SCHOOL OF ENGINEERING - STI
SIGNAL PROCESSING INSTITUTE
Christophe De Vleeschouwer and Pascal Frossard

CH-1015 LAUSANNE

Telephone: +4121 6932601
Telefax: +4121 6937600
e-mail: pascal.frossard@epfl.ch

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

EXPLICIT WINDOW-BASED CONTROL IN LOSSY PACKET NETWORKS

Christophe De Vleeschouwer and Pascal Frossard

Swiss Federal Institute of Technology Lausanne (EPFL)

Signal Processing Institute Technical Report

TR-ITS-2006.016

October 17th, 2006

Part of this work has been submitted to IEEE Transactions on Communications.

This work has been supported by the Swiss NSF, under grant PP002-68737, and by the Belgian NSF.

Explicit window-based control in lossy packet networks

Christophe De Vleeschouwer and Pascal Frossard
Ecole Polytechnique Fédérale de Lausanne (EPFL)
LTS4 - Signal Processing Institute - EPFL
1015 Lausanne, Switzerland

Abstract

This paper addresses the problem of fair allocation of bandwidth resources on lossy channels and heterogeneous networks. It discusses more particularly the ability of window-based congestion control to support non-congestion related losses. We investigate methods for efficient packet loss recovery by retransmission, and builds on explicit congestion control mechanisms to decouple the packet loss detection from the congestion feedback signals. For different retransmission strategies that respectively rely on conventional cumulative acknowledgments or accurate loss monitoring, we show how the principles underlying the TCP retransmission mechanisms have to be adapted in order to take advantage of an explicit congestion control framework. A novel retransmission timer is proposed to deal with multiple losses of data segments and, in consequence, to allow for aggressive reset of the connection recovery timer. It ensures significant benefit from temporary inflation of the send-out window, and hence the fair share of bottleneck bandwidth between loss-prone and lossy connections. Extensive simulations demonstrate the performance of the new loss monitoring and recovery strategies, when used with two distinct explicit congestion control mechanisms. The first one proposes a simple modification of TCP to support explicit congestion control, based on a coarse binary congestion notification from the routers. The second one, introduced in [1], relies on accurate feedback about congestion to compute fine congestion window adjustment. For both congestion control mechanisms, we observe that retransmissions triggered based on a precise monitoring of losses allow for efficient utilization of lossy links, and provide a fair share of the bottleneck bandwidth between heterogeneous connections, even for high loss ratios and bursty loss processes. Explicit congestion control, combined with appropriate error control strategies, can therefore provide a valid solution to reliable and controlled connections over lossy network infrastructures. In addition, our simulations also reveal that triggering retransmissions based on cumulative acknowledgments is only efficient -in terms of bottleneck utilization and fairness- at high loss rates when used in conjunction with an accurate and finely tuned congestion control. Therefore, we finally recommend the implementation of accurate feedback mechanisms either in the routers (about the level of congestion) or at the receivers (about a packet arrival), in order to provide a fair bandwidth allocation in hybrid networks with explicit window-based control.

I. Introduction

Congestion control is certainly imperative in packet networks, as it prevents important bandwidth outage that happens when the network is overwhelmed by too many packets. In addition, it tends to fairly distribute bandwidth resource among simultaneous connections and users, and avoids any one connection from swamping the links and switches between communicating hosts with an excessive amount of traffic. A natural way to achieve this goal is to limit the number of packets that are in transit between the sender and the receiver. Window-based congestion control mechanisms follow this idea by limiting, for each connection, the number of transmitted but yet to be acknowledged packets. A window-based congestion control algorithm not only controls the transmission rate, but also limits the maximum number of outstanding packets according to the congestion window size. It presents a reduced sensitivity, to inaccurate bandwidth estimation, in comparison to rate-based congestion control schemes. Window-based protocol achieve network stability by forcing the connection to obey a 'packet conservation' principle, which means that a new packet is not pushed into the network until an old packet leaves [2].

