A Dynamic Algorithm for Stabilising LEDBAT Congestion Window

Amuda James Abu and Steven Gordon
Sirindhorn International Institute of Technology, Thammasat University,
Pathumthani 12000, Thailand.
james@ict.siit.tu.ac.th, steve@siit.tu.ac.th

***Abstract*—Low Extra Delay Background Transport (LEDBAT) is a delay-based Internet congestion control mechanism developed to allow fair and efficient data transfer when delay-sensitive and file sharing applications co-exist in networks. A LEDBAT source increases its congestion window until a fixed, pre-defined target queue delay is experienced. This paper analyses LEDBAT congestion control showing that the current algorithm, although quickly reaching a steady state (i.e. target delay reached), results in large oscillations of congestion window and queue delay once in steady state. We therefore propose a dynamic calculation of the congestion window gain once in steady state, and show that the proposed modification stabilises the congestion window while still meeting the fairness and efficiency goals of LEDBAT.**

***Keywords*—Internet congestion control, transport protocol, fairness, background file transfer, peer-to-peer applications**

I. Introduction

Low Extra Delay Background Transport (LEDBAT) [14] is a newly proposed one-way delay-based congestion control mechanism for peer-to-peer (P2P) applications and other applications that establish multiple TCP [10] connections for data transfer. LEDBAT is designed so that a source sending rate is similar to TCP-NewReno when no other traffic exists, and yields to newly arriving TCP connections. To be TCP-friendly, a LEDBAT source must not increase its sending rate faster than TCP. Additionally, the source assumes a target queue delay in the path, and aims to adjust its sending rate so that the actual queue delay remains at the target.

The LEDBAT congestion control mechanism is a linear controller where the magnitude of the congestion window increase (and the source sending rate, as the two values are approximately proportional [1]) depends on the difference between the target queuing delay and measured queuing delay, as well as a constant multiplier, $gain$. For the linear controller not to ramp-up faster than TCP, the value of $gain$ must be carefully chosen [14]. Although a $gain$ of 1 packet per round trip time (RTT) in [12], [13] and 10 packets per RTT in [13] have been suggested, no work has analysed the performance impact of different values of $gain$ on LEDBAT.

In this paper we present analysis of the LEDBAT congestion control mechanism, focussing on the rate at which a LEDBAT source changes its sending rate. Ideally, LEDBAT will increase as fast as possible to quickly reach steady state (not faster than TCP), but also maintain a smooth average sending rate, so as not to introduce significant jitter into the network. Our analysis of LEDBAT shows that, using the currently specified constant values of $gain$ in [14], high values allow the source to quickly reach steady state, but result in large variations of sending rate in steady state (hence large variations of the actual queue delay).

To improve LEDBAT performance, we propose an algorithm for dynamically adjusting the $gain$ in steady state. The value of our proposed dynamic $gain$ depends on the congestion window of LEDBAT. Further analysis shows how LEDBAT sources, with our dynamic algorithm, can quickly reach steady state and also maintain a smooth sending rate during steady state. This meets the design objectives of LEDBAT, while improving the throughput and reducing jitter when compared to the original algorithm.

The structure of the rest of this paper is: Section II gives an overview of LEDBAT algorithm, Section III describes the analysis methodology, while Section IV presents results on the performance of LEDBAT with different values of $gain$, our proposed dynamic algorithm is introduced in Section V and compared with the original LEDBAT algorithm, Section VI provides related work while we conclude in Section VII.

