Lifetime Packet Discard for Efficient Real-Time Transport over Cellular Links

Andrei Gurtov[a] *gurtov@cs.helsinki.fi*
Reiner Ludwig[b] *reiner.ludwig@ericsson.com*
[a]Dept. of Computer Science, University of Helsinki, Finland
[b]Ericsson Research, Germany

Mobile cellular users often experience significant delay jitter that undermines quality of real-time applications. Delay jitter can cause unnecessary delivery of stale packets with passed playback deadline and duplicate packets retransmitted by the end host after experiencing a timeout. With Lifetime Packet Discard (LPD) a flow adaptive link can tailor the trade-off between the maximum delay jitter and reliability if quality of service requirements of a flow are known. We propose using an IP option to communicate the flow requirements to the link layer. The packet lifetime is set to the minimum of the data lifetime determined by application and the retransmit timeout value determined by the transport protocol if selective reliability is supported. For congestion-responsive flows, the link transmits only headers of expired packets to prevent unnecessary triggering of end-to-end congestion control. Our simulations show that LPD is efficient in reducing stale data delivery and increases the number of packets delivered in time for real-time flows. For semi-reliable flows throughput and goodput are improved because duplicate packet delivery is prevented.

I. Introduction

By definition, the usefulness of real-time data is limited by a certain deadline. For a video streaming application, the *data lifetime* is determined by the size of the playback buffer at the receiver. For a telemetric application, old measurement samples become obsolete when a new sample is recorded. If delivery across the network takes longer than the data lifetime, packets become *stale* and are typically discarded by the receiver.

Slow last-hop links, especially those provided by wide-area cellular networks, often have significant delay jitter. A measurement study of dial-up connections reported occasional delay jitter of several seconds due to link-layer error recovery by a modem [26]. Frequent delay spikes of 3 to 15 seconds were observed in a wide-area cellular network due to handovers [14].

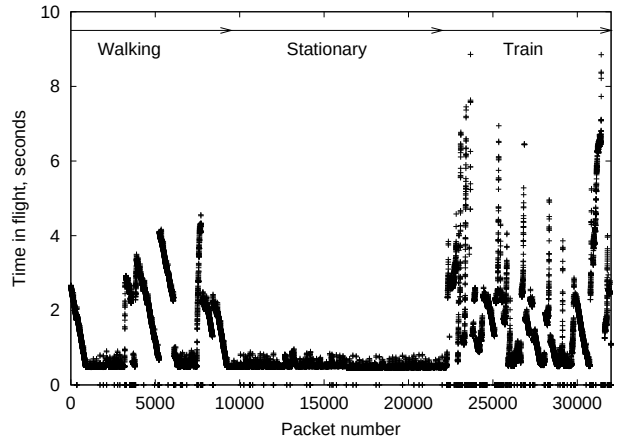
Consider Figure 1(a) showing delay jitter of UDP packets transmitted at 30 kbps over a cellular link. The trace is about an hour long, with 20 minutes of walking, 20 minutes in stationary conditions, and 20 minutes in a moving train. Packets of 500 bytes are transmitted at a constant bit rate downstream on a GPRS link [5]. Delay spikes of several seconds are clearly visible in mobile conditions. About 8% of all packets were lost. To confirm that delays were not only due to congestion, we repeated measurements using a congestion-responsive TFRC flow [11] and still observed significant delay jitter.

Many real-time applications account for delay jitter in the network by buffering data at the receiver. However, extensive buffering increases a start-up delay and harms interactivity for rewind-type operations. For certain types of media, such as live streaming, conversational audio or stock quote updates, significant delaying of the playback may not be an option.

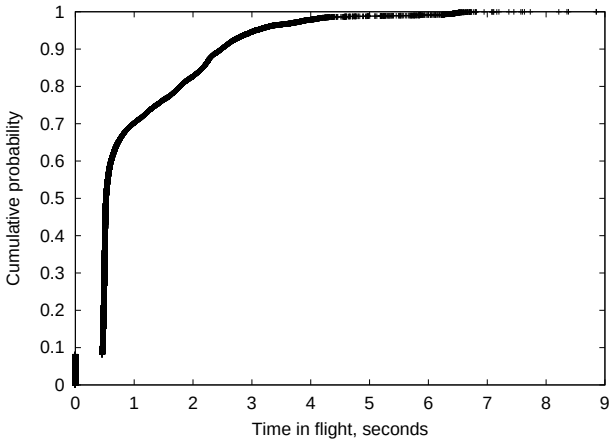
In summary, we believe that eventual disruptions in delivery of real-time data in a wireless environment are inevitable. The goal of our work is to make sure that such disruptions bring minimum dissatisfaction to the user.

The approach that we explore is to assign a lifetime to every packet at the sending host. It gives the link layer the necessary information on how persistent it should be on transmitting a packet. We show that Lifetime Packet Discard (LPD) improves performance by nearly eliminating delivery of stale and duplicate data over an expensive wireless link. Although the idea of LPD is not entirely new [41, 13], we are not aware of its systematic evaluation. We provide extensive simulations of LPD for constant bit rate (CBR), TCP, and TFRC flows.

We show that LPD can unnecessarily trigger end-to-end congestion control and suggest a method to avoid it. Furthermore, a solution to the problem of spurious timeouts in transport protocols is proposed using LPD. The rest of the paper is organized as fol-



(a) One-way delay of data packets



(b) Cumulative distribution of one-way delay

Figure 1: Delay jitter in a streaming test in a live GPRS network.

lows. In Section II, our view of the architecture for delivery of real-time data over wireless links is presented. Section III motivates this work by describing problems that can be solved by our approach. Section IV shows in detail how LPD avoids delivery of stale and duplicate data. In Section V, we evaluate the effect of LPD on performance of various types of flows. Section VI presents ideas for future work. Section VII concludes the paper.

II. The Architecture for Real-Time Transport

Generally speaking, limited bandwidth and battery power are two primary concerns for wireless users. Therefore, an efficient architecture would attempt to satisfy Quality of Service (QoS) requirements of all flows at the minimum cost of bandwidth and battery power.

II.A. Network Architecture

We assume the network architecture illustrated in Figure 2 that resembles the architecture of a GPRS cellular network [5]. The Radio Link Control (RLC) protocol provides recovery of error losses on the radio link between the mobile station (MS) and the Base Station Controller (BSC). Link-layer retransmissions and mobility signaling during handovers are main sources of delay jitter.

The last-hop router implements LPD by dropping packets with a remaining *packet lifetime* of less than it would take to deliver them over the wireless link. The packet lifetime can be used for several other purposes in the last-hop router, such as the earliest deadline scheduling [39]. In this paper, we only consider using the packet lifetime for discarding stale and duplicate data. When the real-time server is located in the Internet, clock synchronization between the last-hop router and the server would typically be required. The Network Time Protocol (NTP) [34] can provide sufficient accuracy for our purposes. When the real-time server is located close to the last-hop router or the network delay to the server is static and known, clock synchronization is not necessary. In our experience, it is a common practice for network operators to locate servers as close as possible to their intended users.

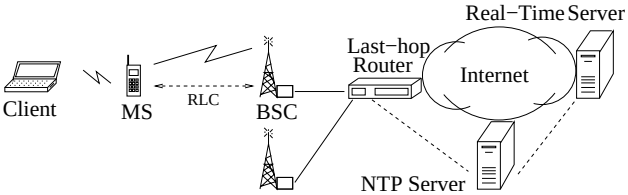


Figure 2: Network architecture.