TCP is certainly the most well-known window-based congestion control protocol. It uses an implicit end-to-end window-based congestion control algorithm [3] where the intermediate network components do not provide any explicit support to the transport layer for congestion-control purposes. Congestion in the network is inferred by the end-systems, exclusively based on the network response (e.g., packet losses and delay). Loss is however known to be a poor signal of congestion, since congestion is not the only source of loss. Moreover, it only provides a late and coarse feedback about the network status. Consequently, TCP becomes inefficient and prone to instability and unfairness when packets are subject to non-congestion related losses [4], or when the delay-bandwidth product increases [5], [6], [7], [8]. Explicit congestion control mechanisms [1], [9], [10], where routers provide explicit feedback to the sender regarding network congestion, have been proved to solve the inefficiency and instability problem due to the imprecise and late feedback provided by congestion-related packet losses. These protocols are more responsive and stable than conventional TCP, and become especially beneficial when the delay-bandwidth product increases.

In this paper, we rather address the problem related to the ambiguous information provided by losses. Interpreting any kind of loss as a congestion notification results in flow starvation on lossy links. This problem is well-known, especially in wireless

This work has been funded by the Swiss NSF, under grant PP002-68737, and by the Belgian NSF.

environments, and has been extensively studied for TCP connections [4], [11], [12], [13]. All these works aim at increasing TCP robustness towards non-congestion related losses. Our work rather considers window-based congestion control protocols in general, and addresses the fundamental question of the ability of such protocols to support non-congestion related losses.

The sources of inefficiency for window-based congestion control protocols in presence of non-congestion related losses are mainly twofolds. First, the erroneous interpretation of a loss as a signal of congestion ends up in congestion window deflation. Second, the anchorage of the congestion window to the last acknowledged data segment prevents to send new data before a previously lost segment has been successfully retransmitted, which creates an indirect coupling between the occurrence of loss, and the effective rate of the connection. To circumvent the first issue, we promote the use of explicit congestion control mechanisms. As long as explicit information about congestion is received by the sender, there is no reason for the sender to infer the congestion state from losses or delay measurements. Hence, losses are not interpreted as a congestion signal anymore, and do not cause congestion window deflation, which in turns prevents all issues related to the discrimination of congestion and non-congestion related losses. In order to address the second issue, we aim at maintaining the connection active and efficient during loss recovery periods. This is done first, by resetting the connection recovery timer each time a packet acknowledgment, either new or duplicate, reaches the source, and second, by inflating the sender window in response to duplicate acknowledgments. In addition, a novel retransmission timer is also proposed to deal with multiple losses of data segments. Hence, relying on the explicit congestion control framework, we propose to implement aggressive retransmission mechanisms in parallel to the emission of novel data segments by the sender.

Two explicit congestion control algorithms are considered in this paper, in order to validate the proposed loss resilience mechanisms. The first one is the eXplicit Control Protocol (XCP), introduced by Katabi and al. [1]. The second one is a novel eXplicit TCP-like congestion control algorithm (XTCP), where routers only provide a coarse binary feedback about congestion. In addition, we have quantified the benefit provided by accurate monitoring of losses, as offered by either selective acknowledgment [14], or the identification of the packet that triggered an acknowledgment [15]. We rely on NS simulations to evaluate the potential of the proposed loss management and retransmission mechanisms in the explicit window-based congestion control framework. Our findings can be summarized as follows.

- In a simple yet representative network topology, where a bottleneck link is shared by lossless and lossy connections, we observe that the proposed loss recovery mechanisms allow to utilize fully the bottleneck link, and provide a fair share of the bottleneck bandwidth between all connections. The conclusion holds both with XCP and XTCP, as long as the loss ratio is moderate (below 5%). Explicit congestion control, combined with appropriate error control strategies, can therefore provide a valid solution to reliable and controlled connections over lossy network infrastructures.
- For high loss ratios or bursty loss processes, we observe that XTCP only achieves fairness between lossless and lossy connections if the sender can rely on precise feedbacks from the receiver about packets arrival. Accurate monitoring of losses and the corresponding anticipation of retransmissions are thus required when the routers only provide a coarse (and delayed) binary feedback about congestion. In contrast, an instantaneous and finely tuned congestion notification allows the connection to face substantial loss rates only based on the information conveyed by cumulative acknowledgments. Therefore, we recommend the implementation of accurate feedback mechanisms either from the routers (about the level of congestion) or from the receivers (about a packet arrival) in explicit window-based controlled protocols, in order to preserve efficiency over lossy links and handle fair bandwidth allocation in hybrid (wired and wireless) networks.
- The strategy proposed to manage the retransmission and recovery timers in the explicit framework, combined with careful inflation and deflation of the send-out window, significantly amplifies the benefit obtained from temporary inflation of the send-out window, as initially introduced by the fast retransmit and fast recovery mechanisms used in TCP Reno [3] or NewReno [16]. We conclude that it is the positive synergy between the creation of a novel retransmission timer, and the careful adaptation of conventional retransmission and recovery mechanisms that allow for the success of explicit window-based congestion control in lossy environements.
- The proposed eXplicit TCP protocol is shown to fairly co-exist with TCP, in a single queue of an XTCP-enabled router, which is certainly an attractive characteristic for its deployment.

