Less-than-Best-Effort Service: A Survey of End-to-End Approaches

David Ros and Michael Welzl

***Abstract*—This paper provides a survey of transport protocols and congestion control mechanisms that 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 and mechanisms provide what is sometimes called a *less-than-best-effort* or *lower than best-effort* service. To a user, such a service can, for instance, be an attractive choice for applications which create traffic that is considered less urgent than that of others—e.g., automatic backup, software updates running in the background, or peer-to-peer applications. The focus of this survey is on *end-host* approaches for achieving a less-than-best-effort service. This includes e.g. upper-layer methods, or techniques that leverage standard transport-layer mechanisms so as to reduce the impact on other competing flows.**

***Index Terms*—Transport protocols; congestion control; quality of service; less-than-best effort service.**

I. Introduction

TCP is the transport protocol of choice for most Internet applications. Such choice may be driven, for instance, by the need of end-to-end data reliability, or because TCP makes it easier for applications to pass through middleboxes. However, due to TCP's built-in congestion control mechanisms, picking TCP as an end-to-end protocol also means picking a specific way of sharing network resources (buffers, bandwidth...) among competing flows.

TCP can be regarded at striving for some form of flow-rate fairness. Under ideal circumstances, long-lived TCP flows that share a bottleneck link tend to obtain an equal share of the link's bandwidth. In practice, an equal share of resources is seldom the outcome of TCP congestion control. It is well known that many factors, like e.g. different round-trip times (RTTs), packet loss rates or flow sizes, result in unequal rates. Nevertheless, in any event the achieved bandwidth share is mainly the outcome of the particular rate-control policies (i.e., congestion control) adopted by TCP.

Trying to equally share network resources in a best-effort manner, as discussed above, is not necessarily always the best objective. Some flows may be less urgent than others; for instance, unattended applications like automatic backup or software updates, or peer-to-peer file transfers, may tolerate longer flow-completion times than interactive web browsing. Besides, by both running for very long periods and opening many TCP connections in parallel, some of these applications may easily end up with a share of resources much higher than what common definitions of fairness would allow. The so-called *bufferbloat* problem of excessive buffering space at bottleneck points aggravates latency issues. TCP-based bulk transfer applications will tend to saturate the bottleneck buffer, inducing huge delays that render interactive applications unusable [2].

Latency-sensitive flows may thus benefit from some flows receiving a *less-than-best-effort* (LBE) service—that is, a service that allows to use only residual network resources[1] and aims at lowering the impact of the latter flows on the former. In other words, users could obtain a more appropriate network behavior (perceived as better overall performance) if an LBE service were used by some applications, as this would allow time-critical data transfers to access more of the available bandwidth when they need it.

A. Goal and outline of the paper

This paper presents a brief survey of proposals to attain a Less-than-Best-Effort service by means of *end-host mechanisms*. For the purposes of this survey, we loosely define an 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 are designed to provide this service as LBE systems. With the exception of TCP Vegas [3], which we present for historical reasons, we exclude systems that have been noted to exhibit LBE behavior under some circumstances but were not designed for this purpose (e.g., RAPID [4]).

TCP's congestion control consists of two phases: (1) *Congestion avoidance*, in which it continuously increases its sending rate by one packet per RTT until at least one packet is lost or ECN-marked [5]. Such loss or marking is then assumed to occur because the queue at the bottleneck router between the sender and receiver has overflown, and hence interpreted as a sign of congestion. In order to react to congestion, a TCP sender halves its sending rate—this policy is called Additive-Increase, Multiplicative-Decrease (AIMD). (2) In the beginning of a connection and in case of very severe congestion, TCP is in its *slow start* phase, in which it (re-)starts with an initial window of (usually) three packets and doubles its send rate every RTT until it reaches a fixed threshold or a previously determined "safe" send rate.

Generally, LBE behavior can be achieved at the transport layer with a mechanism that resembles TCP but exhibits a

Manuscript received 3 November 2011; revised 12 May 2012. A shorter version of this paper appeared as RFC 6297 [1].

D. Ros is with Institut Mines-Télécom / Télécom Bretagne, Rue de la Châtaigneraie, CS 17607, 35576 Cesson Sévigné cedex, France (e-mail: david.ros@telecom-bretagne.eu).

M. Welzl is with the University of Oslo, Department of Informatics, PO Box 1080 Blindern, N-0316 Oslo, Norway (e-mail: michawe@ifi.uio.no).

Digital Object Identifier 10.1109/SURV.2012.060912.00176

[1]For this reason, LBE is sometimes referred to as a *scavenger* service.

more cautious behavior, e.g. by changing Congestion Avoidance to reduce the rate by more than half in the face of congestion or increase it by less than one packet per RTT.

Another, intuitively even less intrusive possibility, is to react to *incipient* congestion before standard TCP would even notice it: before the bottleneck queue overflows, it has to grow, and such growth translates into growing delay for packets travelling from the sender to the receiver. It should therefore not be surprising that the majority of transport-layer LBE proposals is based on measuring this delay. The exact magnitude of this delay depends on the length of the queue and the speed at which it is drained, which is governed by the capacity of the bottleneck link. Delay-based mechanisms can therefore face problems with lacking feedback accuracy when the capacity of this link is very large and/or the bottleneck queue is very small. Moreover, a growing queue only affects traffic in the sender-receiver direction, and hence the easily measurable RTT, which will also include fluctuations that happen on the backward path, can be a too noisy signal. A detailed discussion of such problems is given in Section II-E.

Finally, different from the techniques mentioned above, some end-to-end mechanisms achieve an LBE behavior without modifying transport protocol standards (e.g., by changing the receiver window of standard TCP).

According to this classification, solutions have been categorized in this paper as *delay-based transport protocols* (Section II), *non-delay-based transport protocols* (Section III) and *upper-layer approaches* (Section IV). Some of the schemes in the first two categories could be implemented using TCP without changing its header format; this would facilitate their deployment in the Internet. The schemes in the third category are, by design, supposed to be especially easy to deploy because they only describe a way in which existing transport protocols are used.

An LBE service can be implemented by means of lower-layer (e.g., network layer) techniques only, without any involvement of the endpoints. There has been a substantial amount of work related to LBE mechanisms below the transport layer; for instance, there is a Diffserv-based, Lower-Effort per-domain behavior [6], and similar proposals have been described elsewhere [7], [8]. Going back further in history, one can find many more related traffic discrimination schemes—e.g. ATM's Available Bit Rate (ABR) service [9] was designed to let hosts efficiently use left-over bandwidth that is not used by any other service. ABR required a specific behavior from both the end systems and the network switches. There are also QoS building blocks such as Persistent Class-Based Queuing (P-CBQ) [10] that can be used to give certain traffic an LBE-like treatment. The related literature is vast; a relevant reference for readers interested in such more general mechanisms that are not specific to Internet LBE on the transport layer is [11].

