DualRTT: Detecting Spurious Timeouts in Wireless Mobile Environments

Shaojian Fu and **Mohammed Atiquzzaman**
Telecommunications and Networks Research Lab
School of Computer Science
University of Oklahoma,
Norman, OK 73019-6151, USA.
Email: atiq@ou.edu

***Abstract*— Retransmission ambiguity, arising from delay spikes in a wireless mobile environment, results in poor TCP performance. Eifel improves the performance of TCP by using the timestamp option, which requires additional header bytes, resulting in increased overhead in bandwidth constrained wireless networks. Moreover, the destination needs to support the timestamp option. In this paper, we propose a new algorithm, called DualRTT, which increases the performance of TCP in the presence of delay spikes, without requiring any additional header bytes. It requires changes only at the sender, and hence is *easier to deploy in the existing Internet infrastructure*. It also does not require the destination to support the TCP timestamp option. Results show that DualRTT increases the performance of TCP, and also achieves a higher transport layer efficiency than previous algorithms.**

Keywords: Delay Spikes, Internet protocols, wireless mobile networks, TCP.

Methods Keywords: Simulations.

I. Introduction

TCP was originally designed for wireline environments where packet losses are primarily due to congestion. TCP estimates the Round Trip Time (RTT) to set the Retransmission Time Out (RTO) which is used by TCP's congestion control algorithms [1] to carry out retransmission of packets lost due to congestion. The onset and disappearance of congestion is usually a slow and gradual process; the RTO computation of TCP is therefore based on *slow and gradual* changes in RTT.

In contrast to wireline networks, wireless mobile networks, such as GPRS [2] and CDMA2000 [3], encounter *high bit error rates* and *temporary disconnection*. These networks generally use link layer recovery protocols, such as Radio Link Control (RLC) [4], [5], to recover from packet losses due to errors. Mobility, in conjunction with the use of wireless protocols, can result in *delay spikes* which may render TCP's RTT and RTO estimation inaccurate. A delay spike is defined as a situation where the round-trip time (RTT) suddenly increases for a short duration of time, and then drops to the previous value [6]. Causes of delay spikes in a wireless mobile environment include [7]:

- *Handoff* of a mobile host to a new cell requires the new base station perform channel allocation before data can be transmitted from the mobile host. This causes segments at the mobile host to be queued, giving rise to sudden extra delays.
- *Physical disconnection* of the wireless link during a handoff can result in a sudden increase of the RTT.
- A Radio Link Control (RLC) layer between the LLC and MAC layers to carry out retransmissions at the link layer in wireless mobile networks, such as GPRS and CDMA2000, may result in delay spikes due to repeated retransmission attempts during link outages and high BER periods.
- Higher-priority traffic, such as circuit-switched voice, can preempt (block) the data traffic. The duration of this blocking may be very long as compared to TCP's RTT estimate.

Frequent delay spikes are, therefore, more common in wireless mobile networks than wireline networks. Delay spikes confuse TCP's RTT estimator, because the RTO estimator can't adapt quickly enough to handle sudden RTT changes due to delay spikes. Sudden increase of instantaneous RTT beyond the RTO of the sender results in retransmission ambiguity [8], [9], which will produce *Spurious Timeout*[1] (ST) and *Spurious Fast Retrans-*

This work was supported by NASA grant no. NAG3-2528.

[1] **Spurious timeout** is defined as a timeout which would not have happened if the sender waited long enough. It is a timeout resulting in retransmission due to a segment being delayed (but NOT lost) beyond RTO [9].

mission[2] (SFR), and causes serious end-to-end (transport level) performance penalty [9], [10].

The Eifel algorithm [9], which has been proposed to alleviate TCP's performance penalty, utilizes TCP's timestamp option [11] to solve the retransmission ambiguity by distinguishing between the acknowledgement for the original segment (transmitted before or during the delay spike) and the segment retransmitted after the spurious timeout. Although Eifel increases TCP's performance, the timestamp option adds an additional 12 bytes to the TCP header; this is a *significant overhead, especially for small segments and in bandwidth-limited wireless environments*. Eifel also requires both the sender and receiver to support the TCP timestamp option.

An alternative to Eifel has been proposed by Ludwig [12], where the Retransmit (RXT) flag, a bit taken from the Reserved field of the TCP header, is used to achieve the same function as that of Eifel. The TCP sender sets the RXT flag of segments containing retransmitted data. In response to such a segment, the TCP receiver immediately sends a pure ACK with the RXT flag set. By inspecting the RXT flag of the ACKs that arrive after a retransmission, the TCP sender can resolve retransmission ambiguities. Note that this scheme requires changes at both the sender and the receiver, resulting in deployment issues.

Sarolahti et. al. proposed F-RTO [13], which also monitors received ACKs to determine spurious timeouts. When the first ACK is received after a retransmission, the sender doesn't retransmit the other un-acknowledged segments immediately. If the ACK advances the sender's window, the sender transmits two new segments, then waits for another ACK. The sender infers a spurious timeout if the second incoming ACK advances the sender's window again.

The proxy solution [14] proposed by Kim et. al. introduced a new performance enhancing proxy (PEP), which operates at the border of wireless networks and the Internet. The PEP tracks the data and acknowledge sequence number for each TCP connection. By filtering unnecessarily retransmitted segments and removing duplicate ACKs, the spurious fast retransmissions at the TCP sender can be avoided. This solution needs additional infrastructure change and it suffers scalability problem when the number of TCP connections is large.

Blanton [15] proposed using TCP DSACKs [16] to give the sender more information (than TCP SACKS can provide) about the "spuriously retransmitted" duplicate segments received by the receiver. Since spurious retransmissions occur between spurious timeouts and the notification from the receiver about duplication segments, the mechanism can only detect spurious retransmissions but not spurious timeouts, and hence *cannot prevent spurious retransmissions*. Gurtov et. al. suggests restarting the retransmission timer, and ignoring the DupAcks that arrive after a timeout [6] (more conservative than RFC2581). As in RFC2581 with *bugFix* enabled, this mechanism can only prevent spurious fast retransmissions.

A large body of research, such as AIRMAIL [17], I-TCP [18] and Snoop Protocol [19], have been carried out to improve TCP's performance in wireless environments by alleviating the effects of non-congestion-related losses [20]. This paper focuses on improving TCP performance due to delay spikes (excessive delays) rather than packet losses, and hence is different from the above research efforts.

The *objective* of this paper is to improve the performance of TCP in the presence of delay spikes. We propose a *new TCP sender based algorithm, called* `DualRTT`*, to improve the end to end performance by detecting spurious timeouts*. It has the advantage of not requiring any additional headers in the packets, or any change at the TCP destination or the network infrastructure. `DualRTT` is based on adding a new RTT measurement (at the sender), which records the time between the most recent retransmission of a packet and the acknowledgement of that packet. The minimum value of RTT observed until the current time is also stored in a variable. Spurious timeouts are detected by comparing the new RTT value and the minimum value of the RTT observed so far.