The paper is organized as follows. Section II introduces the framework of our study, and defines the state variables that characterize a window-based connection. We then explain how to implement loss-resilient mechanisms in the explicit congestion control framework, based on information conveyed by acknowledgment packets. Such information either consists in the number of the latest data segment received in-sequence, or the numbers of all packets that have reached the receiver, in Section III and IV, respectively. Section V later presents the XCP and XTCP explicit congestion control algorithms that are used to validate the proposed loss resilience mechanisms. Section VII and VIII validate and compare loss resilience mechanisms in XCP and XTCP, based on NS simulations. Finally, Section IX puts our contribution in perspective with earlier related works, while Section X concludes.

II. Preliminaries

A. Terminology for window based protocols

As the goal of our paper is to explore the ability of explicit window-based congestion control protocols to support transmission losses, we briefly recall here the state variables that characterize a window-based connection. The section follows conventional

TCP terminology, so that readers that are familiar with these notions may skip the section.

In order to regulate packet transmission, the senders uses feedbacks from the receiver about the state of session. We limit our study to window-based control protocols based on positive acknowledgment (ACK), where the receiver sends feedback information in response to correctly received packets. The feedback information is based on the data sequence number and, possibly on the packet sequence number present in the packet header. The **data sequence number** associated to a packet identifies the data segment conveyed by the packet. Two transmissions of the same data are thus characterized by the same data sequence number. The **packet sequence number** identifies each packet transmitted by the sender. It corresponds to a counter incremented by one each time a new packet is sent. Two packets that (re)transmit the same data at different time instants have thus the same data sequence number, but distinct packet sequence numbers. Based on the data sequence number definition, a **cumulative acknowledgment** with a sequence number equal to N, indicates that all the data segments with a data sequence number up to and including N have been correctly received. At any time, the *lack* state variable records the largest cumulative acknowledgement ever received by the sender.

By definition, window based congestion control limits the number of transmitted packets, which have not been acknowledged yet. This number of packets in transit between sender and receiver, is referred to as the **congestion window size**, and is generally denoted *cwnd*. This variable is the one that constraints the rate of the connection based on the network state of congestion. It is adjusted based on the information explicitly received from the routers, or implicitly inferred from the observation of the network behavior (loss and delay). Along with the congestion window, the **send-out window** denoted *swnd*, describes the data that are eligible for transmission, which are the data whose sequence number lies between $lack$ and $lack + swnd$. In practice $swnd \geq cwnd$, and the difference between $swnd$ and $cwnd$ corresponds to data packets that have already left the network, and are stored at the receiver. Note that $swnd$ is upper bounded by the receiver advertised window denoted $rwnd$, which reflects the receiving buffer capacity. Finally, the data sequence number of the next segment to be considered for transmission is given by the *nextseq* state variable. The strategy employed at the server, and in particular the update of $lack$, $swnd$ and $nextseq$ upon reception of receiver acknowledgments or timer expiration, and the monitoring of $swnd$, directly drives the behavior of the congestion control algorithm, as described in the next sections.

B. Overview of TCP retransmission and recovery mechanisms

Before describing in details the mechanisms we propose in order to improve window-based connection over lossy links, we present a brief overview of TCP retransmission and recovery mechanisms. Based on the assumption that the receiver sends out the largest possible cumulative acknowledgment at each packet arrival, TCP implements three mechanisms to recover from a loss, namely two retransmission schemes respectively initiated by a duplicate or a partial acknowledgment, and the connection reset triggered by a recovery timer. We describe below these three mechanisms, and set the framework for the description of our improved solutions. Readers that are familiar with TCP can skip the remaining of this section.

