation decrease their sending rate and retransmit lost segment by timeout, thus, significantly reducing the throughput. In some TCP implementations such as TCP-Tahoe the segment with seq_no=31 will be also retransmitted (indicated by dotted line) by timeout utilizing available bandwidth very inefficiently [13, 14].

WTP flow control shown at Fig.7 follows different logic when loss is detected. Each WTP host maintains a packet jitter variable along with RTT and WND size. So, having received at least one segment it estimates a waiting time for the whole group using following equation:

$$D_{rx} = SRTT + (WND_{sender} - 1) \times f, \quad (11)$$

where $D_{rx}$ – waiting time (max delay) at receiver, ms; $SRTT$ – average smoothed RTT, ms; $WND_{sender}$ – sender’s WND; $f$ – packet jitter, ms. All variables in (11) are statistically collected and known in advance except sender’s WND size for current group of segments. Thus, at time $t_1$ receiver is able to compute time delay for ACK that it must send back to sender. A bitmap field ACK indicates explicitly which segments were delivered by time $t_1$ and which were not. So, sender at time $t_2$ immediately retransmits lost segment. On Fig.7 it is segment with seq_no=7 with increased field CNT that indicates retransmission trial. WTP also redirects the traffic when loss detected to alternative route, if any. As a result, WTP more efficiently utilize bandwidth as no timeout fired because $D_{rx} < RTO_{tcp}$. Moreover, every timeout event in TCP decreases cwnd twice. Upon loss event WTP reduces sending rate to the actual amount of lost segments but recovers afterwards faster than TCP. If all data segments were lost, $WND_{sender}$ drops to 1 MSS and by timeout retransmits first segment. Recovering procedure in WTP is defined in (12):

$$WND_{new} = \min((WND_{old} - L) \times \frac{3}{4}, WND_{rx}), \quad (12)$$

where L – amount of lost segments in the last group; $WND_{rx}$ – receiver window, MSS. New $WND_{sender}$ is recalculated by (12) only when loss is detected. But protocol also takes predicative measures in order to avoid data loss and network congestion by estimating current RTT value as denoted in (13). In contrast to that, TCP constantly increases $cwnd$ until either timeout event happens or 3 duplicate ACKs received only.

$$WND_{new} = WND_{old} - \left(\frac{SRTT - RTT_{last}}{RTT_{last}} \times WND_{old}\right) + 1 \quad (13)$$

Equation (13) applied to both sender and receiver side but with respect to (10). We investigate the effect of the proposed solutions with respect to (10-13) later in this work.

Consider another scenario when a segment is not lost but delivered out of order. Behavior of TCP and WTP is shown on Fig.9 and Fig.10 respectively.
According to (11) it has a delay $D_{rx}$ that allows out of order segment to arrive. As soon as third segment, regardless of its seq_no, reaches receiver an ACK is generated back to sender.

Finally, consider another important scenario which is sometimes assumed to be negligible – a loss of ACK. Fig.11 and Fig.12 present TCP and WTP dialogs when ACK is lost.

When ACK is lost TCP sender has no means to determine it therefore it retransmits segments considered to be lost by timeout as illustrated on Fig.11. At time $t1$ it also decreases cwnd twice reducing sending rate. WTP also uses timeouts to retransmit segments considered to be lost but does it in a more conservative way by sending only first segment in the group. Respective ACK notifies a sender about amount of actually delivered and lost segments. WTP then immediately recovers to the previous sending rate using equation (12) if there were lost segments, otherwise (13). Calculation of the appropriate retransmission timeout (RTO) is a challenging task and WTP uses the same equations (1-5) as TCP. If the segments were transmitted over different routes respective RTO is used (example stated in Table 1).

C. Connection closing phase

Like TCP, proposed protocol requires hosts to close connection by exchanging special segments. In WTP flag FIN is used for that. It might carry a bit indicating that remote host must acknowledge connection closing. If this bit is not set then FIN segment forcibly resets connection like RST segment in TCP. Connection is immediately closed, all allocated resources are freed and no ACK from remote host in this situation expected. It also implies that FIN segments carry no data since they do not guarantee ACK. Hence, in this phase WTP needs maximum 2 segments unlike TCP which requires at least 3: FIN ACK, FIN ACK, ACK.

V. EXPERIMENT RESULTS

In this section we estimate effectiveness of the proposed WTP protocol. The main objective of this section is to compare it with standard TCP implementation compliant to RFC793 and RFC2581. For proper comparison experimental application data transmissions have been carried out on a custom made embedded radio module with Cortex-M3 microcontroller. Respective MANET evaluation kit, shown below, has been developed as well for distributed data visualization and statistics collection.

Total 3 evaluation kits were participating in the experiment: two nodes communicating through a relay, no routing protocol used and no direct communication between hosts. Thus, only
one route was present with 2 hops. Embedded radio module is built around LM3S9D92 microcontroller from TI® and ADF7242 radio transceiver from Analog Devices® working in 2.4GHz ISM band, 1 Mbps data rate (packet mode) and CSMA MAC protocol. Evaluation kit contains OLED screen needed for visualization of essential statistical information in addition to 256 kB of dedicated flash where all socket variables have been written. As for correct TCP implementation the “lwIP” TCP/IP stack has been used [15]. In order to put both protocols in the same conditions and guarantee fair measurements segment drops and “lost ACKs” events have been generated deterministically but at random time. Total amount of such losses per test has been limited to 3% maximum.