II. LEDBAT Congestion Control

A. Motivation of LEDBAT

As a motivation behind the design of LEDBAT, most applications that transfer bulk data in the Internet use TCP [10] as a transport protocol. Some of these applications (e.g. P2P applications such as BitTorrent) are capable of establishing multiple TCP connections for data transfer. Recalling that TCP-NewReno, used widely in the Internet, needs to detect packet loss before it reduces its sending rate [8], thus TCP can potentially fill the buffer of a FIFO bottleneck uplink in the access network of an Internet Service Provider (ISP). The buffer size of most uplinks in home networks are relatively large which can push queuing delay of packets up to several hundreds of milliseconds [14]. When real-time applications such as voice, video, and games co-exist in the same access network with *multiple-connections-initiating* applications that use TCP, the performance of the real-time applications degrades significantly as they cannot withstand high network delay. This led to the effort of a working group in IETF [6] on LEDBAT. A similar *network-latency* minimizing congestion control algorithm has been implemented as Micro Transport Protocol (uTP) used by μTorrent (a UDP-based BitTorrent protocol).

B. LEDBAT Goals and Design

LEDBAT is designed for time-independent applications to provide *lower-than-best-effort* service for end-users towards achieving the following goals [14]:

- To fully saturate the bottleneck while keeping queuing delay low when only LEDBAT is present in the network.
- To quickly yield to traffic traversing the same bottleneck link that uses standard TCP congestion control or UDP.
- To contribute little to the queuing delay caused by TCP.

The LEDBAT algorithm [14], shown in Fig. 1, involves the source estimating the delay to the destination by placing a time stamp in data packets. The destination sends the measured delay of the data packet in a field in the acknowledgement packet. Upon receiving the acknowledgement, the source uses the measured delay to estimate the queue delay in the path. The source assumes the queue delay is the difference between the current delay measurements and a base set of delay measurements. The base one-way delay is taken as the minimum one-way delay from a list of previous one-way delay observations.

```
LEDBAT Destination receives data_packet:
  remote_timestamp = data_packet.timestamp
  acknowledgement.delay =
              local_timestamp() - remote_timestamp
  # fill in other fields of acknowledgement
  acknowledgement.send()

LEDBAT Source receives acknowledgement:
  delay = acknowledgement.delay
  update_base_delay(delay)
  queuing_delay = current_delay() - base_delay()
  off_target = TARGET - queuing_delay
  cwnd += GAIN * off_target / cwnd
```

Fig. 1. LEDBAT algorithm at the sender and receiver

The LEDBAT source has a $target$ queue delay: the source aims not to increase the queue delay above this $target$. The sender increases its sending rate as long as the estimated queuing delay is less than the delay $target$. Otherwise, it reduces its sending rate before the access router buffer is full, in order to allow other applications to obtain a fair share of network resources and experience low queuing delay. Although not explicitly stated in [14], we set the minimum congestion window to be 1 packet. The LEDBAT congestion control algorithm can be implemented with any transport protocol such as TCP, UDP, DCCP, or SCTP.

C. LEDBAT Parameters

LEDBAT uses a linear controller in its design to proportionally modulate LEDBAT congestion window with the estimated queuing delay [14]. A precise description of the controller is given in (1) where w_s is the LEDBAT source congestion window, D is the estimated queuing delay by the LEDBAT source, and $T_{offset} = D - target$.

$$w_s(t+1) = \begin{cases} w_s(t) + \frac{gain \times T_{offset}}{w_s(t)} & \text{if } D < target \\ w_s(t) - \frac{gain \times T_{offset}}{w_s(t)} & \text{if } D > target \\ w_s(t) & \text{if } D = target \\ \frac{w_s(t)}{2} & \text{On packet loss.} \end{cases} \quad (1)$$

LEDBAT aims to achieve friendliness with TCP by: 1) not increasing faster than TCP during start-up phase, 2) quickly yielding to TCP, and 3) halving its congestion window when a packet loss is detected in the path of LEDBAT flow. Carefully choosing a good value of $gain$ is a step towards achieving TCP-friendliness in terms of *not-greater than TCP* ramp-up speed of LEDBAT.

Two parameters, namely $target$ and $gain$, are used in the LEDBAT algorithm. [14] requires $target$ to be set to 25ms while $gain$ should be set such that LEDBAT does not increase faster than TCP. The value of $target$ is justified by the fact that it should not be less than the operating systems accuracy in timestamping packets as queuing delay is estimated from measurement. We use a delay $target$ of 25ms throughout this work. The remainder of the paper analyses the selection of different values of $gain$.

