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

Abstract
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 playout 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 tradeoff between the maximum delay jitter and reliability if quality of service requirements of a flow are known. We propose using the IP timestamp option to communicate the flow requirements to the link layer. The packet lifetime is set to the minimum of the data lifetime determined by the application and the retransmission timeout value determined by the transport protocol if selective reliability is supported. For congestion-sensitive flows, the link transmits only headers of "discarded" 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.

1. Introduction

By definition, the usefulness of real-time data is limited by a certain time limit that we call *lifetime*. For example, for a video streaming application data lifetime is determined by the size of the play out buffer at the receiver. For a telemetric application, old measurement samples become obsolete when a new one is recorded. If delivery across the network takes longer than the data lifetime, data become *stale* and is typically discarded by the receiver.

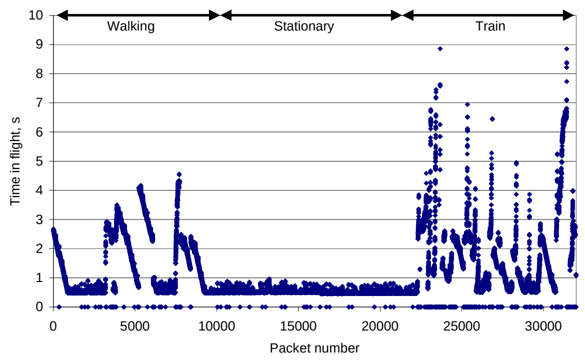
Slow access links, especially in cellular wireless networks, often have significant delay jitter. For example, a measurement study of dial-up connections reports occasional delay jitter of several seconds due to link-layer error recovery by a modem [19]. Frequent delay spikes of 3 to 15 s were observed in a wide-area cellular network due to handovers [11].

Consider Figure 1 showing delay jitter of UDP packets transmitted at 30 kbps over a cellular link. The trace is about an hour long, with 20 min of walking, 20 min in stationary conditions and 20 min in a moving train. Packets of 500 bytes are transmitted at a constant bit rate downstream on a GPRS link [3]. Delay spikes of several seconds are clearly visible in mobile conditions. About 8% of all packets were lost. To confirm that delays were not purely due to congestion, we repeated measurements using a congestion-sensitive TFRC flow [8] and still observed significant delay jitter.

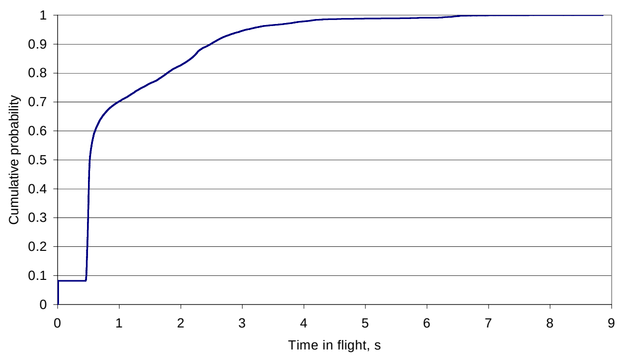
Many real-time applications account for delay jitter in the network by buffering data at the receiver. However, extensive buffering increases a start-up delay and harms interactivity for rewind operations. For certain types of media such as live streaming, conversational audio or stock quote updates,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

In summary, we believe that eventual disruptions to 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 in this paper is to assign a delivery lifetime to each packet at the sending host. This gives the link layer the necessary information on how persistent it should be on transmitting each packet. We show that Lifetime Packet Discard improves performance by nearly eliminating delivery of stale and duplicate data over an expensive cellular link. Although the idea of LPD is not entirely new [37][34], we are not aware of its systematic evaluation. We provide extensive simulations of LPD for CBR, TCP and TFRC flows.



(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.

Furthermore, a solution to the problem of spurious timeouts in transport protocols is proposed using LPD. We show that LPD can unnecessarily trigger end-to-end congestion control and suggest a solution.

The rest of the paper is organized as follows. In Section 2, our view of the architecture for delivery of real-time data over cellular links is presented. Section 3 motivates this paper by describing problems solvable by our approach. Section 4 shows how LPD avoids delivery of stale and duplicate data in practice. In Section 5 we evaluate the effect of LPD on performance of various types of flows. Section 6 presents ideas for future work. Section 7 concludes the paper.

2. 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 do best to satisfy QoS requirements of all flows at the minimum cost of bandwidth and battery power.

2.1 Network Architecture

We assume the network architecture displayed in Figure 2 that resembles the architecture of a GPRS cellular network [3]. 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). The Logical Link Control (LLC) protocol spans from the mobile station to the last-hop router and retransmits lost data primary during handovers.

The last-hop router implements LPD by dropping packets with remaining lifetime less than it would take to deliver the packet over the cellular link. The packet lifetime can be used for several other purposes in the access router such as the earliest deadline scheduling [17]. In this paper, we only consider using packet lifetime for discarding stale and duplicate data.

When the real-time server is located in the Internet, clock synchronization between the access router and the server would be typically required. The network time protocol [27] (NTP) can provide sufficient accuracy for our goals. When the real-time server is located close to the access network 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.

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 access router; therefore, a mechanism to inform the router of packet lifetime is needed.

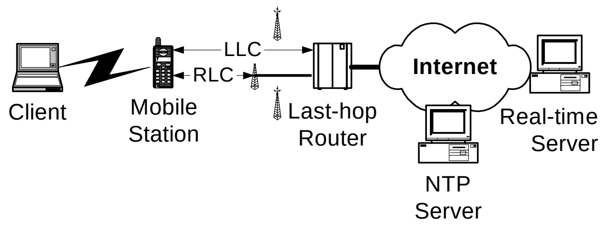


Figure 2. Network architecture.

