

LEDBAT WG
Internet-Draft
Intended status: Experimental
Expires: April 28, 2011

S. Shalunov
G. Hazel
BitTorrent Inc
J. Iyengar
Franklin and Marshall College
October 25, 2010

Low Extra Delay Background Transport (LEDBAT)
draft-ietf-ledbat-congestion-03.txt

Abstract

LEDBAT is an experimental delay-based congestion control algorithm that attempts to utilize the available bandwidth on an end-to-end path while limiting the consequent increase in queueing delay on the path. LEDBAT uses changes in one-way delay measurements to limit congestion induced in the network by the LEDBAT flow. LEDBAT is designed largely for use by background bulk-transfer applications; it is designed to be no more aggressive than TCP congestion control and yields in the presence of competing TCP flows, thus reducing interference with the network performance of the competing flows.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 28, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements notation	3
2. Introduction	3
2.1. Design Goals	3
2.2. Applicability	3
3. LEDBAT Congestion Control	4
3.1. Overview	4
3.2. Preliminaries	4
3.3. Receiver-Side Operation	5
3.4. Sender-Side Operation	5
3.4.1. An Overview	5
3.4.2. The Complete Sender Algorithm	5
3.5. Parameter Values	6
4. Understanding LEDBAT Mechanisms	7
4.1. Delay Estimation	7
4.1.1. Estimating Base Delay	7
4.1.2. Estimating Queueing Delay	8
4.2. Managing the Congestion Window	8
4.2.1. Window Increase: Probing For More Bandwidth	8
4.2.2. Window Decrease: Responding To Congestion	8
5. Choosing Parameter Values	9
5.1. Queueing Delay Target	9
6. Discussion	9
6.1. Framing Considerations	9
6.2. Competing With TCP Flows	10
6.3. Fairness Among LEDBAT Flows	10
7. IANA Considerations	12
8. Security Considerations	12
9. Normative References	12
Appendix A. Timestamp errors	12
A.1. Clock offset	12
A.2. Clock skew	13
A.2.1. Deployed clock skew correction mechanism	13
A.2.2. Skew correction with faster virtual clock	14
A.2.3. Skew correction with estimating drift	14
Authors' Addresses	15

1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

TCP congestion control [RFC2581], the predominant congestion control mechanism used on the Internet, aims to share bandwidth at a bottleneck link equitably among flows competing at the bottleneck. While TCP works well for many applications, applications such as software updates or file-sharing applications prefer to use bandwidth available in the network without interfering with the network performance of other interactive applications. Such "background" traffic can yield bandwidth to TCP-based "foreground" traffic by reacting earlier than TCP to congestion signals.

LEDBAT is an experimental delay-based congestion control mechanism that allows background applications, such as peer-to-peer applications, that send large amounts of data particularly over links with deep buffers, such as residential uplinks, to operate in the background, without interfering with performance of interactive applications. LEDBAT uses one-way delay measurements to determine congestion on the data path, and keeps latency across the tight link in the end-to-end path low while attempting to utilize the available bandwidth on the end-to-end path.

2.1. Design Goals

As a "scavenger" mechanism for the Internet, LEDBAT's design goals are to:

1. Keep delay low when no other traffic is present
2. Add little to the queuing delays induced by TCP traffic
3. Quickly yield to traffic sharing the same bottleneck queue that uses standard TCP congestion control
4. Utilize end-to-end available bandwidth
5. Operate well in networks with FIFO queuing with drop-tail discipline

2.2. Applicability

LEDBAT is a "scavenger" congestion control mechanism---a LEDBAT flow attempts to utilize all available bandwidth and yields quickly to a competing TCP flow---and is primarily motivated by background bulk-transfer applications, such as peer-to-peer file transfers and software updates. It can be used for any application that needs to

run in the "background", to reduce the application's impact on the network and on other interactive network applications.

LEDBAT can be used with any transport protocol capable of carrying timestamps and acknowledging data frequently--LEDBAT can be easily used with TCP, SCTP, and DCCP.

3. LEDBAT Congestion Control

3.1. Overview

A TCP sender increases its congestion window until a loss occurs, which, in the absence of any Active Queue Management (AQM) in the network, occurs only when the queue at the bottleneck link on the end-to-end path overflows. Since packet loss at the bottleneck link is often preceded by an increase in the queueing delay at the bottleneck link, LEDBAT congestion control uses this increase in queueing delay as an early signal of congestion, enabling it to respond to congestion earlier than TCP, and enabling it to yield bandwidth to a competing TCP flow.

LEDBAT employs one-way delay measurements to estimate queueing delay. When the estimated queueing delay is lesser than a pre-determined target, LEDBAT infers that the network is not yet congested, and increases its sending rate to utilize any spare capacity in the network. When the estimated queueing delay becomes greater than a pre-determined target, LEDBAT decreases its sending rate quickly as a response to potential congestion in the network.

3.2. Preliminaries

For the purposes of explaining LEDBAT, we assume a transport sender that uses fixed-size segments and a receiver that acknowledges each segment separately. It is straightforward to apply the mechanisms described here with variable-sized segments and with delayed acknowledgments. A LEDBAT sender uses a congestion window (cwnd) that gates the amount of data that the sender can send into the network in one RTT.

LEDBAT requires that each data segment carries a "timestamp" from the sender, based on which the receiver computes the one-way delay from the sender, and sends this computed value back to the sender.

In addition to the LEDBAT mechanism described below, we note that a slow start mechanism can be used as specified in [RFC2581]. Since slow start leads to faster increase in the window than that specified in LEDBAT, conservative congestion control implementations employing

LEDBAT may skip slow start altogether and start with an initial window of XXX MSS.