Several network-level solutions aim at giving lower priority to "bandwidth hogs", like certain Active Queue Management (AQM) schemes, traffic shaping or scheduling methods; some based on e.g. deep-packet inspection or traffic-volume accounting are already deployed in some networks [12]. Besides, the IETF Congestion Exposure (CONEX) working group[2] is developing a network-layer mechanism which can incentivize the usage of LBE-like applications and/or of LBE-like transports [13].

There are some examples of network-assisted approaches with a transport layer focus, such as Harp [14], which realizes an LBE service by dissipating background traffic to less-utilized paths of the network, based on multipath routing and multipath congestion control. Another example is Network-Friendly TCP (NF-TCP) [15], [16], where a bandwidth-estimation module integrated into the transport protocol allows to rapidly take advantage of free capacity, provided that an especially tuned version of RED [17] with ECN [5] is available along the path to give special treatment to NF-TCP packets. In [18], Venkataraman et al. propose a transport-layer approach called PLT (Priority-Layer Transport) to leverage an existing, network-layer LBE service based on priority queueing. Similar in spirit, [19] proposes simple changes to only the AIMD parameters of TCP for use over a network-layer LBE service, so that such "filler" traffic may aggressively consume unused bandwidth.

What all the latter schemes have in common is that they require some special behavior by an element (usually, but not always, the bottleneck router) along the path between the sender and receiver. This can be a prohibitive deployment obstacle, making pure end-to-end solutions seem much more attractive. In what follows, our discussion is therefore limited to such mechanisms.

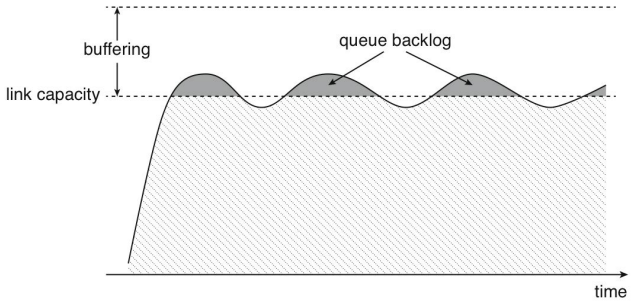
II. Delay-Based Transport Protocols

It is wrong to generally equate "little impact on standard TCP" with "small sending rate". Without Explicit Congestion Notification (ECN) support [5], standard TCP will normally increase its congestion window (and effective sending rate) until a queue overflows, causing one or more packets to be dropped and the effective rate to be reduced. A protocol that stops increasing the rate before this event happens can, in principle, attain a better performance than standard TCP. This can be achieved by performing *delay-based congestion control* at the sender, i.e., monitoring end-to-end delays and using such delay measurements to control the sending rate.

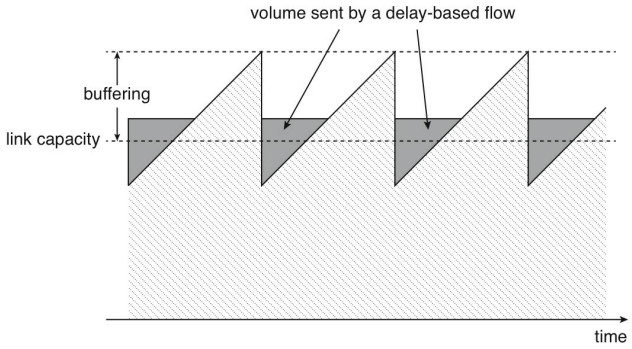
Figure 1 schematically depicts the basic principles of a delay-based LBE transport. The sending rate for a delay-based LBE flow, in absence of other flows sharing the bottleneck, is illustrated by Fig. 1a. The sender tries to send as fast as possible while monitoring the end-to-end delay (corresponding to some amount of buffered packets, shown as gray areas in the figure), and lowers its rate as soon as it detects that the backlog of packets in the end-to-end path exceeds some predefined value. By avoiding to completely fill the buffers by itself, the delay-based flow keeps the latency low for any other flow that may start sending data after the former.

Figure 1b shows the delay-based LBE flow competing with a long-lived non-LBE flow using standard, loss-based TCP. In the presence of another, non-LBE flow, the LBE sender injects data into the network only when the delay due to backlogged packets is below the predefined value, and

[2]http://tools.ietf.org/wg/conex



(a) Sending rate of a delay-based flow: ideal case.



(b) Coexistence of an LBE flow with a non-LBE one.

Fig. 1. Delay-based congestion control.

decreases its sending rate when the other flow increases its own rate; this corresponds to the gray areas in Fig. 1b. When the backlog gets larger than the threshold value, the LBE flow reduces its sending rate to a very low value, to avoid contributing to delay.

A. TCP Vegas

TCP Vegas [3] is one of the first protocols that was known to have a smaller sending rate than standard TCP when both protocols share a bottleneck [20]—yet, it was designed to achieve *more*, not less, throughput than standard TCP[3]. Indeed, when TCP Vegas is the only congestion control algorithm used by flows going through the bottleneck, its throughput is greater than the throughput of standard TCP. However, depending on the bottleneck queue length, TCP Vegas itself can be starved by standard TCP flows. This can be remedied to some degree[4] by the Random Early Detection (RED) Active Queue Management mechanism [17]. Vegas linearly increases or decreases the sending rate, based on the difference between the expected throughput and the actual throughput. The estimation is based on RTT measurements.

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 survey (it has been shown that other elements of the protocol can sometimes play a greater role for its overall behavior [29]). In congestion avoidance, once per RTT, TCP Vegas calculates the expected throughput as: $T_e = \texttt{WindowSize}/\texttt{BaseRTT}$, where `WindowSize` is the current congestion window and `BaseRTT` is the minimum of all measured RTTs. The expected throughput T_e is then compared with the actual throughput T_a, measured based on recent acknowledgements (ACKs). For this, two thresholds α and β (with $\alpha < \beta$) are defined. If $T_a > T_e - \alpha$, this is taken as a sign that the network is underutilized, causing the protocol to linearly increase its rate. If $T_a < T_e - \beta$, this is taken as a sign of congestion, causing the protocol to linearly decrease its rate. When the measured RTT grows to a value larger than `BaseRTT`, the actual throughput T_a, which can be described by $\texttt{WindowSize}/RTT$, gets smaller than T_e, and hence Vegas reacts to delay by reducing its rate when the RTT grows.

TCP Vegas has been analyzed extensively, see e.g. [20], [29]–[33]. 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 (like CARD [34], Tri-S [35], or DUAL [36]) are not discussed here because of the historical role that TCP Vegas has taken in the literature.

B. LEDBAT

Low Extra Delay Background Transport (LEDBAT) is a delay-based congestion control mechanism that is currently being standardized at the IETF [37], in the working group of the same name[5]. A similar algorithm has already been implemented and deployed in BitTorrent peer-to-peer clients, in the so-called uTP protocol [38].

LEDBAT performs window-based congestion control. It aims at interfering as little as possible with standard TCP flows, by being less aggressive than TCP when probing for additional bandwidth, and by trying to keep a small standing queue. LEDBAT has been designed to be used either as part of an existing transport protocol (with the appropriate extensions), or as an application-layer mechanism running on top of UDP (the latter is the approach followed by uTP).