Unfortunately, the time-to-live (TTL) field in IPv4 and IPv6 is too short to provide sufficient accuracy for packet lifetime. The IP timestamp option [36][30] provides exactly what we need to inform the last-hop router about the packet lifetime[1]. The IPv4 standard allows using a custom format for this option[2]. We exploit this opportunity to set the packet lifetime in milliseconds in IP datagrams. 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 [5]. A drawback of the IP timestamp option is that it costs 8 bytes of overhead and may load a router due to slow-path processing.

2.2 End-to-end Real-Time Transport

We believe that future transport protocols for real-time data will support selective reliability and TCP-friendly congestion control.

It is widely recognized that 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 [7]. By recovering important packets within the playout delay, perceived playout quality of the application can be significantly enhanced. SR-RTP is a backward-compatible RTP extension that supports selective reliability [7]. PR-SCTP is another example of a partially reliable transport protocol [35].

Congestion control is a general requirement to all Internet flows in the future [9]. However, the oscillatory nature of TCP AIMD congestion control may not be desirable by real-time applications. The notion of TCP-friendliness permits smoother transmission rate as long as on the average the rate of the flow is same as of a TCP flow in similar conditions. TFRC is one of the proposed slowly responsive equation-based congestion control algorithms [8]. DCCP is a new unreliable transport protocol that allows applications to use TFRC congestion control [16].

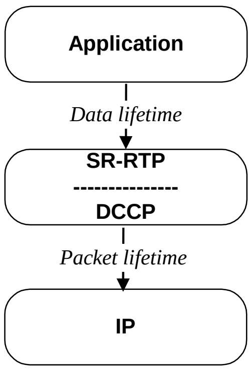


Figure 3. End-to-end transport for real-time flows.

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 (data lifetime). The SR-RTP protocol verifies if data can be delivered with the lifetime. If retransmissions are possible within the data lifetime, SR-RTP sets the packet lifetime to the value of the retransmission timeout. Otherwise, the data lifetime is copied to packet lifetime. The DCCP protocol provides TCP friendly rate control to SR-RTP.

3. Motivation of the Approach

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

3.1 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 deploy retransmissions at the link layer. The link persistency is defined as how long in 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 smaller data blocks than IP MTU is more efficient than relying on purely end-to-end error recovery [23].

[1] The IPv4 option contains a 32-bit timestamp. By default, the time is in milliseconds since midnight UT. However, any custom time format can be used if the high order bit of the timestamp is set.

[2] The timestamp option for IPv6 has not yet been defined. We are working on implementing it.

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 needed 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 tradeoff. A concept of a flow-adaptive link layer was proposed to tailor link behavior to demands of different application flows [22]. The proposed solution is a simple heuristic of being highly persistent on TCP packets and low persistent on UDP packets. It also recognized that this crude discrimination is not sufficient as there are UDP applications (e.g. NFS) that assume nearly reliable delivery from the network. Furthermore, the choice of persistency is arbitrary and may not correspond to actual requirements of applications.

With our approach, each packet has the exact information about its maximum tolerable delay. This allows a flow-adaptive link to optimally control delay vs. loss probability due to wireless errors in the best suitable way to the application.

3.2 Competing Error Recovery

Another problem related with uncontrolled link layer retransmissions is the competition between link layer and end-to-end error recovery. Because link error recovery is seen as delay jitter by the end hosts, the retransmission timer of a transport protocol may expire prematurely triggering unnecessary retransmissions and congestion control.

The Eifel algorithm [20] was proposed as an end-to-end solution for detecting [25] and recovering of spurious TCP timeouts [26][13]. The Eifel algorithm uses the TCP timestamp option to determine if arriving ACKs 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 [33].

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

- End-to-end protocols rely on delivery of old packets in the network after a delay. We allow the end host to retransmit a fresh version of the data after a delay.
- End-to-end solutions proposed so far are for TCP, which is a window-based protocol acknowledging every or at least every other packet. Our solution suits well for rate-based real-time protocols having infrequent ACKs. End-to-end proposals would be inefficient for such a protocol because their operation is affected by waiting for ACKs.
- Ideally, our approach allows to entirely avoid 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.

3.3 Application Empowerment

A popular paper on the next generation of protocols [4] 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 the network should try to deliver the data.

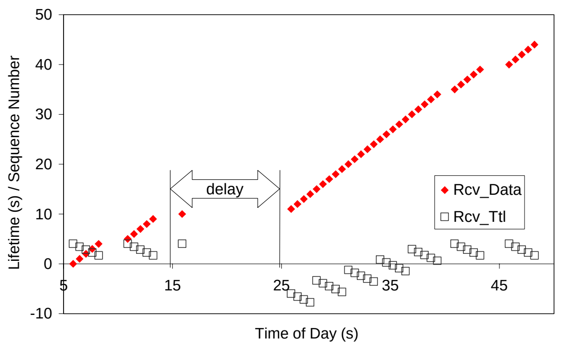
Another tradeoff that should be under control of the application is a maximum transmission burst size vs. a maximum queuing delay. The size of the link buffer sets this tradeoff in the network. Having a small buffer reduces the queuing delay and possibly the amount of stale data delivered to the receiver.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

4. Lifetime Packet Discard

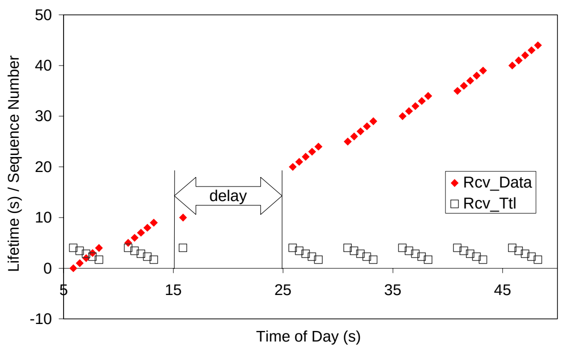
4.1 Preventing Stale Packet Delivery

In this section, we show how LPD can improve performance of CBR flows. A delay spike has two negative effects on such flows:

- Packets buffered in the network become stale and are unnecessarily transmitted after the delay spike ends. This wastes resources.