3.3. Receiver-Side Operation

A LEDBAT receiver operates as follows:

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

3.4. Sender-Side Operation

3.4.1. An Overview

As a first approximation, a LEDBAT sender operates as show below. TARGET is the maximum queueing delay that LEDBAT itself can introduce in the network, and GAIN determines the rate at which the congestion window changes; both constants are specified later.

```
on initialization:
    base_delay = +infinity

on acknowledgement:
    current_delay = acknowledgement.delay
    base_delay = min(base_delay, current_delay)
    queuing_delay = current_delay - base_delay
    off_target = TARGET - queuing_delay
    cwnd += GAIN * off_target / cwnd
```

3.4.2. The Complete Sender Algorithm

The simplified mechanism above ignores noise filtering and base expiration. The full sender-side algorithm is specified below:

```
on initialization:
    set all NOISE_FILTER delays used by current_delay() to +infinity
    set all BASE_HISTORY delays used by base_delay() to +infinity
    last_rollover = -infinity # More than a minute in the past.

on acknowledgement:
    delay = acknowledgement.delay
    update_base_delay(delay)
    update_current_delay(delay)
    queuing_delay = current_delay() - base_delay()
    off_target = TARGET - queuing_delay + random_input()
    cwnd += GAIN * off_target / cwnd
    # flight_size() is the amount of currently not acked data.
    max_allowed_cwnd = ALLOWED_INCREASE + TETHER*flight_size()
    cwnd = min(cwnd, max_allowed_cwnd)

random_input()
    # random() is a PRNG between 0.0 and 1.0
    # NB: RANDOMNESS_AMOUNT is normally 0
    RANDOMNESS_AMOUNT * TARGET * ((random() - 0.5)*2)

update_current_delay(delay)
    # Maintain a list of NOISE_FILTER last delays observed.
    forget the earliest of NOISE_FILTER current_delays
    add delay to the end of current_delays

current_delay()
    min(the NOISE_FILTER delays stored by update_current_delay)

update_base_delay(delay)
    # Maintain BASE_HISTORY min delays. Each represents a minute.
    if round_to_minute(now) != round_to_minute(last_rollover)
        last_rollover = now
        forget the earliest of base delays
        add delay to the end of base_delays
    else
        last of base_delays = min(last of base_delays, delay)

base_delay()
    min(the BASE_HISTORY min delays stored by update_base_delay)
```

3.5. Parameter Values

TARGET parameter MUST be set to 100 milliseconds. GAIN SHOULD be set to 1 so that max rampup rate is the same as for TCP. BASE_HISTORY SHOULD be 10; it MUST be no less than 2 and SHOULD NOT be more than 20. NOISE_FILTER SHOULD be 1; it MAY be tuned so that it is at least 1 and no more than cwnd/2. ALLOWED_INCREASE SHOULD be 1 packet; it

MUST be at least 1 packet and SHOULD NOT be more than 3 packets. TETHER SHOULD be 1.5; it MUST be greater than 1. RANDOMMESS_AMOUNT SHOULD be 0; it MUST be between 0 and 0.1 inclusively.

Note that using the same TARGET value across LEDBAT flows is important, since flows using different TARGET values will not share a bottleneck equitably---flows with higher values will get a larger share of the bottleneck bandwidth.

4. Understanding LEDBAT Mechanisms

This section describes and provides insight into the delay estimation and window management mechanisms used in LEDBAT congestion control.

4.1. Delay Estimation

LEDBAT estimates congestion in the network based on observed increase in queueing delay in the network. To observe an increase in the queueing delay in the network, LEDBAT must separate the queueing delay component from the rest of the end-to-end delay. This section explains how LEDBAT decomposes the observed changes in end-to-end delay into these two components.

LEDBAT estimates congestion in the direction of data flow. To avoid measuring queue build-up on the reverse path (or ack path), LEDBAT uses changes in one-way delay estimates. The extant One-Way Active Measurement Protocol (OWAMP) [XXXcite], can be used for measuring one-way delay, but since LEDBAT is used for sending data, and since LEDBAT requires only changes in one-way delay to infer congestion, simply adding a timestamp to the data segments and a measurement result field in the ack packets seems sufficient. Doing so also avoids the pitfall of measurement packets being treated differently from the data packets in the network.

4.1.1. Estimating Base Delay

End-to-end delay can be decomposed into transmission (or serialization) delay, propagation (or speed-of-light) delay, queueing delay, and processing delay. On any given path, barring some noise, all delay components except for queueing delay are constant; over time, we expect only the queueing delay on the path to change as the queue sizes at the links change. Since queuing delay is always additive to the end-to-end delay, we estimate the sum of the constant delay components, which we call "base delay", to be the minimum delay observed on the end-to-end path. Using the minimum observed delay also allows LEDBAT to eliminate noise in the delay estimation, such as due to spikes in processing delay at a node on the path.

To respond to true changes in the base delay due to route changes, LEDBAT uses only "recent" measurements---measurements over the last N minutes---in estimating the base delay. To implement an approximate minimum over the last N minutes, a LEDBAT sender stores N+1 separate minima---N for the last N minutes, and one for the running current minute. At the end of the current minute, the window moves---the earliest minimum is dropped and the latest minimum is added. When the connection is idle for a given minute, no data is available for the one-way delay and, therefore, no minimum is stored. When the connection has been idle for N minutes, the measurement begins anew.

The duration of the observation window itself is a tradeoff between robustness of measurement and responsiveness to change: a larger observation window yields a more accurate base delay if the true base delay is unchanged, whereas a smaller observation window results in faster response to true changes in the base delay.

4.1.2. Estimating Queueing Delay

Given that the base delay is constant, the queueing delay is represented by the variable component of the measured end-to-end delay. We measure queueing delay as simply the difference between an end-to-end delay measurement and the current estimate of base delay.

4.2. Managing the Congestion Window

4.2.1. Window Increase: Probing For More Bandwidth

A LEDBAT sender increases its congestion window most when the queuing delay estimate is zero. To be friendly to competing TCP flows, we set this highest rate of window growth to be the same as TCP's. In other words, the LEDBAT window grows by at most twice per round-trip time. Since queuing delay estimate is always non-negative, this growth rate ensures that a LEDBAT flow never ramps up faster than a competing TCP flow over the same path.

4.2.2. Window Decrease: Responding To Congestion

When the sender's queuing delay estimate is lower than the target, the sending rate should be increased. When the sender's queuing delay estimate is higher than the target, the sending rate should be reduced. LEDBAT uses a simple linear controller to determine sending rate as a function of the delay estimate, where the response is proportional to the difference between the current queueing delay estimate and the target. In limited experiments with Bittorrent nodes, this controller seems to work well.

To deal with severe congestion when several packets are dropped in

the network, and to provide a fallback against incorrect queuing delay estimates, a LEDBAT sender halves its cwnd when a loss event is detected. As with NewReno, LEDBAT reduces its cwnd by half at most once per RTT. Note that, unlike TCP-like loss-based congestion control, LEDBAT does not induce losses and so it normally does not rely on losses to determine the sending rate. LEDBAT's reaction to loss is thus less important than it is in the case of loss-based congestion control. For LEDBAT, reducing the congestion window on loss is a fallback mechanism in case of severe congestion and in the case of incorrect delay estimates.

5. Choosing Parameter Values

Through this discussion, we hope to encourage informed experimentation with LEDBAT.