In this paper we focus on downlink flows from the real-time server to the client. In practice, the client and the mobile station are often combined in a single device; thus, removing stale data from the bottleneck queue is straightforward for uplink flows. For downlink flows, a real-time server has no direct control over data buffered in the last-hop router; therefore, a mechanism to inform the router of the packet lifetime is needed.

Unfortunately, the time-to-live (TTL) field in IPv4 and IPv6 headers is too short to provide sufficient accuracy for the packet lifetime. Using a custom for-

mat of the IP timestamp option [36] could be one possibility for carrying the packet lifetime. However, a generic method for cross-layer communication would be helpful for future generations of transport protocols and wireless links. As an example, if a transport protocol could inform the link layer that it is tolerant to packet reordering, the link layer could avoid overhead of ordered packet delivery. Therefore, we defined a new IP option to carry the packet lifetime and information about preferences of a transport protocol.

Appendix A gives a detailed format of the proposed IP option. Using the IP option does not cause a layering violation, in contrast to using transport-layer timestamps. As routers operate on the networking layer, they are not supposed to examine other packet headers than the IP header [6]. A drawback of an IP option is that it costs several bytes of overhead and can load a router because of slow-path processing.

Sending the packet lifetime in an IP option is convenient when the server determines the data lifetime. If the receiver determines the data lifetime, a mechanism is needed to convey it to the sender or the last-hop router. Some existing wireless networks already allow a mobile station to send the maximum delay requirement to the base station and last-hop router using link layer signaling [1].

If all packets in a flow have the same lifetime, then only the first packet in the flow could have the IP option. This approach eliminates overhead of having an option in every packet, but raises concerns with managing per-flow state in routers and robustness for unreliable flows. Although these concerns can be partly solved with periodic state refreshments, for example using ICMP messages, including lifetime information in every packet appears more suitable.

II.B. End-to-End Real-Time Transport

We believe that future transport protocols for real-time data will support 1) selective reliability and 2) TCP-friendly congestion control. Real-time data can have tight delivery constrains; recovery of lost packets through retransmissions is not always feasible. However, it was shown that selective reliability is highly beneficial for certain types of data, such as a compressed MPEG-4 video stream [9]. By recovering important packets within the playback delay, the perceived quality of the application can be significantly enhanced. SR-RTP is a backward-compatible RTP extension that supports selective reliability [9]. PR-SCTP is another example of a partially reliable transport protocol [40].

Congestion control is a general requirement for all Internet flows [10]. However, the oscillatory nature of TCP congestion control may not be desirable for real-time applications. The notion of TCP friendliness permits a smoother transmission rate if the average rate is the same as of a TCP flow in similar conditions. TCP Friendly Rate Control (TFRC) is one of proposed slowly responsive congestion control algorithms [11]. Datagram Congestion Control Protocol (DCCP) [23] is a new unreliable transport protocol that allows applications to use TFRC congestion control.

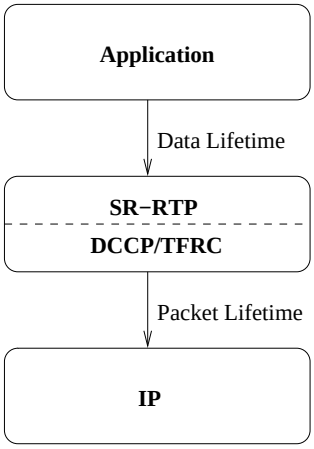


Figure 3: End-to-end transport for real-time flows.

Figure 3 shows how SR-RTP and DCCP fit into our architecture. The application passes a data object to SR-RTP for transmission indicating a maximum tolerable delivery delay (the data lifetime). The SR-RTP protocol verifies if the data object can be delivered within the data lifetime. If retransmissions are feasible within the data lifetime, SR-RTP sets the packet lifetime to the value of the retransmit timeout. Otherwise, the data lifetime is copied to the packet lifetime. The DCCP protocol provides TCP friendly rate control to SR-RTP.

III. Motivation of the Approach

In this section we argue why controlling the packet lifetime in the network is a right approach from an architectural point of view. A practical argument is that since many wireless networks employ billing based on the amount of transferred data, users become particularly concerned about the usefulness of the data they receive.

III.A. Flow Adaptive Link Layer

It is well known that wireless links can potentially introduce high loss rates on data traffic. To effectively carry IP traffic, most modern wireless networks de-

ploy retransmissions at the link layer. The link persistency is defined as time the link protocol attempts to recover a corrupted packet before discarding it and proceeding with transmission of other packets. Previous work has shown that the link layer operating on small data blocks is more efficient than purely end-to-end error recovery [29].

A natural problem for a link carrying a mixture of reliable and real-time traffic is how persistent it should be on a given packet. It was shown that high persistency is beneficial for reliable transport protocols, such as TCP. In opposite, real-time flows favor timely delivery over reliability and require low persistency. Existing link layers do not discriminate among data packets and use the same persistency for reliable and real-time data, thus giving a non-optimal trade-off.

A concept of a flow-adaptive link layer was proposed to tailor link behavior to demands of various application flows [31]. The proposed solution is a simple heuristic of being highly persistent on TCP packets and low persistent on UDP packets. That paper itself recognized that this crude discrimination is not sufficient, as there are UDP applications, such as the Network File System (NFS), that assume nearly reliable delivery from the network. Furthermore, the choice of persistency for each class is arbitrary and may not correspond to actual requirements of applications.

An improvement [42] over a simple TCP/UDP heuristic is to use quality of service information from DiffServ code points [3]. However, there are only 256 code points available and this number may not be sufficient to convey the packet lifetime with sufficient granularity. With our approach, every packet has exact information about its maximum tolerable delay. This information allows a flow-adaptive link to optimally control the delay versus loss probability in the presence of transmission errors.

III.B. Competing Error Recovery

Another problem resulting from uncontrolled link layer retransmissions is possible competition between link layer and end-to-end error recovery. Because link error recovery is observed as delay jitter by the end hosts, the retransmit timer of a transport protocol can expire prematurely. Spurious timeouts trigger unnecessary retransmissions and congestion control.

The Eifel algorithm [28] was proposed as an end-to-end solution for detecting [30] and recovering spurious TCP timeouts [27, 16]. The Eifel algorithm uses the TCP timestamp option to determine if arriving acknowledgments after a timeout refer to original or to retransmitted segments. F-RTO is an alternative end-to-end proposal for the problem of spurious timeouts in TCP and SCTP [38].

There are several advantages of our approach versus end-to-end solutions for real-time protocols:

- End-to-end protocols rely on delivery of old packets in the network and suppress transmission of duplicate packets after a delay. Our approach allows the end host to retransmit a fresh version of data after a delay.

- End-to-end solutions proposed so far are for TCP or SCTP, that are window-based protocols acknowledging every or at least every other packet. Our approach is well suited for rate-based real-time protocols with infrequent acknowledgments. End-to-end proposals would be inefficient for such protocols.

- Ideally, our approach entirely avoids delivery of duplicate packets. End-to-end solutions often require several unnecessary retransmissions to detect that a timeout was spurious.