(a) Without LPD



(b) With LPD

Figure 4. Disruption introduced to a CBR flow by a delay spike.

- Transmission of stale data delays delivery of fresh arriving data.

Figure 4 (a) shows a receiver trace from ns2 of a CBR flow when a 10-second delay spike is introduced. In this example packet lifetime is set to 5 s, which is the interval at which the application regenerates new data objects. The *rcv_data* shows packet sequence numbers at the receiver, and the *rcv_ttl* shows the remaining lifetime of arriving packets. Negative values mean that the packet is stale and should be discarded. When the delay starts at 16th sec, no messages are delivered until the delay ends, but newly arriving messages get queued in the access router. When the delay ends at 26th sec, for the next 15 sec the link delivers only expired messages. The backlog of stale packets prevents fresh updates to be delivered to the receiver.

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

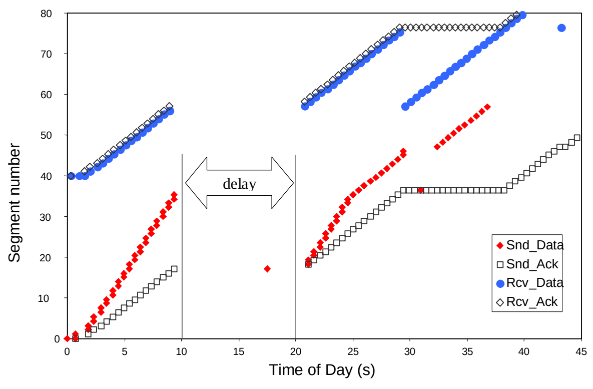
4.2 Preventing Duplicate Packet Delivery

Transports protocols such as TCP, SR-RTP [7], HPF [18] and PR-SCTP [35] 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 varies, the timer can expire prematurely. As a result, two or more duplicate packets can be transmitted over the access link wasting resources. Below we describe this situation for TCP in detail.

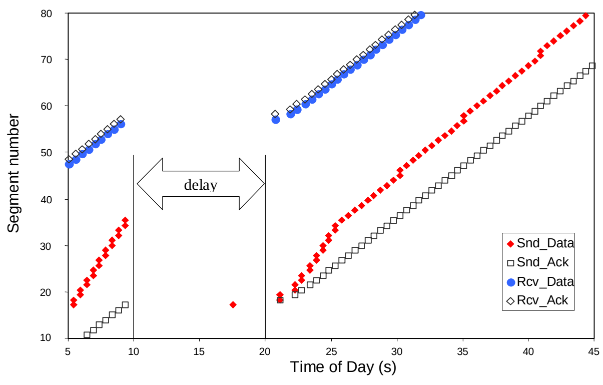
When a sudden delay occurs in the network that exceeds the current value of the TCP retransmission timer, the oldest outstanding segment is retransmitted. Since data segments are delayed but not lost, the retransmission is unnecessary and the timeout is spurious. A spurious TCP timeout is shown in Figure 5 (a). Segment numbers in the receiver trace are offset to prevent an overlap with segment numbers of the sender. The delay in generated between 10th and 20th sec in this test. The first retransmission that happens at the 17th sec is also delayed. The sender interprets the ACK generated by the receiver in response to the delayed segment as related to the retransmission, not the original segment. Because of the retransmission ambiguity problem, an ACK bears no information on which segment, original or retransmitted, has generated it. Encouraged by arriving ACKs, TCP retransmits all outstanding segments using the slow start algorithm. Such a retransmission policy is refereed to as go-back-N since the sender forgets about all segments it has earlier transmitted. At 28th sec retransmitted segments arrive to the receiver and generate duplicate ACKs 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 result into duplicate packets delivery over a cellular link.
- Congestion control is disturbed. In the short run, retransmissions can overload the network if transmitted using the slow start. In the longer run the network can be underutilized.



(a) Without LPD



(b) With LPD

Figure 5. Preventing duplicate packet delivery for a TCP flow after a delay spike.

Figure 5 (b) shows the same TCP flow with LPD. Here, packet lifetime is set to the retransmission timeout value at the TCP sender. When the sender times out and retransmits the first segment at 17th sec, the access router has dropped all old outstanding segments. When the delay ends, the sender immediately gets an ACK for the fresh retransmitted segment and continues retransmitting segments using the 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 presented by spurious timeouts. In this experiment the problem is not significant because the bandwidth-delay product of the path is small. However, reduction of the slow start threshold after a spurious timeout can harm performance on paths with a larger bandwidth-delay product. In Section 4.3 we discuss how this problem can be alleviated.