Different from Vegas, LEDBAT uses one-way delays (OWDs) instead of RTTs to adapt its congestion window `cwnd`. This is done to avoid reacting to delay fluctuations that are caused by reverse cross-traffic.

A LEDBAT sender constantly estimates forward queueing delay, and tries to keep such delay around a target value (specified as $\leq$ 100 ms in [37]) by adapting `cwnd`. With every acknowledgement, the sender increases or decreases `cwnd` linearly, in proportion to the difference between the measured queueing delay and the target value; a fixed coefficient $\texttt{GAIN} \leq 1$ applied to such delay difference allows to control the aggressiveness of window increases/decreases. Moreover, the amount of window increase is computed so that it is never larger than that for a TCP flow—and the closer the queueing

[3]In fact, delay-based congestion control is at the basis of many proposals that aim at generally improving the performance of TCP's congestion avoidance. Some of these proposals, like Compound TCP [21], TCP Illinois [22] and RACC [23], are hybrid loss- and delay-based mechanisms, whereas others (e.g., FAST TCP [24], NewVegas [25], or CODE TCP [26]) are variants of Vegas based primarily on delays.

[4]Though not specific to Vegas, there are also proposals, such as [27], [28], aiming at improving the coexistence of loss-based and delay-based congestion-controlled flows.

[5]http://tools.ietf.org/wg/ledbat

delay is to the target value, the slower the window growth becomes.

In case of packet loss, or of ECN-marking of packets, LEDBAT's congestion control behaves exactly like TCP NewReno [39]—i.e., `cwnd` is halved, and such a window reduction is done at most once per RTT.

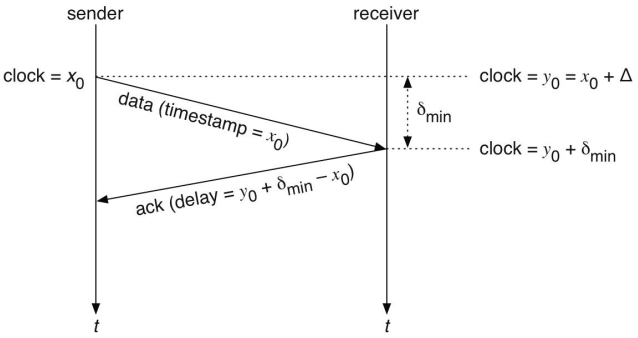
LEDBAT measures forward queueing delays by means of timestamps. At the sender, such timestamps are computed from a local clock and transmitted in data packets. Upon reception of data, the receiver replies with an ACK carrying the difference between a timestamp *local to the receiver* and the timestamp in the data packet. In principle, timestamp clocks at both sender and receiver should be synchronized to measure OWDs in this fashion. However, a LEDBAT sender sidesteps the issue of synchronization, by subtracting such (likely biased) delay values from the minimum delay value received over a time interval (biased in the same way, barring any clock skew); this operation should cancel any clock offset between sender and receiver, and yield an estimation of the "excess" queueing delay. To minimize the impact of outliers in the delay computation, [37] suggests using a simple filtering mechanism.

Figure 2 illustrates the method described above, used by LEDBAT to remove clock bias when estimating queueing delays. The clock at the receiver is offset by a constant value Δ with respect to the sender's clock. Assume the OWD δ_{min} in Fig. 2a is the minimum delay observed; therefore, upon reception of the ACK the sender stores the estimated minimum delay $D_{\text{min}} = y_0 + \delta_{\text{min}} - x_0 = \Delta + \delta_{\text{min}}$. Note that, if there were no bias (i.e., if Δ were $= 0$) then this value would represent the true value of the OWD. Assume now that later on, when another data packet is sent (Fig. 2b), queueing delay has increased by an amount q, so the OWD becomes $\delta = \delta_{\text{min}} + q$. As before, the receiver echoes back the estimated delay $D = y_1 + \delta - x_1 = \Delta + \delta$. The sender estimates thus the current extra queueing delay as the difference between the current delay sample and the minimum observed, that is, $D - D_{\text{min}} = (\Delta + \delta) - (\Delta + \delta_{\text{min}}) = q$.

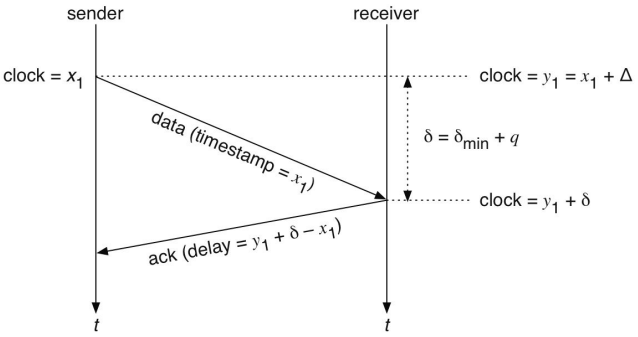
The performance of LEDBAT has been the subject of several recent papers. With the exception of [40]–[42], all these works have evaluated the LEDBAT algorithms by means of simulations and/or simple mathematical models. Besides looking at the overall behavior of LEDBAT, such papers have delved into issues like:

- Coexistence of LEDBAT flows with TCP flows in a bottleneck [40], [41], [43], [44].
- Intra-protocol fairness [40], [45], [46], in particular the so-called *latecomer advantage* (Sec. II-E2).
- Sensitivity to different target queueing delays [40].
- Dynamic adaptation of the `GAIN` parameter [47].
- Impact of delay variations due to e.g. re-routing [48].
- File transfer times in a peer-to-peer swarm [42], [44] using LEDBAT and/or TCP.
- Comparison between LEDBAT and other delay-based proposals: here, [43] seems to be the only available reference, and it arrives at the conclusion that LEDBAT generally achieves a lower priority than TCP Nice [49] and TCP Low Priority (TCP-LP) [50].

Figures 3 and 4, taken from a message to the IETF LEDBAT mailing list [51], illustrate the behavior of LEDBAT in two real-world wireless networking scenarios, obtained with Apple's implementation of the mechanism[6] in Mac OS X. While these data points are only meant as examples, they indicate that LEDBAT fulfills its promise of having significantly less negative influence than TCP on the transfer time of competing TCP data transfers: in most cases, loading the web page took much longer when another TCP was active than in the presence of LEDBAT, and the LEDBAT case does generally not differ much from the case of no other ongoing transfer.



(a) Measuring the minimum OWD.



(b) Measuring a larger OWD due to queueing.

Fig. 2. Estimation of forward queueing delay.

Schneider et al. [40] question the long-term usefulness of LEDBAT (and, seemingly, of similar approaches to LBE), given the current trend in home gateways to add features like traffic shaping and/or scheduling to protect latency- or bandwidth-sensitive flows (e.g., [52], [53]).

C. Other delay-based aproaches

Delay-based transport protocols that were designed to be non-intrusive include TCP Nice [49] and TCP Low Priority (TCP-LP) [50]. TCP Nice [49] 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

[6]See Table II in Section V for code availability.

