This document is a technical report (EED/R-99:128) from Ericsson Research. It is also published as "University of California at Berkeley, Technical Report UCB//CSD-99-1053, May 1999". Please use the latter if you want to reference this document.

A Case for Flow-Adaptive Wireless Links

Reiner Ludwig
Ericsson Research
Herzogenrath, Germany

Abstract-**We study the performance problems that exist when loss responsive flows traverse wireless links, where losses are often unrelated to congestion. We present a novel concept - flow-adaptive wireless links - which provides service differentiation by tailoring link layer error control to the QoS requirements of each flow sharing the link. Flow-adaptive links emphasize local error control as a necessary complement to end-to-end error control, and are independent of transport (or higher) layer protocol semantics. The key idea is that applications use the IP layer as a level of indirection through which QoS requirements are communicated to each link along the path, on a per flow basis. We then demonstrate how this improves performance for the particular class of reliable loss responsive flows. We prove in general that a well engineered, fully reliable wireless link does not interfere with TCP's end-to-end error recovery. Moreover, we propose a new error recovery algorithm (TCP-Eifel) that can optionally be implemented in TCP to further improve performance. By eliminating the retransmission ambiguity problem the algorithm detects spurious timeouts, and uses these as an implicit cross-layer signal to prevent unnecessary retransmissions in TCP.**

1. Introduction

The Internet is undoubtedly evolving to become *the* communication medium of the future. It will not be long before the last circuit switch is taken out of service and virtually all people-to-people, people-to-machine, and machine-to-machine communication are carried in IP [34] packets. The tremendous recent growth of the Internet in terms of connected hosts is only matched by the likewise tremendous growth of cellular telephone subscribers. While most hosts on today's Internet are still wired, the next *big* wave of hosts has yet to hit the Internet. We believe that the predominant Internet access of the future will be wireless. Not only every cellular phone but every *thing* that communicates will have: (1) an IP protocol stack and (2) a wireless interface. Furthermore, we believe that best-effort service classes will remain the most frequently used.

It is well known that congestion control mechanisms used in the Internet today, poorly interact with non-congestion related packet loss above a certain threshold rate. Whereas error rates on state-of-the-art wireline links can be safely neglected, this is not true for wireless links. High performance wireless Internet access is still an unsolved research challenge. Various approaches have been proposed, mostly focusing on the particular problem of TCP [35]. However, non-TCP flows are increasingly found in the Internet. We argue why none of the existing approaches provide a satisfying solution for future wireless Internet access. We present a novel concept - flow-adaptive wireless links - which emphasizes (1) local error control as a necessary complement to end-to-end error control and (2) independence of transport (or higher) layer protocol semantics. The latter allows co-existence with any form of network layer encryption, e.g. [22]. Dynamic adaptation of error control schemes is not limited to changing radio quality. Flow-adaptive wireless links also adapt the error control schemes and local transmission priorities to the individual QoS (Quality of Service) requirements of each flow sharing the link and the network layer priorities among those flows. This extends the range of implementation alternatives to provide differentiated service to the IP layer. This concept is equally applicable to unicast and multicast, reliable and unreliable, and sender- and receiver-driven protocols.

We further suggest a concrete implementation of flow-adaptiveness to support reliable flows (e.g., TCP) making the argument that this requires a fully reliable wireless link. In this context we debunk a frequently made claim and show that a reliable link layer protocol in general does *not* interfere with TCP's end-to-end recovery. Through analysis we reveal how extremely conservative TCP's retransmission timer is, giving rise to think about more aggressive implementations of it. While conservative transport retransmission timers are less likely to interfere with link layer Automatic Repeat reQuest (ARQ), we argue that an adaptive transport retransmission timer should not be tuned to prevent all spurious timeouts. Instead, we believe that it should be "reasonably" conservative while a sender should be able to detect spurious timeouts and react appropriately. Addressing the latter issue we propose a new error recovery algorithm for TCP (TCP-Eifel) that detects spurious timeouts, and uses these as an implicit cross-layer signal to prevent unnecessary retransmissions.

The paper is organized as follows. In Section 2 we explain the conflict between the Internet's congestion signal and wireless links and review related work. In Section 3 we introduce the concept of flow-adaptive wireless links and discuss relevant design considerations. In Section 4 we describe the implementation of the concept for reliable flows, analyse TCP's retransmission timer, and develop the algorithm to prevent spurious retransmissions. We conclude the paper in Section 5 discussing limitations of our solution and outline future work.

2. The Wireless Challenge to Loss Responsive Flows

In this section we first review why wireless links introduce a problem for today's Internet. We then characterize the properties of a wireless link that are relevant for our discussion. This is followed by a summary of related work.

2.1 Congestion or Corruption?

Applications sharing a connection-less best-effort network need to respond to congestion to ensure network stability. Traditionally, congestion control has been implemented at the transport layer. [19] first described the fundamental algorithms that are most used in the Internet today [39], [42]. One of the key elements for any congestion control algorithm is the *congestion signal* that informs senders that congestion has or is about to occur. Throughout this paper we assume a sender-side implementation of transport layer congestion control, and if applicable also error control. The same discussion also applies to receiver-based implementations. One distinguishes between *explicit* congestion signals issued by the network and *implicit* congestion signals inferred from certain network behavior. Nevertheless, routers in today's Internet do not issue explicit congestion signals[1] although this might be implemented in the future [36]. Two approaches have been discussed for senders relying on an implicit congestion signal: delay-based and loss-based. Unfortunately, it is often not possible to draw sound conclusions from network delay measurements. In particular it is difficult to find characteristic measures such as the path's minimum round trip time due to persistent congestion at the bottleneck link or because of route changes [32]. Consequently, "packet loss" is the only signal that senders can confidently use as an indication of congestion. It is implemented either as a direct [39] or an indirect trigger based on a perceived packet loss rate [42] to throttle the flow's send rate. We refer to such flows as being *loss responsive*. In this sense a TCP-based flow is a reliable loss responsive flow, whereas a "TCP-friendly" UDP-based flow is an unreliable loss responsive flow.

However, "packet loss" is not unambiguous. Packets can get lost because of packet drops due to a buffer overflow at the bottleneck link or because of packet corruption due to a transmission error. The former indicates congestion, the latter does not. A sender is not able to discriminate among these events, because packet corruption usually leads to a frame checksum error and subsequent discard of the packet at the link layer. Hence, transmission errors inevitably lead to an underestimation of available bandwidth for loss responsive flows. As a consequence, applications can only fully utilize their share of bandwidth along the path if transmission errors are rare events. This explains why wireless links are often problematic: whereas transmission errors on today's wireline links can be safely neglected, this is not true for wireless links, especially when the end host is mobile.

2.2 Wireless or Broken Networks?

The rate at which packet transmission errors occur for a given flow is called the *damage loss rate*. We can approximate an upper limit for the damage loss rate up to which the flow's send rate is insensitive. A network limited sender cyclically probes the path for more bandwidth. With the additive increase policy of one packet per round trip time [19] this leads to a single - in the ideal case - dropped packet at the end of each cycle. Thus, the reciprocal of the number of packets that are sent per cycle determines the *probing loss rate*[2]. This rate is different for every path, depending on its *bandwidth/delay product* and *Maximum Transmission Unit* (MTU) [40]. Hence, a sender is insensitive to transmission errors as long as the damage loss rate stays below the probing loss rate. It is worth pointing out that [39] and [40] misinterpret [19] by stating that TCP's congestion avoidance algorithm assumes that packet loss caused by damage is much less than 1 percent. This is not correct: if the bandwidth/delay product is already exhausted with a few packets, the damage loss rate may be much higher than 1 percent without considerably affecting performance.

Researchers have argued that wireless links should be designed with an average damage loss rate below some worst case (minimum) probing loss rate. This approach might work for certain high bandwidth/delay links like satellites that alone account for most of the bandwidth/delay product of any path over that link. However, finding a worst case probing loss rate is virtually impossible without choosing one that is vastly conservative, i.e., low enough for those paths that require many packets to exhaust the bandwidth/delay product. For the other paths the wireless link would be overly protected, wasting of *the* most valuable resource of many wireless networks: spectrum. In addition, over-protection of the wireless link can negatively affect higher layer protocol performance [24][27]. Therefore, we deem this overprovisioning approach as inappropriate. In Section 3.1 we instead argue for a more differentiated approach.