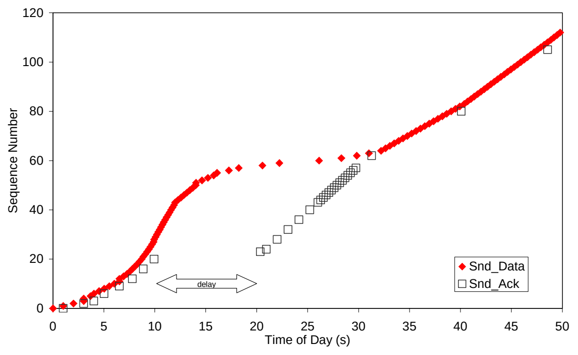
Another possible complication is that setting packet lifetime correctly can be difficult for transport protocols for which the retransmit timer is updated *after* a segment had been sent. During bulk data transmission in TCP, the retransmit timer is offset by one RTT because the timer is restarted upon a new ACK [24][29]. Furthermore, the retransmit timeout value can be updated more frequently than once per RTT. For optimal performance (to avoid occasional dropping of valid packets and delivery of duplicates) the transport protocol should not collect new RTT samples nor restart the timer until the oldest outstanding segment is acknowledged. However, if RTT suddenly increases, the timer has to be updated to avoid plentiful spurious timeouts.

4.3 Interactions with End-to-end Congestion Control

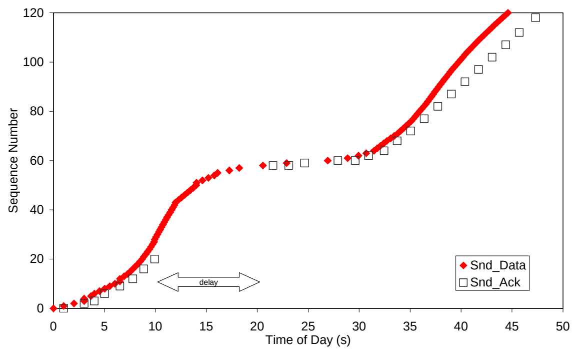
End-to-end congestion control in the Internet is based on an assumption that almost all packet losses are due to congestion [15]. Hence, discarding expired packets in the network incorrectly triggers reduction of the transmission rate at the sender. In this section we examine how an end-to-end congestion control reacts to packets drops by LPD.

Figure 6 (a) shows a sender trace of a TFRC flow when a 10-second delay spike is introduced. We assume application data lifetime of 5 s in this example. Thus, at the sender lifetime in packets is set to 5 s. The sender gets no feedback during the delay and gradually slows down to eventually transmit one packet per RTT. When the delay spike ends, a burst of ACKs that were delayed comes to the sender. It takes about 15 sec after the delay ends for the sender to get to the normal transmission rate. Because the receiving application discards stale data in this case, the transport protocol does not see any data loss in the network. Since the receiver reports no loss events, the sender does not further invoke congestion control.

Figure 6 (b) shows the same flow when the router drops stale packets. Until the delay 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 harms performance.



(a) Without LPD



(b) With LPD

Figure 6. Undesired interactions with end-to-end congestion control in a TFRC flow after a delay spike.

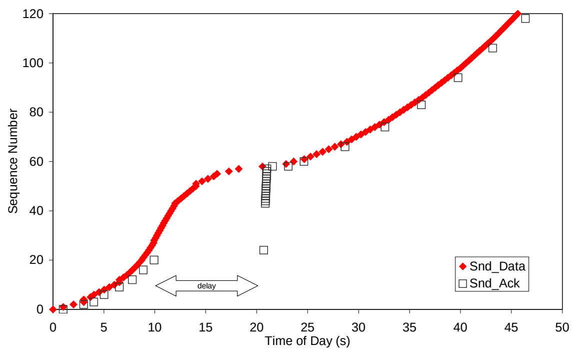


Figure 7. Interference of LPD with end-to-end congestion control is resolved by transmitting headers of stale packets.

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 'discarded' packets[3]. 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. A flow uses a bit in the IP timestamp option to indicate if it is interested in headercasting or its packets may be simply dropped.

5. Performance Evaluation

We run an extensive set of ns2 simulations to explore 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 acknowledgements, limited transmit, timestamps, MSS of 1000 bytes and unlimited receiver window. Delay jitter was introduced by inserting a 7 s delay spike every 30 s according to real-world traces in Figure 1. All queues are Drop-Tail; existing active queue management algorithms are not easily applicable when the level of statistical multiplexing is low [37]. LPD is executed in R1. Our simulation scripts are publicly available.

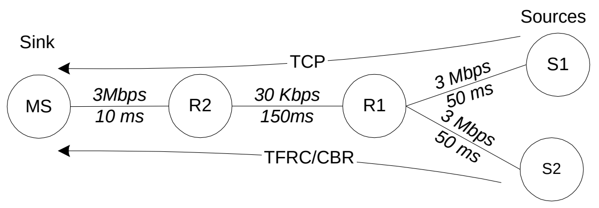


Figure 8. Simulation setup in ns2.

5.1 CBR Flows

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

Figure 9 shows performance of a 27 kbps CBR flow with different values of data lifetime. In this test, packet lifetime is determined by the application according to the playback buffer available. With a Drop-Tail buffer fewer packets are delivered valid when data lifetime becomes tighter. Furthermore, the number of valid packets quickly decreases with increasing of the buffer size in the bottleneck buffer. The number of stale packets increases with a smaller data lifetime or a larger buffer size. Eventually, almost all packets are delivered stale.

When LPD is enabled, performance for different data lifetimes and buffer sizes is stable and nearly optimal. About 90% of packets are delivered valid. At

[3] This may require re-computing the packet checksum and may not work in presence of IPsec.