Our work *differs from previous work* in the sense that `DualRTT` takes into account the dynamics of packet queueing at the wireless link during a delay spike. To detect spurious timeouts, it exploits the fact that packets, delayed due to a delay spike, are queued consecutively at the sender side of the wireless link. Similar to F-RTO, the algorithm has the *advantage* that it does not require TCP timestamp option support at the sender and receiver, thereby eliminating the timestamp option overhead, which is desirable in bandwidth limited wireless environments.

Real world measurements by National Laboratory for Applied Network Research (NLANR) show that *58.5% of the uplink web traffic packets have a small payload (between 0-64 bytes) [21]*. For small packet sizes, `DualRTT` results in a higher transport layer efficiency, as will be shown in Sec. VII-E.

The main *contributions* of this paper can be summarized as follows:

[2]**Spurious fast retransmission** occurs when segments get reordered beyond the DUPACK-threshold in the network before reaching the receiver [9], i.e. the reordering length is greater than the DUPACK threshold (three for TCP).

- *Proposed a new algorithm* (DualRTT) that can detect TCP spurious timeouts caused by delay spikes in a wireless mobile environment without the support of TCP timestamp option, thereby eliminating the dependence on the TCP timestamp option. [3]
- DualRTT is *sender-based*, and no modification is required at the receiver or the network infrastructure, thereby making it easier to deploy in existing networks.
- Shown that the transport layer efficiency of DualRTT is higher than previous algorithms for small packet sizes. Note that a higher transport layer efficiency translates to greater network bandwidth being available for carrying the payload.
- *Demonstrated* that the new algorithm can enhance TCP performance without using any additional header bytes.

The rest of the paper is organized as follows. In Sec. II, we lay the groundwork for the motivation of the problem by discussing the effect of delay spikes on transport protocols. To facilitate comparison between our algorithm and Eifel, we review the Eifel algorithm in Sec. III. Our proposed algorithm (DualRTT) for detecting spurious timeouts is described in Secs. IV, followed by an analytical model to determine the parameters of our algorithm in Sec. V. Sec. VI compares the proposed algorithm with Eifel. Performance comparison of the proposed algorithm and Eifel, based on *ns-2* simulation, is presented in Sec. VII, followed by concluding remarks in Sec. VIII.

II. Effect of Delay Spike on Transport Protocols

In this section, we use segment trace plots obtained from the *ns-2* simulator [23] to illustrate the adverse effect of a delay spike on the throughput of TCP.

A. Simulation setup

The following parameters are used for the simulation topology shown in Fig. 1.

- The end-to-end delay between source (S) to destination (D) is 1.4 second (corresponding to the average delay in GPRS networks [4]) for both uplink and downlink.
- Without loss of generality, for illustration purpose, we use a link bandwidth of 46.8Kbps in this section. A range of link bandwidths will be used when we evaluate DualRTT in Sec. VII.

[3]Recent Internet measurements (April 2003) shows that even though 80.51% current server OSs support the timestamp option by default, most common client OSs do not have the option enabled by default during the connection setup [22].

TABLE I
Simulation Parameters.

Protocol:	TCP Reno
TCP Header size:	20 bytes
Payload size:	536 bytes
rwnd limit	20 segments
Initial *cwnd*	1 segment
Initial *ssthresh*	20 segments

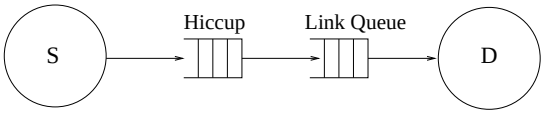

Fig. 1. Simulation Topology.

- A delay spike occurs in the uplink, beginning at time $t = 28.0$s and lasting for 12 seconds. The delay spike was simulated using an *ns-2* module called "hiccup" [24] which holds all the arriving segments for 12 seconds.
- To ensure a continuous supply of data, an FTP source was used at the sender.

TCP parameters used in our simulation are shown in Table I.

B. Spurious Timeout (ST) and Spurious Fast Retransmission (SFT)

The time plot of the simulation is shown in Fig. 2. After

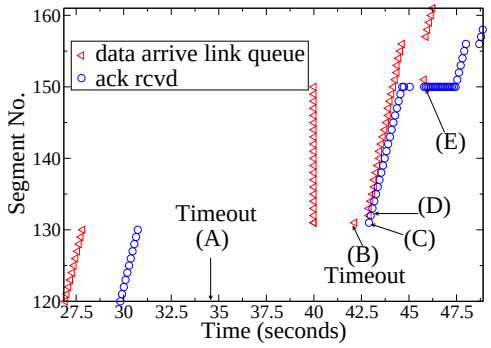

Fig. 2. Spurious Transmission and Spurious Fast Retransmission in TCP Reno.

the delay spike begins, the first segment leaving the sender is segment 131 at time $t = 28.99$s when $RTO = 4$s (see Figs. 2). This segment, as well as later segments 132-150, are held up in the hiccup queue until $t = 28 + 12 = 40$s, when all the segments in the hiccup queue are released to the link queue. Because only one timer is maintained for this connection, and ACKs for earlier segments (prior to 131) arrive at the sender during the delay spike, the timer doesn't time out until $t = 34.54$s (point Fig2-A[4]) when segment 131 is spuriously retransmitted, RTO is updated

[4]We use the notation Fig*x*-*y* to represent point (y) in Figure x

to 7.6s, and the congestion window of sender is reduced to one segment. Because original segment 131 is not lost, but only delayed, *this timeout is a spurious timeout*.

The above spurious retransmission of segment 131 occurs during the delay spike, and is thus also delayed until the end of the delay spike at $t = 40.0$s, when all the segments which are held up in the hiccup queue (segments 131-150) are released to the link queue, as shown in Fig. 2. Note that the original and spuriously retransmitted segment 131 are overlapped at $t = 40.0$s, and hence can not be distinguished from one another.

The sender again times out at $t = 42.14$s (which equals the time of last timeout (34.54s) plus the RTO value of 7.6s) as shown at point Fig2-B. The sender, on receiving the ACK for original segment 131 at $t = 42.89$s (point Fig2-C), starts retransmitting the outstanding segments using the Slow Start algorithm. The effect of go-back-N behavior of the Slow-Start algorithm is evident from the retransmission, starting at point Fig2-D, of segments 132-150.

Although not shown in the figure, up to $t = 47.5$s the receiver receives segments in the following order:

$$131, 132, \ldots, 150, \underbrace{131, 132, \ldots, 150}_{\text{spuriously retransmitted segments}}, 151, \ldots$$

On receipt of the spuriously retransmitted (duplicate) segments 131-150, the receiver generates a series of DupAcks acknowledging segment 150. When the Reno TCP sender receives the 3rd DupAck, it does fast retransmission of segment 151, as shown in Fig2-E. Since segment 151 is not lost, this is a *Spurious Fast Retransmission* resulting in the congestion window being halved unnecessarily. It's important to note that the above Spurious Fast Retransmission is a consequence of the go-back-N Spurious Retransmission which started at point Fig2-D. It has been pointed out by Ludwig et.al. [9] that the fundamental reason for ST and SFR is *retransmission ambiguity* arising from TCP sender's inability to distinguish between ACKs from an original segment and the corresponding retransmitted segment.