Nevertheless, wireless networks can be broken. In particular we exclude from our studies the problem of intermittent connectivity, where the wireless link suddenly becomes unavailable for a duration that exceeds the order of the wireless link's round trip time. This can be due to several reasons. A cellular network might have "bad spots" where it does not provide sufficient coverage, or a cell handover process might introduce excessive delay or data loss. In either case this is *not* a networking problem. The first example is a radio problem that should be fixed with more or better tuned base transceiver stations. The second indicates a wireless network that was not designed to support seamless mobility in the first place. This includes many of the early WLAN (Wireless Local Area Network) systems, which were mainly targeted at reducing cabling costs, i.e., supported wireless but stationary hosts. This issue is further discussed in Section 3.2.

2.3 Related Work

The most elegant solution to loss responsive flows over wireless links is to eliminate "packet loss" to signal congestion, and in-

1. At least after the *source quench* [40] has been banned.

2. In [19] the number of packets sent per cycle is called the window equilibration length and is approximated as $W^2/3$ where W is the window size at the end of a cycle. More detail can e.g. be found in [30].

stead use an explicit signal issued from the network [36]. Applications could then fully utilize their share of bandwidth irrespective of any damage loss rate. Also, this allows sources of reliable flows to clearly separate error from congestion control. However, even if such mechanisms get deployed, it will still take years before a sender can safely assume that every router along the path is upgraded accordingly. For the next several years, we must deal with loss responsive flows.

While we are not aware of any work that studies the problem of loss responsive flows over wireless links in general, the particular problem of TCP over wireless links has been investigated in several studies discussed in this section. We have categorized the proposed solutions as shown in Figure 1. Note that the dark shaded areas indicate whether a transport protocol or its implementation must be changed, or whether transport protocol dependent state has to be maintained in the network. The lightly shaded areas indicate changes required at the link layer. Conceptual design considerations that favour one or another solution are further discussed in Section 3.2. We ignore solutions for flows that are not loss responsive, e.g. [9].

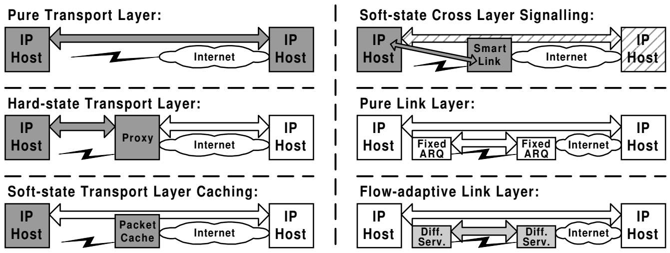

Figure 1: Approaches to solve "TCP over Wireless".

Pure transport layer solutions try to solve the problem solely on an end-to-end basis. Adding the notion of selective acknowledgements (SACK) to TCP [29] is a way to deal with damage loss [3]. The advantage is that a sender can quickly recover from multiple lost packets in a single round trip time and that such an event is treated as *one* congestion signal instead of one signal for each lost packet. In case a particular packet must be retransmitted more than once, [38] proposes a further enhancement to the TCP sender assuming a SACK receiver.

Hard-state transport layer approaches encompass all forms of splitting in that end-to-end semantics are sacrificed for better performance. The concept was initially proposed in [2], and has been used in other work including transit satellite links [15]. Any protocol can be chosen for the wireless link, e.g., [23] combines splitting with a pure link layer approach. The major benefit of hard-state transport layer solutions is that the end-to-end flow is shielded from damage loss on the wireless link, and the flow can fully utilize its share of bandwidth over the entire path. The concept of splitting lends itself well to non-TCP, e.g., unreliable loss responsive flows.

The Snoop protocol developed in [4] implements "TCP-aware" local loss recovery. Variation of the Snoop protocol were studied in [3]. Its advantage over split solutions is that the network state is soft, i.e., it is not crucial for the end-to-end connection, and thus preserves the end-to-end semantics. One problem is that it can only be applied to the those edges of a path that are free of congestion. The reason is the suppression of duplicate acknowledgements (DUPACKs), which filters out a congestion signal, and a proposed negative acknowledgement (NACK) scheme. When sending *to* the mobile host, packets dropped at a bottleneck link between the wireless link and the mobile host are mistaken for damage loss by the TCP-aware cache. The congestion signal (the three DUPACKs) is not propagated to the sender. For packets sent *from* the mobile host, the NACK scheme causes a problem. If the wireless link itself (or any other link between the mobile host and the wireless link) becomes the bottleneck, packets lost due to congestion[3] cannot be discriminated from those lost due to damage. Consequently a NACK is sent in either case, and the sender again relies on external means to get the congestion signal (e.g., the *source quench*).

Soft-state cross layer approaches make the flow's sender aware of the wireless link. This is achieved by having the link layer (or network layer in the case of Mobile-IP [33]) inform the transport layer sender about specific events so that it can adapt accordingly. The solution proposed in [11] uses ICMP (Internet Control Message Protocol) [40] to signal all active receivers that the link is in a bad state. The receiver reflects the signal to the sender using a dedicated TCP option field. In the network that was studied in [11], the reverse path did not traverse the "problem link". A similar idea is proposed in [7] which focuses on the problem of frequent and long disconnections. In case of disconnections a transport layer proxy issues TCP acknowledgements (ACKs) which shrink the advertised window to zero. This forces the TCP sender into *persist mode* [40]. In this mode the TCP sender will not suffer from timeouts nor from exponential back-off of the retransmission timer value. [8] and [28] focus on the problem of data loss or delay caused by cell handovers. Both solutions are based on the deployment of [33] and suggest informing the transport layer sender about a cell handover to trigger, e.g., the fast retransmit algorithm [8]. [3] proposes an explicit loss notification, which the link layer piggy-backs onto TCP acks as a TCP option to inform the sender that a particular packet was lost due to damage[4]. This solution, however, has the same problem as the above-mentioned NACK scheme.

Pure link layer solutions aim at hiding the artifacts of the wireless link to higher layer flows. The techniques include adaptive forward error correction, interleaving, adaptive power control, and fully-reliable [1][13][14][31] and semi-reliable [20] link layer ARQ protocols[5]. Some wireless networks use none of those (e.g., early commercially available 802.11 WLANs), while others

3. In [26] these effects were measured where packets got dropped locally at the mobile host because of congestion at the first-hop wireless link.
4. This requires that the IP *and* the TCP checksum be re-computed.
5. Note that none of the variations of the Snoop protocol discussed in [3] as "link layer solutions" are considered in this context. The main difference being that pure link layer solutions are not tied into the specific semantics of any higher layer protocol.

use combinations, e.g., the GSM (Global System for Mobile communications) digital cellular network. Pure link layer solutions can yield excellent TCP bulk data throughput without interfering with end-to-end error recovery [5][12][26]. Interactions between link layer and end-to-end error recovery have been studied in [3][10][12][26]. A key advantage is that local knowledge about the link's error characteristics, which can vary largely over short time scales, can be exploited to optimize error control efficiency. The second advantage is that it does not require any changes to the IP-based protocol stacks. The drawback is that the error control schemes are applied irrespective of the QoS requirements of individual flows sharing the link. A flow that requires link layer ARQ cannot share the link with a delay-sensitive flow intolerable of delays introduced by link layer retransmissions. On the other hand, an adaptive application might be able to tolerate higher loss rates in return for higher available bit rates than the link's channel coding scheme provides.

We describe the flow-adaptive link layer approach in Section 3.1. We are not aware of any related work that suggests implementations of this approach other than [27]. Similar ideas have been briefly mentioned in [12], but without developing any concrete implementation.

3. Making the Case for Flow-adaptive Wireless Links

Spectrum required for wireless links is often a valuable resource, demanding most efficient error control scheme. Traditionally, those schemes have been designed towards fixed parameters such as residual bit error rates or link latency. State of art wireless networks are further capable of distinguishing between voice and data to choose appropriate error control. However, there is more to distinguish than just voice and data. Flows in the Internet can have more differentiated QoS requirements including various degrees of reliability or delay sensitivity. Moreover, a flow's QoS requirements may change dynamically over the duration of its "life time". Yet, today's wireless networks are not designed flexible enough to adapt appropriately, leaving potential performance improvements unexploited. Besides, in today's Internet, flows do not carry enough information to allow for such differentiated treatment. Recent developments [6] might change that in the future opening new possibilities for more fine-grained adaptation of error control on wireless links. In this section we further develop this idea proposing the concept of flow-adaptive wireless links.