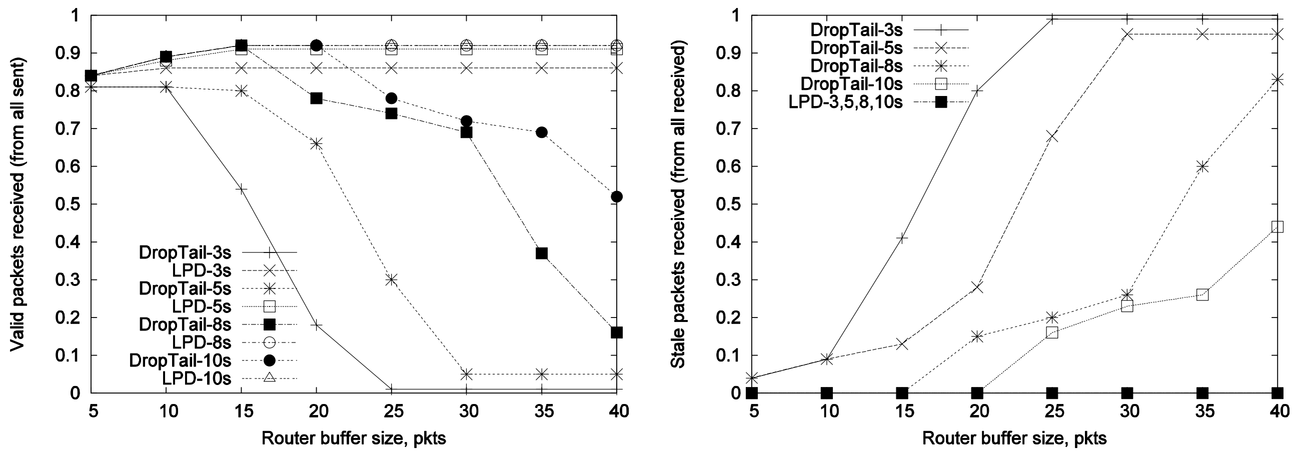


Figure 9. Effect of LPD on performance of a single CBR flow with different packet lifetimes.

the same time, no stale packets are sent over the cellular link.

5.2 TCP Flows

In this section we examine how LPD can prevent delivery of duplicate segments in presence of spurious TCP timeouts. We compare performance of TCP over LPD with the standard TCP and TCP with the Eifel algorithm. The download time reflects perceived user performance of the application. The number of duplicate segments received shows how resource-efficient a solution is. The lifetime in packets is set to the retransmission timeout value of the TCP sender.

In Figure 10 the download time of TCP over Drop-Tail is 10-30% higher than over LPD. TCP with the Eifel algorithm has only slightly higher download time than TCP over LPD. However, when the congestion control is not undone after a spurious timeout for the Eifel algorithm ("Eifel-CC"), then the download time is variable and up to 100% higher than of TCP over LPD.

The number of duplicate segments for standard TCP grows quickly with a larger buffer size. Up to 15% of all delivered segments are duplicates. Both the Eifel algorithm and TCP over LPD perform equally well in reducing the number of duplicate segments delivered.

We want to note that our goal was not to significantly beat end-to-end solutions such as Eifel for TCP. We believe that Eifel works fine for fully reliable protocols. Instead, the main benefits of LPD are for real-time flows as described in Section 3.1. The reason why we used TCP is because semi-reliable protocols are not yet well supported in ns2.

5.3 TFRC and TCP Flows

In this section we compare concurrent TCP and TFRC flows with and without LPD. For TCP flows, we use the download time and the number of duplicate segments received as performance metrics. The packet

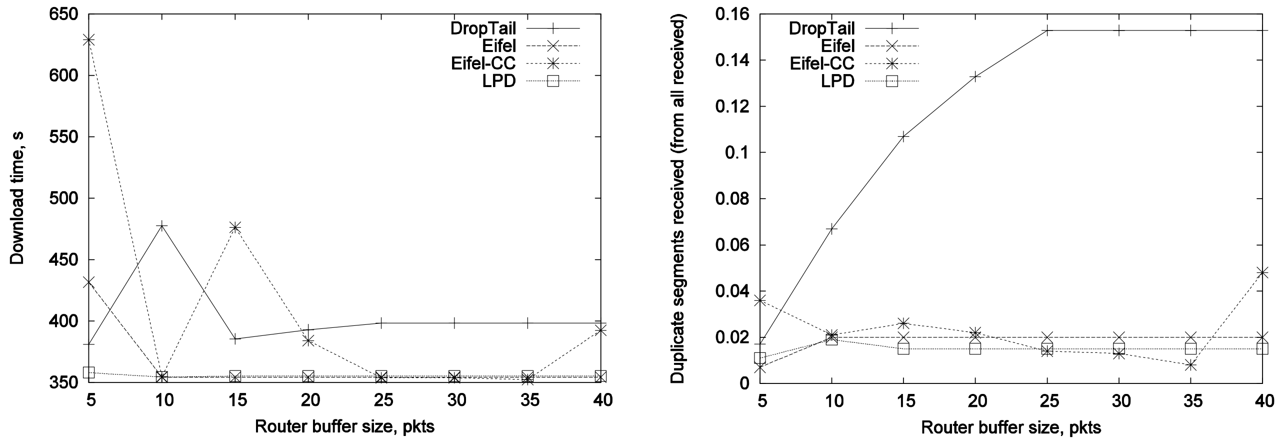


Figure 10. Comparison of TCP with LPD, TCP with the Eifel algorithm and standard TCP. Packet lifetime equals TCP RTO.