Totally in experiment 5 data transmission tests have been performed, 100 transmission rounds each. Both TCP and WTP started at the same time where sender generated dummy application data trying to maximize data rate. Actual application data throughput, protocol effectiveness as well as other socket statistics have been measured and presented in Table 2. For the first test behavior of the congestion windows for both protocols is shown on Fig.14, round trip time values are presented on Fig.15, max application data throughput is shown on Fig.16.

Table II. Transmission statistics

<table>
<thead>
<tr>
<th>Test</th>
<th>Total</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>TCP</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packets sent</td>
<td>621</td>
<td>599</td>
</tr>
<tr>
<td>Data segments sent</td>
<td>399</td>
<td>481</td>
</tr>
<tr>
<td>ACK segments sent</td>
<td>211</td>
<td>107</td>
</tr>
<tr>
<td>Data segments lost (timeouts)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ACK segments lost</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Retransmitted segments</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Time elapsed, ms</td>
<td>5359</td>
<td>4917</td>
</tr>
<tr>
<td>Traffic, b</td>
<td>65140</td>
<td>68540</td>
</tr>
<tr>
<td>Average RTT (including RTO), ms</td>
<td>55,8</td>
<td>51,2</td>
</tr>
<tr>
<td>Average throughput, kbps</td>
<td>12,1</td>
<td>13,9</td>
</tr>
<tr>
<td>Effectiveness (app data/total traffic)</td>
<td>0,41</td>
<td>0,53</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average RTT (including RTO), ms</td>
<td>53,2</td>
<td>49,5</td>
</tr>
<tr>
<td>Average throughput, kbps</td>
<td>12,9</td>
<td>15,1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average RTT (including RTO), ms</td>
<td>67,6</td>
<td>60,6</td>
</tr>
<tr>
<td>Average throughput, kbps</td>
<td>9,4</td>
<td>12,3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average RTT (including RTO), ms</td>
<td>69,5</td>
<td>56,2</td>
</tr>
<tr>
<td>Average throughput, kbps</td>
<td>9,3</td>
<td>12,8</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average RTT (including RTO), ms</td>
<td>48,2</td>
<td>43,6</td>
</tr>
<tr>
<td>Average throughput, kbps</td>
<td>14,0</td>
<td>17,1</td>
</tr>
</tbody>
</table>

In every test run presented in the table 2 amount of lost packets varies from 1% to 3% out of totally generated. Round trip time in both protocols is almost the same (difference in 1-2 ms) when network delivers the packets but it changes drastically when segment or its ACK is lost (Fig.15). The latter is especially important to take into consideration since in MANETs probability of small packet loss is much higher than in fixed network and caused mainly not by a router buffer overflow but by other MANET specific reasons, such as mobility.

Fig.14 shows the behavior of congestion window for TCP and WTP. As can be seen when data segment is lost TCP timeout triggers retransmission and decreases cwnd size twice of its size regardless of the actual loss reason. WTP at this point differentiates losses. It acts the same when ACK is lost slowing down sending rate to TCP level or even lower, though it recovers much faster.

As it was mentioned before classical TCP interprets every packet drop as network congestion, triggering retransmission of the lost segment by timeout. Calculation of the appropriate retransmission timeout (RTO) is a challenging task. Retransmission timeout in WTP is calculated in the same manner as in TCP but in combination with changing processing logic at the receiver it gives faster reaction to the loss in the network. Using equation (11) and the fact that $D_A < RTO_{TCP}$ proposed protocol enables more adequate adjustment to data
losses, delay and current situation in the connection, thus, improving connection throughput. In addition to that, WTP takes preventive measures to avoid network congestion either when RTT grows or when data segment is lost but ACK has been delivered successfully using (13) and (12), respectively.

As a result average throughput of WTP is higher in all 5 tests – about 19% in average. This is mainly due to lower header size, bigger MSS, changing processing logic at both sides and introducing receiver side deterministic mechanisms of delayed ACK computation that excludes unnecessary ACK transmissions.

![Fig.16. TCP and WTP throughput at sender](image)

VI. CONCLUSIONS AND FURTHER WORK

Efficient transport layer protocol is important part of the TCP/IP protocol stack as it directly interacts with user application, especially in MANETs. In this paper we investigate main problems of reliable data transfer in MANETs as well as general reliability architecture at transport layer. Since every node is mobile a loss of a packet can be caused not only by network congestion but also by many other reasons, first of all, by the node mobility and topology change.

We proposed wireless transport protocol (WTP) – a new transport layer protocol that was designed specifically for mobile ad hoc networks. It has low overhead and capable to adjust its performance depending on different parameters in the connection such as RTT, different the loss events and other. One of the biggest innovations in the proposed protocol is introducing at the receiver side deterministic delayed ACK computation which shortens the reaction time to the segment loss, thus, utilizing available bandwidth more efficiently. Experiments show that having up to 3% of lost segments in the connection WTP performs roughly 19% better than standard TCP with no optimization to MANETs.

However, in this work during the experimental measurements it was not fully discovered the effects of the cross-layer optimization due to limited amount of equipment. Hence, impact of cross-layer support in WTP is the subject for further experiments and research.

REFERENCES