C. Effect of Delay Spike on TCP Reno

In this section, we start with the assumption of a loss-free network to present the effect of delay spikes on the behavior of Reno. The simulation topology, link delay, and link bandwidth are the same as stated in Sec. II-A.

It was shown in Sec. II-B that the current Reno congestion control algorithm results in Spurious Timeout and the associated go-back-N behavior in the presence of delay spikes. However, a bug fix proposed in RFC 2582 [25] implements a "more careful policy" than that of standard Reno in treating the TCP DupAck series. The policy disables fast retransmissions until all the segments outstanding at timeout are Acked. When this bug fix is used, fast retransmissions can be eliminated. Fig. 3 shows the segment plot for this scenario. Note that the sender still retransmits all the outstanding segments starting from point Fig3-B, but at $t = 45.8$s (point Fig3-C there is no retransmission of segment 151, and the sender's congestion window is not halved.

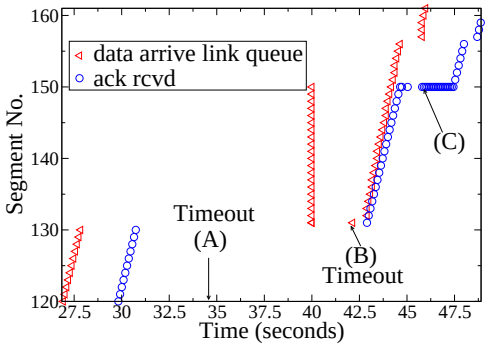

Fig. 3. TCP Reno behavior with RFC 2582 bug fix.

III. Eifel Algorithm

Eifel [9] was designed specifically to improve TCP performance in the presence of delay spikes. The fundamental reason for a go-back-N retransmission in TCP is the retransmission ambiguity (see Sec. II-B) when the sender gets acknowledgements after timeout. The idea of Eifel is straight-forward: use the TCP timestamp option to eliminate this ambiguity.

In Eifel, every TCP segment sent by the sender is timestamped using the TCP timestamp option. The sender also stores the timestamp of the first retransmitted segment, irrespective of whether the retransmission is triggered by a timeout or a fast retransmission. The receiver echoes back the timestamp in the ACK segment. When the ACK for the retransmitted segment comes back, the sender compares the ACK's timestamp with the one it stored earlier. If the ACK's timestamp is smaller than the one stored, the sender concludes that the timeout and retransmission were spurious and unnecessary. The sender then restores *cwnd* and *ssthresh* to the values before the timeout, and *transmits new segments instead of going through go-back-N*.

Fig. 4 shows the segment plot for the Eifel algorithm using the same topology and simulation parameters as in Sec. II-A. In Fig. 4, two timeouts occur at $t = 34.1$s and $t = 40.5$s. Note that when the sender gets the ACK for the original segment 131 at $t = 42.9$s (point Fig4-A), it detected the spurious timeout. As a result, in contrary to Fig. 2 for TCP Reno, segments 132-150 are *not retransmitted*, and the congestion window is restored to the pre-

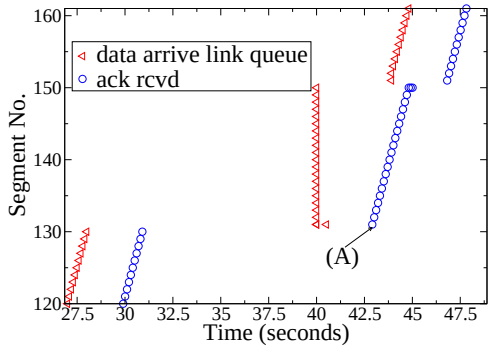

Fig. 4. Detection of spurious timeout by Eifel.

vious value. No DupAcks are generated by the receiver, thereby eliminating Spurious Retransmissions.

The Eifel algorithm uses the same congestion control mechanisms (Slow start, Congestion Avoidance, Fast Retransmit and Fast Recovery) which are used by TCP Reno. One deviation of Eifel from TCP Reno is the action taken after detection of ST (Sec. I): On detection of a spurious timeout, Eifel restores the congestion window and slow start threshold as if the timeout hadn't occurred [9].

The problem with Eifel is the header overhead incurred by additional 12 bytes required for the TCP timestamp option field in the TCP header. This reduces the transport layer efficiency (see Sec. VII), which measures the actual amount of the link bandwidth used for carrying useful data (payload). Eifel also requires the receiver to support the timestamp option, giving rise to deployment issues.

IV. `DualRTT`: The Proposed Algorithm to Detect Spurious Timeouts

In this section, we describe our proposed `DualRTT` algorithm for the detection of spurious timeouts arising from delay spikes in mobile wireless environments.

A. TCP retransmission timer variables

TCP uses Karn's algorithm [8] to carry out RTT measurements and RTO updates when a timeout occurs. The algorithm restricts RTO updates for retransmitted segments as follows:

> .... When an acknowledgement arrives for a packet that has been sent more than once (i.e., retransmitted at least once), ignore any round-trip measurement based on this packet, thus avoiding retransmission ambiguity ....

Note that Karn's algorithm avoids incorrect RTT measurements by avoiding retransmission ambiguity, i.e. the sender does not perform RTT measurements on retransmitted segments. The reason is that if RTT measurement are based on the transmission time of the original packet, the RTT estimate may be too pessimistic. On the other hand, an RTT measurement based on the transmission time of the most recent retransmitted packet may result in a too optimistic estimate. Therefore, neither RTT is taken into account for updating RTO.

TABLE II
RTT AND RTO MEASUREMENT BY KARN'S ALGORITHM.

Time	RTO	RTT
17.545	4.6	2.9
30.840	3.8	2.9
42.905	15.2	2.9
43.004	15.2	2.9
43.102	15.2	2.9
43.791	15.2	2.9
43.890	15.2	2.9
43.988	15.2	2.9
44.481	15.2	2.9
44.579	15.2	2.9
47.780	4.0	3.3
52.704	3.8	2.9

Table II shows several RTT and RTO values near the long delay (which occurs between 28 to 40 seconds) corresponding to the TCP simulation in Fig. 2. Between $t = 30.840$s and 42.905s, two timeouts occurred, and the RTO doubled twice to $3.8 \times 2 \times 2 = 15.2$s. Following that, although the sender received some acknowledgements, it didn't update the RTT and RTO values because the acknowledgements were for retransmitted segments which were ineligible for updating RTT and RTO. After $t = 47.780$s, the acknowledgement of new segments (not retransmitted) are used to update RTT and RTO.

B. The `DualRTT` algorithm

In our proposed `DualRTT` algorithm, we assume the time interval between the arrival of adjacent delayed segments at the receiver is small. This assumption is based on the observation that during a delay spike in a wireless mobile communication system, the segments are queued at the link buffer of the wireless link [26]. When these segments are released from the buffer at the end of the delay spike, they will arrive at the receiver almost back-to-back, the arrival interval being approximately equal to the queueing delay in the buffer.