III. Analysis Methodology

The aims of our analysis are to: analyse and quantify the impact of different values of $gain$ on the performance of LEDBAT and other traffic; and improve LEDBAT performance with high values of $gain$.

We consider a scenario of a user uploading a file (FTP) to a remote computer (Fig. 2): First in the absence of other traffic, and secondly in the presence of a real-time video traffic from another user in the same access network. In both cases, we consider the link connecting the access router to the next router as a bottleneck with relatively large buffer size. Using the network simulator ns-2.34 [9] we analyse the performance of using LEDBAT for the file transfer.

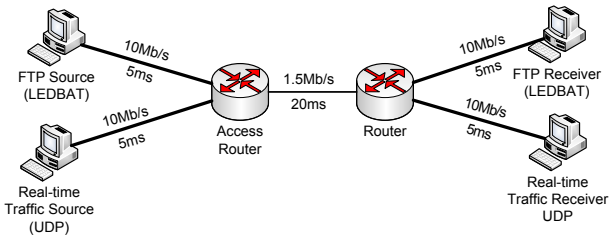

Fig. 2. Simulation network topology

A. Simulation Setup

The network topology is depicted in Fig. 2. We set all links to 10Mb/s and 5ms except the link connecting the access router to the next router. This is set to 1.5Mb/s in order to make it the bottleneck link, and the delay is 20ms because

we assume there are multiple links connecting the end-users' access networks. We use a buffer size of 50 packets at the router with a FIFO droptail queue discipline. Link MTU is given as 1500B. Key parameter values used in the simulation are given in TABLE I. Other parameters take their default values in ns-2.34. In the case of $gain$, 40 is the default value.

TABLE I
SIMULATION PARAMETERS WITH RESPECTIVE DEFAULT VALUES

Category	Parameters	Values
LEDBAT	$gain$	5, 10, 20, 40, 80, 160, 240, 320, 400
	$target$	25ms
	Traffic	FTP
	Packet size	1500B (including IP header)
UDP	Traffic	Constant Bit Rate (CBR)
	Packet size	500B
	Rate	750Kb/s

There are three scenarios analysed:

1) LEDBAT source starts at time 0sec and runs until the end of simulation after 30 seconds. A UDP traffic source starts at time 20sec. We record the instantaneous values of LEDBAT congestion window and queuing delay of LEDBAT packets with each value of $gain$.
2) LEDBAT source runs, on its own, for a duration of 180 seconds.
3) LEDBAT source runs for 180sec, sharing the bottleneck link with a UDP for the entire duration.

For the last two scenarios the performance metrics are:

- Average queuing delay of LEDBAT packets at the access router, denoted as *Queue Delay*.
- Average congestion window of LEDBAT, *Cwnd*.
- Average link utilization of LEDBAT traffic.
- Standard deviation of queuing delays which defines how much the queuing delay of each LEDBAT packet deviates from the queuing delay $target$, denoted as σ_{delay}.
- Standard deviation of congestion window which defines how much each computed LEDBAT congestion window deviates from the average value denoted as σ_{cwnd}.
- Time taken by LEDBAT to reach steady state.

Since one of our objectives is to analyse the performance of LEDBAT with different values of $gain$ in steady state, average values of metrics collected are taken over the duration of the steady state of LEDBAT.

B. Implementation of LEDBAT in ns2

We implement the LEDBAT congestion control algorithm [14] in the network simulator ns-2.34 [9]. It is implemented as a new variant of TCP congestion control mechanism in order to ensure retransmission in case of packet loss. We use TCP timestamping option [16] for the timestamping of packets at the LEDBAT sender.

IV. ANALYSIS OF CONGESTION WINDOW GAIN

In this section, we present our analysis results when we simulate the topology shown in Fig. 2 with different values of $gain$ as given in TABLE I.