a) Retransmission based on duplicate acknowledgment.: In the absence of loss or packet reordering, any acknowledgment indicates the correct reception of novel data at the receiver, and is referred to as a **new ACK**. In presence of packet losses however, duplicate acknowledgments may be generated in response to the reception of data that have a higher data sequence number than any data segments that have not been received yet. Upon reception of a **duplicate ACK**, the sender infers that the data immediately following the largest acknowledged data, i.e., the $(lack + 1)^{th}$ data segment, has either been delayed or lost by the network. In practice, similar to the principle adopted by TCP Reno [3], our sender waits for several duplicate ACKs before it concludes that a packet has been lost, and retransmits it. We denote $dupackthreshold$ the number of duplicate ACKs necessary for the sender to infer the loss of the $(lack + 1)^{th}$ data segment. In the same time, the arrival of a duplicate acknowledgment at the sender indicates that a packet has reached the receiver and left the network. In accordance with the congestion window definition, the send-out window $swnd$ is incremented by one and a new data packet can be sent out in response to the arrival of a duplicate acknowledgment. Specifically, $swnd$ is monitored as the sum of $cwnd$ and $dupwnd$, where $dupwnd$ denotes a counter initialized to zero and incremented by one every time a duplicate ACK reaches the sender. In TCP Reno [3], the $dupwnd$ counter is reset to zero when the connection is reset, or each time a new ACK is received. We explain in Section III-A that an explicit congestion control framework advantageously supports fine adjustment of the $dupwnd$ counter.

b) Retransmission based on partial acknowledgment.: This retransmission mechanism has been proposed by the NewReno version of TCP [16], and is expected to help when multiple packets are lost from a single window of data. Among new data acknowledgments, NewReno distinguishes between **complete** and **partial acknowledgments**. Let $olack$ and $nlack$ respectively denote the largest data sequence number acknowledged before, respectively after the reception of a new ACK. A new ACK is defined to be a complete ACK if it acknowledges all the data segments that have been sent before the last (re)transmission of the $(olack + 1)^{th}$ segment. On the contrary, a new ACK is a partial acknowledgment if it only indicates the correct reception of a subset of these segments. To decide whether a new ACK is a partial or a complete acknowledgment, the sender uses the **recover** state variable, which indicates the largest sequence number sent out before the last retransmission of the next data to acknowledge (i.e., the $(olack + 1)^{th}$ data segment). Specifically, every time a data segment is retransmitted, either because a timer expires or a retransmission mechanism becomes active, $recover$ records the largest data sequence number

ever sent out by the sender. Upon reception of a new ACK, the sender compares $nlack$ with $recover$. If $nlack$ is strictly smaller than the $recover$ value, the sender infers that the new ACK is a partial acknowledgment. The sender interprets this information as the loss of the $(nlack + 1)^{th}$ data segment, and retransmits it. It is worth noting that in presence of the partial acknowledgment retransmission mechanism, the $dupacks$ counter defined above should only be reset upon reception of a complete acknowledgment in order to avoid multiple retransmissions of the same packet during the same round trip time.

c) Mechanism of last resort: the recovery timer.: In complement to retransmission mechanisms, all TCP implementations use a **recovery timer** as a recovery mechanism of last resort. The expiration of this timer indicates that the connection stayed idle for a while, and has to be reset. A **connection reset** simply consists in setting $cwnd$ to one, and $nextseq$ to $lack + 1$. All counters and timers are also reset to zero. The timeout that indicates the expiration of the recovery timer is generally set to a conservative estimation of the round trip time (RTT). In TCP [17], this timeout is defined as the sum of an exponential weighted average of the RTT samples, and of an estimate of the deviation of RTT samples from this average. In TCP, the recovery timer is only reset upon reception of a new ACK or upon reception of a duplicate ACK that causes a retransmission. We use the same timeout value in the next sections, where we propose to reinforce the recovery mechanisms in an explicit congestion control framework.

C. Explicit congestion control

The next sections describe how retransmission mechanisms can be adapted in the context of explicit congestion control, for improved performance and fairness in resource allocation. Fundamentally, the main difference between the implicit and explicit congestion control [1], [9], [10] lies in the fact that the explicit framework can rely on the explicit feedback provided by the routers to the sender about the level of congestion. Explicit congestion control has been shown to solve the inefficiency and instability problem due to the imprecise and late feedback provided by congestion-related packet losses. As an example, Katabi and al. [1] have designed an eXplicit Control Protocol, XCP, where the network uses precise and explicit congestion signaling to drive the reaction of senders to congestion information. The resulting protocol is more responsive and stable than conventional TCP, but was not designed to cope with non-congestion related losses.