3.1 Service Differentiation through Error Control

Flow-adaptive link layers build on the concepts of pure link layer solutions. They emphasize (1) independence of transport (or higher) layer protocol semantics and (2) local error control as a necessary complement to end-to-end error control. Dynamic adaptation of error control schemes is not limited to changing radio quality. Flow-adaptive wireless links also adapt the error control schemes and local transmission priorities to the individual QoS requirements of each flow sharing the link and the network layer priorities among those flows. This concept is new although a precursor of the idea was introduced in [27], which developed a coarse grained differentiation between elastic (TCP) and realtime (UDP-based) flows. In that study the protocol identifier field in the IP header is used to choose whether or not to run fully-reliable ARQ at the link layer.

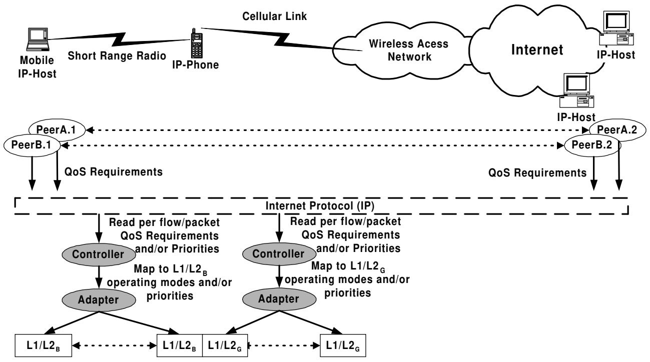

Figure 2: The concept of flow-adaptive wireless links.

In this section we generalize this concept by following the framework proposed in [6] to exploit QoS-related information - mainly the proposed differentiated service field but also any other field - derived from the IP header of each flow at the link layer. The key idea is that applications use the IP layer as a level of indirection through which QoS requirements and/or flow priorities are communicated to each link along the path, on a per flow/packet basis. [6] proposes to use such information to implement so-called *Per Hop Behaviors* (PHBs) by employing a range of queue service and/or queue management disciplines on a network node's output interface queue, e.g., weighted round-robin (WRR) queue servicing or drop-preference queue management. We propose to extend this notion for wireless links, making such links flow-adaptive by leveraging of existing error control schemes implemented at the link layer. Flow-adaptive link layers in wireless networks are what the end-to-end argument calls "an incomplete version of the function provided by the communication system [that] may be useful as a performance enhancement". We believe that carrying communication-related QoS requirements as part of the flow's headers, and adapting lower layer functions such as error control accordingly, advances the discussion provided in section 2.3 of [37]. This basic idea is depicted in Figure 2. It shows a mobile host that connects wireless to the Internet through a cellular network. The connection between the mobile host and the cellular network access device is yet a second (different) wireless link. Both wireless links are flow-adaptive but independent of each other. Each link deploys its own error control schemes (called $L1/L2_B$ and $L1/L2_G$ in Figure 2) which are optimized for the particular radio. A controller has access to the IP headers of each flow traversing the link. It reads the per packet QoS requirements and maps them to link specific parameters which are then used to adapt the local error control schemes accordingly.

Even today's most advanced wireless data networks lack flow-adaptive link layers. Consider the General Packet Radio Service (GPRS), a new packet-switched data service for the GSM digital cellular network. The GPRS link layer [13] can simultaneously support four different priority classes (one best-effort and three predictive-QoS classes). Each priority class has its own error control schemes which might be different from those of the other classes and provides fixed QoS. However, the granularity of adaptation is coarse and is unrelated to the QoS required by each flow but rather related to the charges the operator can impose onto each priority class. For example a file transfer and a real-time voice stream get the same QoS (e.g., through the same channel coding scheme) if both are assigned to the same priority class. Thus, GPRS lacks the concept that different flows in the same priority class are provided with differentiated QoS. This *can* but might not necessarily mean that a flow's QoS requirements are not satisfied but are "over-satisfied", e.g., by adding too much redundancy through channel coding, leading to wasted radio resources like spectrum and transmission power. In the remainder of this section we list examples of possible applications of flow-adaptive links. An implementation supporting fully-reliable end-to-end flows (e.g., TCP) is described and analysed in Section 4.

The Internet is currently changing dramatically and non-TCP flows are becoming increasingly important. One example is semi-delay-sensitive flows [42], used by adaptive play-back audio/video applications, or the even more delay-sensitive flows of real-time audio/video applications. If the range of required bandwidth, acceptable loss-rates, and tolerable per packet delays could be communicated to the wireless link, then much smarter error control decisions could be made. These concern the amount of channel coding, interleaving, and maximum permissible delay for ARQ that is dynamically applied while the radio quality changes. Another example is semi-reliable end-to-end flows (e.g., issued by a stock quote broadcasting application that periodically refreshes obsolete information). A "time-to-live" field in milliseconds with respect to a global clock could be used to drive a semi-reliable link layer ARQ protocol. Yet, another example of differentiation between flows allows a link layer ARQ protocol to perform out-of-order delivery of packets belonging to distinct or also identical[6] flows. This is acceptable as packets in a connection-less network get re-ordered anyway. Also certain fractions of an IP packet (e.g., the compressed IP header) could be protected with a higher amount of channel coding. This list could be continued much further.

3.2 Design Considerations

In this section we discuss the design considerations when solving the problem of loss responsive flows over wireless links. We use these guidelines to assess the approaches presented in Section 2.3 and the concept of flow-adaptive wireless links we proposed above.

Deployment

This concerns the required effort to deploy a particular solution, the incentives for the involved players to do so, but also the interworking with other network elements and protocols. Solutions that require changes to transport layer protocols, or implementations thereof, rely on a large scale effort to be incorporated into operating system software of wireless hosts and/or wireless network gateways (see dark shaded boxes in Figure 1). Pure transport solutions have additional drawbacks. They not only need consensus in the wireless industry, but also require upgrading, e.g., the large base of existing web servers to become effective. Also, introducing transport layer state in the network fails when network layer encryption [22] spans the gateway[7], and hard-state solutions further complicate cell handover. Deployment is also a concern for flow-adaptive approaches as their viability depends on the deployment of proposals like [6]. However, the advantage of flow-adaptive solutions is that they can be realized solely by the manufacturers of wireless networks and access devices. Also, flow-adaptive link layer solutions are orthogonal to the deployment of explicit congestion notification mechanisms [36], which would obsolete some of the solutions of Section 2.3.

Error Control Performance

Poor error recovery performance for reliable end-to-end flows like TCP comes in two forms. First, the size of the units that are re-/transmitted over the wireless link may not be optimized to the error characteristics of the link, leading to low goodput[8], a waste of radio resources (e.g., spectrum and transmission power) and useless load onto a potentially shared link. Secondly, with decreasing goodput over the wireless link, schemes that only rely on end-to-end error recovery pose an increasing unfair load onto a shared best-effort network due to a potentially high fraction of packets that have to retransmitted over the entire path. Both topics are discussed in detail for the case of a GSM network in [24]. This rules out pure transport layer solutions [29][38] as a general solution because the path MTU is often not the right choice as the retransmission unit size. Error control performance is the strongest argument in favour of solutions which are based on pure link layer error control. A similar line of argumentation applies to unreliable but delay-sensitive flows. The challenge here is to find the optimal amount of channel coding required to achieve a target range of user data bandwidths versus residual loss rates. Again, the most efficient solution requires knowledge of the ever changing local error characteristics. This favours flow-adaptive solutions, and is supported by the arguments in [37].

General Purpose vs. Dedicated Solutions

We believe that it is a wrong design decision to make the network, transport or any higher layer protocol, aware of mobility (cell handovers) [8][28] or wireless links [3][7][11]. A wireless network must hide the error characteristics of wireless links, while supporting seamless mobility[9]. Developers of existing and future networking protocols should be able to abstract from these purely

6. Note that this may interfere with differential encodings operating over that link [26].

7. Unfortunately, this is also true for transport layer header compression schemes.

8. The fraction of useful data over total amount of data transmitted over a given time.

9. A wireless access network which provides seamless mobility must "look like" *one* single-hop link.

local issues. We believe that it is also a wrong design decision to make link layer protocols aware of higher layer protocol semantics [4] or to install protocol-dependent gateways [2][15][23]. This would require upgrading for every new/changed higher layer protocol, adding to the deployment problems mentioned before. Also pure link layer solutions [1][13][14][20][31] have the problem of not being general purpose solutions as mentioned in Section 2.3 with respect to the undifferentiated use of ARQ. Flow-adaptive wireless links, on the other hand, are truly general purpose.