`DualRTT` adds *two new variables at the sender*:

- A new RTT measurement variable called $NRTT$. $NRTT$ records the time between the "most recent retransmission" and the "arrival of acknowledgement" of the corresponding segment at the sender. Note that if the segment is not a retransmitted segment, $NRTT = RTT$. The RTO update still uses Karn's algorithm, i.e. $NRTT$ is *not* used to update RTO. *The function of $NRTT$ is to detect spurious timeouts.*

TABLE III

RTO, RTT, $MinRTT$ AND $NRTT$ DURING A DELAY SPIKE.

Time	RTO	RTT	$NRTT$	$MinRTT$
17.545	4.6	2.9	2.9	2.8
30.840	3.8	2.9	2.9	2.8
42.905	15.2	2.9	13.9	2.8
43.004	15.2	2.9	0.1	2.8
43.102	15.2	2.9	0.1	2.8
43.791	15.2	2.9	0.2	2.8
43.890	15.2	2.9	0.3	2.8
43.988	15.2	2.9	0.3	2.8
44.481	15.2	2.9	0.4	2.8
44.579	15.2	2.9	0.4	2.8
47.780	4.0	3.3	3.3	2.8
52.704	3.8	2.9	2.9	2.8

- A new variable, called $MinRTT$, which records the minimum value of RTT observed so far since the transport level connection was established.

To get a better understanding of the two new variables, we show the values of $NRTT$ and $MinRTT$ near the long delay in Table III. To illustrate the relationship between the two new variables, RTO and RTT, we also reproduce the values of RTO and RTT from Table II.

We can see from Table III that before the long delay starting at $t = 28$s, $NRTT = RTT$, and $MinRtt$ is a good estimate of the smallest time the sender can expect for a segment to be acknowledged. In our example, the round trip propagation delay was 2.8s (1.4×2). The function of $MinRtt$ is to protect the algorithm against RTT oscillations caused by temporal changes in network conditions.

Detection of a spurious timeout by `DualRTT` is shown in Fig. 5. At $t = 42.905$s (see point Fig 5-A), the sender receives the acknowledgement for the first retransmitted segment (segment 131). The sender increases $cwnd$ from 1 to 2 and sends out two segments: segments 132 and 133 (point Fig5-B). Shortly after the transmission of segment 132, it is acknowledged at $t = 43.004$s, resulting in $NRTT = 0.1$s. Compared to $MinRtt$ at this time (2.8s), $NRTT$ is only 1/28-th of $MinRtt$, which is apparently impossible in a normal network. *We use this as an indication of spurious timeout.* More specifically, `DualRTT` uses the condition that if

$$NRTT < \tau * MinRtt \quad (1)$$

then spurious timeout is detected. τ is a threshold which depends on network conditions such as link bandwidth, path delay, and segment size. In response to detection of spurious transmission, the sender restores *cwnd* and *ssthresh* to the values before the timeout, and resumes sending new segments starting from point Fig5-C.

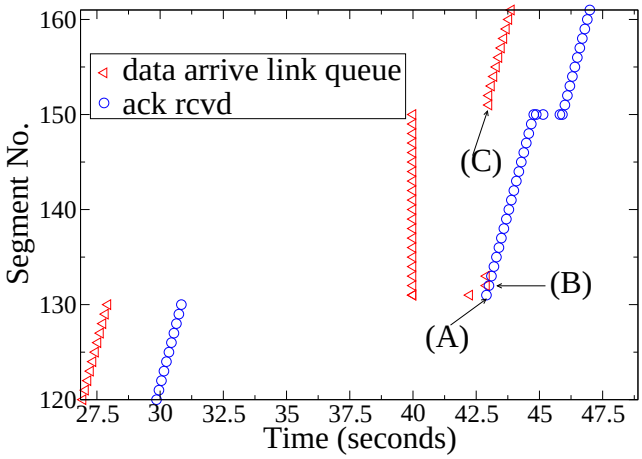

Fig. 5. Detection of spurious timeout by `DualRTT`.

```
BEGIN
Initialization:
        MinRTT=65535
        NRTT=0
New Ack segment arrives:
        if acked segment retransmitted
        then
            NRTT = current_time − last_sent_time
        else
            NRTT = RTT
        end if
        update RTT, MinRTT
        if NRTT < τ * MinRTT
        then
            /*spurious timeout detected*/
            restore saved cwnd and ssthresh
            start transmitting new data
        end if
END
```

Fig. 6. The `DualRTT` algorithm.

The `DualRTT` algorithm is shown in Fig. 6. At the start of the connection (the initialization phase), a large value of $MinRTT$ should be used to prevent it from being assigned a wrong value when the actual path delay is large. Our chosen value (65535 ticks) should be enough for almost all networks, and is easy to implement.

C. Choice of Threshold τ

It is very important to select an optimal value of τ. A low value of τ results in a conservative algorithm. This is because, for given values of $NRTT$ and $MinRTT$ at any instant of time, the lower the value of τ, the harder it is to satisfy Eqn. (1). For example, for $\tau = 0.025$ in the example given in Sec. IV-B, the sender will not detect the spurious timeout because Eqn. (1) will not be satisfied. The value of τ needs to be adaptively adjusted depending on network conditions. In Sec. V, we develop an algorithm to adaptively determine the optimal value of τ to minimize the detection error as seen in Eqn. (2).

V. DETERMINING THE OPTIMAL VALUE OF τ

Now we turn our attention to the problem of dynamically finding an optimal value of τ. We first analyze the

relationship between τ and wireless bandwidth, propagation delay along the path, and path MTU in Sec. V-A. In Sec. V-B, we develop a linear model for τ.

A. Log-Linear relationship between τ and wireless bandwidth, propagation delay, and PMTU

From Eqn. (1), we can observe that a small value of $NRTT$ should allow us to use a small value of τ, and likewise a large value of $NRTT$ implies a larger τ, i.e. the value of τ should reflect NRTT, which represents the proximity in time of adjacent Acks come back just after the delay spike. This time interval between two Acks depends on the network bandwidth, propagation delay along the path, and segment size. So we express τ as a function of bandwidth, propagation delay and PMTU. In order to develop a model for this function, we find the relationship between τ and network bandwidth, propagation delay, and path MTU through simulations.

The simulations were performed with the following values: wireless bandwidth (B) was varied between 31.2 Kbps -1.5 Mbps, path delay (D) ranged from 100 ms to 2000 ms, path MTU (M) ranged from 576 Bytes to 4352 Bytes, and τ varied between 0.01 and 0.6. All values are chosen as discrete values. For every combination of B, D, and M, we simulated 300 randomly generated delay spikes.