5.1. Queuing Delay Target

Consider the queuing delay on the bottleneck. This delay is the extra delay induced by congestion control. One of our design goals is to keep this delay low. However, when this delay is zero, the queue is empty and so no data is being transmitted and the link is thus not saturated. Hence, our design goal is to keep the queuing delay low, but non-zero.

How low do we want the queuing delay to be? Because another design goal is to be deployable on networks with only simple FIFO queuing and drop-tail discipline, we can't rely on explicit signaling for the queuing delay. So we're going to estimate it using external measurements. The external measurements will have an error at least on the order of best-case scheduling delays in the OSes. There's thus a good reason to try to make the queuing delay larger than this error. There's no reason that would want us to push the delay much further up. Thus, we will have a delay target that we would want to maintain.

6. Discussion

6.1. Framing Considerations

The actual framing and wire format of the protocol(s) using the LEDBAT congestion control mechanism is outside of scope of this document, which only describes the congestion control part.

There is an implication of the need to use one-way delay from the sender to the receiver in the sender. An obvious way to support this

is to use a framing that timestamps packets at the sender and conveys the measured one-way delay back to the sender in ack packets. This is the method we'll keep in mind for the purposes of exposition. Other methods are possible and valid.

Another implication is the receipt of frequent ACK packets. The exposition below assumes one ACK per data packet, but any reasonably small number of data packets per ACK will work as long as there is at least one ACK every round-trip time.

The protocols to which this congestion control mechanism is applicable, with possible appropriate extensions, are TCP, SCTP, DCCP, etc. It is not a goal of this document to cover such applications. The mechanism can also be used with proprietary transport protocols, e.g., those built over UDP for P2P applications.

6.2. Competing With TCP Flows

Consider competition between a LEDBAT connection and a connection governed by loss-based congestion control (on a FIFO bottleneck with drop-tail discipline). Loss-based connection will need to experience loss to back off. Loss will only occur after the connection experiences maximum possible delays. LEDBAT will thus receive congestion indication sooner than the loss-based connection. If LEDBAT can ramp down faster than the loss-based connection ramps up, LEDBAT will yield. LEDBAT ramps down when queuing delay estimate exceeds the target: the more the excess, the faster the ramp-down. When the loss-based connection is standard TCP, LEDBAT will yield at precisely the same rate as TCP is ramping up when the queuing delay is double the target.

LEDBAT is most aggressive when its queuing delay estimate is most wrong and is as low as it can be. Queuing delay estimate is nonnegative, therefore the worst possible case is when somehow the estimate is always returned as zero. In this case, LEDBAT will ramp up as fast as TCP and halve the rate on loss. Thus, in case of worst possible failure of estimates, LEDBAT will behave identically to TCP. This provides an extra safety net.

6.3. Fairness Among LEDBAT Flows

The design goals of LEDBAT center around the aggregate behavior of LEDBAT flows when they compete with standard TCP. It is also interesting how LEDBAT flows share bottleneck bandwidth when they only compete between themselves.

LEDBAT as described so far lacks a mechanism specifically designed to equalize utilization between these flows. The observed behavior of

existing implementations indicates that a rough equalization, in fact, does occur.

The delay measurements used as control inputs by LEDBAT contain some amount of noise and errors. The linear controller converts this input noise into the same amount of output noise. The effect that this has is that the uncorrelated component of the noise between flows serves to randomly shuffle some amount of bandwidth between flows. The amount shuffled during each RTT is proportional to the noise divided by the target delay. The random-walk trajectory of bandwidth utilized by each of the flows over time tends to the fair share. The timescales on which the rates become comparable are proportional to the target delay multiplied by the RTT and divided by the noise.

In complex real-life systems, the main concern is usually the reduction of the amount of noise, which is copious if not eliminated. In some circumstances, however, the measurements might be "too good" -- since the equalization timescale is inversely proportional to noise, perfect measurements would result in lack of convergence.

Under these circumstances, it may be beneficial to introduce some artificial randomness into the inputs (or, equivalently, outputs) of the controller. Note that most systems should not require this and should be primarily concerned with reducing, not adding, noise.

With delay-based congestion control systems, there's a concern about the ability of late comers to measure the base delay correctly. Suppose a LEDBAT flow saturates a bottleneck; another LEDBAT flow starts and proceeds to measure the base delay and the current delay and to estimate the queuing delay. If the bottleneck always contains target delay worth of packets, the second flow would see the bottleneck as empty start building a second target delay worth of queue on top of the existing queue. The concern ("late comers' advantage") is that the initial flow would now back off because it sees the real delay and the late comer would use the whole capacity.

However, once the initial flow yields, the late comer immediately measures the true base delay and the two flows operate from the same (correct) inputs.

Additionally, in practice this concern is further alleviated by the burstiness of network traffic: all that's needed to measure the base delay is one small gap. These gaps can occur for a variety of reasons: the OS may delay the scheduling of the sending process until a time slice ends, the sending computer might be unusually busy for some number of milliseconds or tens of milliseconds, etc. If such a gap occurs while the late comer is starting, base delay is

immediately correctly measured. With small number of flows, this appears to be the main mechanism of regulating the late comers' advantage.

7. IANA Considerations

There are no IANA considerations for this document.

8. Security Considerations

A network on the path might choose to cause higher delay measurements than the real queuing delay so that LEDBAT backs off even when there's no congestion present. Shaping of traffic into an artificially narrow bottleneck can't be counteracted, but faking timestamp field can and SHOULD. A protocol using the LEDBAT congestion control SHOULD authenticate the timestamp and delay fields, preferably as part of authenticating most of the rest of the packet, with the exception of volatile header fields. The choice of the authentication mechanism that resists man-in-the-middle attacks is outside of scope of this document.

9. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

Appendix A. Timestamp errors

One-way delay measurement needs to deal with timestamp errors. We'll use the same locally linear clock model and the same terminology as Network Time Protocol (NTP). This model is valid for any differentiable clocks. NTP uses the term "offset" to refer to difference from true time and "skew" to refer to difference of clock rate from the true rate. The clock will thus have a fixed offset from the true time and a skew. We'll consider what we need to do about the offset and the skew separately.

A.1. Clock offset

First, consider the case of zero skew. The offset of each of the two clocks shows up as a fixed error in one-way delay measurement. The

difference of the offsets is the absolute error of the one-way delay estimate. We won't use this estimate directly, however. We'll use the difference between that and a base delay. Because the error (difference of clock offsets) is the same for the current and base delay, it cancels from the queuing delay estimate, which is what we'll use. Clock offset is thus irrelevant to the design.