4. Dealing with Reliable Flows

In this section we propose that flows providing full end-to-end reliability (e.g., TCP) should encode that application layer QoS requirement in the IP header, as outlined in Section 3.1. We further argue in Section 4.1 why this requirement is best served by running fully-reliable ARQ over the wireless link. Explicitly encoding this information in the IP header allows a finer grained differentiation than proposed in [27]. There it is suggested to use the protocol identifier to derive the reliability requirement. It is assumed that the protocol identifier for UDP indicates that the flow is delay-sensitive and does not provide end-to-end reliability. This assumption fails when full reliability is implemented above UDP. Related work on this topic, briefly discussed in Section 4.2, suggests that fully-reliable ARQ at the link layer is likely to cause interference with TCP's error recovery. However, the analysis we present in Section 4.3 debunks this claim. Our analysis allows us to formulate guidelines according to which link layer error control should be implemented to minimize the probability of such interference. In Section 4.4 we discuss mechanisms that can be implemented at the link layer to fulfil those guidelines. In Section 4.5 we propose a new error recovery algorithm (TCP-Eifel) that can be implemented in TCP, further improving throughput and decreasing useless load onto the Internet should a more aggressive retransmission timer be implemented. The proposed mechanism eliminates the retransmission ambiguity problem [21], this enabling the TCP sender to detect spurious timeouts. In fact the algorithm uses spurious timeouts as an implicit cross-layer signal to the flow's sender, allowing it to prevent excessive spurious retransmissions.

4.1 Why Fully-Reliable ARQ at the Link Layer?

The end-to-end argument [37] tells us that it is not worth the effort to implement "perfect" reliability at the link layer. Yet, our design should eliminate non-congestion related packet loss to avoid the problems outlined in Section 2.1.

Semi-reliable link layer ARQ [20] solves this conflict by "giving up" retransmitting at a certain point, emphasizing end-to-end error recovery. The problem with this is to allow the link layer sender to decide *when* to "give up" and discard the packet. Optimizing this solution requires knowledge of the path's round trip time, which cannot be known at the link layer (unless it is carried in the IP header).

Fully-reliable link layer ARQ [1][13][14][31] does not have this problem but instead potentially interferes with end-to-end error recovery. Its advantage, however, is that it guarantees that any loss[10] at the link is due to congestion. This is exactly the right signal to give to the sender of a loss responsive flow (see Section 2.1). A link can have a transient outage that does not last longer than the order of the wireless link's round trip time (see Section 2.2), or its bandwidth drops instantly due to a suddenly increasing number of link layer retransmissions. Either case leads to congestion at the link and that fact should be signalled to the source of the flow.

If we can provide convincing arguments that interactions of fully-reliable link layer ARQ with end-to-end error recovery either happens only rarely (Section 4.3), the probability of their occurrence can be minimized by a well engineered wireless link (Section 4.4), or those interaction can be exploited to prevent excessive spurious retransmissions (Section 4.5), then fully-reliable link layer ARQ is the best approach to support fully-reliable end-to-end flows. Also, only fully-reliable link layer ARQ can avoid the potentially disastrous impact of lossy links on differential encodings, e.g., [18], as demonstrated in [26]. This might appear to be a minor issue, but header compression on a bottleneck link (likely to be the wireless access link) considerably increases throughput.

4.2 The Myth of Competing Error Recovery

Competing error recovery implies that two or more ARQ protocols concurrently transfer the same data object while operating independent of each other, usually on different protocol layers. The classic example is a reliable transport layer protocol operating over a link protected by a reliable link layer protocol. The risk is that this might lead to so-called *spurious timeouts* and subsequent *spurious retransmissions* on the transport layer. For the case of TCP, this was first investigated in [10]. The study concludes that at small link transmission error rates almost all packets retransmitted at the link layer are also retransmitted by the TCP sender (!). However, the presented analysis and simulation are flawed. The analysis assumes that the retransmission timeout value (RTO) is fixed. This is an unrealistic precondition that increases the likelihood of spurious timeouts at the TCP sender. The simulation ignores increases in the round trip time and RTO due to congestion; flow level packets are retransmitted entirely at the link layer, i.e., link layer segmentation is ignored; the link layer implements a stop-and-wait protocol; and only 2 instead of 4 times the variation is used for calculating the RTO. These are again unrealistic preconditions. Despite these questionable results, [4] uses them to argue against reliable link layer protocols that operate independent of a higher layer reliable flow like TCP. A tight coupling between link layer and transport layer error recovery is proposed, introducing the notion of "TCP-awareness" at the link layer (see Section 2.3). Without presenting sufficient evidence, the study presented

10. Apart from the more unlikely events of link layer error detection failures.

in [23] concludes that competing error recovery is the prevalent reason for TCP timeouts. Yet, another form of competing error recovery caused by interactions with TCP's fast retransmit algorithm [19] is discussed in [3]. But that problem is only caused by the retransmission scheme the authors propose. It allows out-of-order delivery and data duplication, leading to the mentioned interactions. This cannot happen with a reliable link layer protocol in the classical sense, as it only delivers data in-sequence and has to remove duplicate data.

To the contrary, a measurements-based analysis [26] concludes that spurious timeouts are rare. Although not explicitly focusing on the problem of competing error recovery, the studies in [5] and [12] indicate the same result. The analysis in [26] cannot be generalized, though, but only proves the case for the specific wireless link that was investigated. The next section provides a general analysis of the problem taking TCP as an example of a fully-reliable flow.

4.3 Analysing TCP's Retransmission Timer

When running a fully-reliable link layer ARQ protocol that guarantees in-order delivery and removal of duplicate data, the only concern with respect to competing error recovery are spurious timeouts at the transport layer sender[11]. The more conservative a transport layer retransmission timer is, the more "slack" is given for link layer retransmissions. We investigate this question for TCP which undoubtedly is the most widely deployed reliable transport protocol in the Internet today. We use [41] as the reference implementation of TCP in our analysis.

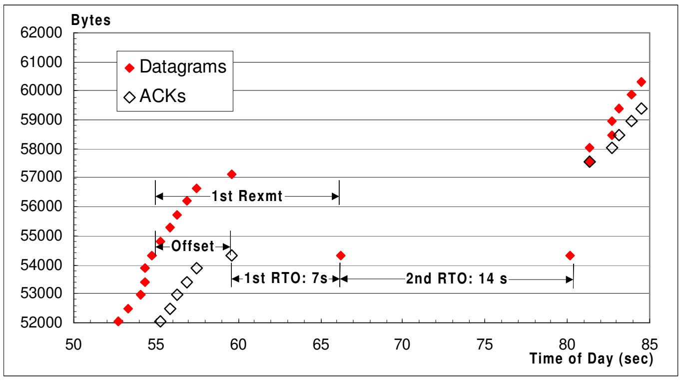

Figure 3: The RTO is *not* the retransmission timer.

We briefly review how TCP's adaptive retransmission timer works [19] and introduce some abbreviations. While data is in transit the sender samples the *round trip time* (RTT) by timing the difference between sending a particular byte and receiving the first acknowledgment (ACK) which covers that byte. The *retransmission timer* (Rexmt) for a particular byte[12] is the maximum time that the sender waits for the corresponding ACK before a retransmission is triggered. Each RTT sample is run through a filter and updates the so-called smoothed RTT estimator (SRTT), i.e., the average RTT as seen over some recent past. In addition, the sender measures how much the RTT varies over time and maintains that value in the so-called smoothed mean deviation estimator (RTTVar). SRTT and RTTVar are used to calculate the retransmission timeout value (RTO), as $RTO = SRTT + 4 \times RTTVar$. Only unambiguous samples are used to update the estimators otherwise a backed-off (doubled) RTO is used due to the *retransmission ambiguity problem* [21]. The RTO is then finally used to start/re-start the retransmission timer with every received ACK for new data. If implemented this way, the RTO is *not* the same as the retransmission timer. This is depicted in Figure 3, which shows a sender-side time/sequence plot. As can be seen the retransmission timer is always offset by roughly one RTT[13]. We call this offset the *retransmission timer offset*. Thus, the retransmission timer is the sum of two terms: (1) the RTO and (2) the retransmission timer offset[14]. Another important aspect - especially when considering wireless links - is how often RTT samples are measured by the TCP sender. Most implementations only time one packet per RTT, whereas a proposed extension to TCP [17] suggests timing every packet to much more closely track changes in the RTT.

In brief the retransmission timer is a prediction of the upper limit of the RTT. Spurious timeouts will not be seen as long as the RTT never grows faster than the retransmission timer can adapt. Hence, the goal of our analysis is to model the worst-case to determine the maximum *sudden* delay, i.e., from one packet to the other, that may be introduced by link layer ARQ without triggering a spurious timeout at the TCP sender. That is, we need to determine the minimum difference between the retransmission timer and the RTT. This worst-case approach leads to a fairly simple yet realistic model for our analysis based on the following assumptions.