The optimal value of τ for each combination of B, D, and M is determined by minimizing the detection error of `DualRTT`. The detection errors can be of two types: True timeouts which are misinterpreted as Spurious timeouts (TMS), and Spurious timeouts which are misinterpreted as True timeouts (SMT). Generally speaking, `DualRTT` is more conservative for a smaller value of τ as described in Sec. IV-C, thereby resulting in a higher SMT. On the contrary, a larger value of τ makes the algorithm more aggressive, and therefore tends to generate a higher TMS.

We determine the value of τ such that the overall detection error (ε), given by

$$\varepsilon = \phi_1 * SMT + \phi_2 * TMS \quad (2)$$

is minimized, where ϕ_1 and ϕ_2 are weighting coefficients, and $\phi_1 + \phi_2 = 1$. Because a TMS error means that a segment loss was not detected by the sender before transmitting a new window of data, and it is very expensive to recover from such a loss [27], we assign a higher value to ϕ_2. A higher value of ϕ_2 will allow the TMS errors in Eqn. (2) to get more priority during the error minimization, resulting in a more conservative algorithm. In our simulation, we selected $\phi_2 = 0.8$.

For example, the simulation results obtained by varying B, with $D = 200$ms and $MTU = 1500$ for 300 delay spikes, are shown in Tables IV and V. Table IV shows the actual number of Spurious Timeouts detected by Eifel during 300 delay spikes. Since Eifel uses the TCP timestamp option to detect Spurious Timeouts reliably, we obtain the table using Eifel. Table V shows the number of Spurious Timeouts detected by DualRTT for various value of τ. To make the comparison fair, we ensured that Tables V and IV were based on the simulations having exactly the same long delay patterns.

TABLE IV
ACTUAL NUMBER OF SPURIOUS TIMEOUT DETECTED BY EIFEL.

B (bps)	*D* (ms)	*M* (KB)	Number of Spurious Timeouts
31.2K	200	1.5	156
62.4K	200	1.5	300
130K	200	1.5	300
360K	200	1.5	300
1.0M	200	1.5	300
1.5M	200	1.5	300

TABLE V
NUMBER OF SPURIOUS TIMEOUT DETECTED BY `DualRTT`.

B (bps)	*D* (ms)	*M* (KB)	τ						
			0.01	0.025	0.04	0.08	0.1	0.2	0.6
31.2K	200	1.5	0	1	3	1	3	6	**158**
62.4K	200	1.5	0	0	0	0	3	**300**	300
130K	200	1.5	0	1	0	**300**	300	300	300
360K	200	1.5	1	0	**300**	300	300	300	300
1.0M	200	1.5	0	**300**	300	300	300	300	300
1.5M	200	1.5	**300**	300	300	300	300	300	300

By comparing Tables IV and V, we can determine the optimal value of τ for each case; as shown by the bold numbers in Table V. The size of the table depends on the number of combinations of B, D, and M used in the simulation. For example, if we choose six B values, seven D, and four M values, then the table will have 168 rows.

We now want to establish the relationship between τ and B, D, & M by averaging τ over B, D, and M in Table V. For example, in order to obtain the B-τ relationship, we plot optimal τ values versus different B values for one D and M combination. We repeat this process for four sets of D and M combinations; the resulting relationship is shown in Fig. 7. Similarly, we can obtain the relationship between τ and D, M as shown in Figs. 8 and 9, respectively.

From Figs. 7, 8, and 9, we can observe that the relationship of τ versus B and D is exponential, while τ versus M is rather close to linear. This analysis justifies our selection of a log-linear model in Sec. V-B.

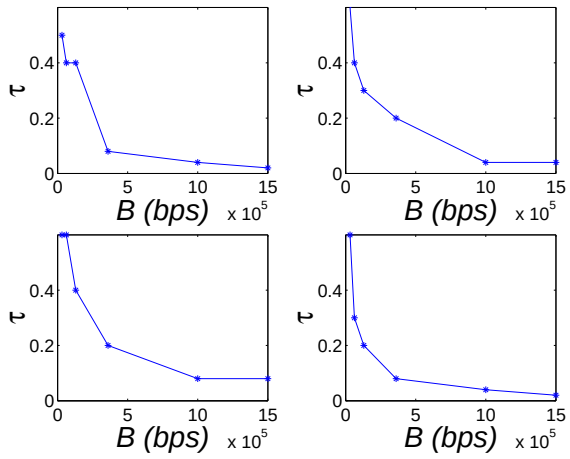

Fig. 7. Relationship between B and τ.

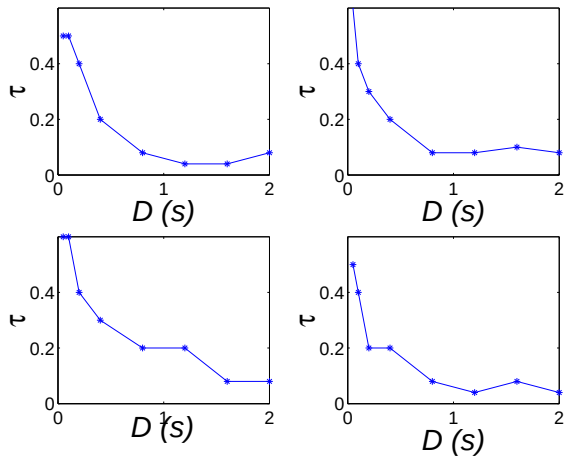

Fig. 8. Relationship between D and τ.

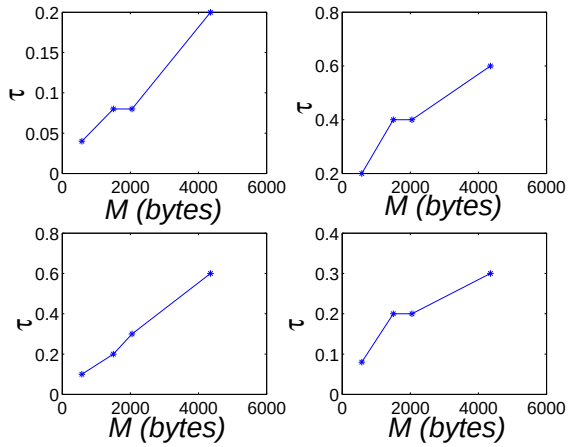

Fig. 9. Relationship between M and τ.

B. A log-linear model for τ

Based on the analysis in Sec. V-A, we can express τ as a linear combination of $\log(B)$, $\log(D)$, and M:

$$\tau = \alpha \log(B) + \lambda \log(D) + \omega M \tag{3}$$

where α, λ, and ω are constant coefficients. Next, we determine the empirical values of α, λ, and ω from simulation data.

We can now rewrite Eqn. (3) in terms of a matrix expression as follows:

$$\tau = H * \begin{pmatrix} \alpha \\ \lambda \\ \omega \end{pmatrix} \tag{4}$$

Here, the columns of H represent the values of $\log(B)$, $\log(D)$, and M. The size of H and τ in this equation depend on the number of combinations of B, D & M used in the simulation. For example, if we choose six B values, seven D values, and four M values, H will have a size of 168×3, and τ will have a size of 168×1.