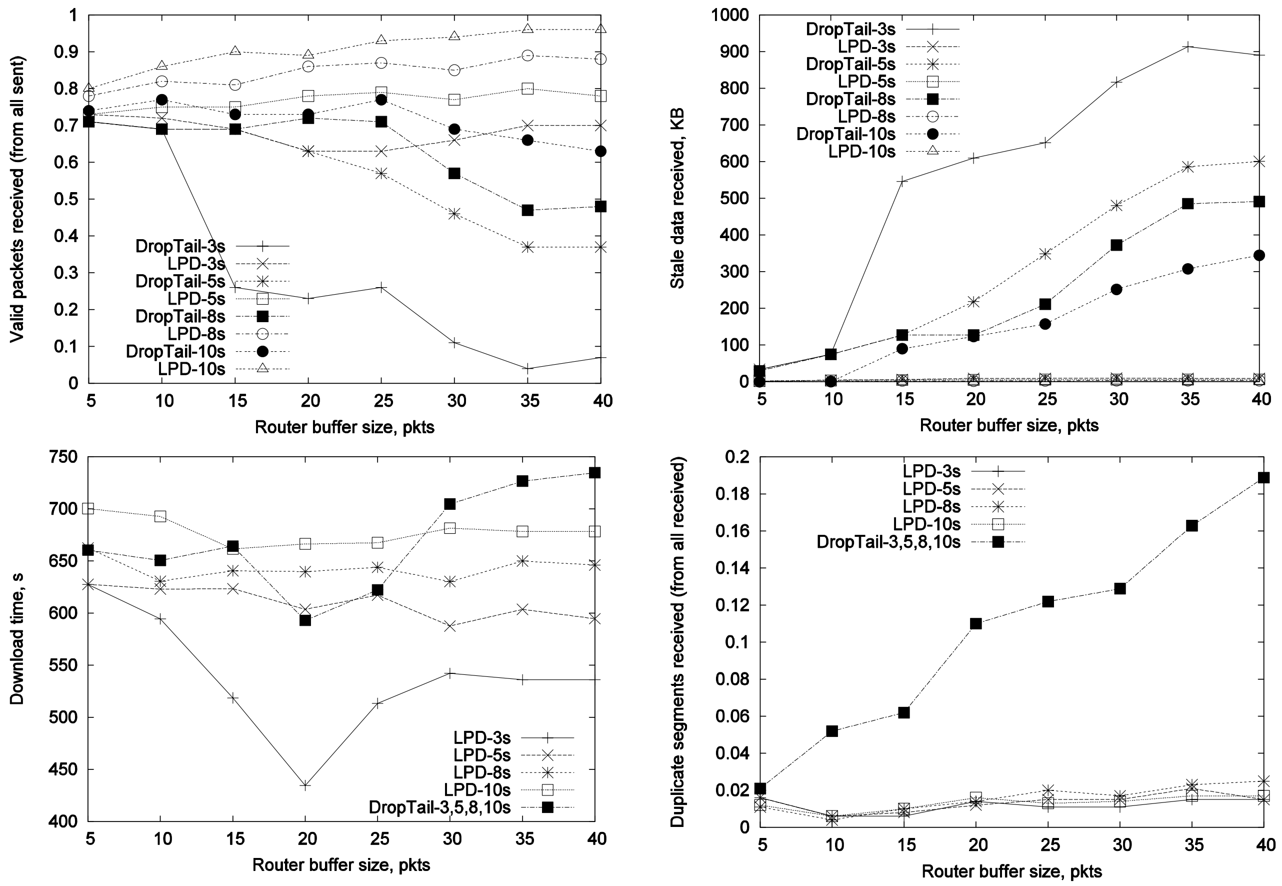


Figure 11. Effect of LPD on performance on concurrent TCP and TFRC flows with different packet lifetimes.

lifetime of TCP segments is set to the TCP retransmission timeout value. For TFRC flows, we look at the total size of stale packets received to reflect the behavior of headercasting. Packet lifetime of TFRC packets is set to 3, 5, 8, or 10 s to correspond to different application data lifetime.

In Figure 11 two top graphs show TFRC performance and two bottom graphs show TCP performance. TFRC flows over Drop-Tail have less valid packets than over LPD for a given data lifetime. The difference is growing with an increase in the router buffer size. Similarly, the number of bytes delivered stale is high for Drop-Tail. This number is small for LPD indicating that headercasting is a feasible way to avoid unnecessary triggering of end-to-end congestion control.

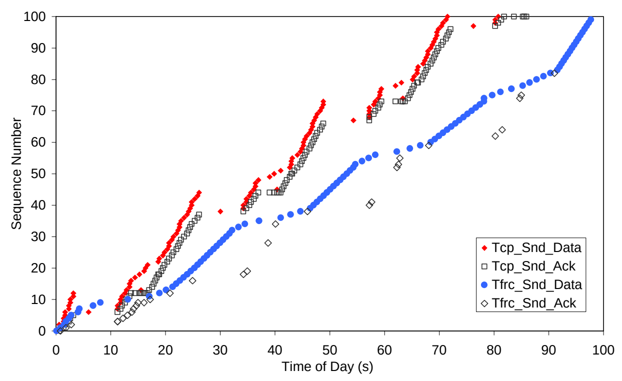
For TCP, the download time over LPD is similar to Drop-Tail, but decreases when the lifetime of the TFRC flow is lower. The number of unnecessary retransmissions is low and stable for all TCP flows over LPD, but grows quickly over Drop-Tail.

Figure 12 shows in detail why performance with LPD is better. Unnecessary go-back-N retransmissions are prevented for TCP flows when LPD is enabled. Stale packets of the TFRC flow are transmitted only as headers when LPD is enabled, which saves bandwidth and prevents problems with end-to-end congestion control. Finally, throughputs of TCP and TFRC flows are closer to each other, which is a crude indicator that LPD improves fairness between flows.