- *Steady state analysis*
  We need to model a situation where the least amount of noise affects the RTT samples as this leads to a minimum value for RTTVar and thus to a minimum RTO, i.e., as close to RTT as possible. Therefore, we model a bulk data transfer in steady state where the sender does not compete with cross traffic at the bottleneck link. Any cross traffic leads to RTT variation to which the RTO responds rather sensitive, i.e., gets more conservative. We also assume ideal radio conditions, i.e., no delay variation caused by link layer ARQ. In fact we do not have to model link layer ARQ for our analysis at all.

11. Recall from Section 2.1 that throughout this paper we assume that flow level error and congestion control is implemented at the transport layer sender.
12. This might be misleading. In fact [41] implements only a single retransmission timer for each TCP connection.
13. The exact value depends on whether delayed-ACKs [40] are used and/or whether the sender has just send two or more back-to-back packets when probing for more bandwidth during slow-start or congestion avoidance.
14. Note that this is implementation dependent, e.g., LINUX 1.0 does not reset the retransmission timer with every ACK for new data.

• *Fixed bandwidth, latency, and IP packet size*
The relevant factors of the wireless link are its bandwidth and its *round trip latency* (RTL). As we have assumed that the sender does not compete with cross traffic all the link's bandwidth is available to the sender. The RTL is the component of the RTT that is independent of bandwidth and packet size. We assume that the bandwidth, RTL, and IP packet size are constant. This eliminates any noise that variation in these factors contribute. We assume the default IP packet size (576 bytes).

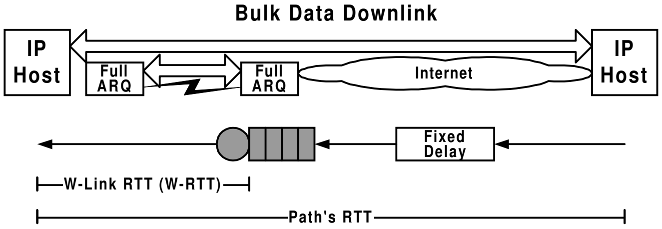

Figure 4: Model of a flow over a fully-reliable link.

• *Network-limited sender*
We assume a network-limited sender where the *congestion window* [19] limits the load the sender may impose onto the network at any time. The alternative is to model a receiver-limited sender where the window advertised by the receiver limits the load. The analysis of the latter is trivial when assuming the connection to be in steady state as the RTO will be equal to RTT. In this case the minimum difference between the retransmission timer and the RTT is the retransmission timer offset. However, connections are often not receiver-limited.

• *Minimal bottleneck buffer*
Large *bottleneck buffers* cause inflated RTTs [26]. As a side effect this leads to a more conservative retransmission timer due to an inflated retransmission timer offset. We model a "reasonable" bottleneck buffer as 1.5 times the pipe capacity. The *pipe capacity* is the minimal amount of data the sender needs to have in transit to fully utilize its bandwidth share of the bottleneck link. Due to the additive increase and multiplicative decrease algorithm that TCP uses to control the congestion window the chosen size of the bottleneck buffer guarantees that the pipe is always full while still some packets are always queued to cover transients. We assume that the queue is large enough to absorb any delay variation that packets experience on the segment of the path before the queue. Together with the above assumptions the latter allows us to model the "Internet cloud" as a fixed delay component (see Figure 4). If that seemed unrealistic we could increase the bottleneck buffer but that would lead to a more conservative retransmission timer offset. Note that this queue does not get in the way of link layer ARQ as that is implemented on a layer below.

• *The wireless link dominates the RTT*
The probability for link layer ARQ to cause spurious timeouts is low if the fraction that the wireless link contributes to the overall path's RTT (denoted as W-RTT in Figure 4) is small. Thus, in a worst-case scenario the wireless link itself is the entire path, i.e., the above mentioned fixed delay component of the "Internet cloud" is zero. This implies that the wireless link is the bottleneck link.

• *RTT sampling rate*
In our model we assume that every packet is timed to measure the RTT using the policies suggested in [17]. This makes the retransmission timer adapt much faster to RTT changes than it would otherwise which may in some cases counter our worst-case approach. However, we strongly believe that especially in a wireless environment where link characteristics can change considerably over short time scales it is important to track the RTT as close as possible. This is also discussed in [26].

• *Explicit congestion signal*
We assume that congestion is signalled explicitly at the end of each congestion avoidance cycle instead of through a dropped packet. This not only makes the analysis simpler but is also a pessimistic assumption. The reason for the latter is that the usual periodic packet drop at the end of each congestion avoidance cycle in most cases triggers the fast retransmit algorithm [19]. This leads to a series of DUPACKs that return to the sender. These in turn inflate the SRTT and cause RTT variation (RTTVar) because of the policy that [17] requires for echoing the timestamp option in DUPACKs.

• *Timer granularity*
It has been argued that one reason why TCP's retransmission timer is so conservative lies in the operating system's timer granularity, e.g., 500 milliseconds as used in BSD UNIX[15]. Some studies [5] even conclude that this is the main reason why spurious timeouts are rare. Although timer granularity contributes to the retransmission timer being more conservative, our analysis shows that it is only a minor factor. We assume a timer granularity of one millisecond.

• *Transfer direction*
Although Figure 4 suggests a downlink transmission, the transfer direction does not make a difference in our analysis.

In summary we are left with two parameters for our analysis: (1) the bandwidth of the wireless link, and (2) its round trip latency. As our analysis deliberately eliminates any variabilities, the sender-side connection state (congestion window, RTT, retransmission timer, etc.) evolves deterministically over time. In addition, this state is recursive as we are looking at a network-limited connection in steady-state. Hence, our analysis does not require any simulation but instead we modelled the recursion as an iteration on a spread sheet [25]. We defined three target metrics for our analysis: MINSlack, MEDSlack, and MAXSlack which are the minimum, median, and the maximum difference between the retransmission timer and the RTT. The sender-side connection state

15. There are other operating systems, though, that implement a granularity of 10 milliseconds.

is periodically in one of those three states. One could view MEDSlack as the typical connection state because half of the time of one congestion avoidance cycle is spent "below" and the other half "above" that state.

We first analysed why [26] came to the conclusion that spurious timeouts are so rare. The result is depicted in Figure 5 using the parameters of the network that was measured in that study. As expected the retransmission timer is extremely conservative even in steady state and with the least amount of noise in the RTT samples. It can be seen how the queue that builds up during congestion avoidance inflates the RTT until a congestion signal is received. In response the TCP sender "pauses" for about *1/2* of the RTT which allows the queue to drain. The next RTT sample causes a huge variation (RTTVar) as the corresponding packet finds the queue at its lowest size. This fires up the RTO because RTTVar has the same sign as a RTT increase and the opposite sign of a RTT decrease[16]. It is apparent how hyper-sensitive the RTO is to a high RTTVar while it comes down slowly when the RTTvar stays low. As the RTT grows it can be seen that the retransmission timer grows its distance to the RTT, due to increases in both the RTO and the retransmission timer offset. The retransmission timer offset prevents the retransmission timer from converging too close to the RTT which at one point almost exceeds the RTO[17]. A spurious timeout would be seen if the RTT graph crossed the Rexmt (retransmission timer) graph. Figure 5 also shows MINSlack, MEDSlack, and MAXSlack, both in milliseconds and also normalized to the sum of packet transmission delay - the *Pipe Clock* - and the RTL. The wireless link that was studied in [26] could impossibly cause such a huge sudden delay, i.e., from one packet to the other. The link would have been considered broken and terminated long before that. Note that we were looking at the absolute worst-case. In a real world measurement, noise in the RTT would have made the retransmission timer even more conservative, making a spurious timeout caused by competing error recovery in this network close to impossible. When the connection state at the TCP sender is in MEDSlack roughly 60 percent of the difference between the retransmission timer and the RTT comes from the RTO. The remaining part comes from the retransmission timer offset.