Extracting optimal values of τ from Table V, we get:

$$\begin{pmatrix} \dots \\ 0.6 \\ 0.2 \\ 0.08 \\ 0.04 \\ 0.025 \\ 0.01 \\ \dots \end{pmatrix} = \begin{pmatrix} & \dots & \\ \log(31.2K) & \log(0.2) & 576 \\ \log(62.4K) & \log(0.2) & 576 \\ \log(130K) & \log(0.2) & 576 \\ \log(360K) & \log(0.2) & 576 \\ \log(1.0M) & \log(0.2) & 576 \\ \log(1.5M) & \log(0.2) & 576 \\ & \dots & \end{pmatrix} \begin{pmatrix} \alpha \\ \lambda \\ \omega \end{pmatrix} \tag{5}$$

By using the least square method, we can determine the best estimation of α, λ, and ω as:

$$\begin{pmatrix} \alpha \\ \lambda \\ \omega \end{pmatrix} = (H^T H)^{-1} * H^T * \tau = \begin{pmatrix} 8.022 * 10^{-3} \\ -5.803 * 10^{-2} \\ 1.463 * 10^{-6} \end{pmatrix} \tag{6}$$

where H^T means the transpose of matrix H.

For given values of B, D, and M, and using the values of α, λ, ω obtained from Eqn. (6), we can determine an optimal τ using Eqn. (3). B and D can be estimated from the sender's statistics about the network path properties [28], and M can be found through a PMTU discovery mechanism as discussed in [29]. Note that during the startup period of TCP connection, or when the mobile host has just moved to a new cell, B and D cannot be obtained accurately from earlier statistics. At these times, a conservative value of τ should be used to start, simulation results from Tables IV and V indicate that a value of τ=0.1 results in no TMS errors and low SMT errors, therefore it is suitable in such cases.

C. Detection error of the model

We examined the accuracy of the above log-linear model for τ by measuring the detection errors for the simulation setup of Sec. V-A. In each of the 168 configurations, we simulated 300 delay spikes. Among a total number of 50400 delay spikes, there were 37500 actual spurious timeouts as measured by Eifel. DualRTT produces an SMT error of 11.3%, and a TMS error of 0.12%, which is consistent with our objective of minimizing the TMS error (see Sec. V-A).

VI. Comparison of Eifel and DualRTT

The time line of DualRTT and Eifel are shown in Figs. 10 and 11 which correspond to the time plots in Figs. 5 and 4 respectively. Every segment is labelled as "S#", where "S" represents the segment type which can be one of the following: "S" for original transmission of a segment; "R" for retransmission of a segment; and "A" for an acknowledgment of a segment. "#" represents the sequence number of the segment.

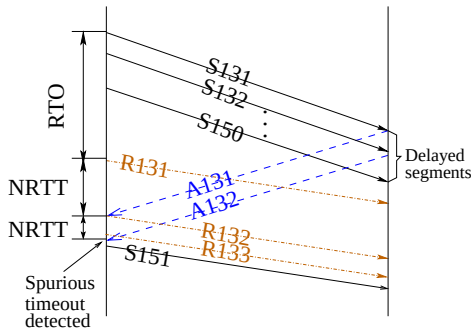

Fig. 10. Detection of spurious timeout in DualRTT.

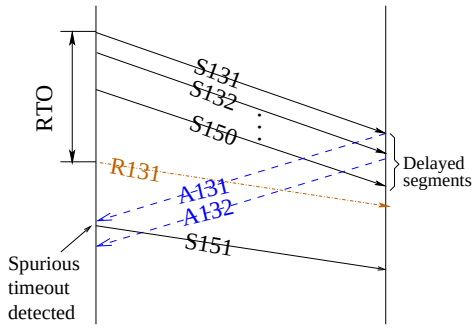

Fig. 11. Detection of spurious timeout in Eifel.

Referring to Fig. 10 for DualRTT, T1 and T2 correspond to the *NRTT* for segments 131 and 132 respectively. DualRTT detects spurious timeout when A132 arrives. In comparison, Eifel detects spurious timeout when A131 arrives. DualRTT therefore needs to wait for slightly longer (time T2) than Eifel to detect spurious timeout. After the detection of ST, both DualRTT and Eifel start transmitting new segments starting at S151. Table VI summarizes the pros and cons of Eifel and DualRTT in detecting spurious timeout.

TABLE VI
Comparison of Eifel and DualRTT.

Algorithm	Advantages	Disadvantages
Eifel	• More robust under certain network conditions. • Detects spurious timeout after receiving acknowledgement from the first retransmitted segment.	• Needs TCP Timestamp option support at both endpoints. • 12 bytes of header overhead.
DualRTT	• No requirement for Timestamp option. • Less header overhead, and hence more efficient than Eifel in the case of wireless networks.	• Less robust in congested networks. • Needs acknowledgement from two retransmitted segments before detecting spurious timeout.

VII. Performance Evaluation

To measure the performance of our proposed DualRTT algorithm, we implemented the algorithm as a subclass of Agent/TCP/FullTCP in the *ns-2* simulator [23]. In this section, we evaluate the performance of DualRTT to determine the increase in the transport layer throughput in the presence of delay spikes. We then compare the transport layer efficiency (defined in Sec. VII-E) of DualRTT and Eifel.

A. Network topology and traffic sources

To evaluate the performance of the new algorithm, we use the parking lot network topology shown in Fig. 12 with three traffic flows: MH→W9 and W7→W2 carry TCP/FTP traffic, and W8→W1 has a TCP/Exponential traffic. MH→W9 represents traffic originating from a Mobile Host (MH) which is affected by delay spikes, and W7→W2 and W8→W1 simulate background traffic. Both the FTP traffic are greedy sources that try to consume as much network resource as possible. The Exponential traffic is an ON/OFF source with burst time 1500ms, idle time 50ms, and sending rate 4.0Mbps. The propagation delay and bandwidth for the links are shown in Table VII.

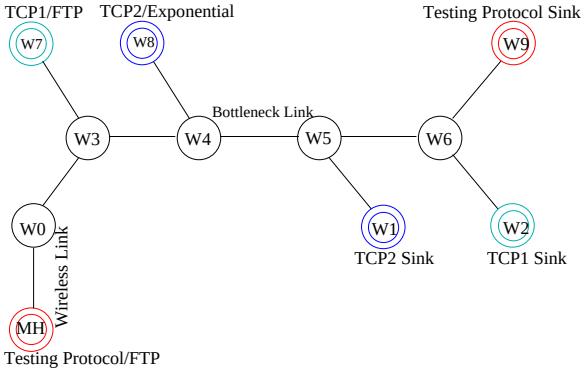

Fig. 12. Network topology for performance evaluation.

TABLE VII

LINK BANDWIDTH AND DELAY OF THE SIMULATION TOPOLOGY.