A.2. Clock skew

Now consider the skew. For a given clock, skew manifests in a linearly changing error in the time estimate. For a given pair of clocks, the difference in skews is the skew of the one-way delay estimate. Unlike the offset, this no longer cancels in the computation of the queuing delay estimate. On the other hand, while the offset could be huge, with some clocks off by minutes or even hours or more, the skew is typically not too bad. For example, NTP is designed to work with most clocks, yet it gives up when the skew is more than 500 parts per million (PPM). Typical skews of clocks that have never been trained seem to often be around 100-200 PPM. Previously trained clocks could have 10-20 PPM skew due to temperature changes. A 100-PPM skew means accumulating 6 milliseconds of error per minute. The expiration of base delay related to route changes mostly takes care of clock skew. A technique to specifically compute and cancel it is trivially possible and involves tracking base delay skew over a number of minutes and then correcting for it, but usually isn't necessary, unless the target is unusually low, the skew is unusually high, or the base interval is unusually long. It is not further described in this document.

For cases when the base interval is long or the skew is high or the target is low, a technique to correct for skew can be beneficial. The technique described here or a different mechanism MAY be used by implementations. The technique described is still experimental, but it is actually currently used. The pseudocode in the specification below does not include any of the skew correction algorithms.

A.2.1. Deployed clock skew correction mechanism

Clock skew can be in two directions: either the sender's clock is faster than the receiver's, or vice versa. We refer to the former situation as clock drift "in sender's favor" and to the latter as clock drift "in receiver's favor".

When the clock drift is "in sender's favor", nothing special needs to be done, because the timestamp differences (i.e., the raw delay estimates) will grow smaller with time, and thus the base delay will be continuously updated with the drift.

When the clock drift is "in receiver's favor", the raw delay estimates will drift up with time, suppressing the throughput needlessly. This is the case that can benefit from a special mechanism. Assume symmetrical framing (i.e., same information about delays transmitted in both way). If the sender can detect the receiver reducing its base delay, it can infer that this is due to clock drift "in receiver's favor". This can be compensated for by increasing the sender's base delay by the same amount. Since, in our implementation, the receiver sends back the raw timestamp estimate, the sender can run the same base delay calculation algorithm it runs for itself for the receiver as well; when it reduces the inferred receiver's delay, it increases its own by the same amount.

A.2.2. Skew correction with faster virtual clock

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

Since having a faster clock on the sender is, relatively speaking, a non-problem, one can use two different virtual clocks on each LEDBAT implementation: use, for example, the default machine clock for situations where the instance is acting as a receiver, and use a virtual clock, easily computed from the default machine clock through a linear transformation, for situations where the instance is acting as a sender. Make the virtual clock, e.g., 500 PPM faster than the machine clock. Since 500 PPM is more than the variability of most clocks (plus or minus 100 PPM), any sender's clock is very likely to be faster than any receiver's clock, thus benefitting from the implicit correction of taking the minimum as the base delay.

Note that this approach is not compatible with the one described in the preceding section.

A.2.3. Skew correction with estimating drift

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

The history of base delay minima we already keep for each minute provides us with direct means of computing the clock skew difference between the two hosts. Namely, we can fit a linear function to the set of base delay estimates for each minute. The slope of the function is an estimate of the clock skew difference for the given pair of sender and receiver. Once the clock skew difference is estimated, it can be used to correct the clocks so that they advance at nearly the same rate. Namely, the clock needs to be corrected by half of the estimated skew amount, since the other half will be corrected by the other endpoint. Note that the skew differences are

then maintained for each connection and the virtual clocks used with each connection can differ, since they do not attempt to estimate the skew with respect to the true time, but instead with respect to the other endpoint.

A.2.3.1. Byzantine skew correction

This is an alternative skew correction algorithm, currently under consideration and not deployed in the wild.

When it is known that each host maintains long-lived connections to a number of different other hosts, a byzantine scheme can be used to estimate the skew with respect to the true time. Namely, calculate the skew difference for each of the peer hosts as described in the preceding section, then take the median of the skew differences.

This inherent clock drift can then be corrected with a linear transformation before the clock data is used in the algorithm from the preceding section, the currently deployed algorithm, or nearly any other skew correction algorithm.

While this scheme is not universally applicable, it combines well with other schemes, since it is essentially a clock training mechanism. The scheme also acts the fastest, since the state is preserved between connections.

Authors' Addresses

Stanislav Shalunov
BitTorrent Inc
612 Howard St, Suite 400
San Francisco, CA 94105
USA

Email: shalunov@bittorrent.com
URI: <http://shlang.com>

Greg Hazel
BitTorrent Inc
612 Howard St, Suite 400
San Francisco, CA 94105
USA

Email: greg@bittorrent.com

Janardhan Iyengar
Franklin and Marshall College
415 Harrisburg Ave.
Lancaster, PA 17603
USA

Email: jiyengar@fandm.edu

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 12, 2011

M. Welzl
University of Oslo
D. Ros
Telecom Bretagne
October 9, 2010

A Survey of Lower-than-Best-Effort Transport Protocols
draft-ietf-ledbat-survey-01.txt

Abstract

This document provides a survey of transport protocols which are designed to have a smaller bandwidth and/or delay impact on standard TCP than standard TCP itself when they share a bottleneck with it. Such protocols could be used for low-priority "background" traffic, as they provide what is sometimes called a "less than" (or "lower than") best-effort service.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 12, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Delay-based transport protocols	3
2.1. Accuracy of delay-based congestion predictors	6
2.2. Delay-based congestion control = LBE?	7
3. Non-delay-based transport protocols	7
4. Application layer approaches	8
5. Orthogonal work	9
6. Acknowledgements	11
7. IANA Considerations	11
8. Security Considerations	11
9. Informative References	11
Authors' Addresses	16

1. Introduction

This document presents a brief survey of proposals to attain a Less than Best Effort (LBE) service without help from routers. We loosely define a LBE service as a service which results in smaller bandwidth and/or delay impact on standard TCP than standard TCP itself when sharing a bottleneck with it. We refer to systems that provide this service as Less than Best Effort (LBE) systems. Generally, LBE behavior can be achieved by reacting to queue growth earlier than standard TCP would, or by changing the congestion avoidance behavior of TCP without utilizing any additional implicit feedback. Some mechanisms achieve a LBE behavior at the application layer, e.g. by changing the receiver window of standard TCP, and there is also a substantial amount of work that is related to the LBE concept but not presenting a solution that can be installed in end hosts or expected to work over the Internet. According to this classification, solutions have been categorized in this document as delay-based transport protocols, non-delay-based transport protocols, application layer approaches and orthogonal work.