Looking at a bandwidth of 2 Mb/s and a RTL of 2 milliseconds, which should reflect the parameters of a WLAN system, yields a similar normalized result: a MINSlack of 2.3, a MEDSlack of 3.8, and a MAXSlack of 6.3. These are still vastly conservative upper limits for a sudden delay that a link layer ARQ protocol may introduce without causing any problems, especially when the link layer implements segmentation (see Section 4.4). This explains why [3] and [12] did not report of any spurious timeouts under realistic measurement conditions. In general we find that the more the RTL exceeds the Pipe Clock the lesser the normalized distance between the RTT and the retransmission timer becomes. Still when expressing the RTL as twice the Pipe Clock - which should be a reasonable ratio - the bandwidth can be scaled from 10 kb/s up to 5 Mb/s while the normalized results still remain extremely conservative: a MINSlack of 2.1-2.4, a MEDSlack of 3.5-3.6, and a MAXSlack of 5.8-5.9. Also, in this general case the fraction of the RTO in the difference between the retransmission timer and the RTT remains at 60 percent when the connection state is in MEDSlack.

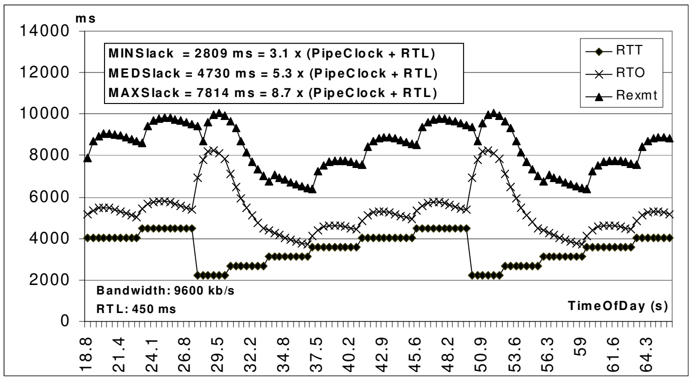

Figure 5: The retransmission timer in steady state.

Our model [25] allows to instantly explore a wide parameter space of which we only presented a small part due to space limitations. However, it allows us to conclude in general that TCP's retransmission timer is conservative enough to make a spurious timeout caused by a fully-reliable wireless link an unlikely event. Many of the worst-case assumptions described above have to come together to make such an event at all possible. A completely different question, though, is whether TCP's retransmission timer is too conservative and how it would change our result if it was tuned to be more aggressive. This question is studied in Section 4.5.

4.4 Preventing Spurious Timeouts at the Link Layer

So far we have argued why fully-reliable flows over wireless links are best served by operating fully-reliable ARQ at the link layer. This does not say anything about how link layer Forward Error Control (FEC) schemes[18] should be operated. For reliable flows the design and operational goal for the link layer is to transfer the flow's data reliably and as fast as possible. Metrics that might be of interest to delay-sensitive flows (e.g., packet jitter, range of available bit rate, residual packet error rate) are irrelevant for fully-reliable flows. This allows to tune FEC schemes aggressively[19], i.e., protecting the channel as weakly as possible to minimize both the overall transfer time of the flow's data and the radio resource usage (e.g. spectrum and transmission power). The only constraint is that most (e.g., 95 percent) sudden delay increases

16. This effect has already been pointed out in [19].
17. A case can be constructed where this actually happens which shows how important the retransmission timer offset is.
18. We use the abbreviation FEC where 'C' stands for 'Control' not 'Correction' to refer to all other (adaptive or non-adaptive) transmission schemes (power control, forward error correction, interleaving, frame length control, spreading factor control, etc.) but *not* ARQ.
19. Other design metrics also play an important role in this respect, e.g. the latency introduced by interleaving [27].

from one packet to the other should not exceed the limits given in Section 4.3. This design goal lends itself well to be analysed in wireless link simulators. The scenario to be evaluated is a link which operates at ideal radio conditions for a certain time - long enough for the RTO to converge to its minimum - and then drops to a known worst-case radio condition. The resulting sudden delay increase per packet should then be evaluated to ensure that it stays within the above mentioned constraint. The results of [24] and [26] show that, e.g., the GSM circuit-switched data link fulfils these guidelines. To the contrary the results indicate that FEC on that link is not performed aggressively enough, leaving potential for further improvements. The results presented in [12] draws similar conclusions for a WLAN link.

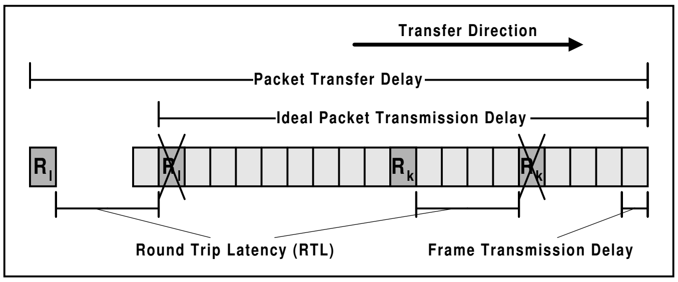

Figure 6: Impact of retransmissions of link layer segments.

Nevertheless, some techniques can be applied at the link layer to limit sudden delay increases. Below we list sources of delay caused by link layer ARQ and suggest techniques to limit their effect. In the following the term *packet* implies an IP protocol data unit and the term *frame* implies a link layer protocol data unit.

• *Delay caused by frame retransmissions*
Packet are often too big to yield efficient ARQ performance over wireless links [24]. Therefore link layer protocols often perform segmentation of packets into smaller frames. For example a frame size of 30 bytes (24 bytes payload) is used in [14]. The impact of frame retransmissions on packet transfer delay depends on the "position" of the frame within the packet. As long as the transmission delay for the remaining frames belonging to a particular packet is larger than the link's RTL (defined in Section 4.3), the packet transfer delay is only increased by the frame transmission delay. On the other extreme, if the last frame of a particular packet has to be retransmitted then the packet transfer delay is additionally increased by the link's RTL. Both cases are illustrated in Figure 6. The latter has an especially pronounced effect if such a frame has to be retransmitted more than once. This emphasizes the importance of minimizing the latency when designing wireless links. In addition, a technique can be implemented where the per frame FEC protection level is incrementally increased for frames that have to be retransmitted multiple times and/or for frames at the end of a packet.

• *Stalled window*
Link layer ARQ commonly implements sliding-window flow control. Hence, the maximum window size of the ARQ protocol is of key importance if the frame error rate is considerable. Together with the RTL and frame transmission delay, the maximum window size determines the maximum number of retransmissions for a particular frame before the sender cannot transmit any further but has to wait until that frame is acked. For example [14] uses a maximum window of 61, the RTL of commercially deployed systems is about 320 milliseconds, and the frame transmission delay is 20 ms. Thus, the sender is blocked from sending whenever a frame has to be retransmitted more than twice. [24] measured these events, although only when the radio quality was low. The same technique mentioned above can be applied for this situation in that the per frame FEC protection level is incrementally increased as soon as stalled window situation becomes imminent.

4.5 Preventing Spurious Retransmissions in TCP

The analysis provided in Section 4.3 and related work [32] suggest that TCP's retransmission timer is too conservative. The main reasons lie in the sensitivity of the RTO to variation in the RTT and the retransmission timer offset. A certain retransmission timer offset is needed[20], however, to allow the fast retransmit algorithm [19] to become effective in case the RTO has converged too close to the RTT. But we believe that the current retransmission timer offset of one RTT is too conservative[21] especially because it scales with the amount of buffering at the bottleneck [26]. Also, if the sender times every packet - which we promote when a wireless link is part of the end-to-end path (see Section 4.3) - the justification given in [19] (appendix C) to multiply the RTTVar with a factor of 4 (instead of the earlier proposal of 2) in the RTO calculation does not hold any more. The reason is that by timing every packet, the RTO adapts fast enough so that it is not required to value variation in RTT so high. Thus, reasons exist to change TCP's retransmission timer to be more aggressive which in turn challenges the results we obtained in Section 4.3.

We argue that an adaptive transport layer retransmission timer should not be tuned to prevent all spurious timeouts as this results in a retransmission timer which is overly conservative. This has a negative impact on TCP's performance whenever the sender has to resort to a (long) timeout to recover a lost packet. This affects interactive applications but also bulk data transfers as soon the receiver's receive buffer is exhausted to absorb any further out-of-order packets. As a result the sender is blocked from sending any further packets. Instead, we believe that an adaptive transport layer retransmission timer should be "reasonably" conservative while a sender should be able to detect spurious timeouts and react appropriately. The latter is studied in the remainder of this section leaving research on a new algorithm to control TCP's retransmission timer for further study.

---

20. As long as at least 4 segments are outstanding (one that is lost and three that trigger DUPACKs).
21. The time required to receive three DUPACKs would be more appropriate. This could, e.g., be approximated as 4 x SRTT/cwnd, where cwnd is the current congestion window in multiples of MSS.