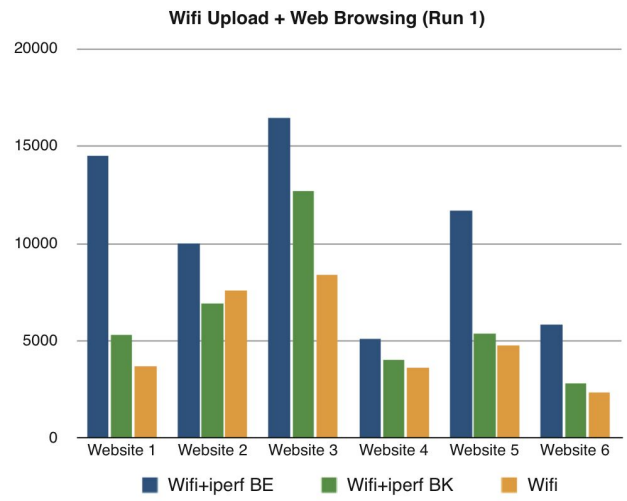


Fig. 3. Web page load times (in milliseconds) to different websites when a standard TCP upload (blue), LEDBAT (green), or no upload flow (yellow) was in progress to another host on a WiFi connection (taken from [51]).

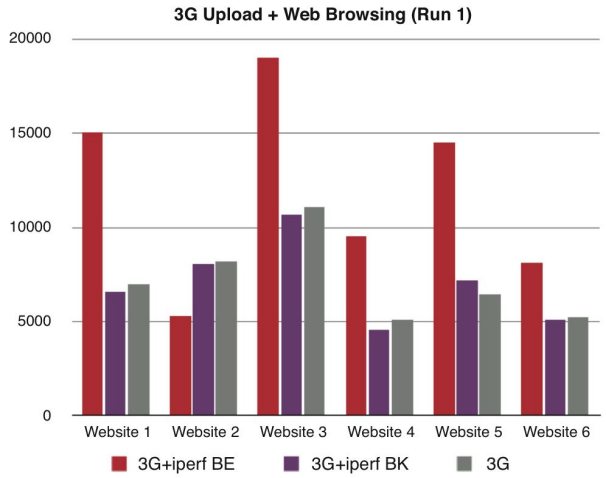


Fig. 4. Web page load times (in milliseconds) to different websites when a standard TCP upload (red), LEDBAT (purple), or no upload flow (grey) was in progress to another host on a 3G connection (taken from [51]).

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 [49] show that TCP Nice achieves its goal of efficiently utilizing spare capacity while being non-intrusive to standard TCP.

Like LEDBAT, TCP-LP [50] uses only the one-way delay instead of the RTT as an indicator of incipient congestion. Using the TCP Timestamps option [54], 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 connection'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 simply halving the congestion window like standard TCP; TCP-LP tries to delay the former action by an additional RTT, to see if there is persistent congestion or not. It does so by halving the window at first in response to an early congestion indication, then initializing an "inference time-out timer" and maintaining the current congestion window until this timer fires. If another early congestion indication appeared during this "inference phase", the window is then set to 1; otherwise, the window is maintained and TCP-LP continues to increase it in 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 that, like standard TCP, TCP-LP reacts to congestion at most once per RTT.

Using a simple analytical model, the authors of TCP-LP [50] illustrate the feasibility of a delay-based LBE transport by showing that, due to the non-linear relationship between throughput and RTT, it is possible to avoid interfering with standard TCP traffic even when the flows under consideration have a larger RTT than standard TCP flows. With ns-2 simulations and real-life experiments using a Linux implementation, the authors of [50] 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[7].

Sync-TCP [57] 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 Additive-Increase Multiplicative-Decrease (AIMD) parameters depending on the trend followed by the average delay over an observation window. Even though the authors of [57] 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 standard TCP in congestion-avoidance mode.

D. Proposals combining delay and available bandwidth

TCP Westwood Low-Priority (TCPW-LP) [58], [59] is an LBE variant of Westwood TCP [60]. Like Vegas, Westwood

[7]There exists a version of TCP-LP, called HSTCP-LP [55], designed for bulk data transfer in networks with large bandwidth-delay products. HSTCP-LP behaves roughly like TCP-LP for low available capacities, and otherwise reverts to a HSTCP-like behavior [56] for large congestion windows.

keeps track of the minimum RTT (RTTmin) observed during the lifetime of the connection. A Westwood sender estimates the available bandwidth BWE based on incoming TCP acknowledgements. When there is moderate congestion (detected by means of duplicate ACKs), the sender adjusts TCP's cwnd and slow-start threshold by setting them to $\texttt{BWE} \times \texttt{RTTmin}$, so they reflect the size of the end-to-end pipe in the absence of congestion.

TCPW-LP adds to Westwood a so-called early window-reduction mechanism based on the number of backlogged packets, computed as $\texttt{cwnd} - \texttt{BWE} \times \texttt{RTTmin}$. Whenever this backlog (called a "virtual queue length") exceeds a threshold Q_{th}, cwnd is set to $\texttt{BWE} \times (\texttt{RTTmin} - D_l)$, with D_l being a (smoothed) estimation of the minimum queueing delay. The threshold Q_{th} is not fixed, but tries to take into account the congestion induced by "foreground" (i.e., non TCPW-LP) flows—when congestion is mainly due to foreground flows, a smaller Q_{th} is used, so early-window reduction kicks in with smaller increases in backlog.

ImTCP-bg [61] is a delay-based scheme close in spirit to TCPW-LP. A congestion control algorithm, based on estimating the available bandwidth, is complemented with a congestion detector based on RTTs. First, the congestion window is always kept upper-bounded by $\bar{A} \times \texttt{RTTmin}$, where $\bar{A}$ is an averaged bandwidth estimation. Whenever SRTT, the smoothed (average) RTT—computed as in standard TCP—grows above $\delta \times \texttt{RTTmin}$ (with $\delta > 1$ a fixed parameter of the mechanism), cwnd is reduced in proportion to the ratio $\texttt{RTTmin}/\texttt{SRTT}$.

E. Issues with Delay-Based Congestion Control

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 for instance [62]–[67]. 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:

- *Bottleneck buffer size:* A delay-based mechanism can be given an LBE behavior only 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.
- *RTT measurement issues:* 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., when 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 [68]. A congested reverse path may also result in an erroneous assessment of the congestion state of the forward path. Finally, in the case of fast or short-distance links, the majority of the measured delay can in fact be due to processing in the involved hosts; typically, this processing delay is not of interest, and it can underlie fluctuations that are not related to the network at all.
- *OWD measurement issues:* Most of the RTT measurement issues above apply to OWD measurements too, but, as discussed in Sec. II-B, the latter would in principle also require synchronized clocks. While the problem of clock offsets—i.e., clock A being ahead or behind clock B—disappears when the metric of interest is not the real OWD but only its change, the problem of clock drift—i.e., clock A being faster or slower than clock B—does not. In LEDBAT, this problem is addressed by ensuring that some parameters (e.g., the validity of a base delay estimate) are set such that the typically small clock drifts seen in practice do not cause a problem. The appendix of [37] also briefly describes some clock-skew correction mechanisms that could be incorporated in a LEDBAT implementation.
- *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 in Figure 1 of [65].
- *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 timescales [69].