Links	Link BW (Kbps)	Prop. Delay (ms)
W0-W3, W3-W7, W5-W4 W4-W8 W5-W1, W6-W2, W6-W9	1500	200
MH-W0	15.6-1500	400
W4-W5	200-3500	200

The bandwidth of the wireless link (MH-W0) was varied between 15.6Kbps and 1.5Mbps to investigate the impact of different wireless bandwidth; the bandwidth of the bottleneck link (W4-W5) was varied between 0.2Mbps and 3.5Mbps to investigate the effect of varying bandwidth at the bottleneck link. The wireless link (MH-W0) delay was set to 400ms to take into account the RLC layer ARQ handling delay [4]. Wired link delays were chosen to make the end-to-end delay of TCP traffic equal to 1.4sec, a commonly encountered end-to-end link delay in GPRS networks [4].

B. Delay Spikes

We used the *ns-2* "hiccup" module [24] to randomly insert three delay spikes in the MH→W0 connection during a 150 second FTP session. Large delay spikes (due to cell re-selection) with small interval between spikes (arising from frequent handoffs) makes it difficult for TCP to adapt to RTT changes. To simulate such difficult scenarios [27], our simulation uses delay spikes whose lengths are uniformly distributed between (3, 15) seconds, with the interval between the delay spikes also being uniformly distributed between (20, 40) seconds.

C. Transport protocols

Extensive simulation was performed for the following three protocols at the Mobile Host, using the same payload size for all the protocols.

1) TCP Reno (*ns*-2 ver. 2.1.b.8 implementation);
2) Eifel (implemented by Technical University of Berlin [24]);
3) `DualRTT`.

TABLE VIII

PROTOCOLS PARAMETERS FOR THE THREE PROTOCOLS.

Header size (Bytes):	20 (Reno) 32 (Eifel) **20** (`DualRTT`)
Payload size:	536 bytes
rwnd limit	20 segments
Initial *cwnd*	1 segment
Initial *ssthresh*	20 segments

To obtain a comprehensive comparison among the three protocols, the bandwidth of the wireless link (MH-W0) and bottleneck link (W4-W5) were varied to generate a total of 65 simulation scenarios, with each scenario run for 50 times independently to ensure the statistical fairness of the results. Each simulation run consisted of a 150-second FTP session. Results presented in this section represent the average of all the simulation runs. To ensure fairness among the protocols, the parameters were kept the same for the three protocols as shown in Table VIII.

D. Transport Layer Throughput

We define the Transport Layer Throughput (TLT) of a protocol as the total number of segments delivered to the destination during a fixed duration of FTP session.

Figs. 13 and 14 show the TLT of the three protocols for a bottleneck bandwidth of 200kbps and 1.5Mbps respectively obtained from a 150 second FTP session. Fig. 13 shows that the TLT of TCP Reno, Eifel and `DualRTT` initially increase with an increase in the wireless link bandwidth. However, if the wireless link bandwidth is further increased, the bottleneck link becomes congested and starts dropping packets. Timeouts in delay spikes increase the RTO to a large value, and if packets are lost in the same window as the delay spike, Eifel has to wait a long time to retransmit the lost packet and, therefore, becomes very *sensitive to packet losses occurring after a delay spike* as reported in [27]. As a result, in Fig. 13 the TLT of Eifel drops with an increase in the wireless link bandwidth above 200Kbps. Since packet losses in `DualRTT` are handled the same way as Eifel, the TLT of `DualRTT` also drops when packets are lost after a delay spike. However, when the bottleneck link bandwidth is sufficiently large (for example, 1.5 Mbps), the probability of packet losses after a delay spike due to congestion is very small.

The above negative impact of packet losses on Eifel and DualRTT is not seen in Fig. 14.

We can see in Figs. 13 and 14 that the TLT reaches a saturation point when the wireless link bandwidth reaches around 31.2 Kbps. This is because the receiver window size of 20 segments (see Table VIII) and an end to end round trip delay of 2.8s (Sec. VII-A) limits the TLT of a connection to a maximum of $(20 * 576 * 8)/2.8 = 32.9$ Kbps for TCP Reno and DualRTT, and 33.6Kbps for Eifel, where 576 is the payload size 536 bytes plus 40 bytes of TCP/IP header size.

Fig. 15 shows the 150-second FTP session TLT averaged over different wireless link bandwidths ranging from 15.6Kbps-1.5Mbps for various combinations of protocol and bottleneck link bandwidths. From Fig. 15, we can see that DualRTT significantly increases the TLT of TCP Reno. The TLT of DualRTT is better than Eifel for low bottleneck link bandwidths (under 1Mbps); for other cases, its performance is at least equal to that of Eifel. *It is to be noted that although Eifel detects spurious timeout slightly earlier than* DualRTT, *the TLT of* DualRTT *is better than Eifel because of the fewer header bytes required by* DualRTT. The TLT enhancement of DualRTT over Eifel is not significant because we used a payload size of 536 bytes in our simulation which is large as compared to the 12-byte TCP timestamp option.

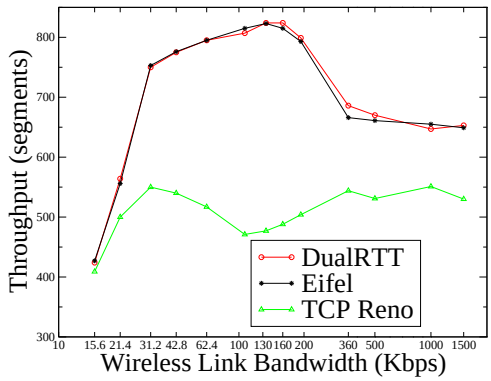

Fig. 13. Comparison of TLT of TCP Reno, Eifel and DualRTT for bottleneck bandwidth of 0.2 Mbps.

E. Transport Layer Efficiency

We now compare the Transport Layer Efficiency of Eifel and DualRTT. We define Transport Layer Efficiency (TLE) as the ratio of bandwidth used by the transport layer segment payload to the total size of a segment as follows:

$$\text{TLE} = \frac{\text{Payload Size of a segment}}{\text{Total Size of a segment}} \quad (7)$$

Table IX shows the TLE of Eifel and DualRTT for various values of payload sizes, with segment header size of

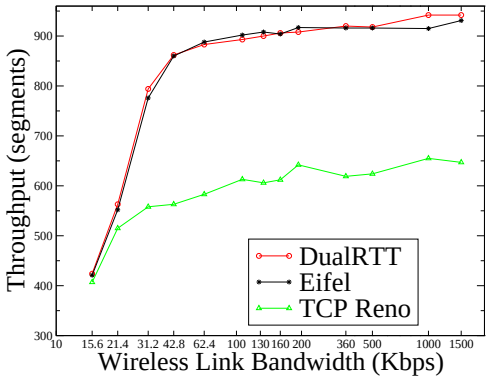

Fig. 14. Comparison of TLT of TCP Reno, Eifel and DualRTT for bottleneck bandwidth of 1.5 Mbps.