Spurious timeouts caused by a sudden excessive RTT increase have a disastrous impact causing an entire window's worth of packets to be retransmitted. Examples of sudden excessive RTT increases include route changes and delay caused by link layer ARQ. This becomes a concern if a more aggressive retransmission timer is used at the transport layer than the one currently implemented in [41] for TCP as discussed in Section 4.3. As shown in Figure 7, the problem with spurious timeouts is that a sender cannot resolve the retransmission ambiguity problem [21]. This fools the sender into believing that all the (too late) original ACKs returning after the timeout are ACKs for the retransmissions leading to the retransmission of an entire window. Furthermore, the duplicate data packets trigger DUPACKs which in turn trigger the fast retransmit algorithm [19]. This causes yet another unnecessary retransmission and a decrease in the sender's transmission rate. Figure 7 visualizes this effect which we forced on a direct cable connection between two hosts. We developed the 'hiccup' routine that pauses the outbound interface for a user supplied delay. Figure 7 shows both the TCP sender and receiver time/sequence plot in which the receiver trace is offset by 10,000 bytes so that it does not overlap with the sender trace. The sender and receiver traces are *not* correlated in time. We "zoomed in" the relevant part of the trace which is shown entirely in the upper left corner of Figure 7.

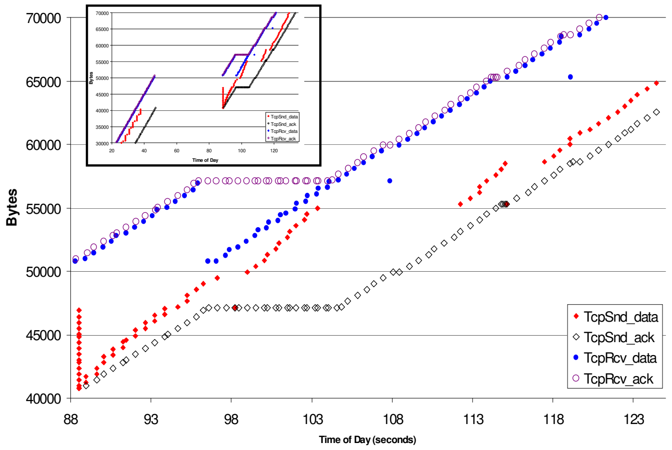

Figure 7: The effect of excessive packet delay.

Detailed explanation of Figure 7 *for the interested reader: We connected two BSDi 3.0 UNIX hosts via a direct serial cable running PPP (Point-to-Point Protocol) at 9.6 kbit/s. Half way through the measurement we called 'hiccup' which blocked the outbound interface for about 50 seconds. All ACKs that are outstanding at that moment return to the sender as usual and clock out further data packets. The reason why these are not seen in the plot until after the block is released, is that 'hiccup' queues the packets "above" the packet filter. When the block is released all packets queued by 'hiccup' are placed into the interface buffer at once. That's why they all get the same timestamp by the packet filter. Blocking the interface for 50 seconds actually caused two retransmissions of the "oldest" outstanding packet which both are placed in the 'hiccup' queue. This can be seen in the receiver trace as the same packet is received in second 96 and again in second 97. The rest of the trace has been explained above, apart from one artifact in second 114. At that moment the receiver sends three DUPACKs without any apparent reason. This in turn triggers yet another fast retransmit. We do not have an explanation for this phenomena but believe that this is "non-standard" TCP behavior.*

We propose a new TCP algorithm (TCP-Eifel) that solves the problem by eliminating the retransmission ambiguity to allow the TCP sender to detect spurious timeouts and prevent unnecessary retransmissions. I.e. the goal of the algorithm is to avoid "the network equivalent of pouring gasoline on a fire" [19]. In fact one could view spurious timeouts as an *implicit cross-layer signal* through which, e.g., a wireless bottleneck link can signal that the radio quality has decreased too much to allow for the sender's current transmission rate. The key advantage of implicit over explicit cross-layer signals (see Section 2.3) is that implicit signals are reliable as they cannot get lost.

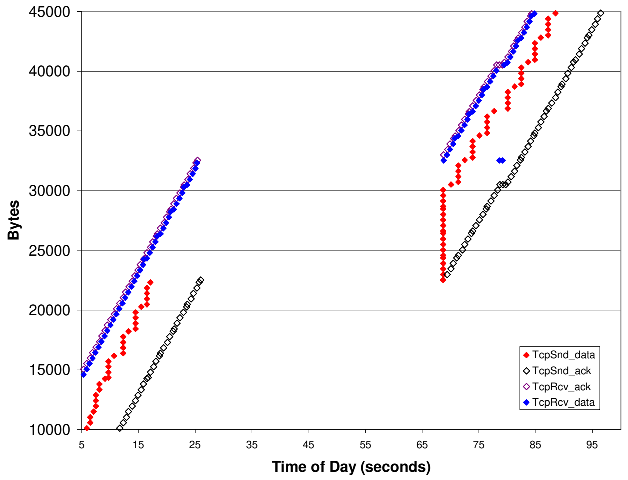

Figure 8: Spurious timeout detection (TCP-Eifel) at work.

To eliminate the retransmission ambiguity problem, the sender has to be able to discriminate between an ACK for the first transmission of a packet and an ACK for its retransmission. This is trivial when using the TCP timestamp option [17] where the timestamp in each packet can serve as a unique packet identifier which the receiver reflects in the corresponding ACKs. In fact our first implementation of the algorithm uses the timestamp option for that purpose. The timestamp of the first ACK that acks a retransmission triggered by a timeout determines whether that ACK corresponds to the original transmission of a packet or its retransmission. In the former case the algorithm detects that the timeout that the sender had taken was a spurious timeout and performs the following actions:

- the congestion window and the slow-start treshold [41] are restored to the value that they had before the spurious timeout occurred, taking the late ACK into account, and
- the RTT sample that can be derived from the (late) original ACK is used to re-calculate the RTO and re-start the retransmission timer, i.e., the backed-off RTO [21] is not used in this case.

Under "normal" circumstances this prevents all but the first spurious retransmission. This is shown in Figure 8 which is based on the same "hiccup measurement" described above. After the spurious timeout has been detected, the sender resumes normal transmission at the same rate as before. In case only a single spurious retransmission occurred, every late original ACK following the detection clocks out a new data packet. If more than one spurious retransmission occurred this has to be taken into account to comply with the conservation law [19]. The same idea upon which this algorithm is based can also be used to detect packet re-orderings that falsely triggered the fast retransmit algorithm.

Evaluating the performance improvement of this algorithm is difficult, as it depends on the target network environment and the implementation of a more aggressive TCP retransmission timer. We would have to implement such a retransmission timer and introduce "realistic" sudden delay increases to provoke spurious timeouts. Hence, we could make the performance improvement look as good or as bad as we wanted. It merely depends on how aggressive the retransmission timer is implemented and/or how "realistic" the sudden delay increases are chosen. We therefore deem a performance evaluation as inappropriate as long as TCP's retransmission timer stays as conservative as it is today (see Section 4.3).

Although timestamps work well as a first solution, we think that the required 12 bytes option field in every TCP header is too much overhead. Instead, we propose using 2 bits each way using 4 of the 6 reserved bits in the TCP header[22]. This allows the sender to unambiguously mark the first three transmissions of a packet[23]. This should be sufficient, as a TCP sender that has to transmit a single packet more than three times has more serious problems than spurious timeouts. Otherwise, the timestamp algorithm remains mostly unchanged. As with the timestamp field, the 2-bit solution also requires the receiver to reflect these bits in the corresponding ACKs. However, unlike with timestamps, no further intelligence needs to be implemented at the receiver. The 2 bits of a received packet are merely echoed in the corresponding ACK. Instead of including the timestamp in every packet the sender stores it with each unacked packet. The same policies outlined in section 3.4 of [17] apply.

The algorithm we propose, be it based on timestamps or on the 2-bit alternative, certainly has the same deployment concerns as any change to TCP (see discussion in Section 3.2). Nevertheless, we believe that any future reliable transport protocol must prevent spurious retransmissions.

5. Conclusion and Future Work

In this paper, we introduced the important new concept of flow-adaptive wireless links. Its novelty lies in the use of explicit QoS information in the IP header to dynamically adapt physical and link layer error control schemes for each flow sharing the link. The key advantages of our approach over related work are its applicability to any type of flow and the interoperability with network layer encryption. Other advantages of flow-adaptive wireless links is that implementations of this concept are confined to the "wireless edges" and wireless transit links of the Internet. It is also orthogonal to the deployment of explicit congestion notification mechanisms [36]. The limitation of our solution is that applications today do not explicitly include their QoS requirements in their flow's headers. Thus, making this approach viable requires further standardization, adoption, and deployment of the differentiated services framework [6].