In the explicit congestion control framework, the sender can however strictly and uniquely rely on the routers feedback to control the size of its congestion window. Hence, it does not have to interpret losses as a signal of congestion, and does not have to undertake actions to slow down the connection upon reception of duplicate ACKs. It should rather try to maintain the connection active and effective as long as (duplicate or new) ACKs are regularly received, as ACKs convey information about congestion and indicate that the connection is still alive. These are the characteristics that are exploited in the next sections, in order to improve the performance of explicit window-based congestion control in presence of losses. Detailed examples of practical implementations of explicit congestion control algorithms are presented later in Section V, where the routers use the packet header and acknowledgment to return congestion-related information to the source.

III. Improved retransmission mechanisms with explicit congestion control

This section proposes original retransmission and send-out window management mechanisms to recover from losses while preserving connection efficiency, in the context of an explicit congestion control framework. In summary, connection efficiency is preserved by (i) partial deflation of the send-out-window upon reception of a new ACK, (ii) reset of the recovery timer in response to any ACK, and (iii) definition of a novel retransmission timer. We describe these three mechanisms and explain how they interact and complement each others. As a main outcome, we show that the introduction of a new **retransmission timer** induces significant changes in the behavior of the retransmission and recovery mechanisms defined for TCP and generally accepted in the context of implicit congestion control. In short, the retransmission timer offers an improved strategy to handle multiple losses of a segment, and allows for more frequent resets of the recovery timer. This in turn increases the benefit obtained from a careful management of the inflation of the send-out window in presence of losses.

A. Partial deflation of the send-out-window

As explained in Section II-B, the arrival of a duplicate acknowledgment at the sender indicates that a packet has reached the receiver, and thus left the network. Hence, the send-out window $swnd$ is progressively and artificially inflated upon reception of duplicate ACKs. Upon reception of new data acknowledgment, the head of the send-out window is moved to the largest acknowledged data segment. It possibly oversteps a number of data segments whose earlier receptions have triggered duplicate ACKs, and incremented $dupwnd$. In order to maintain the ensure that the number of packets in transit is equal to the congestion window, $dupwnd$ has thus to be decremented, accordingly to the number of segments that which triggered duplicate ACKs in the past, but which are implicitly acknowledged by the reception of a new ACK. In other words, $dupwnd$ should be decremented by $nlack - (olack + 1)$ in the explicit congestion control framework, so as to preserve connection efficiency while recovering from losses. In contrast, $dupwnd$ is simply reset to zero upon reception of a new ACK in the conventional implementation proposed by TCP Reno [3]. This is because, in an implicit congestion control environment, the $cwnd$ back off anyway alleviates the potential advantage taken from a partial deflation of $dupwnd$.

B. Aggressive reset of the recovery timer

While duplicate ACKs should not be interpreted as a signal of congestion in an explicit congestion control framework, they indicate that the connection is alive, and even convey fresh information about congestion state. It is therefore important to keep the connection active upon reception of a duplicate ACK, and to postpone the expiration of the recovery timer. Hence, we propose to reset the recovery timer both in response to a new ACK and to a duplicate ACK when the congestion control is explicit. Note that, even if aggressive resets of the recovery timer ensure that the connection is maintained, the efficiency of that strategy strongly depends on the careful management of the temporary inflation of the send-out window described above. Both mechanisms closely interact and complement each others.

C. Retransmission based on a timer

This section introduces a novel retransmission timer in order to deal efficiently with multiple losses of a segment. It is well-known from the TCP literature that retransmission mechanisms based on duplicate or partial ACKs do not support multiple losses of the same segment, typically because the $dupacks$ state variable introduced in Section II-B is only reset to zero after the acknowledgment of new data[1].

To deal with multiple losses of a segment, TCP uses a recovery timer whose expiration causes the retransmission of that segment in the absence of a new ACK. In an explicit congestion control framework, we claim however that the recovery timer should be reset at every duplicate ACK, and that the connection efficiency should be preserved by progressive inflation of the send-out window. In these conditions, the multiple losses of a data segment result in a situation where $dupwnd$ goes to infinity (or at least to a value that pushes $swnd$ to $rwnd$), before the recovery timer gets the opportunity to expire and cause the retransmission of the lost segment. To circumvent the problem, we propose to implement a retransmission mechanism based on a novel **retransmission timer**. When the timer expires, the $(lack + 1)^{th}$ data segment is retransmitted. The timer is reset every time new data are acknowledged, and every time a data segment is retransmitted.