As shown in Fig. 3a with a $gain$ of 40, LEDBAT significantly increases its congestion window from 2 packets to 12 packets where it detects that the queuing delay has built-up to about 25ms as shown in Fig. 3b. The congestion window stabilizes at this point in order to keep queuing delay as low as 25ms until the arrival of UDP flow at 20s where LEDBAT estimates queuing delay is greater than $target$. In response to this, LEDBAT reduces its congestion window until estimated queuing delay is less than $target$ where it increases its congestion window to about 6 packets. LEDBAT reaches steady state at this value because UDP traffic arrives at a constant rate of 750Kb/s. This means that LEDBAT makes use of the remaining half of the bottleneck bandwidth of 1.5Mb/s. However, different behaviour of both the congestion window and queuing delay curves are observed in Fig. 3c and 3d respectively as both curves oscillates significantly because of high value of $gain$ (i.e. $G = 320$). This is because, according to (1), higher $gain$ means more growth or shrink of $Cwnd$.

In Fig. 3e, the time taken by LEDBAT to reach steady state (i.e. when estimated queuing delay is approximately equal to $target$) in the presence and absence of UDP in the network decreases as we increase the value of $gain$ from 5 to 240. Beyond $gain = 240$, the time-taken remains constant because the network resources such as bandwidth and link delay remain unchanged. Time to reach steady state when LEDBAT shares the bottleneck with UDP is less than when only LEDBAT is present in the network. This is due to reduced network resources by UDP traffic (i.e. available shared link bandwidth for LEDBAT is reduced from 1.5Mb/s to 750Kb/s). In (1), increasing $gain$ can increase the growth of LEDBAT congestion window during start-up phase, thus reducing the time-taken to reach steady state.

V. PROPOSED DYNAMIC GAIN ALGORITHM

A. Motivation

Our proposed extension to LEDBAT is motivated by the fact that high value of $gain$ in LEDBAT does not achieve a stable congestion window at steady state. This leads to large oscillations in queuing delay experienced by packets belonging to LEDBAT flow and other flows (especially real-time) that share the same bottleneck link with LEDBAT as shown in Fig. 3c and Fig. 3d. This is because the value of $gain$ is significant (i.e. a multiplier) in the equation of LEDBAT congestion window as shown in (1).

B. A Dynamic Gain Algorithm

In the LEDBAT source congestion control algorithm in Fig. 1, before the congestion window is calculated, we introduce a new calculation of gain when steady state has been reached:

$$gain = \frac{10^{\lceil \log_{10} \lfloor cwnd \rfloor \rceil}}{cwnd} \tag{2}$$

Equation (2) shows that the value of $gain$ will always be in the range of 1 to 9 for all values of $cwnd$. The reason for this is that we want the expression `off_target/cwnd` to be