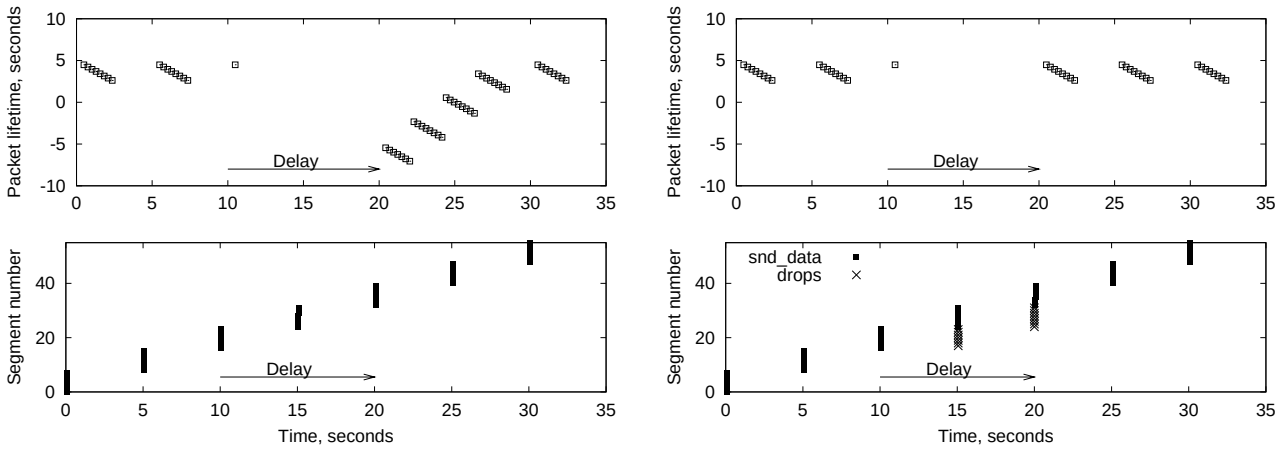
The strong side of end-to-end solutions is that false congestion control actions can be easily undone at the sender. However, our approach can be complemented by a mechanism to undo congestion control at the end hosts.

III.C. Application Empowerment

A popular paper on the next generation of protocols [7] argues that the application should be given control over recovery from lost and delayed packets. Indeed, the application may not need recovery of that particular data object or can re-generate a fresh version of it for retransmission. Our approach supports this principle by empowering the application to control for how long time the network should try to deliver the data.

Another trade-off that should be under the control of the application is a maximum transmission burst size versus a maximum queuing delay. The size of the link buffer sets this trade-off in the network. Having a small buffer reduces the queuing delay and possibly the amount of stale data delivered to the receiver.

However, a small link buffer can also cause undesirably high packet loss rates for a bursty real-time application with a variable bit rate data encoding. With our approach, the network buffer can be sufficiently large to accommodate bursty sources, because the application can explicitly limit the maximum queuing delay.



(a) Without LPD

(b) With LPD

Figure 4: Disruption introduced to an unresponsive ON/OFF flow by a delay spike.

Discarding stale data in the network is particularly useful when the data lifetime is comparable with round trip time of the path. In this case, end-to-end mechanisms to compensate for delay variation cannot adapt effectively. However, a situation when a significant fraction of packets persistently expires in the network should be avoided. On a network path with multiple congestion points, packets that expire after crossing a congested router consume resources from useful traffic. Therefore, an application or transport protocol should detect persistent expiration and adapt by increasing the packet lifetime.

IV. Lifetime Packet Discard

IV.A. Preventing Stale Packet Delivery

In this section, we show how LPD can improve the performance of UDP flows that are unresponsive to congestion. Packets buffered in the network become stale and are unnecessarily transmitted after a delay spike producing two negative effects:

- Transmitting stale data wastes resources.
- Transmitting stale data delays delivery of packets that contain fresh data.

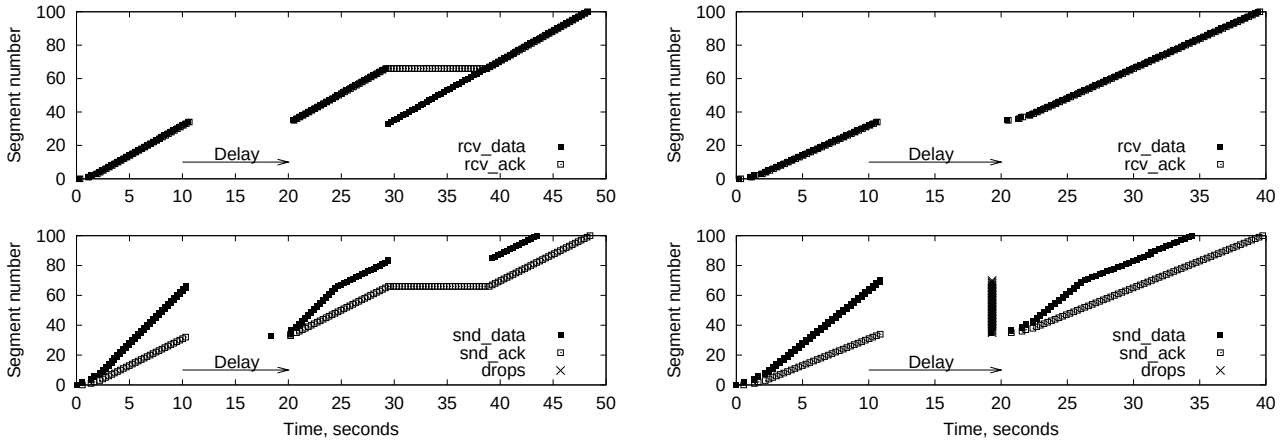
Figure 4(a) shows a trace from ns2 simulator of an ON/OFF UDP flow experiencing a 10 second delay spike. In this example, the packet lifetime is set to five seconds that equals the interval at which the application generates new data objects. The bottom graph shows packet sequence numbers at the sender and the top graph shows the remaining lifetime of arriving packets at the receiver. Negative values mean that the packet is stale and should be discarded. When the delay spike starts on the 10th second, no packets are delivered until it ends, but newly arriving packets get queued in the last-hop router. When the delay spike ends on the 20th second, for following seven seconds the link delivers only stale packets. The backlog of stale packets prevents fresh updates to be delivered to the receiver.

Figure 4(b) shows a similar scenario when the router implements LPD. Immediately when the delay spike ends, fresh updates are delivered to the receiver. Furthermore, no stale packets are sent over the wireless link that saves resources.

IV.B. Preventing Duplicate Packet Delivery

Transports protocols, such as TCP, SR-RTP [9], HPF [25], and PR-SCTP [40], provide some degree of reliability by retransmitting lost packets. A packet is considered lost when a retransmission timer expires at the sender. When the delay in the network increases, the timer can expire prematurely. As a result, two or more duplicate packets can be transmitted over the wireless link wasting resources. Below we describe this situation for TCP in detail.

When a sudden delay occurs in the network that exceeds the current value of the TCP retransmission timer, the oldest outstanding segment is retransmitted. Since data segments are delayed but not lost, the retransmission is unnecessary and the timeout is spurious. A spurious TCP timeout is shown in Figure 5(a). The delay spike is generated between the 10th and 20th seconds in this test. The first retrans-



(a) Without LPD

(b) With LPD

Figure 5: Duplicate packet delivery in a TCP flow after a spurious timeout. The receiver trace (top) and the sender trace (bottom).

mission that occurs on the 17th second is also delayed. The sender interprets the acknowledgment generated by the receiver in response to the delayed segment as related to the retransmission, not the original segment. TCP retransmits all outstanding segments using the slow start algorithm. Such a retransmission policy is referred to as *go-back-N*, because the sender forgets about all segments it has earlier transmitted. On the 30th second, retransmitted segments arrive to the receiver and generate duplicate acknowledgments as the original segments have already been delivered.