2. Delay-based transport protocols

It is wrong to generally equate "little impact on standard TCP" with "small sending rate". Unless the sender's maximum window is limited for some reason, and in the absence of ECN support, standard TCP will normally increase its rate until a queue overflows, causing one or more packets to be dropped and the rate to be reduced. A protocol which stops increasing the rate before this event happens can, in principle, achieve a better performance than standard TCP. In the absence of any other traffic, this is even true for TCP itself when its maximum send window is limited to the bandwidth*round-trip time (RTT) product.

TCP Vegas [Bra+94] is one of the first protocols that was known to have a smaller sending rate than standard TCP when both protocols share a bottleneck [Kur+00] -- yet it was designed to achieve more, not less throughput than standard TCP. Indeed, when it is the only protocol on the bottleneck, the throughput of TCP Vegas is greater than the throughput of standard TCP. Depending on the bottleneck queue length, TCP Vegas itself can be starved by standard TCP flows. This can be remedied to some degree by the RED Active Queue Management mechanism [RFC2309].

The congestion avoidance behavior is the protocol's most important feature in terms of historical relevance as well as relevance in the context of this document (it has been shown that other elements of the protocol can sometimes play a greater role for its overall

behavior [Hen+00]). In Congestion Avoidance, once per RTT, TCP Vegas calculates the expected throughput as $\text{WindowSize} / \text{BaseRTT}$, where WindowSize is the current congestion window and BaseRTT is the minimum of all measured RTTs. The expected throughput is then compared with the actual (measured) throughput. If the actual throughput is smaller than the expected throughput minus a threshold β , this is taken as a sign of congestion, causing the protocol to linearly decrease its rate. If the actual throughput is greater than the expected throughput minus a threshold α (with $\alpha < \beta$), this is taken as a sign that the network is underutilized, causing the protocol to linearly increase its rate.

TCP Vegas has been analyzed extensively. One of the most prominent properties of TCP Vegas is its fairness between multiple flows of the same kind, which does not penalize flows with large propagation delays in the same way as standard TCP. While it was not the first protocol that uses delay as a congestion indication, its predecessors (which can be found in [Bra+94]) are not discussed here because of the historical "landmark" role that TCP Vegas has taken in the literature.

Transport protocols which were designed to be non-intrusive include TCP-LP [Kuz+06] and TCP Nice [Ven+02]. Using a simple analytical model, the authors of [Kuz+06] illustrate the feasibility of this endeavor by showing that, due to the non-linear relationship between throughput and RTT, it is possible to remain transparent to standard TCP even when the flows under consideration have a larger RTT than standard TCP flows.

TCP Nice [Ven+02] follows the same basic approach as TCP Vegas but improves upon it in some aspects. Because of its moderate linear-decrease congestion response, TCP Vegas can affect standard TCP despite its ability to detect congestion early. TCP Nice removes this issue by halving the congestion window (at most once per RTT, like standard TCP) instead of linearly reducing it. To avoid being too conservative, this is only done if a fixed predefined fraction of delay-based incipient congestion signals appears within one RTT. Otherwise, TCP Nice falls back to the congestion avoidance rules of TCP Vegas if no packet was lost or standard TCP if a packet was lost. One more feature of TCP Nice is its ability to support a congestion window of less than one packet, by clocking out single packets over more than one RTT. With ns-2 simulations and real-life experiments using a Linux implementation, the authors of [Ven+02] show that TCP Nice achieves its goal of efficiently utilizing spare capacity while being non-intrusive to standard TCP.

Other than TCP Vegas and TCP Nice, TCP-LP uses only the one-way delay (OWD) instead of the RTT as an indicator of incipient congestion.

This is done to avoid reacting to delay fluctuations that are caused by reverse cross-traffic. Using the TCP Timestamps option [RFC1323], the OWD is determined as the difference between the receiver's Timestamp value in the ACK and the original Timestamp value that the receiver copied into the ACK. While the result of this subtraction can only precisely represent the OWD if clocks are synchronized, its absolute value is of no concern to TCP-LP and hence clock synchronization is unnecessary. Using a constant smoothing parameter, TCP-LP calculates an Exponentially Weighted Moving Average (EWMA) of the measured OWD and checks whether the result exceeds a threshold within the range of the minimum and maximum OWD that was seen during the connections's lifetime; if it does, this condition is interpreted as an "early congestion indication". The minimum and maximum OWD values are initialized during the slow-start phase.

Regarding its reaction to an early congestion indication, TCP-LP tries to strike a middle ground between the overly conservative choice of immediately setting the congestion window to one packet and the presumably too aggressive choice of halving the congestion window like standard TCP. It does so by halving the window at first in response to an early congestion indication, then initializing an "interference time-out timer", and maintaining the window size until this timer fires. If another early congestion indication appeared during this "interference phase", the window is then set to 1; otherwise, the window is maintained and TCP-LP continues to increase it the standard Additive-Increase fashion. This method ensures that it takes at least two RTTs for a TCP-LP flow to decrease its window to 1, and, like standard TCP, TCP-LP reacts to congestion at most once per RTT.

With ns-2 simulations and real-life experiments using a Linux implementation, the authors of [Kuz+06] show that TCP-LP is largely non-intrusive to TCP traffic while at the same time enabling it to utilize a large portion of the excess network bandwidth, which is fairly shared among competing TCP-LP flows. They also show that using their protocol for bulk data transfers greatly reduces file transfer times of competing best-effort web traffic.

Sync-TCP [Wei+05] follows a similar approach as TCP-LP, by adapting its reaction to congestion according to changes in the OWD. By comparing the estimated (average) forward queuing delay to the maximum observed delay, Sync-TCP adapts the AIMD parameters depending on the trend followed by the average delay over an observation window. Even though the authors of [Wei+05] did not explicitly consider its use as an LBE protocol, Sync-TCP was designed to react early to incipient congestion, while grabbing available bandwidth more aggressively than a standard TCP in congestion-avoidance mode.

Delay-based congestion control is also at the basis of proposals aiming at adapting TCP's congestion avoidance to very high-speed networks. Some of these proposals [Tan+06][Sri+08][Liu+08] are hybrid loss- and delay-based mechanisms, whereas others [Dev+03][Wei+06][Cha+10] are variants of Vegas based solely on delays.

