# 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 Retransmission*[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.

The *objective* of this paper is to improve the performance of TCP in the presence of delay spikes. We propose a *new*

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].

[2] **Spurious fast retransmission** occurs when segments get re-ordered 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).

0-7803-8991-3/05/$20.00 © 2005 IEEE

*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.

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

- *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. Our proposed algorithm (DualRTT) for detecting spurious timeouts is described in Secs. III. Performance comparison of the proposed algorithm and Eifel, based on *ns-2* simulation, is presented in Sec. IV, followed by concluding remarks in Sec. V.

## II. Effect of Delay Spike on Transport Protocols

In this section, we use segment trace plots obtained from the *ns-2* simulator [13] to illustrate the adverse effect of a delay spike on the throughput of TCP. The time plot of the simulation where ST and SFR occur due to a delay spike is shown in Fig. 1.

After 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. 1). This segment, as well as later segments 132-150, are held up in the hiccup queue until $t = 28 + 12 = 40$s,

[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 [12].

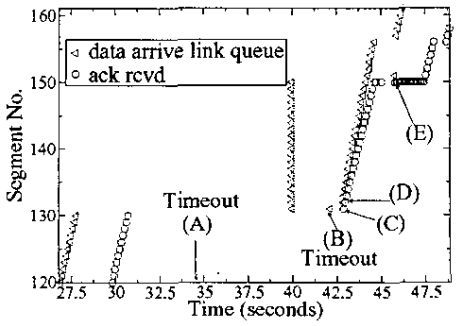



Fig. 1. Spurious Transmission and Spurious Fast Retransmission in TCP Reno.

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 Fig1-A[4]) when segment 131 is spuriously retransmitted, $RTO$ is updated 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. 1. 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 Fig1-B. The sender, on receiving the ACK for original segment 131 at $t = 42.89$s (point Fig1-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 Fig1-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:

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

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 Fig1-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 Fig1-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.

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

TABLE I

*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

*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 |

## III. 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 I shows several *RTT* and *RTO* values near the long delay (which occurs between 28 to 40 seconds) corresponding to the TCP simulation in Fig. 1. 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 [14]. 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.*
- 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 II. To illustrate the relationship between the two new variables, *RTO* and *RTT*, we also reproduce the values of *RTO* and *RTT* from Table I.

We can see from Table II 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 ($1.4 \times 2$). 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. 2. At $t = 42.905$s (see point Fig 2-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 Fig2-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. $\tau$ 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 Fig2-C.

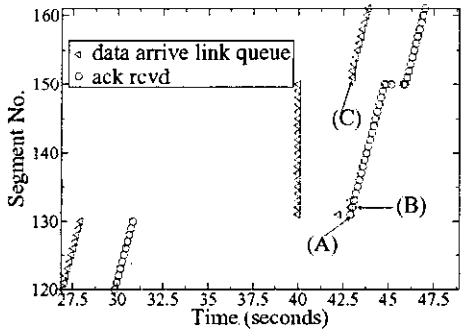



Fig. 2. Detection of spurious timeout by DualRTT.

The DualRTT algorithm is shown in Fig. 3. 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, $\tau$

It is very important to select an optimal value of $\tau$. A low value of $\tau$ 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 $\tau$, the harder it is to satisfy Eqn. (1). For example, for $\tau = 0.025$ in the example given in Sec. III-B, the sender will not detect the spurious timeout because Eqn. (1) will not be satisfied. The value of $\tau$ needs to be adaptively adjusted depending on network conditions. We have developed an algorithm to adaptively determine the optimal value of $\tau$ to minimize the detection error, interested readers may refer our full-sized version of this paper [15] for more details.

```
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. 3. The DualRTT algorithm.

## IV. 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 [13]. In this section, we evaluate the performance of DualRTT to determine the increase in the transport layer throughput in the presence of delay spikes.

### A. Network topology and traffic sources

To evaluate the performance of the new algorithm, we use the parking lot network topology shown in Fig. 4 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.

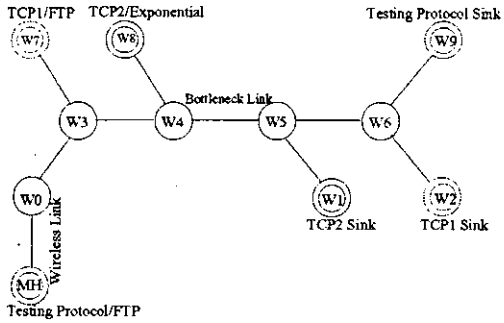



Fig. 4. Network topology for performance evaluation.

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 [16] 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 [17], 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 [16]);
3) DualRTT.

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.

### 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.

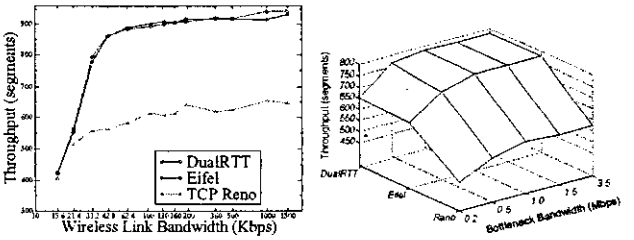
Fig. 5(a) shows the TLT of the three protocols for a bottleneck bandwidth of 1.5Mbps obtained from a 150 second FTP session. Fig. 5(a) shows that the TLT of TCP Reno, Eifel and DualRTT initially increase with an increase in the wireless link bandwidth. However, if the wireless link bandwidth is further increased, the bottleneck link becomes congested and starts dropping packets.

We can see in Fig. 5(a) 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 used in our simulation and an end to end round trip delay of 2.8s (Sec. IV-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. 5(b) 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. 5(b), 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.

## V. Conclusion

In this paper, we have proposed DualRTT, a new algorithm to improve the end-to-end performance of TCP 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, TCP Reno and Eifel shows that DualRTT has a higher transport layer throughput for payload data.




(a) TLT of TCP Reno, Eifel and DualRTT at bottleneck bandwidth 1.5Mbps.

(b) Average TLT of TCP Reno, Eifel and DualRTT for different bottleneck bandwidth.

Fig. 5. Comparison of TLT of TCP Reno, Eifel and DualRTT

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

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

[14] 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.

[15] S. Fu and M. Atiquzzaman, "DualRTT: Detecting spurious timeouts in wireless mobile environments," Tech. Rep., Computer Science, University of Oklahoma, www.cs.ou.edu/~atiq, July 2003.

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

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