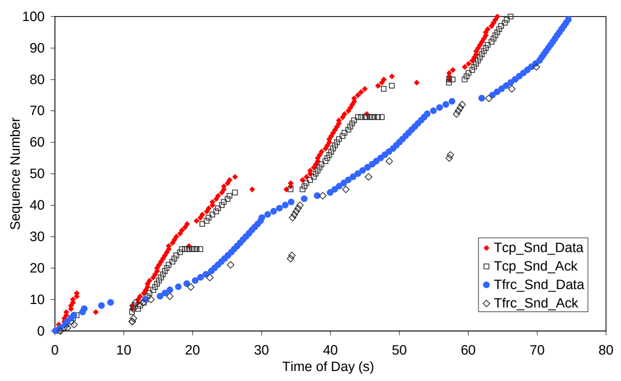
6. Considerations for Future Work

6.1 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 reliably are still in the development stage. Therefore, it is not a significant burden to complement them now with packet lifetime



(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. Packet lifetime of the TFRC flow is 5 s. Packet lifetime of the TCP flow is set to the RTO value.

functionality. Furthermore, in our experience software in infrastructure nodes in cellular networks is updated frequently which facilitates deploying LPD.

We recognize that it is not feasible to require that *all* nodes 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 sending end host can discover if routers in the path to destination support, or at least tolerate, carrying packet lifetime as an IP option. The IP timestamp option is a part of the standard [30] and should be supported by all Internet routers. However, some broken routers or firewalls may 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 acknowledgements or ICMP 'packet discarded' notifications the sender host can decide whether to send the IP timestamp option with packet lifetime for a given destination.

6.2 Congestion Control with Packet Lifetime

Capability of fine-grain control of packet lifetime in the network brings exciting opportunities for new developments in end-to-end congestion control. Actually, ideas for using packet lifetime for congestion control appeared in the early days of the Internet [28]. They were seemingly forgotten after a widespread deployment of loss-based congestion control [15].

A possible approach is to set the packet lifetime close to the minimum observed RTT of the path. If queuing delay increases in routers due to congestion then low-priority traffic gets expired and dropped preserving the capacity for best effort traffic. Such low-priority traffic is useful for 'scavenger' applications that utilize leftover bandwidth without slowing down other traffic.

Another possible application of packet lifetime is congestion control for ACK packets of a transport protocol. Currently, TCP suffers on asymmetric paths due to fixed rate of ACKs to the data segments [2]. Including packet lifetime in ACKs can prevent heavy congestion on the thin backward link.

6.3 Discard of Application Data Units

The concept of application-level framing argues that data objects should be delivered for an application over the network as application data units (ADU) [4]. One ADU corresponds to the data entity convenient for the application, such as a video frame. The ADU size can exceed the IP MTU of the network. Large ADU are transmitted as several IP datagrams, or as fragments of a single datagram.

A congested router in today's Internet drops IP datagrams arbitrary between ADUs. When the application has tight real-time constrains, there is no possibility to recover lost ADU fragments. Therefore, the receiver host might drop many large ADUs with only a few missing fragments. Consequently, the playout quality of the application can detiorate to an unacceptable level. A similar problem appeared previously in the context of dropping ATM cells from TCP segments [32].

To prevent this problem, the router could drop entire ADUs while possibly taking their priority and lifetime in consideration. However, if ADU framing is done at the transport layer, the router cannot sort packets into ADUs without snooping into transport headers. Such 'layering violation' is undesirable form a point of view of the Internet architecture [5].

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 datagram id without snooping into upper level headers. We are investigating possible performance gains of this approach for real-time data transport.

7. Conclusions

Emerging real-time transport protocols combine selective retransmissions with TCP-friendly congestion control. In our architecture for efficient real-time transport over cellular links, new transport protocols are reinforced with Lifetime Packet Discard at the wireless link layer. In this paper we evaluated effects of LPD on bulk 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 retransmission timeout value determined by the transport protocol. The packet lifetime coordinates operation of the link and transport-layers. A flow adaptive link can use packet lifetime for deciding on the number of retransmission attempts, data encoding and scheduling of transmissions. The transport protocol can recover lost packets quicker by deploying a more aggressive retransmission timer [24] because the cost of spurious timeouts is minimal with LPD.

End-to-end congestion control in the Internet is based on the assumption that most packet losses are due to congestion. Therefore, discarding stale data in the network can incorrectly trigger end-to-end congestion control causing a reduction of 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 remarkable efficiency gains.

The benefit of LPD is most significant for larger sizes of the bottleneck buffer. Here 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 buffers is inadequate for smoothing bursty traffic generated by variable bit rate codecs.
- A larger buffer can accommodate bandwidth variation occurring during vertical handovers [14].
- The current practice is to use very large buffers in cellular links [21]. A per-user buffer of 50-200 KB in live GPRS networks was measured [11].

Using an active queue management algorithm such as RED [10] may seem an attractive alternative to LPD. However, RED does not work well in an environment with a low level of statistical multiplexing [11][37]. Drop From Head (DFH) could be used together with LPD to increase performance.

In future work, we will consider use of explicit loss notification [1] as an alternative to headercasting for avoiding unnecessary triggering of congestion control by expiration losses. The cumulative explicit transport error notification [6] takes a different approach from providing fine-grain feedback per each discarded packet. The idea is that routers tell to the sender the average fraction of lost packets due to transmission errors. The sender makes a smaller decrease in the congestion window 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.