2.1. Accuracy of delay-based congestion predictors

The accuracy of delay-based congestion predictors has been the subject of a good deal of research, see e.g. [Bia+03], [Mar+03], [Pra+04], [Rew+06], [McC+08]. The main result of most of these studies is that delays (or, more precisely, round-trip times) are, in general, weakly correlated with congestion. There are several factors that may induce such a poor correlation:

- o Bottleneck buffer size: in principle, a delay-based mechanism could be made "more than TCP friendly" *_if_* buffers are "large enough", so that RTT fluctuations and/or deviations from the minimum RTT can be detected by the end-host with reasonable accuracy. Otherwise, it may be hard to distinguish real delay variations from measurement noise.
- o RTT measurement issues: in principle, RTT samples may suffer from poor resolution, due to timers which are too coarse-grained with respect to the scale of delay fluctuations. Also, a flow may obtain a very noisy estimate of RTTs due to undersampling, under some circumstances (e.g., the flow rate is much lower than the link bandwidth). For TCP, other potential sources of measurement noise include: TCP segmentation offloading (TSO) and the use of delayed ACKs [Hay10].
- o Level of statistical multiplexing and RTT sampling: it may be easy for an individual flow to "miss" loss/queue overflow events, especially if the number of flows sharing a bottleneck buffer is significant. This is nicely illustrated e.g. in Fig. 2 of [McC+08].
- o Impact of wireless links: several mechanisms that are typical of wireless links, like link-layer scheduling and error recovery, may induce strong delay fluctuations over short time scales [Gur+04].

Whether a delay-based protocol behaves in its intended manner (e.g., it is "more than TCP friendly", or it grabs available bandwidth in a very aggressive manner) may therefore depend on the accuracy issues listed above. Moreover, protocols like Vegas need to keep an estimate of the minimum ("base") delay; this makes such protocols highly sensitive to eventual changes in the end-to-end route during

the lifetime of the flow [Mo+99].

TODO: incorporate [Bha+07] and any references therein that may be missing.

2.2. Delay-based congestion control = LBE?

Regarding the issue of false positives/false negatives with a delay-based congestion detector, most studies focus on the loss of throughput coming from the erroneous detection of queue build-up and of alleviation of congestion. Arguably, for a LBE transport protocol it's better to err on the "more-than-TCP-friendly side", that is, to always yield to perceived congestion whether it is "real" or not; however, failure to detect congestion (due to one of the above accuracy problems) would result in behavior that is not LBE. For instance, consider the case in which the bottleneck buffer is small, so that the contribution of queueing delay at the bottleneck to the global end-to-end delay is small. In such a case, a flow using a delay-based mechanism might end up consuming a good deal of bandwidth with respect to a competing standard TCP flow, unless it also incorporates a suitable reaction to loss.

Consider also the case in which the bottleneck link is already (very) congested. In such a scenario, delay variations may be quite small, hence, it may be very difficult to tell an empty queue from a heavily-loaded queue, in terms of delay fluctuation. Therefore, a newly-arriving delay-based flow may start sending faster when there is already heavy congestion, eventually driving away loss-based flows [Sha+05].

3. Non-delay-based transport protocols

4CP [Liu+07], which stands for "Competitive and Considerate Congestion Control", is a protocol which provides a LBE service by changing the window control rules of standard TCP. A "virtual window" is maintained, which, during a so-called "bad congestion phase" is reduced to less than a predefined minimum value of the actual congestion window. The congestion window is only increased again once the virtual window exceeds this minimum, and in this way the virtual window controls the duration during which the sender transmits with a fixed minimum rate. The 4CP congestion avoidance algorithm allows for setting a target average window and avoids starvation of "background" flows while bounding the impact on "foreground" flows. Its performance was evaluated in ns-2 simulations and in real-life experiments with a kernel-level implementation in Microsoft Windows Vista.

Some work was done on applying weights to congestion control mechanisms, allowing a flow to be as aggressive as a number of parallel TCP flows at the same time. This is usually motivated by the fact that users may want to assign different priorities to different flows. The first, and best known, such protocol is MultTCP [Cro+98], which emulates N TCPs in a rather simple fashion. Improved versions of the parallel-TCP idea are presented in [Hac+04] and [Hac+08], and there is also a variant where only one feedback loop is applied to control a larger traffic aggregate by the name of Probe-Aided (PA-)MultTCP [Kuo+08]. Another protocol, CP [Ott+04], applies the same concept to the TFRC protocol [RFC5348] in order to provide such fairness differentiation for multimedia flows.

The general assumption underlying all of the above work is that these protocols are "N-TCP-friendly", i.e. they are as TCP-friendly as N TCPs, where N is a positive (and possibly natural) number which is greater than or equal to 1. The MultFRC [Dam+09] protocol, another extension of TFRC for multiple flows, is however able to support values between 0 and 1, making it applicable as a mechanism for a LBE service. Since it does not react to delay like the mechanisms above but adjusts its rate like TFRC, it can probably be expected to be more aggressive than mechanisms such as TCP Nice or TCP-LP. This also means that MultFRC is less likely to be prone to starvation, as its aggression is tunable at a fine granularity even when N is between 0 and 1.

4. Application layer approaches

A simplistic, application-level approach to a background transport service may just consist in scheduling automated transfers at times when the network is lightly loaded, as described in e.g. [Dyk+02]. An issue with such a technique is that it may not necessarily be appropriate to applications like peer-to-peer file transfer, since the notion of an "off-peak hour" is not meaningful when end-hosts may be located anywhere in the world.

TCP's built-in flow control can be used as a means to achieve a low-priority transport service. For instance, the mechanism described in [Spr+00] controls the bandwidth by letting the receiver intelligently manipulate the receiver window of standard TCP. This is done because the authors assume a client-server setting where the receiver's access link is typically the bottleneck. The scheme incorporates a delay-based calculation of the expected queue length at the bottleneck, which is quite similar to the calculation in the above delay based protocols, e.g. TCP Vegas. Using a Linux implementation, where TCP flows are classified according to their application's needs, it is shown that a significant improvement in

packet latency can be attained over an unmodified system while maintaining good link utilization.

A similar method is employed by Mehra et al. [Meh+03], where both the advertised receiver window and the delay in sending ACK messages are dynamically adapted to attain a given rate. As in [Spr+00], Mehra et al. assume that the bottleneck is located at the receiver's access link. However, the latter also propose a bandwidth-sharing system, allowing to control the bandwidth allocated to different flows, as well as to allot a minimum rate to some flows.

Receiver window tuning is also done in [Key+04], where choosing the right value for the window is phrased as an optimization problem. On this basis, two algorithms are presented, binary search -- which is faster than the other one at achieving a good operation point but fluctuates -- and stochastic optimization, which does not fluctuate but converges slower than binary search. These algorithms merely use the previous receiver window and the amount of data received during the previous control interval as input. According to [Key+04], the encouraging simulation results suggest that such an application level mechanism can work almost as well as a transport layer scheme like TCP-LP.