In summary, spurious timeouts in a semi-reliable transport protocol cause two problems:

- Unnecessary end-to-end retransmissions produce duplicate packet delivery over a wireless link.
- Congestion control is disturbed. In the short run, retransmissions using slow-start can overload the network. In the long run, the network can be underutilized.

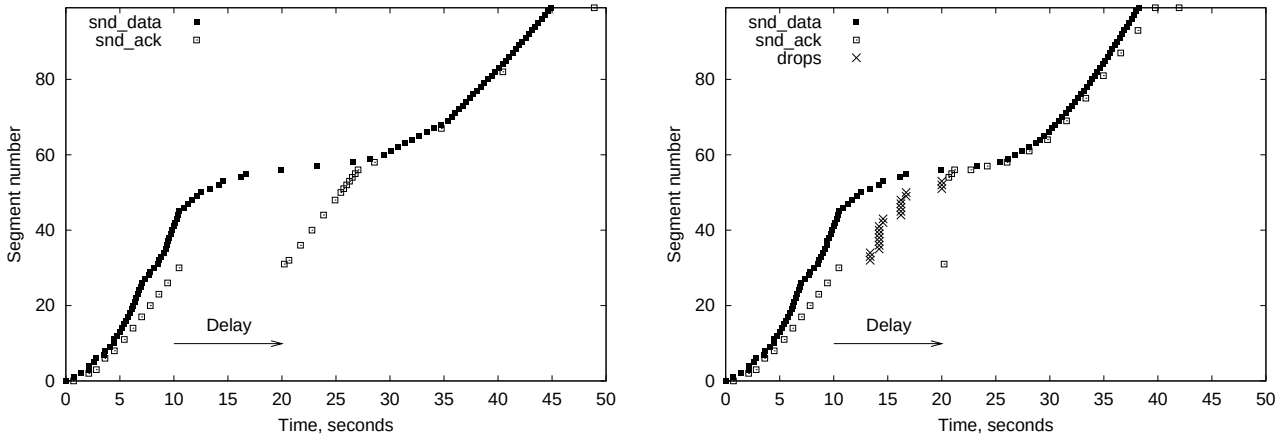
Figure 5(b) shows the same scenario with LPD. The packet lifetime is set to the retransmit timeout value at the TCP sender. When the sender times out and retransmits the first segment on the 17th second, the last-hop router has dropped all outstanding segments from the flow. When the delay spike ends, the sender immediately gets an acknowledgment for the retransmitted segment and continues retransmitting segments using go-back-N. Since all originally transmitted segments are dropped, no duplicate packets are delivered to the receiver.

LPD alone does not solve the second problem triggered by spurious timeouts, unnecessary congestion control at the end hosts. In Figure 5(b) the problem is not significant because the bandwidth-delay product of the path is small. However, a reduction of the transmission rate after a spurious timeout can reduce performance on paths with a larger bandwidth-delay product [16]. In the next section we discuss how this problem can be alleviated.

Another possible complication is that setting the packet lifetime correctly can be difficult for transport protocols that update the retransmit timer after sending a segment. During bulk data transmission in TCP, the retransmit timer is offset by one RTT because the timer is restarted upon every acknowledgment [33, 35]. Furthermore, the retransmit timeout value can be updated more frequently than once per RTT [19]. For optimal performance (to avoid occasional dropping of valid packets and delivery of duplicates) the transport protocol should not update the retransmit timeout value nor restart the timer until the oldest outstanding segment is acknowledged. However, if RTT increases significantly, the timer has to be updated to avoid many spurious timeouts.

IV.C. Interactions with End-to-End Congestion Control

End-to-end congestion control in the Internet is based on an assumption that almost all packet losses are due to congestion [18]. Hence, discarding expired packets in the network incorrectly triggers a reduction of the transmission rate at the sender. In this section, we ex-



(a) Without LPD

(b) With LPD

Figure 6: Behavior of end-to-end TFRC congestion control after a delay spike.

plore how TFRC congestion control reacts to packet drops by LPD.

Figure 6(a) shows a sender trace of a TFRC flow when a 10 second delay spike is introduced. We assume the data and packet lifetime of five seconds in this example. The sender gets no feedback during the delay spike and gradually slows down to eventually transmit one packet per RTT. When the delay spike ends, several feedback packets that were delayed arrive to the sender. It takes about 15 seconds after the delay spike ends for the sender to return to the normal transmission rate. Because the receiving *application* discards stale data, the transport protocol does not see any data loss in this example. The receiver reports no loss events and the sender does not further invoke congestion control.

Figure 6(b) shows the same flow with LPD. Until the delay spike ends, the sender's behavior is identical to Figure 6(a), as expected. But later, feedback packets report packet losses and the sender keeps the transmission rate reduced. Still, the sender is able to reach a higher sequence number than without LPD because unnecessary transmission of stale packets is eliminated. However, in tests with higher link bandwidth we observed that unnecessary triggering of congestion control reduces performance.

Figure 7 shows how the undesired triggering of end-to-end congestion control can be avoided. We call our solution *headercasting*. The idea of headercasting is to transmit only IP and transport headers of stale packets. The receiver knows upon getting a header that there was no packet loss due to congestion and there is no reason to trigger congestion control at the sender. When a feedback packet arrives to the sender, it can undo the reduction of the transmission rate that occurred because of no-feedback timeouts. A flow uses a bit in the IP option to indicate if it is interested in headercasting or its packets can be simply dropped. Headercasting requires re-computing checksums and may not work in the presence of IPsec.

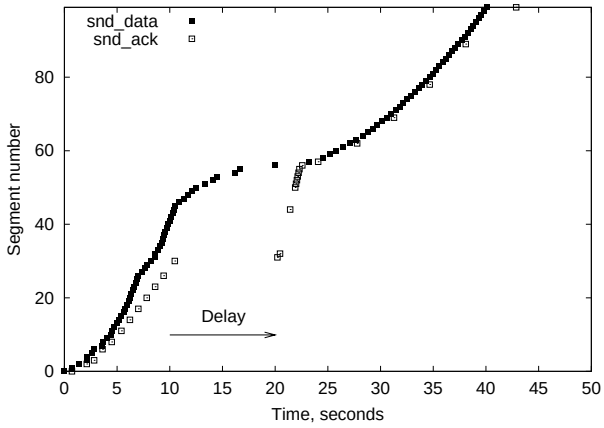


Figure 7: Interference of LPD with end-to-end congestion control is resolved by transmitting headers of stale packets.

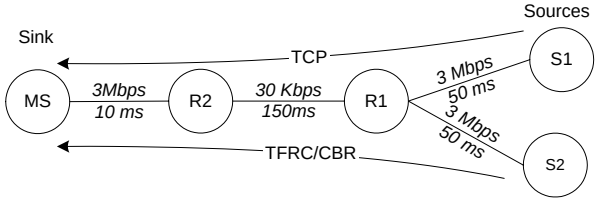


Figure 8: Simulation setup in ns2.