Bhandarkar et al. [27] claim that it is possible to significantly improve the accuracy of delay-based congestion prediction by adopting some simple techniques (smoothing of RTT samples, increasing the RTT sampling frequency). Nonetheless, they acknowledge that even with such techniques, it is not possible to eradicate detection errors. Their proposed delay-based congestion-avoidance method, PERT (Probabilistic Early Response TCP), mitigates the impact of residual detection errors by means of a probabilistic response mechanism to congestion-detection events.

2) Other Issues: 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 depend on the accuracy issues listed in Sec. II-E1. 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 [70].

Regarding the issue of false positives or 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 an LBE transport protocol it is 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.

A delay-based mechanism may also suffer from the so-called *latecomer advantage* or *latecomer unfairness* problem. Consider the case in which the bottleneck link is already 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 [46], [71].

III. Non-Delay-Based Transport Protocols

There exist a few transport-layer proposals that achieve an LBE service without relying on delay as an indicator of congestion. In the algorithms discussed below, the loss rate of the flow determines, either implicitly or explicitly, the sending rate; this rate is adapted so as to obtain a lower share of the available bandwidth than standard TCP. Such mechanisms likely cause more queuing delay and react to congestion more slowly than delay-based ones.

4CP [72], which stands for "Competitive and Considerate Congestion Control", is a protocol that provides an LBE service by changing the window control rules of standard TCP. A virtual window is maintained that, 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. Whether the congestion state is "bad" or "good" depends on whether the loss event rate is above or below a threshold (or target) value. 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.

The MulTFRC [73], [74] protocol is an extension of TCP-Friendly Rate Control (TFRC) [75] for multiple flows. MulTFRC takes the main idea of MulTCP [76] and similar proposals, e.g., [77]–[79], a step further. A single MulTCP flow tries to emulate—and be as friendly as—a number $N > 1$ of parallel TCP flows. By supporting values of $N \in (0, 1)$, MulTFRC can be used as a mechanism for an LBE service. Since it does not react to delay like the protocols described in Sec. II but adjusts its rate like TFRC, MulTFRC can probably be expected to be more aggressive than mechanisms such as LEDBAT, TCP Nice or TCP-LP. On the positive side, this also means that MulTFRC is less likely to be prone to starvation, and its aggressiveness is tunable at a fine granularity, even when N is between 0 and 1.

IV. Upper-Layer Approaches

The proposals described in this section do not require modifying transport-protocol standards. Most of them can be regarded as running on top of an existing transport, even though they may be implemented either at the application layer (i.e., in user-level processes), or in the kernel of the end-hosts' operating systems. Such upper-layer mechanisms may arguably be easier to deploy than transport-layer approaches, since they do not require any changes to the transport itself.

A. Receiver-Oriented, Flow-Control-Based Approaches

Some proposals for achieving an LBE behavior work by exploiting existing transport-layer features—typically, at the receiving side. In particular, TCP's built-in flow control can be used as a means to achieve a low-priority transport service.

The mechanism described in [80] is an example of the above technique. It controls the bandwidth by letting the receiver intelligently manipulate the receiver window of standard TCP. This is possible 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, Spring et al. show in [80] 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. [81], where both the advertised receiver window and the delay in sending ACK messages are dynamically adapted to attain a given rate. As in [80], 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 control of the bandwidth allocated to different flows, as well as allotment of a minimum rate to some flows.

Receiver window tuning is also done in [82], 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 [82], 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 web prefetching, is to rate-limit the transfer of such bursty traffic [83]. Note that one of the techniques used in [83] is, precisely, to have the downloading application adapt the TCP receiver window, so as to reduce the data rate to the minimum needed. This way, other flows are disturbed as little as possible while a deadline is respected for the transfer of the data.

B. Approaches Based on Scheduling at the End-Hosts

A simplistic, application-level method for a background transport service may consist in scheduling automated transfers at times when the network is lightly loaded, e.g., as described in [84] for cooperative proxy caching. An issue with such a technique is that it may not necessarily be applicable 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.

TABLE I
TAXONOMY OF MAIN END-TO-END APPROACHES FOR IMPLEMENTING AN LBE SERVICE.

Family of approaches	Based on	Examples of proposals	Control done at
Delay-based transport protocols	OWD-based CC[a]	LEDBAT [37] and uTP [38]	Sender
		TCP-LP [50] and HSTCP-LP [55]	Sender
		Sync-TCP [57]	Sender
	RTT-based CC	TCP Nice [49]	Sender
	RTT- and available bandwidth-based CC	TCPW-LP [58], [59]	Sender
		ImTCP-bg [61]	Sender
Non-delay-based transport protocols	Window- and loss-based CC	4CP [72]	Sender
	Rate-based CC	MulTFRC [73], [74]	Sender
Upper-layer approaches	TCP flow control	Spring et al. [80]	Receiver
		Mehra et al. [81]	Receiver
		Key et al. [82]	Receiver
		Crovella and Barford [83]	Receiver
	End-host scheduling	Off-peak-hour scheduling [84]	Sender
		BITS [85]	Sender
		OS-level idletime scheduling [86]	Sender and Receiver

[a] CC stands for congestion control.

The so-called Background Intelligent Transfer Service (BITS) [85] is implemented in several versions of Microsoft Windows. BITS 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 at a given end-host give way to both high-priority foreground transfers and traffic from interactive applications at the same host.

A different approach is taken in [86]—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 that originates from any host where the presented scheduling scheme may not be implemented.

V. CONCLUDING REMARKS

Table I summarizes the main LBE systems discussed before. Despite the amount and diversity of work in this area, research on end-to-end LBE mechanisms often leaves a few key questions unanswered.

One potential problem for layer-4 LBE systems is that of *encapsulation* and, more generally, of *integration* of such mechanisms into existing transport protocols. The TCP-based proposals described in Sections II and III could likely be implemented either without any changes to transport headers, or by appropriately using TCP options (be they already-defined options or new ones). However, given today's prevalence of middleboxes in the Internet, even using existing options like TCP timestamps may not necessarily be completely trouble-free [87]. Note also that the IETF has focused only on standardizing the congestion-control mechanisms of LEDBAT; how to integrate LEDBAT into standard transports has been left undefined so far.

An application flow that chooses a scavenger end-to-end service would, in principle, experience a degraded performance with respect to using standard TCP. Hence, the issue of deployment and usage *incentives*, though not a technical problem, has to be considered.