We demonstrated how flow-adaptive wireless links can be implemented to support reliable flows and revealed how extremely conservative TCP's retransmission timer is. We argued why adaptive transport layer retransmission timers should not be tuned to prevent all spurious timeouts. We proposed a new error recovery algorithm for TCP (TCP-Eifel) that detects spurious timeouts, and uses these as an implicit cross-layer signal to prevent unnecessary retransmissions. As with any potential change to widely used protocols, our proposed algorithm faces deployment challenges, even if we use TCP options for incremental deployment. Nevertheless, we believe that any future reliable transport protocol must prevent spurious retransmissions.

Finally, we argued that flow-adaptive wireless links are a true end-to-end solution. In this paper, we have concentrated on reliable flows, leaving unaddressed issues related to other kinds of flows. In particular, we have not dealt with various forms of real-time flows or semi-reliable flows as discussed in Section 3.1. Flow-adaptation for these kinds of flows remains an open research challenge, as the use of Internet-based applications like radio, IVR-based (Interactive Voice Recognition) user interfaces, and multi-user video games increase in importance. A key future challenge is to identify the right information required in the IP header, to drive error control decisions at wireless links along the path. We continue to investigate these open questions.

Acknowledgments

Randy Katz, Bela Rathonyi, Keith Sklower, Jean Bolot, Michael Meyer, Almudena Konrad, Anthony Joseph.

References

[1] Ayanoglu E., Paul S., LaPorta T. F., Sabnani K. K., Gitlin R. D., A*IRMAIL: A link-layer protocol for wireless networks*, Wireless Networks, Vol. 1, No. 1, pp. 47-60, February 1995.

[2] Bakre A., Badrinath B. R., *I-TCP: Indirect TCP for Mobile Hosts*, In Proceedings of ICDCS 95, May 1995.

[3] Balakrishnan H., Padmanabhan V., Seshan S., Katz R. H., *A Comparison of Mechanisms for Improving TCP Performance over Wireless Links*, In Proceedings of ACM SIGCOMM 96.

[4] Balakrishnan H., Seshan S., Katz R. H., Improving reliable transport and handoff performance in cellular wireless networks, Wireless Networks, Vol. 1, No. 4, pp. 469-481, December 1995.

22. Two of these 6 reserved bits are used by [36].
23. A similar solution has been proposed in [16] using the so-called subsequence field. This was pointed out by Phil Karn on the PILC (Performance Implications of Link Characteristics) mailing list.

[5] Bhagwat P., Bhattacharya P., Krishna A., Tripathi S. K., *Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs*, Wireless Networks, Vol. 3, No. 1, pp. 91-102, 1997.

[6] Blake S. et al., *An Architecture for Differentiated Services*, RFC 2475, December 1998.

[7] Brown K., Singh S., *M-TCP: TCP for Mobile Cellular Networks*, Computer Communications Review, 27(5), October 1997.

[8] Cáceres R., Iftode L., *Imroving the Performance of Reliable Transport Protocols in Mobile Computing Environments*, IEEE JSAC, Vol. 13, No. 5, pp. 850-857, June 1995.

[9] Clark D. D., Lambert M. L., Zhang L., *NETBLT: A bulk data transfer protocol*, In Proceedings of ACM SIGCOMM 87.

[10] DeSimone A., Chuah M. C., Yue O.-C., *Throughput Performance of Transport-Layer Protocols over Wireless LANs*, In Proceedings of the IEEE Globecom 93.

[11] Durst R. C., Miller G. J., Travis E. J., *TCP Extensions for Space Communications*, In Proceedings of ACM MOBICOM 96.

[12] Eckhardt D. A., Steenkiste P., *Improving Wireless LAN Performance via Adaptive Local Error Control*, In Proceedings of IEEE ICNP 98.

[13] ETSI, *Digital cellular communications system (Phase 2+); General Packet Radio Service (GPRS); Mobile Station (MS) Base Station System (BSS) interface; Radio Link Control / Medium Access Control (RLC/MAC) protocol*, GSM Specification 04.60, Version 6.1.0, August 1998.

[14] ETSI, *Radio Link Protocol for data and telematic services on the Mobile Station - Base Station System (MS-BSS) interface and the Base Station System - Mobile Switching Center (BSS-MSC) interface*, GSM Specification 04.22, Version 5.0.0, December 1995.

[15] Henderson T. R., Katz R. H., *Transport Protocols for Internet-Compatible Satellite Networks*, To appear in IEEE JSAC in 1999.

[16] ISO/IEC, *Information processing systems - Open Systems Interconnection - Connection oriented transport protocol specification*, International Standard ISO/IEC 8073, December 1988.

[17] Jacobson V., Braden R., Borman D., *TCP Extensions for High Performance*, RFC 1323, May 1992.

[18] Jacobson V., *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144, February 1990.

[19] Jacobson, V., *Congestion Avoidance and Control*, In Proceedings of ACM SIGCOMM 88.

[20] Karn P., *The Qualcomm CDMA Digital Cellular System*, In Proceedings of the USENIX Mobile and Location-Independent Computing Symposium, USENIX Association, August 1993.

[21] Karn P., Partridge C., *Improving Round-Trip Time Estimates in Reliable Transport Protocols*, In Proceedings of ACM SIGCOMM 87.

[22] Kent S., Atkinson R., *Security Architecture for the Internet Protocol*, RFC 2401, November 1998.

[23] Kojo M., Raatikainen K., Liljeberg M., Kiiskinen J., Alanko T., *An Efficient Transport Service for Slow Wireless Telephone Links*, IEEE JSAC, Vol. 15, No. 7, pp. 1337-1348, September1997.

[24] Ludwig R., Konrad A., Joseph A., *Optimizing the End-to-End Performance of Reliable Flows over Wireless Links*, To appear in Proceedings of ACM/IEEE MOBICOM 99.

[25] Ludwig R., *Model of the TCP Sender Connection State in Equilibrium*, available at http://iceberg.cs.berkeley.edu/publications.html, January 1999.

[26] Ludwig R., Rathonyi B., Konrad A., Oden K., Joseph A., *Multi-Layer Tracing of TCP over a Reliable Wireless Link*, In Proceedings of ACM SIGMETRICS 99.

[27] Ludwig R., Rathonyi B., *Link Layer Enhancements for TCP/IP over GSM*, In Proceedings of IEEE INFOCOM 99.

[28] Manzoni P., Ghosal D., Serazzi G., *Impact of Mobility on TCP/IP: An Integrated Performance Study*, IEEE JSAC, Vol. 13, No. 5, pp. 858-867, June 1995.

[29] Mathis M., Mahdavi J., Floyd S., Romanow A., *TCP Selective Acknowledgement Options*, RFC 2018, October 1996.

[30] Mathis M., Semke J., Mahdavi J., Ott T., *The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm*, Computer Communications Review, 27(3), July 1997.

[31] Nanda S., Ejzak R., Doshi B. T., *A Retransmission Scheme for Circuit-Mode Data on Wireless Links*, IEEE JSAC, Vol. 12, No. 8, October 1994.

[32] Paxson, V., *End-to-End Routing Behavior in the Internet*, IEEE/ACM Transactions on Networking, Vol.5, No.5, pp. 601-615, October 1997.

[33] Perkins C., *IP Mobility Support*, RFC 2002, October 1996.

[34] Postel, J., Internet Protocol, RFC 791, September 1981.

[35] Postel, J., Transmission Control Protocol, RFC793, September 1981.

[36] Ramakrishnan K. K., Floyd S., *A Proposal to add Explicit Congestion Notification (ECN) to IP*, Internet Draft, Work in progress, January 1999.

[37] Saltzer J. H., Reed D. P., Clark D. D., *End-To-End Arguments in System Design*, ACM Transactions on Computer Systems, Vol. 2, No. 4, November 1984.

[38] Samaraweera N. K. G., Fairhurst G., *Reinforcement of TCP Error Recovery for Wireless Communication*, Computer Communications Review, 28(2), April 1998.

[39] Stevens W. R., *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, RFC2001, January 1997.

[40] Stevens W. R., *TCP/IP Illustrated, Volume 1 (The Protocols)*, Addison Wesley, November 1994.

[41] Stevens W. R., *TCP/IP Illustrated, Volume 2 (The Implementation)*, Addison Wesley, November 1994.

[42] The TCP-Friendly Website, http://www.psc.edu/networking/tcp_friendly.html.