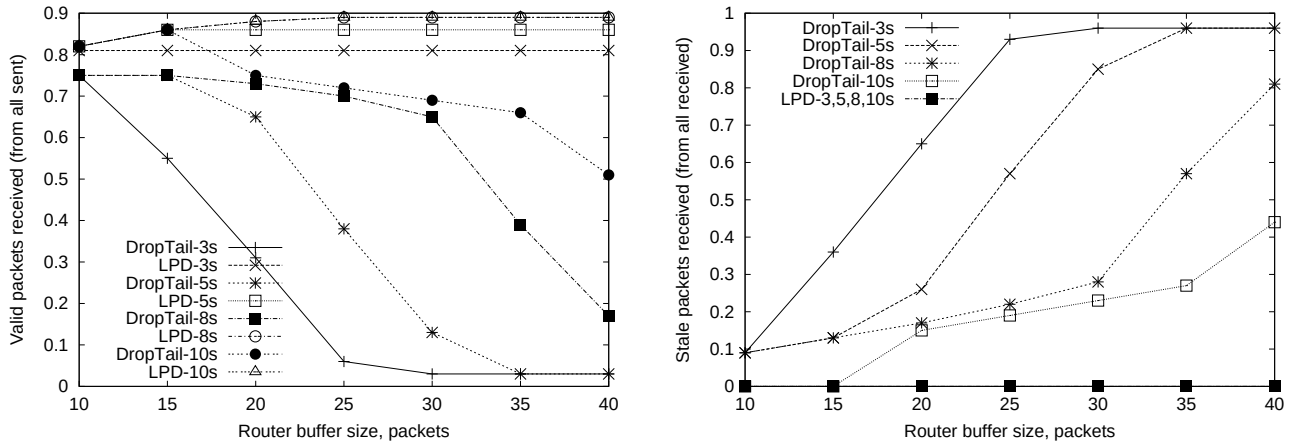


Figure 9: Effect of LPD on performance of a single CBR flow with various packet lifetimes.

V. Performance Evaluation

We run an extensive set of ns2 simulations to explore the effects of LPD on CBR, TCP, and TFRC flows. Figure 8 shows the topology we have used; it resembles the setup of a GPRS user. We used the state-of-the-art TCP with SACK, delayed acknowledgments, limited transmit, segments of 1000 bytes, the retransmit timer compliant to RFC2988, and the receiver window of 50 segments. Delay jitter was introduced by inserting delay spikes of 7-10 seconds at a 30-second interval according to real-world traces in Figure 1(a). We present simulations using Drop-Tail buffers, but we also experimented with Random Early Detection (RED) [12] with similar results. LPD is executed in R1. Our simulation scripts are publicly available [15].

V.A. Constant Bit Rate Flows

In this section, we evaluate performance of a CBR flow in the presence of delay jitter in the network. We are interested in two main performance metrics. A fraction of valid packets (received within their lifetime) describes perceived quality of the application. A fraction of packets delivered stale describes how efficient are the use of wireless link bandwidth and battery power of the mobile station.

Figure 9 shows performance of a 25-kbps CBR flow with various values of data lifetime. In this test, the packet lifetime is assumed to be determined by the application according to the available playback buffer. With a Drop-Tail buffer, fewer packets are delivered valid when the data lifetime becomes tighter. Furthermore, the number of valid packets quickly decreases with increase of the buffer size in the last-hop router. The fraction of stale packets increases with a smaller data lifetime or a larger buffer size. Eventually, nearly all packets are delivered stale.

When LPD is enabled, performance for different data lifetimes and buffer sizes is stable and nearly optimal. About 85% of the packets are delivered valid. At the same time, no stale packets are sent over the wireless link.

V.B. TCP Flows

In this section, we explore how LPD can prevent delivery of duplicate segments in the presence of spurious TCP timeouts. We compare performance of TCP over LPD, the standard TCP, and TCP with the Eifel algorithm. The download time reflects the perceived performance of the application. The number of duplicate segments received shows efficiency of the resource use. The packet lifetime is set to the retransmit timeout value of the TCP sender.

In Figure 10, download time of TCP over Drop-Tail is 10-20% higher than over LPD. TCP with the Eifel algorithm has only a slightly higher download time than TCP over LPD for large buffers. However, when the buffer size is small, Eifel suffers from genuine retransmission timeouts due to congestion losses [16]. Its download time is variable and up to 60% higher than of TCP over LPD. When congestion control is not undone after a spurious timeout with the Eifel algorithm (Eifel-CC), then its performance is even worse compared to TCP over LPD.

The number of duplicate segments for standard TCP grows with a larger buffer size. Up to 16% of all delivered segments are duplicates. Both the Eifel algorithm and TCP over LPD perform well in reducing the number of duplicate delivered segments below 3%.

We would like to note that the goal of LPD was not

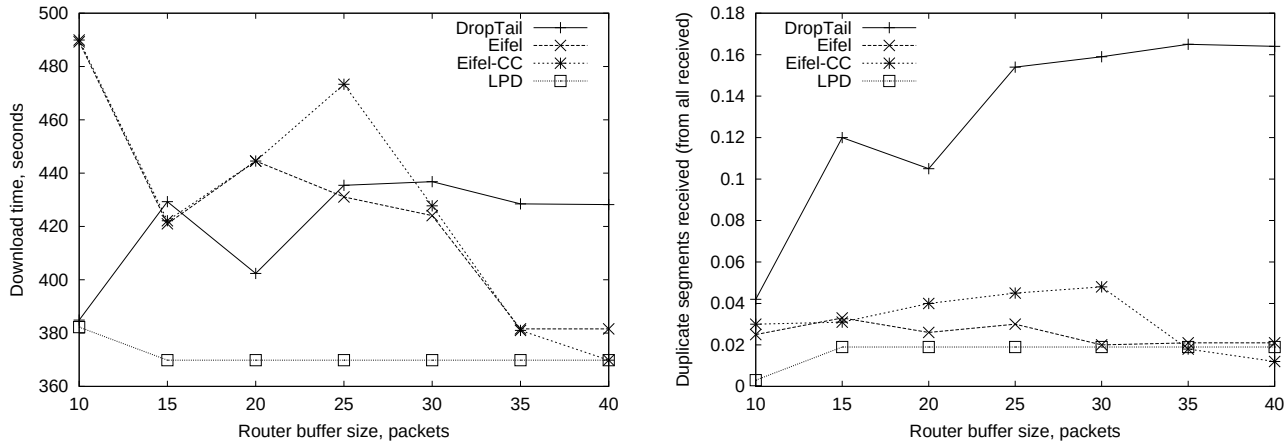


Figure 10: Comparison of TCP with LPD, TCP with the Eifel algorithm, and standard TCP. The packet lifetime is set to the TCP retransmit timeout.

to beat end-to-end solutions, such as Eifel, for non-real-time flows. We believe that end-to-end solutions work well for fully reliable protocols. Instead, main benefits of LPD are for real-time flows as described in Section III.A. The reason we used TCP for evaluation is lack of support for semi-reliable transport protocols in ns2.

V.C. TFRC and TCP Flows

In this section, we compare concurrent TCP and TFRC flows with and without LPD. For TCP flows, we use download time and the number of received duplicate segments as performance metrics. The packet lifetime of TCP segments is set to the TCP retransmit timeout value. For TFRC flows, we look at the total size of received stale packets. The packet lifetime of TFRC packets is set to 3, 5, 8 or 10 seconds to correspond to various data lifetimes.

In Figure 11, the two top graphs show TFRC performance and the two bottom graphs show TCP performance. TFRC flows over Drop-Tail receive less valid packets than over LPD for given data lifetime. The difference is growing with an increase in the router buffer size. Similarly, the number of bytes delivered stale is large for Drop-Tail and small for LPD.

For TCP, download time over LPD is lower than over Drop-Tail, with a growing difference when the data lifetime in the TFRC flow gets lower. The number of unnecessary retransmissions is low and stable for a TCP flow over LPD, but grows quickly over Drop-Tail with the increasing buffer size.