Another key question is that of *signaling*. In a sender-side scheme based on TCP, should a receiver be able to tell a sender that it would like to use an LBE service? The bottleneck of an Internet connection is often near the end-hosts, i.e., near a client that obtains data from a server; it therefore seems that the answer to that question should probably be "yes". But how should this be done? There is no such signaling defined in TCP—should a different protocol be used to convey this message?[8] And should such a message also include configuration parameters, or would a simple "use LBE service" signal suffice? Despite being a simple problem, the lack of a standard LBE receiver-to-sender signaling solution could become a serious issue, as it could turn out to be a deployment obstacle for LEDBAT.

Finally, while most of the transport-layer protocols that we have covered in this survey do a good job at showing that they exhibit LBE properties in competition with standard TCP, it is not generally clear how they compare in terms of aggressiveness. Would e.g. LEDBAT push aside TCP-LP or vice versa? So far, [43] appears to be the only investigation of this matter, discussing TCP-LP, TCP Nice and LEDBAT. Code for several of the approaches discussed in this paper is available on the Internet, and could directly be used to carry out a more extensive comparison; Table II provides a list of URLs that worked at the time of writing.

[8] BitTorrent clients supporting uTP seem to tackle this by attempting to establish both TCP and uTP connections, depending on some user-configurable options and hard-coded policies [42].

TABLE II
CODE AVAILABILITY FOR SOME END-TO-END LBE SYSTEMS.

Mechanism	Platform	Runs in[a]	URL[b]
LEDBAT [37]	Mac OS X	Kernel	http://opensource.apple.com/source/xnu/xnu-1699.22.81/bsd/netinet/tcp_ledbat.c
	Linux, ns-2	Kernel	http://www.infres.enst.fr/~drossi/index.php?n=Software.LEDBAT [c]
uTP [38]	Linux, Mac OS X, Windows	User space	https://github.com/bittorrent/libutp
TCP-LP [50]	Linux, ns-2	Kernel	http://www.ece.rice.edu/networks/TCP-LP/
	Linux	Kernel	http://sourceforge.net/projects/tcp-lp-mod/
	Linux	User space	http://sourceforge.net/projects/lpt-lib/
HSTCP-LP [55]	Linux, ns-2	Kernel	http://www.ece.rice.edu/networks/TCP-LP/
ImTCP-bg [61]	FreeBSD	Kernel	http://www.ane.cmc.osaka-u.ac.jp/~hasegawa/imtcp/imtcp_freebsd.html
MulTFRC [73], [74]	Linux, ns-2	User space	http://heim.ifi.uio.no/~michawe/research/projects/multfrc/index.html
BITS [85]	Windows (XP and later)	Kernel	http://en.wikipedia.org/wiki/Background_Intelligent_Transfer_Service [d]

[a] This column does not apply to ns-2 code.
[b] All URLs were tested on June 12, 2012.
[c] A statement on this page requests citing [41].
[d] This page provides some relevant information and links. The code is a component of Windows.

In today's Internet, TCP congestion control has already turned from a homogenous, standardized solution into a heterogeneous world of competing mechanisms. LEDBAT and its relatives have the potential to make this bouquet much more colorful. Whether they will contribute only one new color, or a whole range of colors to the behaviors of real Internet transfers remains to be seen.

ACKNOWLEDGMENTS

D. Ros would like to thank Stein Gjessing and the Department of Informatics (IFI) at the University of Oslo for their kind support.

We thank Padma Bhooma for allowing us to use his figures from [51], and all the people who have contributed to RFC 6297 [1]: Melissa Chavez and Yinxia Zhao for reference pointers, as well as Jari Arkko, Mayutan Arumaithurai, Dragana Damjanovic, Elwyn Davies, Wesley Eddy, Stephen Farrell, Mirja Kuehlewind, Tina Tsou and Rolf Winter for their comments and suggestions.

REFERENCES

[1] M. Welzl and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols," RFC 6297 (Informational), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6297.txt
[2] J. Gettys, "Bufferbloat: Dark buffers in the Internet," *IEEE Internet Computing*, vol. 15, no. 3, pp. 95–96, 2011.
[3] L. S. Brakmo and L. L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.
[4] V. Konda and J. Kaur, "RAPID: Shrinking the congestion-control timescale," in *Proc. IEEE INFOCOM*, Rio de Janeiro, Apr. 2009.
[5] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," Internet Standards Track RFC 3168, IETF, Sep. 2001. [Online]. Available: http://tools.ietf.org/html/rfc3168
[6] R. Bless, K. Nichols, and K. Wehrle, "A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services," Informational RFC 3662, IETF, Dec. 2003.
[7] T. Chown, T. Ferrari, S. Leinen, R. Sabatino, N. Simar, and S. Venaas, "Less than Best Effort: Application Scenarios and Experimental Results," in *Proc. QoS-IP 2003*, ser. Lecture Notes in Computer Science, no. 2601, Feb. 2003, pp. 131–144. [Online]. Available: http://www.cnaf.infn.it/~ferrari/tfngn/lbe/
[8] "QBone Scavenger Service (QBSS)," Internet2 QBone Initiative, http://qbone.internet2.edu/qbss/.
[9] *Traffic Management Specification Version 4.1*, The ATM Forum, 1999.
[10] K. Carlberg, P. Gevros, and J. Crowcroft, "Lower than best effort: a design and implementation," *ACM SIGCOMM Computer Communications Review*, vol. 31, no. 2 supplement, pp. 244–265, Apr. 2001.
[11] G. Armitage, *Quality of service in IP networks: foundations for a multi-service Internet*. Indianapolis, IN, USA: Macmillan Publishing Co., Inc., 2000.
[12] C. Bastian, T. Klieber, J. Livingood, J. Mills, and R. Woundy, "Comcast's protocol-agnostic congestion management system," Informational RFC 6057, IETF, Dec. 2010. [Online]. Available: http://tools.ietf.org/html/rfc6057
[13] B. Briscoe and R. Woundy (eds.), "ConEx concepts and use cases," Internet Draft draft-ietf-conex-concepts-uses, *work in progress*, Mar. 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-conex-concepts-uses
[14] R. Kokku, A. Bohra, S. Ganguly, and A. Venkataramani, "A multipath background network architecture," in *Proc. IEEE INFOCOM*, Anchorage, May 2007.
[15] M. Arumaithurai, X. Fu, and K. Ramakrishnan, "NF-TCP: Network Friendly TCP," in *Proc. 17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)*, May 2010.
[16] ——, "NF-TCP: A network friendly TCP variant for background delay-insensitive applications," in *Proc. IFIP Networking*, Valencia (Spain), May 2011.
[17] B. Braden *et al.*, "Recommendations on queue management and congestion avoidance in the Internet," Informational RFC 2309, IETF, Apr. 1998. [Online]. Available: http://tools.ietf.org/html/rfc2309
[18] V. Venkataraman, P. Francis, M. Kodialam, and T. V. Lakshman, "A priority-layered approach to transport for high bandwidth-delay product networks," in *Proc. ACM CoNEXT*, Madrid, 2008.
[19] B. Ott, T. Warnky, and V. Liberatore, "Congestion control for low-priority filler traffic," in *SPIE QoS 2003 (Quality of Service over Next-Generation Internet)*, ser. Proc. SPIE, Vol. 5245, 154, Monterey (CA), USA, Jul. 2003.
[20] K. Kurata, G. Hasegawa, and M. Murata, "Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas," in *Proc. INET 2000*, Yokohama, Jul. 2000. [Online]. Available: http://www.isoc.org/inet2000/cdproceedings/2d/2d_2.htm
[21] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM 2006*, Barcelona, Spain, Apr. 2006.
[22] S. Liu, T. Basar, and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks," *Performance Evaluation*, vol. 65, no. 6-7, pp. 417 – 440, 2008.
[23] K. Shi, Y. Shu, O. Yang, and J. Luo, "Receiver-assisted congestion control to achieve high throughput in lossy wireless networks," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 491–496, Apr. 2010.
[24] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.
[25] A. De Vendictis, A. Baiocchi, and M. Bonacci, "Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP