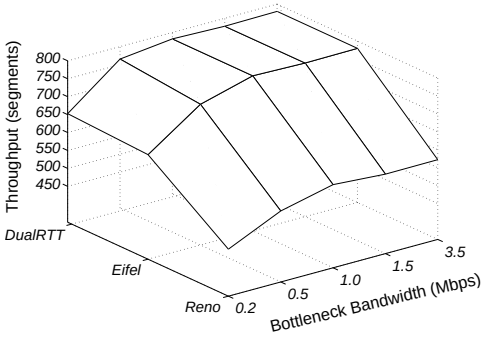

Fig. 15. Average TLT of TCP Reno, Eifel and DualRTT for different bottleneck bandwidth.

20+12=32 bytes for Eifel and 20 bytes for DualRTT. It also shows the percentage increase of TLE of DualRTT as compared to Eifel. The first and second column of the table show the payload size distribution of an NLANR Passive Measurement [21]. The payload size in the first column is the average payload for each group of packets measured: for example, 32 bytes is used for the payloads of length 0-64 bytes.

As can be seen from the table, the 12-bytes of header required by Eifel, due to the use of the timestamp option, results in low TLE for small payloads. For example, for a payload of 32 bytes, the TLE of DualRTT is (0.615-0.5)/0.5 = 23.1% higher than Eifel. Note that higher TLE results in less wastage of network bandwidth, which translates to greater availability of the bandwidth for the transmission of real data (payload). The average percentage TLE increase is calculated by taking the weighed average of column 5, where the weights are taken from the column 2 which shows the percentage of the traffic for a specific payload. We have calculated the average percentage TLE increase is 16.86%.

VIII. Conclusion

In this paper, we have proposed DualRTT, a new algorithm to improve the end-to-end performance of TCP

TABLE IX
COMPARISON OF TLE FOR EIFEL AND DualRTT.

Payload (Bytes)	% Traffic	TLE		% Increase
		Eifel	DualRTT	
32	58.49	0.500	0.615	23.1
96	29.73	0.750	0.828	10.3
192	1.72	0.857	0.906	5.7
376	3.98	0.922	0.949	3.0
768	3.37	0.960	0.975	1.5
1460	2.70	0.979	0.986	0.8

in the presence of delay spikes in wireless mobile environments. DualRTT does not require any additional header bytes, and is therefore suitable for bandwidth constrained mobile wireless networks. DualRTT also *does not require any change at the destination or the Internet infrastructure, nor does it require the destination to support the TCP timestamp option*; it requires changes only at the sender, and hence is *easy to deploy in the existing Internet infrastructure*.

Performance comparison of DualRTT and Eifel shows that DualRTT has a higher transport layer efficiency which translates to more network bandwidth being available to carry the payload data (useful information).

REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," IETF RFC 2581, April 1999.

[2] J. Cai and D. Goodman, "General packet radio service in GSM," *IEEE Communication Magazine*, pp. 122–131, October 1997.

[3] *CDMA2000 Standards for Spread Spectrum Systems*, TIA/EIA/IS-2000.

[4] R. Ludwig and D. Turina, "Link Layer Analysis of the General Packet Radio Service for GSM," in *ICUPC*, San Diego, April 1997, pp. 525–530.

[5] W. Ajib and P. Godlewski, "Acknowledgment operations in the rlc layer of GPRS," in *The Sixth IEEE International Workshop on Mobile Multimedia Communications (MOMUC'99)*, San Diego, California, USA, November 1999, pp. 311–317.

[6] A. Gurtov and R. Ludwig, "Making TCP robust against delay spikes," Internet Draft, draft-gurtov-tsvwg-tcp-delay-spikes-00.txt, February 2002.

[7] A. Gurtov, "Effect of delays on TCP performance," in *IFIP Personal Wireless Communications*, August 2001.

[8] P. Karn and C. Partridge, "Improving Round-Trip Time estimates in reliable transport protocols," *ACM Computer Communications Review*, vol. 17, no. 5, pp. 67–73, August 1987.

[9] R. Ludwig and R. H. Katz, "The Eifel algorithm: Making TCP robust against spurious retransmission," *ACM Computer Communications Review*, vol. 30, no. 1, pp. 30–36, January 2000.

[10] D. S. Eom, H. Lee, and M. Sugano et. al., "Improving TCP handoff performance in mobile IP based networks," *Computer Communications*, vol. 25, no. 7, pp. 635–646, May 2002.

[11] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," IETF RFC 1323, May 1992.

[12] R. Ludwig, "The TCP retransmit (rxt) flag," Internet Draft, draft-ludwig-tsvwg-tcp-rxt-flag-02.txt, November 2001.

[13] P. Sarolahti and M. Kojo, "F-RTO: An algorithm for detecting spurious retransmission timeouts with TCP and SCTP," Internet Draft, draft-sarolahti-tsvwg-tcp-frto-04.txt, June 2003.

[14] Y.C. Kim and D.H. Cho, "Considering spurious timeout in proxy for improving TCP performance in wireless networks," *IEEE Communications Letters*, vol. 8, no. 1, pp. 30–32, Jan 2004.

[15] E. Blanton and M. Allman, "Using TCP DSACKs and SCTP Duplicate TSNs to detect spurious retransmissions," Internet Draft, draft-blanton-dsack-use-01.txt, August 2001.

[16] S. Floyd et. al., "An Extension to the Selective Acknowledgement (SACK) Option for TCP," IETF RFC2883, July 2000.

[17] E.Ayanoglu, S.Paul, T.F. LaPorta, K.K. Sabnani, and R.D. Gitlin, "A link-layer protocol for wireless networks," *ACM Wireless Networks*, February 1995.

[18] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *Proc. 15th International Conf. on Distributed Computing Systems (ICDCS)*, May 1995.

[19] H. Balakrishnan, S. Seshan, and R.H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," in *ACM Wireless Networks*, December 1995.

[20] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," in *ACM SIGCOMM'96*, Stanford, CA, August 1996.

[21] *NLANR Passive Measurement and Analysis (packet counts, port statistics, packet lengths)*, http://www.cs.columbia.edu/~hgs/internet/traffic.html.

[22] Richard Wendland, "How prevalent is Timestamp option and PAWS?," www.postel.org/pipermail/end2end-interest/2003-May/003220.html, May 2003.

[23] *The Network Simulator - ns-2*, http://www.isi.edu/nsnam/ns/.

[24] *NS TCP Eifel Page*, http://www-tkn.ee.tu-berlin.de/~morten/eifel/ns-eifel.html.

[25] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," IETF RFC 2582, April 1999.

[26] G. Racherla et. al., "Performance evaluation of wireless TCP schemes under different rerouting schemes in mobile networks," in *Proc. IEEE TenCon*, New Delhi, India, December 1998, pp. 93–96.

[27] A. Gurtov and R. Ludwig, "Evaluating the Eifel algorithm for TCP in a GPRS network," in *Proceedings of European Wireless*, Florence, Italy, February 2002.

[28] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *SIGCOMM*, Cambridge, MA, September 1999, pp. 263–274.

[29] J. Mogul and S. Deering, "Path MTU Discovery," IETF RFC 1191, November 1990.