Figure 12(a) and 12(b) show in detail why performance with LPD is better. Unnecessary go-back-N retransmissions are prevented for the TCP flow when LPD is enabled. Stale packets of the TFRC flow are discarded when LPD is enabled that saves bandwidth. Finally, throughput of TCP and TFRC flows are closer, which is a crude indicator that LPD improves fairness between flows.

VI. Considerations for Future Work

In this section, we discuss issues that should be considered in more detail in future work. First, we discuss concerns of using LPD in the presence of IP security and incremental deployment. Second, a possibility of dropping entire application data units with stale fragments is considered.

VI.A. Operation with IPsec

The use of IP security (IPsec) [22] can affect LPD if the router is unable to read the packet lifetime from an IP option or recompute checksums for headercasting.

The IPsec architecture provides two modes for payload encryption, the transport mode and the tunnel mode. In the transport mode, IP options for IPv4 and IPv6 are not encrypted [20]. Therefore, the router can read the packet lifetime but cannot recompute checksums. In the tunnel mode, some outer fields are constructed from the original header, but IP options are "never copied" according to RFC2401 [22]. Therefore, the router can have difficulties executing LPD in the tunnel IPsec mode.

A similar problem with the Explicit Congestion Notification (ECN) and DiffServ was solved by a correction to RFC2401 that requires copying the relevant fields to the outer header. A similar correction for IP options was proposed stating that the "post-IPsec code may insert/construct options for the outer header" [21]. Furthermore, the IPsec tunnel is often

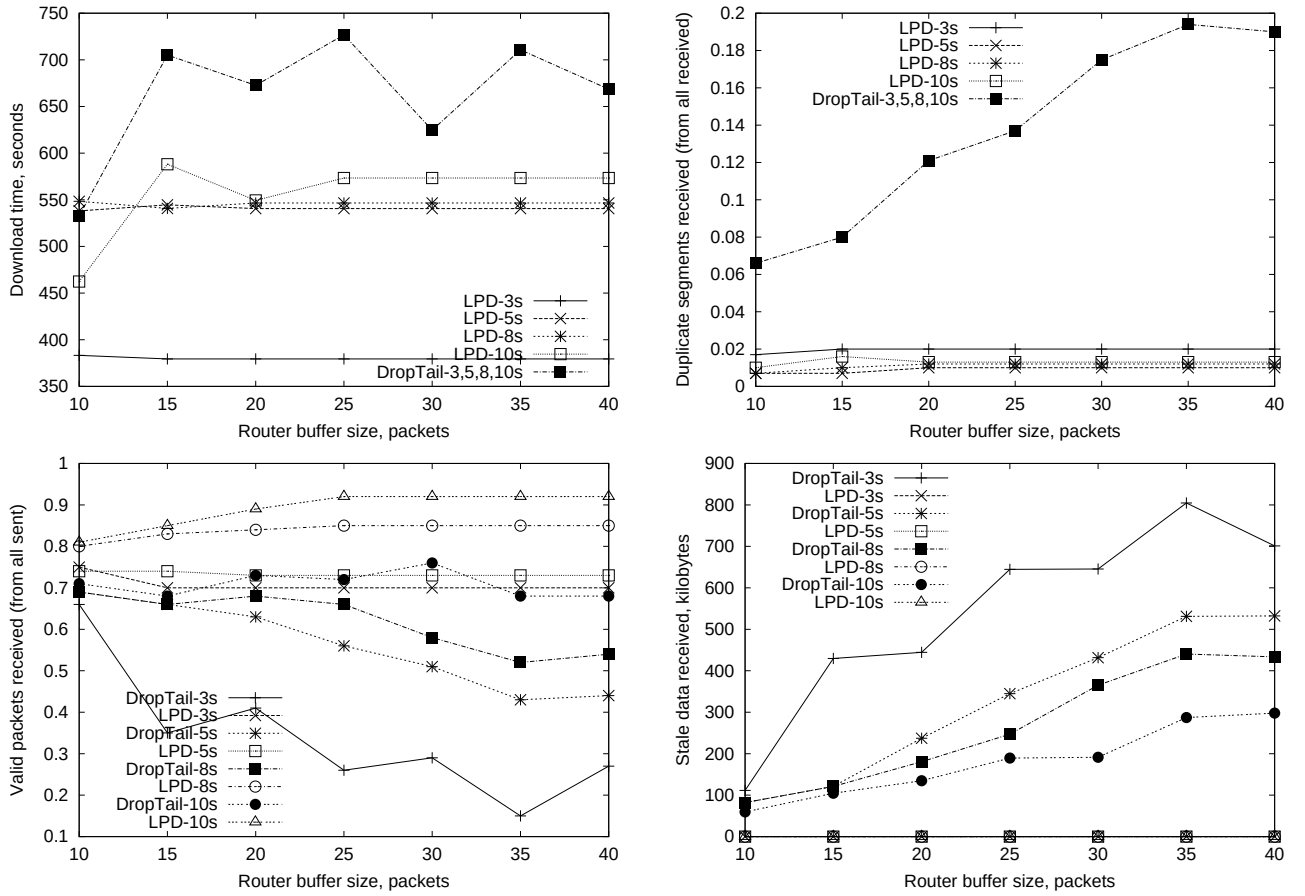


Figure 11: Effect of LPD on performance on concurrent TFRC (top) and TCP (bottom) flows for various data lifetimes.

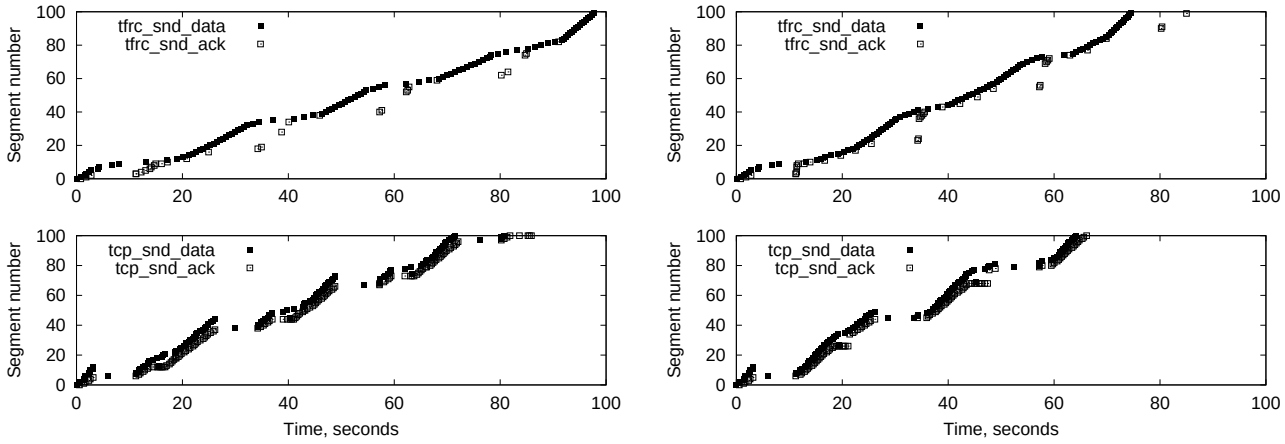
terminated before the last-hop router that allows LPD execution without problems.