Acknowledgements

Thanks to Rajiv Chakravorty and members of the Sahara project at UCB for valuable comments.

References

[1] H. Balakrishnan, R. Katz, Explicit Loss Notification and Wireless Web Performance, In Proceedings of IEEE Globecom Internet Mini-Conference, November 1998.

[2] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, M. Sooriyabandara, TCP Performance Implications of Network Path Asymmetry, RFC 3449, December 2002.

[3] G. Brasche, B. Walke. Concepts, services and protocols of the new GSM phase 2+ General Packet Radio Service, IEEE Communications Magazine, Vol 35, No 8, pages 94-104, August 1997.

[4] D. D. Clark, D. L. Tennenhouse. Architectural considerations for a new generation of protocols, In Proceedings of ACM SIGCOMM’90, August 1990.

[5] D. D. Clark, The Design Philosophy of the DARPA Internet Protocols, CCR, Vol. 18, No. 4, August 1988, pp. 106–114.

[6] W. Eddy, S. Ostermann, M. Allman, New Techniques for Making Transport Protocols Robust to Corruption-Based Loss. July 2003. Under submission.

[7] N. Feamster, H. Balakrishnan, Packet Loss Recovery for Streaming Video, 12th International Packet Video Workshop, April 2002.

[8] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, SIGCOMM, August 2000.

[9] S. Floyd, K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, August 1999.

[10] S. Floyd, V. Jacobson, Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993.

[11] A. Gurtov, TCP Performance in the Presence of Congestion and Corruption Losses, Master's Thesis, University of Helsinki, December 2000.

[12] A. Gurtov, M. Passoja, O. Aalto, M. Raitola, Multi-Layer Protocol Tracing in a GPRS Network, In Proceedings of IEEE Vehicular Technology Conference (VTC'02), September 2002.

[13] A. Gurtov, R. Ludwig, Responding to Spurious Timeouts in TCP, IEEE INFOCOM'03, March 2003.

[14] A. Gurtov, J. Korhonen, Measurement and Analysis of TCP-Friendly Rate Control for Vertical Handovers, submitted for publication.

[15] V. Jacobson, Congestion Avoidance and Control, ACM SIGCOMM, August 1988.

[16] E. Kohler, M. Handley, S. Floyd, J. Padhye, Datagram Congestion Control Protocol (DCCP), http://www.icir.org/kohler/dcp/.

[17] K. Lee, M. Zarki, Scheduling real-time traffic in IP-based cellular networks, The 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, 2000.

[18] J. Li, S. Ha, V. Bharghavan, HPF: A Transport Protocol for Heterogeneous Packet Flows in the Internet, INFOCOM 1999.

[19] D. Loguinov, H. Radha, Measurement Study of Low-bitrate Internet Video Streaming, In Proceedings of the ACM SIGCOMM Internet Measurement Workshop, November 2001.

[20] R. Ludwig, R. Katz, The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions, ACM Computer Communications Review, Vol. 30, No. 1, January 2000.

[21] R. Ludwig, B. Rathonyi, A. Konrad, K. Oden, A. Joseph, Multi-layer Tracing of TCP over a Reliable Wireless Link. ACM SIGMETRICS 1999.

[22] R. Ludwig, B. Rathonyi, Link Layer Enhancements for TCP/IP over GSM, INFOCOM 1999.

[23] R. Ludwig, A. Konrad, A. D. Joseph, R. H. Katz, Optimizing the End-to-End Performance of Reliable Flows over Wireless Links, Kluwer/ACM Wireless Networks Journal Vol. 8, Nos. 2/3 , March-May 2002.

[24] R. Ludwig, K. Sklower, The Eifel Retransmission Timer. Appears in ACM Computer Communications Review, Vol. 30, No. 3, July 2000.

[25] R. Ludwig, M. Meyer, The Eifel Detection Algorithm for TCP, RFC 3522, April 2003.

[26] R. Ludwig, A. Gurtov, The Eifel Response Algorithm for TCP, draft-ietf-tsvwg-tcp-eifel-response-03.txt , work in progress.

[27] D. Mills, Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI, RFC2030, October 1996.

[28] J. Nagle, On Packet Switches With Infinite Storage. RFC970, December 1985.

[29] V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 2988, November 2000.

[30] J. Postel, Internet Protocol (IP), RFC-791, September 1981.

[31] K. Ramakrishnan, S. Floyd, D. Black, The Addition of Explicit Congestion Notification (ECN) to IP, RFC3168.

[32] A. Romanow, S. Floyd, Dynamics of TCP Traffic over ATM Networks, IEEE JSAC, V. 13 N. 4, May 1995, p. 633-641.

[33] P. Sarolahti, M. Kojo, K. Raatikainen. F-RTO: A New Recovery Algorithm for TCP Retransmission Timeouts, University of Helsinki, Department of Computer Science, Technical Report C-2002-07.

[34] B. Smith, Cyclic-UDP: A Priority-Driven Best-Effort Protocol, Tech. report, Cornell University, 1995

[35] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, P. Conrad, SCTP Partial Reliability Extension, http://www.ietf.org/internet-drafts/draft-stewart-tsvwg-prsctp-03.txt.

[36] Z. Su, A Specification Of The Internet Protocol (IP) Timestamp Option, RFC 781, May 1981.

[37] M. Sågfors, R. Ludwig, M. Meyer, J. Peisa, Queue Management for TCP Traffic over 3G Wireless Links, In Proceedings of IEEE WCNC, March 2003.

[38] J. Wong, Y. Liu, Deadline based network resource management, In Proceedings of the Ninth International Conference on Computer Communications and Networks, 2000.