Reno network scenario," *Performance Evaluation*, vol. 53, no. 3-4, pp. 225 – 253, 2003.

[26] Y.-C. Chan, C.-L. Lin, C.-T. Chan, and C.-Y. Ho, "CODE TCP: A competitive delay-based TCP," *Computer Communications*, vol. 33, no. 9, pp. 1013–1029, Jun. 2010.

[27] S. Bhandarkar, A. Narasimha Reddy, Y. Zhang, and D. Loguinov, "Emulating AQM from end hosts," in *Proc. ACM SIGCOMM*, Kyoto, Aug. 2007.

[28] L. Budzisz, R. Stanojevic, A. Schlote, R. Shorten, and F. Baker, "On the fair coexistence of loss- and delay-based TCP," in *Proc. IEEE IWQoS*, Jul. 2009, pp. 1–9.

[29] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," in *Proc. IEEE INFOCOM*, Tel Aviv, Mar. 2000.

[30] J. Ahn, P. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: emulation and experiment," in *Proc. ACM SIGCOMM*, Cambridge (MA), USA, Aug. 1995, pp. 185–195.

[31] O. Ait-Hellal and E. Altman, "Analysis of TCP Vegas and TCP Reno," in *Proc. IEEE ICC*, Montreal, Jun. 1997, pp. 495–499.

[32] S. Low, L. Peterson, and L. Wang, "Understanding TCP Vegas: A duality model," *Journal of the ACM*, vol. 49, no. 2, pp. 207–235, Mar. 2002.

[33] A. Tang, J. Wang, S. Hegde, and S. Low, "Equilibrium and fairness of networks shared by TCP Reno and Vegas/FAST," *Telecommunication Systems*, vol. 30, no. 4, Dec. 2005.

[34] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM SIGCOMM Computer Communications Review*, vol. 19, no. 5, pp. 56–71, Oct. 1989.

[35] Z. Wang and J. Crowcroft, "A new congestion control scheme: slow start and search (Tri-S)," *ACM SIGCOMM Computer Communications Review*, vol. 21, no. 1, pp. 56–71, Jan. 1991.

[36] ——, "Eliminating periodic packet losses in the 4.3-Tahoe BSD TCP congestion control algorithm," *ACM SIGCOMM Computer Communications Review*, vol. 22, no. 2, pp. 9–16, Jan. 1992.

[37] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind, "Low extra delay background transport (LEDBAT)," Internet Draft draft-ietf-ledbat-congestion, *work in progress*, Oct. 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-ledbat-congestion

[38] A. Norberg, *uTorrent transport protocol*, BitTorrent, Jan. 2010, http://www.bittorrent.org/beps/bep_0029.html.

[39] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," Internet Standards Track RFC 3782, IETF, Apr. 2004. [Online]. Available: http://tools.ietf.org/html/rfc3782

[40] J. Schneider, J. Wagner, R. Winter, and H.-J. Kolbe, "Out of my way – evaluating Low Extra Delay Background Transport in an ADSL access network," in *Proc. 22nd International Teletraffic Congress (ITC 22)*, Amsterdam, Sep. 2010.

[41] D. Rossi, C. Testa, and S. Valenti, "Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm," in *Proc. Passive and Active Measurement Workshop (PAM'10)*, Zürich, Apr. 2010.

[42] C. Testa, D. Rossi, A. Rao, and A. Legout, "Experimental assessment of BitTorrent completion time in heterogeneous TCP/uTP swarms," in *4th International Workshop on Traffic Measurement and Analysis (TMA)*, ser. Lecture Notes in Computer Science, vol. 7189. Vienna: Springer, Mar. 2012, pp. 52–65.

[43] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa, "A hands-on assessment of transport protocols with lower than best effort priority," in *Proc. IEEE LCN*, Denver, Oct. 2010, pp. 8–15.

[44] C. Testa and D. Rossi, "The impact of uTP on BitTorrent completion time," in *Proc. IEEE P2P'11*, Kyoto, Sep. 2011.

[45] D. Rossi, C. Testa, S. Valenti, and L. Muscariello, "LEDBAT: The new BitTorrent congestion control protocol," in *Proc. ICCCN*, Zürich, Aug. 2010.

[46] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti, "The quest for LEDBAT fairness," in *Proc. IEEE GLOBECOM*, Miami, Dec. 2010.

[47] A. Abu and S. Gordon, "A dynamic algorithm for stabilising LEDBAT congestion window," in *Second International Conference on Computer and Network Technology (ICCNT)*, Bangkok, Apr. 2010, pp. 157–161.

[48] ——, "Impact of delay variability on LEDBAT performance," in *Proc. IEEE AINA*, Singapore, Mar. 2011, pp. 708–715.

[49] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: a mechanism for background transfers," in *Proc. USENIX OSDI '02*, Boston, Dec. 2002.

[50] A. Kuzmanovic and E. Knightly, "TCP-LP: low-priority service via end-point congestion control," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 739–752, Aug. 2006.

[51] P. Bhooma, Message to the IETF LEDBAT mailing list, Oct. 19, 2011. [Online]. Available: http://www.ietf.org/mail-archive/web/ledbat/current/msg00532.html

[52] T. Bonald and L. Muscariello, "Shortest queue first: implicit service differentiation through per-flow scheduling," Demo at IEEE LCN, Oct. 2009.

[53] T. Bonald, L. Muscariello, and N. Ostallo, "Self-prioritization of audio and video traffic," in *Proc. IEEE ICC*, Kyoto, Jun. 2011.

[54] V. Jacobson, B. Braden, and D. Borman, "TCP Extensions for High Performance," Internet Standards Track RFC 1323, IETF, May 1992. [Online]. Available: http://tools.ietf.org/html/rfc1323

[55] A. Kuzmanovic, E. Knightly, and R. Cottrell, "HSTCP-LP: A protocol for low-priority bulk data transfer in high-speed high-RTT networks," in *Proc. PFLDnet*, Argonne (IL), USA, Feb. 2004.