VI.B. Deployment Concerns

As for many new proposals, LPD raises some deployment concerns. The sender end host has to be modified to set the packet lifetime. The last-hop router needs a modification to check for expired packets. However, real-time transport protocols supporting TCP-friendly congestion control and selective reliability are still in development. Therefore, it is not a significant burden to complement them now with packet lifetime functionality. Furthermore, in our experience software in last-hop routers in cellular networks is updated frequently that facilitates deploying LPD. We recognize that it is not feasible to require that all hosts and routers be upgraded in order to deploy a new solution. Fortunately, LPD can be incrementally deployed starting from a limited number of real-time servers.

The server can discover if routers in the path to the destination support, or at least tolerate, carrying the packet lifetime as an IP option. The IP timestamp option is a part of the standard [36] and should be supported by all Internet routers. However, some misbehaving routers or firewalls can discard IP datagrams with IP options. To avoid unnecessary overhead if LPD is not supported and prevent dropping of packets if routers do not tolerate an IP option, the sender host should first probe the destination by sending a train of knowingly stale and valid packets. Based on the transport layer acknowledgments or ICMP "packet discarded" notifications the sender host can decide whether to send the IP option with packet lifetime for a given destination.

As a simple experiment, we sent ICMP ping messages from a host in ICSI, Berkeley to a host in UH, Helsinki. The path contained 19 hops. All pings containing the IP timestamp option returned successfully, but the RTT was 214-515 milliseconds, which is more variable than a nearly constant RTT of 191 milliseconds with "normal" pings. This variability is probably because of slow-path processing for packets with IP options and overhead of adding a timestamp to an ICMP message in routers. Finally, when we pinged a host located behind a firewall in Helsinki, pings con-



(a) Without LPD

(b) With LPD

Figure 12: LPD improves goodput, throughput and fairness of TCP and TFRC flows when delay spikes occur in the network. The packet lifetime of the TFRC flow is five seconds. The packet lifetime of the TCP flow is set to the retransmit timeout.

taining an IP option did not get through, but "normal" pings did.

VI.C. Discarding Application Data Units

The concept of application-level framing suggests that data objects should be delivered over the network as application data units (ADU) [7]. One ADU corresponds to the data entity convenient for the application, such as a video frame. The ADU size can exceed the maximum transmit unit (MTU) of an IP network. Therefore, large ADUs are transmitted as several IP datagrams, or as fragments of a single datagram.

A congested router in today's Internet drops IP datagrams arbitrarily between ADUs. When the application has tight real-time constraints, there is no possibility to recover lost ADU fragments. Therefore, the receiver host can drop large ADUs with only a few missing fragments. Consequently, the playback quality of the application can deteriorate to an unacceptable level. A similar problem appeared previously in the context of dropping ATM cells from TCP segments [37].

To prevent this problem, the router could drop entire ADUs while possibly taking their priority and the lifetime into consideration. However, if ADU framing is at a transport or upper layer, the router cannot classify packets into ADUs without snooping into transport or application headers. Such "layering violation" is undesirable from the point of view of the Internet architecture [6].

A possible solution is to rely on IP fragmentation for delivery of large ADUs. In this case, the router can identify and discard an entire ADU based on the standard datagram identifier field in the IP header without snooping into upper layer headers. We are investigating possible performance benefits of this approach for real-time data transport.

VII. Conclusions

Emerging real-time transport protocols combine selective retransmissions and TCP-friendly congestion control. In our architecture for efficient real-time transport over wireless links, new transport protocols are reinforced with Lifetime Packet Discard at the link layer. In this paper we evaluated effects of LPD on CBR, TCP, and TFRC flows.

The lifetime of a packet is set to the minimum of the data lifetime determined by the application and the retransmit timeout value determined by the transport protocol. The packet lifetime coordinates operation of link and transport layers. A flow adaptive link can use the packet lifetime for determining the number of retransmission attempts, data encoding, and scheduling of transmissions. The transport protocol can recover lost packets faster by deploying a more aggressive retransmission timer [33], because the cost of spurious timeouts is minimal with LPD.

End-to-end congestion control in the Internet is based on the assumption that all packet losses result from congestion. Therefore, discarding stale data in the network can incorrectly trigger end-to-end congestion control reducing the transmission rate at the

sender. We show that transmitting only headers of packets with expired lifetime prevents interactions with congestion control and still provides significant efficiency gains. The benefit of LPD is higher for a larger size of the bottleneck buffer. Below we provide some arguments on why using a very small buffer is not a desirable solution.

- A small buffer causes a short congestion avoidance cycle that generates frequent packet drops.
- A small buffer is inadequate for smoothing bursty traffic generated by variable bit rate codecs.
- A larger buffer can accommodate bandwidth variation occurring during vertical handovers.
- The current practice is to use large buffers in cellular links [32]. A per user buffer of 50-200 kilobytes was measured in a GPRS network [17].

Using an active queue management algorithm, such as RED [12], may seem an attractive alternative to LPD. However, we run tests using RED instead of Drop-Tail and obtained similar results. Drop From Head (DFH) [24] could be used together with LPD to increase performance.

In future work, we will consider use of explicit loss notification [2] as an alternative to headercasting for avoiding unnecessary triggering of congestion control by expiration losses. The cumulative explicit transport error notification [8] takes a different approach from providing fine-grain feedback per every discarded packet. Instead, routers tell the sender the average fraction of lost packets due to transmission errors. The sender makes a smaller decrease in the transmission rate on individual loss events. Preliminary evaluation of this approach suggests that it is effective in improving TCP throughput over links with error losses while remaining congestion-friendly to other TCPs. We expect these results to be directly applicable to our work.

Acknowledgments

Authors thank Rajiv Chakravorty and members of the Sahara project at UC Berkeley for valuable suggestions. Many thanks to Sally Floyd and Randall Stewart for comments on the paper.

References

[1] 3GPP. TS 23.107: QoS concept and architecture, Mar. 2002.

[2] H. Balakrishnan and R. Katz. Explicit loss notification and wireless web performance. In *Proc. of Globecom Internet Mini-Conference*, Nov. 1998.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. IETF RFC 2475, Dec. 1998.

[4] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM Computer Communication Review*, 32(1):20–30, Jan. 2002.

[5] G. Brasche and B. Walke. Concepts, services and protocols of the new GSM phase 2+ general packet radio service. *IEEE Communications Magazine*, 35(8):94–104, Aug. 1997.

[6] D. D. Clark. The design philosophy of the DARPA internet protocols. In *Proc. of ACM SIGCOMM'88*, Aug. 1988.

[7] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. of ACM SIGCOMM'90*, Sept. 1990.

[8] W. Eddy, S. Ostermann, and M. Allman. New techniques for making transport protocols robust to corruption-based loss. Submitted for publication, July 2003.

[9] N. Feamster and H. Balakrishnan. Packet loss recovery for streaming video. In *Proc. of 12th International Packet Video Workshop*, Apr. 2002.

[10] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. on Networking*, 7(4):458–472, Aug. 1999.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. of ACM SIGCOMM'00*, Aug. 2000.

[12] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. on Networking*, 1(4):397–413, Aug. 1993.

[13] D. J. Goodman, R. A. Valenzuela, K. T. Gayliard, and B. Ramamoorthi. Packet reservation multiple access for local wireless communications. *IEEE Trans. on Communications*, 37(8):885–890, Aug. 1989.

[14] A. Gurtov. Effect of delays on TCP performance. In *Proc. of IFIP Personal Wireless Communications (PWC'01)*, Aug. 2001.

[15] A. Gurtov. Extensions of ns2 simulator. Available at http://www.cs.helsinki.fi/u/gurtov/ns/, Nov. 2003.