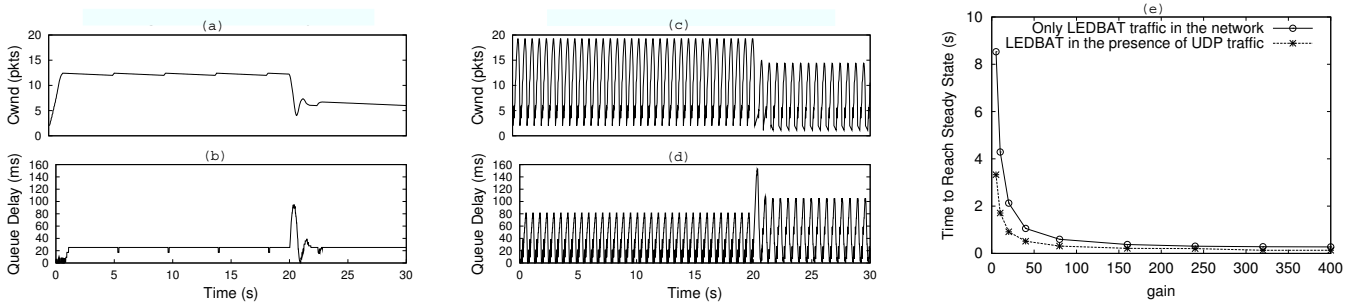

Fig. 3. (a) LEDBAT congestion window when $gain = 40$; (b) Queue delay of LEDBAT packets when $gain = 40$; (c) LEDBAT congestion window when $gain = 320$; (d) Queue delay of LEDBAT packets when $gain = 320$; (e) Time for LEDBAT source to reach steady state for different values of $gain$. In (a)-(d) a UDP source starts at time 20s.

multiplied by a value of $gain < 10$ (a one-digit multiplier) for all values of $cwnd$ in steady state. This will significantly reduce the high oscillation caused by high value of $gain$. Using (2) makes our algorithm scalable especially in high-speed networks because the power of 10 in the numerator increases as the number of digits of $cwnd$ increases.

C. Performance Results

After implementing the dynamic $gain$ algorithm in ns-2.34, we repeated the simulation experiments described in Section III. Selected results are reported here.

Comparing the results in Fig. 4a and 4b with Fig. 3c and 3d, our proposed extension to LEDBAT reduces queuing delay at the access router significantly from 80ms to stabilize at about 25ms after 5s unlike the unmodified LEDBAT in Fig. 3d that oscillates largely between 80ms and approximately 0ms. This is because LEDBAT with the proposed extension resets $gain$ to a dynamic value computed as shown in equation 2 as soon as it reaches steady state. As shown in Fig. 4a, LEDBAT congestion window is quickly increased from 2 packets to about 19 packets because $gain = 320$, thus inducing a maximum queuing delay of 80ms. As $T < 80ms$, LEDBAT congestion window is decreased with a value of $gain$ determined by $cwnd$ until after 6s where the congestion window and queuing delay stabilize at approximately 12 packets and 25ms respectively. LEDBAT with the proposed extension yields to UDP traffic just after 20s by reducing its congestion window until estimated queuing delay is less than $target$.

As shown in Fig. 4d, average congestion window of LEDBAT without dynamic value of $gain$ starts to increase significantly after $gain = 240$ from 12 packets to about 15 packets while it remains almost at 12 packets with our proposed extension as we increase $gain$. Fig. 4e shows that with our proposed extension, we obtain standard deviation that is less than 1 packet for all values of $gain$ while without the extension the standard deviation increases significantly after $gain = 80$ from 1 packet to 5 packets.

The increasing average and standard deviation of LEDBAT congestion window is responsible for the increasing average and standard deviation of queuing delay of LEDBAT packets without the proposed extension from 25ms to 50ms and from 0ms to 35ms respectively as we increase the value of $gain$ as shown in Fig. 4h and 4i. However, with the proposed extension, the average and standard deviation of queuing delay remain constant as we increase the value of $gain$ at about 27ms and 5ms respectively as given in the same figure.

Similar results are obtained when LEDBAT shares the bottleneck with UDP as given in Fig. 4f, 4g, 4j, and 4k. They only differ in the average congestion window (of ≈ 12 packets in Fig. 4d and ≈ 6 packets in Fig. 4f) of LEDBAT with the proposed extension. This is because only half of the bottleneck bandwidth is available for LEDBAT as UDP traffic arrives at the access router at a constant rate of 750Kb/s.

LEDBAT, without the proposed extension, achieves 100% and 50% link utilization until $gain = 160$ and 80 respectively, beyond which the average link utilizations decrease below 100% and 50% to $\approx 70\%$ and $\approx 45\%$ in the absence and presence of UDP traffic respectively as shown in Fig. 4c. However, in the case of LEDBAT with the proposed extension, average link utilizations remain at 100% and 50% in the absence and presence of UDP traffic respectively as we increase the value of $gain$ also shown in the same figure.