[56] S. Floyd, "HighSpeed TCP for Large Congestion Windows," Experimental RFC 3649, IETF, Dec. 2003. [Online]. Available: http://tools.ietf.org/html/rfc3649

[57] M. Weigle, K. Jeffay, and F. Smith, "Delay-based early congestion detection and adaptation in TCP: impact on web performance," *Computer Communications*, vol. 28, no. 8, pp. 837–850, May 2005.

[58] H. Shimonishi, M. Sanadidi, and M. Gerla, "Service differentiation at transport layer via TCP Westwood low-priority (TCPW-LP)," in *Proc. IEEE ISCC*, Alexandria, Egypt, Jun. 2004, pp. 804–809.

[59] H. Shimonishi, T. Hama, M. Sanadidi, M. Gerla, and T. Murase, "TCP-Westwood low-priority for overlay QoS mechanism," *IEICE Trans. Commun.*, vol. E89-B, no. 9, pp. 2414–2423, Sep. 2006.

[60] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang, "TCP Westwood: bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM MOBICOM*, Rome, Jul. 2001, pp. 287–297.

[61] T. Tsugawa, G. Hasegawa, and M. Murata, "Background TCP data transfer with inline network measurement," *IEICE Trans. Commun.*, vol. E89-B, no. 8, pp. 2152–2160, Aug. 2006.

[62] M. Rodríguez-Pérez, S. Herrera-Alonso, M. Fernández-Veiga, and C. López-García, "Common problems in delay-based congestion control algorithms: a gallery of solutions," *European Trans. Telecommunications*, vol. 22, no. 4, pp. 168–178, 2011. [Online]. Available: http://dx.doi.org/10.1002/ett.1485

[63] S. Biaz and N. Vaidya, "Is the round-trip time correlated with the number of packets in flight?" in *Proc. 3rd ACM SIGCOMM conference on Internet measurement (IMC '03)*, Miami Beach (FL), USA, 2003, pp. 273–278.

[64] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for TCP," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 356–369, Jun. 2003.

[65] G. McCullagh and D. Leith, "Delay-based congestion control: Sampling and correlation issues revisited," Hamilton Institute, Technical Report, 2008.

[66] R. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. PFLDnet*, 2004.

[67] S. Rewaskar, J. Kaur, and D. Smith, "Why don't delay-based congestion estimators work in the real-world?" University of North Carolina at Chapel Hill, Dept. of Computer Science, Technical report TR06-001, Jan. 2006.

[68] D. Hayes, "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 100219A, Feb. 2010. [Online]. Available: http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf

[69] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM Computer Communications Review*, vol. 34, no. 2, pp. 85–96, Apr. 2004.

[70] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and TCP Vegas," in *Proc. IEEE INFOCOM*, New York, Mar. 1999, pp. 1556–1563.

[71] S. Shalunov, L. Dunn, Y. Gu, S. Low, I. Rhee, S. Senger, B. Wydrowski, and L. Xu, "Design space for a bulk transport tool," Internet2 Transport Group, Technical Report, May 2005.

[72] S. Liu, M. Vojnović, and D. Gunawardena, "Competitive and considerate congestion control for bulk data transfers," in *Proc. IEEE IWQoS*, Evanston (IL), USA, Jun. 2007.

[73] D. Damjanovic and M. Welzl, "MulTFRC: Providing weighted fairness for multimedia applications (and others too!)," *ACM SIGCOMM Computer Communications Review*, vol. 39, no. 9, Jul. 2009.

[74] ——, "An extension of the TCP steady-state throughput equation for parallel flows and its application in MulTFRC," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1676–1689, Dec. 2011.

[75] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): protocol specification," Internet Standards Track RFC 5348, IETF, Sep. 2008. [Online]. Available: http://tools.ietf.org/html/rfc

[76] J. Crowcroft and P. Oechslin, "Differentiated end-to-end internet services using a weighted proportional fair sharing TCP," *ACM SIGCOMM Computer Communications Review*, vol. 28, no. 3, pp. 53–69, Jul. 1998.
[77] T. Hacker, B. Noble, and B. Athey, "Improving throughput and maintaining fairness using parallel TCP," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004.
[78] T. Hacker and P. Smith, "Stochastic TCP: A statistical approach to congestion avoidance," in *Proc. PFLDnet 2008*, Manchester, Mar. 2008.
[79] F. Kuo and X. Fu, "Probe-aided MulTCP: an aggregate congestion control mechanism," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 1, pp. 17–28, Jan. 2008.
[80] N. Spring, M. Chesire, M. Berryman, V. Sahasranaman, T. Anderson, and B. Bershad, "Receiver based management of low bandwidth access links," in *Proc. IEEE INFOCOM*, Tel Aviv, Mar. 2000, pp. 245–254.
[81] P. Mehra, A. Zakhor, and C. De Vleeschouwer, "Receiver-driven bandwidth sharing for TCP," in *Proc. IEEE INFOCOM*, San Francisco, Apr. 2003.
[82] P. Key, L. Massoulié, and B. B. Wang, "Emulating low-priority transport at the application layer: a background transfer service," in *Proceedings of ACM SIGMETRICS*, New York, Jun. 2004.
[83] M. Crovella and P. Barford, "The network effects of prefetching," in *Proc. IEEE INFOCOM*, San Francisco, Apr. 1998, pp. 1232–1239.
[84] S. Dykes and K. Robbins, "Limitations and benefits of cooperative proxy caching," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 7, pp. 1290–1304, Sep. 2002.
[85] *Background Intelligent Transfer Service (Windows)*, Microsoft. [Online]. Available: http://msdn.microsoft.com/library/bb968799(VS.85).aspx
[86] L. Eggert and J. Touch, "Idletime scheduling with preemption intervals," in *Proc. 20th ACM Symposium on Operating Systems Principles (SOSP 2005)*, Brighton, Oct. 2005, pp. 249–262.
[87] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proc. 11th ACM Internet Measurement Conference (IMC'11)*, Berlin, Nov. 2011.

David Ros received his B.Sc. (with honors) and M.Sc. degrees, both in Electronics Engineering, from the Simón Bolívar University, Caracas, Venezuela, and his Ph.D. in Computer Science from the Institut National de Sciences Appliquées (INSA), Rennes, France. He is currently working as an Associate Professor at Télécom Bretagne, Rennes, in the Networks, Security and Multimedia (RSM) Department. His active research interests include: transport protocols and congestion control, network simulation and modeling, as well as quality of service issues in IP networks.

Michael Welzl passed his Ph.D. defense at the University of Darmstadt, Germany, with distinction in November 2002, and received his habilitation from the same University in June 2007. He spent two years as a research assistant at the Telecooperation department, University of Linz, Austria, before joining the faculty of the newly founded Institute of Computer Science at the University of Innsbruck, Austria in November 2001, where he led a research team on Network Support for Grid Computing. In May 2009, he joined the Department of Informatics of the University of Oslo as Associate Professor. He was appointed to a Full Professorship in September 2009. Since October 2011, he is also an Adjunct Professor at Swinburne University of Technology.