[16] A. Gurtov and R. Ludwig. Responding to spurious timeouts in TCP. In *Proc. of IEEE INFOCOM'03*, Apr. 2003.

[17] A. Gurtov, M. Passoja, O. Aalto, and M. Raitola. Multi-layer protocol tracing in a GPRS network. In *Proc. of IEEE Vehicular Technology Conference (VTC'02 Fall)*, Sept. 2002.

[18] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM'88*, Aug. 1988.

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

[20] S. Kent. IP Encapsulating Security Payload (ESP). Work in progress, draft-ietf-ipsec-esp-v3-06.txt, July 2003.

[21] S. Kent. Security architecture for the Internet Protocol. Work in progress, draft-ietf-ipsec-rfc2401bis-00.txt, Oct. 2003.

[22] S. Kent and R. Atkinson. Security architecture for the Internet Protocol. IETF RFC 2401, Nov. 1998.

[23] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion control without reliability. Available at http://www.icir.org/kohler/dccp/, May 2003.

[24] T. Lakshman, A. Neidhardt, and T. J. Ott. The Drop from Front Strategy in TCP and in TCP over ATM. In *Proc. of IEEE INFOCOM'96*, Mar. 1996.

[25] J. Li, S. Ha, and V. Bharghavan. HPF: a transport protocol for heterogeneous packet flows in the Internet. In *Proc. of IEEE INFOCOM'99*, Mar. 1999.

[26] D. Loguinov and H. Radha. Measurement study of low-bitrate internet video streaming. In *Proc. of the First ACM SIGCOMM Internet Measurement Workshop (IMW-01)*, Nov. 2001.

[27] R. Ludwig and A. Gurtov. The Eifel response algorithm for TCP. Work in progress, draft-ietf-tsvwg-tcp-eifel-response-04.txt, Oct. 2003.

[28] R. Ludwig and R. H. Katz. The Eifel algorithm: Making TCP robust against spurious retransmissions. *ACM Computer Communication Review*, 30(1):30–36, Jan. 2000.

[29] R. Ludwig, A. Konrad, A. D. Joseph, and R. H. Katz. Optimizing the end-to-end performance of reliable flows over wireless links. *ACM/Baltzer Wireless Networks*, 8(2):289–299, Mar. 2002.

[30] R. Ludwig and M. Meyer. The Eifel detection algorithm for TCP. IETF RFC 3522, Apr. 2003.

[31] R. Ludwig and B. Rathonyi. Link layer enhancements for TCP/IP over GSM. In *Proc. of IEEE INFOCOM'99*, Mar. 1999.

[32] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, and A. Joseph. Multi-layer tracing of TCP over a reliable wireless link. In *Proc. of ACM SIGMETRICS'99*, May 1999.

[33] R. Ludwig and K. Sklower. The Eifel retransmission timer. *ACM Computer Communication Review*, 30(3):17–27, July 2000.

[34] D. Mills. Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI. IETF RFC 2030, Oct. 1996.

[35] V. Paxson and M. Allman. Computing TCP's retransmission timer. IETF RFC 2988, Nov. 2000.

[36] J. Postel. Internet protocol. IETF RFC 791, Sept. 1981.

[37] A. Romanow and S. Floyd. The dynamics of TCP traffic over ATM networks. In *Proc. of ACM SIGCOMM'94*, Aug. 1994.

[38] P. Sarolahti, M. Kojo, and K. Raatikainen. F-RTO: A new recovery algorithm for TCP retransmission timeouts. Technical Report C-2002-07, University of Helsinki, Feb. 2002.

[39] S. Shakkottai and R. Srikant. Scheduling real-time traffic with deadlines over a wireless channel. *ACM/Baltzer Wireless Networks*, 8(1):13–26, 2002.

[40] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. SCTP partial reliability extension. Work in progress, draft-ietf-tsvwg-prsctp-01.txt, Aug. 2003.

[41] J. Wong and Y. Liu. Deadline based network resource management. In *Proc. of the International Conference on Computer Communications and Networks (ICCCN'00)*, Oct. 2000.

[42] G. Xylomenos. *Multi Service Link Layers: An Approach to Enhancing Internet Performance over Wireless Links*. PhD thesis, University of California at San Diego, 1999.

A. IP Option for Cross Layer Communication

Figure 13 is a proposal for an IPv4 option for cross-layer communication. Using an IP option seems to be the only feasible way to implement the concept. Enabling the router to access headers above IP would be a poor design choice and changing the basic format of IPv4 or IPv6 header is not realistic. However, the proposed format is preliminary. There are different alternatives to explore, such as creating a separate IP option for the lifetime, reordering, and corruption information versus combining them all in a single IP option.

Fields *type* and *length* have the same meaning as for all IPv4 options. The type could be any unallocated number; the length is eight bytes.

The field *reorder* tells how deep reordering the transport protocol is willing to tolerate. As an example, for standard TCP the value would be three, that is the number of duplicate acknowledgments required to trigger fast retransmit. This field could be set dynamically if the value of the duplicate acknowledgment threshold is made adaptive [4].

The *lifetime* field carries the packet lifetime in milliseconds and determines for how long time a packet can be kept in the network. If the value is less than 65535, the lifetime value is relative to the sender's clock. In this case every router can decrement it by the delay introduced to the packet. If the timestamp value is larger than 65535, it is absolute deadline time (modulo 24 hours from midnight UT plus 65535). This format requires that participating routers have synchronized clocks with the sender.

The *flags* field contains four flags. The *corruption-tolerable* bit tells that it is acceptable to deliver a packet with corrupted payload (that otherwise would be dropped by the link layer). This flag is useful when an application is resilient to errors and the transport protocol implements partial checksums. The *data corrupted* bit is set by the link layer if some data in the packet is known to be corrupted. Two *stale-treatment* bits determine desired treatment of a packet with expired lifetime. A value 00 indicates that the packet should be forwarded, 01 that it should be discarded, 10 that the packet should be forwarded with truncated payload. The *slow* bit is set if the packet belongs to a slowly-responsive flow.

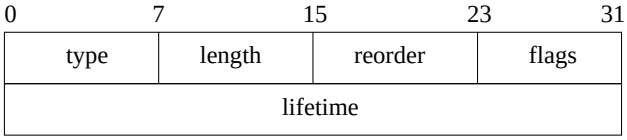


Figure 13: An IP option for cross-layer communication.

The IPv6 option follows the same format as above. It is a hop-by-hop option. The highest two order bits of the *type* field are set to 00 to inform the router to ignore this option if not understood and forward the packet. The third highest order bit is set to 0 if the timestamp field is in a deadline form. It indicates that the option does not change in transit. If the timestamp field is in a lifetime form, this bit is set to 1 to indicate that the option can change in transit (when routers decrement the packet lifetime).

Biographies

Andrei Gurtov *received his master's and licentiate's degrees in Computer Science at the University of Helsinki in 2000 and 2002. He is currently finalizing a doctoral thesis at the same university. In 2003 he spent six months in the International Computer Science Institute at Berkeley working in a project on building better simulation models. He is currently a senior researcher at TeliaSonera.*

Reiner Ludwig *received his diploma and doctoral degrees in computer science from the University of Technology in Aachen, Germany, in 1994 and 2000. He pursues his research as an employee of Ericsson Research where he joined in 1994. He has spent over two years, between 1997 and 1999, at the University of California at Berkeley, working as a guest researcher on behalf of Ericsson Research. He currently focuses his research on the performance analysis of end-to-end protocols, and their interaction with wireless link and physical layer functions.*