VI. Related Work

The first congestion collapse that hit the Internet over 3 decades ago motivated the work in [17], after which several other congestion control algorithms have been proposed. Related to LEDBAT are delay-based and low-priority congestion control algorithms. Jain first introduced a delay-based congestion control mechanism in [11]. In fact, none of the existing delay-based congestion control algorithms, even the famous TCP-Vegas [4], has been designed to minimize network delay to a defined value.

Due to the recent dominance of the Internet by non-interactive bulk data carrying traffic, several low-priority congestion control protocols have been developed to adjust their sending rate based on loss rate [5], delay [3], [18] and an inline network [15] measurements, adjusting the receiver advertised window at the application layer [2]. Further comparison of these low priority protocols can be found in [7].

The only work that has reported the performance evaluation of LEDBAT in [12] does not addressed the impact of different values of $gain$ on the performance.

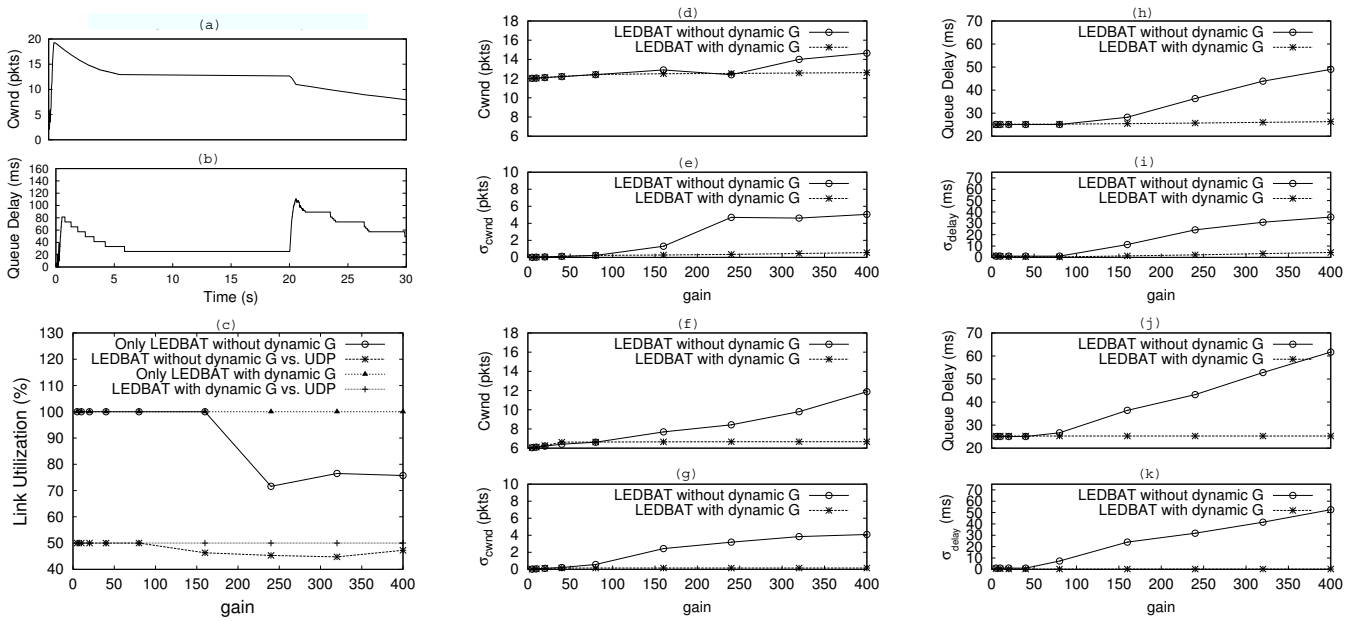

Fig. 4. (a) and (b): LEDBAT congestion window and queue delay when using dynamic gain with initial value 320 (UDP starts at 20s); (c) Bottleneck link utilization in the presence and absence of UDP and comparing fixed $gain$ and dynamic $gain$; (d) and (e): LEDBAT congestion window and its standard deviation in the absence of UDP; (f) and (g): LEDBAT congestion window and its standard deviation in the presence of UDP; (h) and (i): Queue delay and its standard deviation in the absence of UDP; (j) and (k): Queue delay and its standard deviation in the presence of UDP.