Another way of dealing with non-interactive flows, like e.g. web prefetching, is to rate-limit the transfer of such bursty traffic [Cro+98b]. Note that one of the techniques used in [Cro+98b] is, precisely, to have the downloading application adapt the TCP receiver window, so as to reduce the data rate to the minimum needed.

The so-called Background Intelligent Transfer Service (BITS) [BITS], implemented in several versions of Microsoft Windows, uses a system of application-layer priority levels for file-transfer jobs, together with monitoring of bandwidth usage of the network interface (or, in more recent versions, of the network gateway connected to the end-host), so that low-priority transfers give way to both high-priority (foreground) transfers and traffic from interactive applications.

5. Orthogonal work

Various suggestions have been published for realizing a LBE service by influencing the way packets are treated in routers. One example is the Persistent Class Based Queuing (P-CBQ) scheme presented in [Car+01], which is a variant of Class Based Queuing (CBQ) with per-flow accounting. RFC 3662 [RFC3662] defines a DiffServ per-domain behavior called "Lower Effort". Similar Lower-Effort PDBs have been tested and deployed, at least in research networks [Cho+03], [QBSS].

Harp [Kok+04] realizes a LBE service by dissipating background traffic to less-utilized paths of the network. This is achieved without changing routers by using edge nodes as relays. According to the authors, these edge nodes should be gateways of organizations in order to align their scheme with usage incentives, but the technical solution would also work if Harp was only deployed in end hosts. It detects impending congestion by looking at delay, similar to TCP Nice [Ven+02], and manages to improve utilization and fairness over pure single-path solutions.

An entirely different approach is taken in [Egg+05]: here, the priority of a flow is reduced via a generic idletime scheduling strategy in a host's operating system. While results presented in this paper show that the new scheduler can effectively shield regular tasks from low-priority ones (e.g. TCP from greedy UDP) with only a minor performance impact, it is an underlying assumption that all involved end hosts would use the idletime scheduler. In other words, it is not the focus of this work to protect a standard TCP flow which originates from any host where the presented scheduling scheme may not be implemented.

In [Ven+08], Venkataraman et al. propose a transport-layer approach to leverage an existing, network-layer LBE service based on priority queueing. The transport protocol, which they call PLT (Priority-Layer Transport), splits a layer-4 connection into two flows, a high-priority one and a low-priority one. The high-priority flow is sent over the higher-priority queueing class (in principle, offering a best-effort service) using an AIMD, TCP-like congestion control mechanism. The low-priority flow, which is mapped to the LBE class, uses a non TCP-friendly congestion control algorithm. The goal of PLT is thus to maximize its aggregate throughput by exploiting unused capacity in an aggressive way, while protecting standard TCP flows carried by the best-effort class. [Ott+03] proposes simple changes to the AIMD parameters of TCP for use over a network-layer LBE service, so that such "filler" traffic may aggressively consume unused bandwidth. Note that [Ven+08] also considers a mechanism for detecting the lack of priority queueing in the network, so that the non-TCP friendly flow may be inhibited. The PLT receiver monitors the loss rate of both flows; if the high-priority flow starts seeing losses while the low-priority one does not experience 100% loss, this is taken as an indication of the absence of strict priority queueing.

Another technique is that used by protocols like NF-TCP [Aru+10b], where a bandwidth-estimation module integrated into the transport protocol allows to rapidly take advantage of free capacity. NF-TCP combines this with an early congestion detection based on Explicit Congestion Notification (ECN) [RFC3168] and RED [RFC2309]; when congestion starts building up, appropriate tuning of a RED queue

allows to mark low-priority (i.e., NF-TCP) packets with a much higher probability than high-priority (i.e., standard TCP) packets, so low-priority flows yield up bandwidth before standard TCP flows.

6. Acknowledgements

The authors would like to thank Dragana Damjanovic, Melissa Chavez, Yinxia Zhao and Mayutan Arumaithurai for reference pointers.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

This document introduces no new security considerations.

9. Informative References

- [Aru+10b] Arumaithurai, M., Fu, X., and K. Ramakrishnan, "NF-TCP: A Network Friendly TCP Variant for Background Delay-Insensitive Applications", Technical Report No. IFI-TB-2010-05, Institute of Computer Science, University of Goettingen, Germany, September 2010, <<http://www.net.informatik.uni-goettingen.de/publications/1718/NF-TCP-techreport.pdf>>.
- [BITS] Microsoft, "Windows Background Intelligent Transfer Service", <[http://msdn.microsoft.com/library/bb968799\(VS.85\).aspx](http://msdn.microsoft.com/library/bb968799(VS.85).aspx)>.
- [Bha+07] Bhandarkar, S., Reddy, A., Zhang, Y., and D. Loguinov, "Emulating AQM from end hosts", Proceedings of ACM SIGCOMM 2007, 2007.
- [Bia+03] Biaz, S. and N. Vaidya, "Is the round-trip time correlated with the number of packets in flight?", Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC '03) , pages 273-278, 2003.
- [Bra+94] Brakmo, L., O'Malley, S., and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", Proceedings of SIGCOMM '94, pages 24-35, August 1994.

- [Car+01] Carlberg, K., Gevros, P., and J. Crowcroft, "Lower than best effort: a design and implementation", Workshop on Data communication in Latin America and the Caribbean 2001, San Jose, Costa Rica, Pages: 244 - 265, 2001.
- [Cha+10] Chan, Y., Lin, C., Chan, C., and C. Ho, "CODE TCP: A competitive delay-based TCP", Computer Communications , 33(9):1013-1029, June 2010.
- [Cho+03] Chown, T., Ferrari, T., Leinen, S., Sabatino, R., Simar, N., and S. Venaas, "Less than Best Effort: Application Scenarios and Experimental Results", Proceedings of QoS-IP , pages 131-144, February 2003.
- [Cro+98] Crowcroft, J. and P. Oechslein, "Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP", ACM SIGCOMM Computer Communication Review vol. 28, no. 3 (July 1998), pp. 53-69, 1998.
- [Cro+98b] Crovella, M. and P. Barford, "The network effects of prefetching", Proceedings of Infocom 1998, April 1998.
- [Dam+09] Damjanovic, D. and M. Welzl, "MultFRC: Providing Weighted Fairness for Multimedia Applications (and others too!)", ACM Computer Communication Review vol. 39, no. 3 (July 2009), 2009.
- [Dev+03] De Vendictis, A., Baiocchi, A., and M. Bonacci, "Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario", Performance Evaluation , 53(3-4):225-253, 2003.
- [Dyk+02] Dykes, S. and K. Robbins, "Limitations and benefits of cooperative proxy caching", IEEE Journal on Selected Areas in Communications 20(7):1290-1304, September 2002.
- [Egg+05] Eggert, L. and J. Touch, "Idletime Scheduling with Preemption Intervals", Proceedings of 20th ACM Symposium on Operating Systems Principles SOSP 2005, Brighton, United Kingdom, pp. 249/262, October 2005.
- [Gur+04] Gurtov, A. and S. Floyd, "Modeling wireless links for transport protocols", ACM SIGCOMM Computer Communications Review 34(2):85-96, April 2004.
- [Hac+04] Hacker, T., Noble, B., and B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP", Proceedings

of Infocom 2004, March 2004.