VII. Conclusion

We have analysed the rate at which the LEDBAT congestion window changes, both while only a single LEDBAT source is running, as well as when LEDBAT shares a bottleneck link with a real-time UDP application. Focussing on the congestion window $gain$, our results show that a LEDBAT source can quickly reach a steady state with a high $gain$ (where the steady state is when the $target$ queue delay is reached). However, once in steady state the high $gain$ leads to large oscillations in the sending rate, and large peaks in queue delay. Therefore we propose calculating a dynamic $gain$ once in steady state, where the $gain$ depends on the current congestion window size. Further analysis shows how our proposed dynamic $gain$ algorithm stabilises the sending rate, without compromising on the fast start-up, thereby still meeting the LEDBAT design goal of TCP-friendliness.

Further work is needed on understanding the performance of LEDBAT congestion control. In particular, the interaction between multiple LEDBAT flows with other TCP and UDP flows needs to be analysed, as fairness problems may arise between LEDBAT flows as well as between LEDBAT and non-LEDBAT flows. Also LEDBAT assumes the only variable component of the RTT is the queue delay at the bottleneck router. The impact of other variable components (e.g. due to re-routing, variable link delay) must be analysed before LEDBAT can be considered as a suitable congestion control mechanism in the Internet.

References

[1] F. Kelly. Mathematical modelling of the internet. In *Fourth International Congress on Industrial and Applied Mathematics*, July 1999.

[2] P. Key, L. Massoulié, and B. Wang. Emulating low-priority transport at the application layer: a background transfer service. In *ACM SIGMETRICS*, pages 118–129. ACM New York, NY, USA, 2004.

[3] A. Kuzmanovic and E. W. Knightly. TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking (TON)*, 14(4):752, 2006.

[4] L. S. Brakmo and L. L. Peterson. TCP Vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[5] S. Liu, M. Vojnovic, and D. Gunawardena. Competitive and considerate congestion control for bulk-data transfers. In *Proceedings of IEEE International Workshop on QoS, Evanston, IL, USA*, volume 9, 2007.

[6] Low Extra Delay Background Transport (LEDBAT) Working Group (WG) Charter. http://www.ietf.org/dyn/wg/charter/ledbat-charter.html.

[7] M. Welzl. A survey of lower-than-best effort transport protocols. IETF Internet Draft, (work-in-progress), March 2009.

[8] M. Allman, V. Paxson, and W. R. Stevens. TCP congestion control. RFC 2581, IETF, April 1999.

[9] ns-2. Network Simulator. http://www.isi.edu/nsnam.

[10] J. Postel. Transmission Control Protocol (TCP)-RFC 793, 1981.

[11] R. Jain. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communication Review*, 19(5), October 1989.

[12] D. Rossi, C. Testa, S. Valenti, P. Veglia, and L. Muscariello. News from the internet congestion control world. *CoRR*, abs/0908.0812, August 2009. http://arxiv.org/abs/0908.0812.

[13] S. Shalunov. Low extra delay background transport. In *IETF 75*, Stockholm, July 2009.

[14] S. Shalunov. Low Extra Delay Background Transport (LEDBAT). IETF Internet Draft, (work-in-progress), October 2009. http://tools.ietf.org/pdf/draft-ietf-ledbat-congestion-00.pdf.

[15] T. Tsugawa, G. Hasegawa, and M. Murata. Background TCP data transfer with inline network measurement. *IEICE Transactions on Communications*, 89(8):2152–2160, 2006.

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

[17] M. J. Karels, and V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review*, 18(4):314–329, 1988.

[18] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. *ACM SIGOPS Operating Systems Review*, 36(SI):343, 2002.