- [Hac+08] Hacker, T. and P. Smith, "Stochastic TCP: A Statistical Approach to Congestion Avoidance", Proceedings of PFLDnet 2008, March 2008.
- [Hay10] Hayes, D., "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms", Technical Report 100219A , Centre for Advanced Internet Architectures, Swinburne University of Technology, February 2010.
- [Hen+00] Hengartner, U., Bolliger, J., and T. Gross, "TCP Vegas revisited", Proceedings of Infocom 2000, March 2000.
- [Key+04] Key, P., Massoulie, L., and B. Wang, "Emulating Low-Priority Transport at the Application Layer: a Background Transfer Service", Proceedings of ACM SIGMETRICS 2004, January 2004.
- [Kok+04] Kokku, R., Bohra, A., Ganguly, S., and A. Venkataramani, "A Multipath Background Network Architecture", Proceedings of Infocom 2007, May 2007.
- [Kuo+08] Kuo, F. and X. Fu, "Probe-Aided MultTCP: an aggregate congestion control mechanism", ACM SIGCOMM Computer Communication Review vol. 38, no. 1 (January 2008), pp. 17-28, 2008.
- [Kur+00] Kurata, K., Hasegawa, G., and M. Murata, "Fairness Comparisons Between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas", Proceedings of INET 2000, July 2000.
- [Kuz+06] Kuzmanovic, A. and E. Knightly, "TCP-LP: low-priority service via end-point congestion control", IEEE/ACM Transactions on Networking (ToN) Volume 14, Issue 4, pp. 739-752., August 2006, <<http://www.ece.rice.edu/networks/TCP-LP/>>.
- [Liu+07] Liu, S., Vojnovic, M., and D. Gunawardena, "Competitive and Considerate Congestion Control for Bulk Data Transfers", Proceedings of IWQoS 2007, June 2007.
- [Liu+08] Liu, S., Basar, T., and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks", Performance Evaluation , 65(6-7):417-440, 2008.

- [Mar+03] Martin, J., Nilsson, A., and I. Rhee, "Delay-based congestion avoidance for TCP", IEEE/ACM Transactions on Networking , 11(3):356-369, June 2003.
- [McC+08] McCullagh, G. and D. Leith, "Delay-based congestion control: Sampling and correlation issues revisited", Technical report , Hamilton Institute, 2008.
- [Meh+03] Mehra, P., Zakhor, A., and C. De Vleeschouwer, "Receiver-Driven Bandwidth Sharing for TCP", Proceedings of Infocom 2003, April 2003.
- [Mo+99] Mo, J., La, R., Anantharam, V., and J. Walrand, "Analysis and Comparison of TCP Reno and TCP Vegas", Proceedings of Infocom 1999, March 1999.
- [Ott+03] Ott, B., Warnky, T., and V. Liberatore, "Congestion control for low-priority filler traffic", SPIE QoS 2003 (Quality of Service over Next-Generation Internet), In Proc. SPIE, Vol. 5245, 154, Monterey (CA), USA, July 2003.
- [Ott+04] Ott, D., Sparks, T., and K. Mayer-Patel, "Aggregate congestion control for distributed multimedia applications", Proceedings of Infocom 2004, March 2004.
- [Pra+04] Prasad, R., Jain, M., and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance", Proceedings of PFLDnet , 2004.
- [QBSS] "QBone Scavenger Service (QBSS)", Internet2 QBone Initiative .
- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3662] Bless, R., Nichols, K., and K. Wehrle, "A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services",

RFC 3662, December 2003.

- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.
- [Rew+06] Rewaskar, S., Kaur, J., and D. Smith, "Why don't delay-based congestion estimators work in the real-world?", Technical report TR06-001, University of North Carolina at Chapel Hill, Dept. of Computer Science, January 2006.
- [Sha+05] Shalunov, S., Dunn, L., Gu, Y., Low, S., Rhee, I., Senger, S., Wydrowski, B., and L. Xu, "Design Space for a Bulk Transport Tool", Technical Report, Internet2 Transport Group, May 2005.
- [Spr+00] Spring, N., Chesire, M., Berryman, M., Sahasranaman, V., Anderson, T., and B. Bershad, "Receiver based management of low bandwidth access links", Proceedings of Infocom 2000, pp. 245-254, vol.1, 2000.
- [Sri+08] Sridharan, M., Tan, K., Bansala, D., and D. Thaler, "Compound TCP: A new TCP congestion control for high-speed and long distance networks", Internet Draft draft-sridharan-tcpm-ctcp, work in progress, November 2008.
- [Tan+06] Tan, K., Song, J., Zhang, Q., and M. Sridharan, "A Compound TCP approach for high-speed and long distance networks", Proceedings of IEEE INFOCOM 2006, Barcelona, Spain, April 2008.
- [Ven+02] Venkataramani, A., Kokku, R., and M. Dahlin, "TCP Nice: a mechanism for background transfers", Proceedings of OSDI '02, 2002.
- [Ven+08] Venkataraman, V., Francis, P., Kodialam, M., and T. Lakshman, "A priority-layered approach to transport for high bandwidth-delay product networks", Proceedings of ACM CoNEXT, Madrid, December 2008.
- [Wei+05] Weigle, M., Jeffay, K., and F. Smith, "Delay-based early congestion detection and adaptation in TCP: impact on web performance", Computer Communications 28(8):837-850, May 2005.
- [Wei+06] Wei, D., Jin, C., Low, S., and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance", IEEE/

ACM Transactions on Networking , 14(6):1246-1259,
December 2006.

Authors' Addresses

Michael Welzl
University of Oslo
Department of Informatics, PO Box 1080 Blindern
N-0316 Oslo,
Norway

Phone: +43 512 507 6110
Email: michawe@ifi.uio.no

David Ros
Telecom Bretagne
Rue de la Chataigneraie, CS 17607
35576 Cesson Sevigne cedex,
France

Phone: +33 2 99 12 70 46
Email: david.ros@telecom-bretagne.eu