In more detail, a short timeout (i.e., time before timer expiration) results in fast retransmission, and rapid loss recovery. However, one should take care not to trigger retransmissions for packets that are still in transit. Stability becomes an important issue with retransmission based on a timer. A bad choice of the timeout value might cause the sender to inject a new packet into the network before an old one has exited. This violates the 'packet conservation principle' that guarantees stability for window-based transport protocols [2]. To avoid this problem, we propose to chose the retransmission timeout larger than the recovery timeout value[2], defined as a conservative estimation of the round trip time (see Section II-B). A retransmission timeout that is larger than the recovery timer guarantees that, in case of trouble, typically due to the underestimation of the round trip time estimation, the connection ends up in a recovery phase, and does not swamp the network with inadequate retransmissions. To validate this statement, and provide emperical evidence of the stability of our system, we have run simulations for which some connections deliberatly under-estimate the round trip time when computing the recovery and, consequently, the retransmission timeouts. We have observed that these connections did not strangle other connections, and that they were themselves strongly penalized by regular expirations of the recovery timer. We have also observed in the simulations presented in Sections VII-A and VIII-B that the expiration of the retransmission timer is always appropriate. Hence, we conclude that defining the retransmission timeout as a larger and scaled version of the recovery timeout prevents unstable behavior of the system.

From a functional point of view, it is worth noting that the retransmission timer triggers the required retransmissions before expiration of the recovery timer, even if its timeout is larger than the recovery timeout. This is because the explicit congestion control authorizes frequent resets of the recovery timer, i.e. each time either a duplicate or a new ACK is received, while only resetting the retransmission timer in response to a new ACK. That complementarity between the recovery and the retransmission timer is fundamental and probably represents one of the most important findings of our study. We later show in Sections VII-A and VIII-B that the proposed retransmission timer improves the connection efficiency significantly beyond the benefit already provided by partial deflation of $swnd$ and aggressive reset of the recovery timer.

D. Summary and discussion

The main differences between the retransmission and recovery mechanisms proposed in this paper, and the ones implemented in TCP are twofold. First, losses are not interpreted as a signal of congestion in the explicit control framework, which avoids the need to sharply reduce the congestion window in response to duplicate ACKs. It permits to keep the connection active and efficient as long as sufficient ACKs are received. The connection efficiency is preserved since the congestion window stays unchanged upon reception of a duplicate ACK, and the send-out window is only partially deflated upon reception of a new ACK. Second, the presence of a retransmission timer also contributes to preserve the connection efficiency as it allows for a reset of the recovery timer each time a new ACK or a duplicate ACK is received. Such an aggressive reset strategy is especially beneficial in conjunction with the partial deflation of the congestion window. Hence, we conclude that the window

[1]If $dupacks$ was reset to zero immediately after the retransmission of a packet, subsequent duplicate acknowledgments that correspond to the same window of emission would trigger an additional and probably useless retransmission.

[2]During our simulations, the retransmission timeout has been defined twice as large as the recovery timeout.

and timer management mechanisms introduced before nicely complement each others, and contribute to the robustness of the explicit congestion control framework in the presence of packet loss.

As a last comment, we emphasize that the proposed mechanisms are strictly dedicated to an explicit congestion framework, but can not contribute to improve the performance of an implicit congestion control algorithm. As an example, our simulations have revealed that the partial deflation of $dupwnd$, already proposed by TCP NewReno, brings a significant benefit when implemented in the context of an explicit congestion control, but does not significantly help in a classical TCP context. In TCP, duplicate ACKs cause a decrease of the congestion window size, which alleviates the benefit obtained from a partial deflation of $dupwnd$. Similarly, in an implicit congestion control framework, it is far better to reset the connection in presence of multiple losses, rather than to use a retransmission timer that keeps the connection alive, but severely strangled due to $cwnd$ back off.

IV. Loss monitoring with rich feedback

With the limited information available from cumulative acknowledgments, a sender can only learn about a single lost packet per round trip time. We consider here that the receiver sends a more detailed feedback than cumulative acknowledgments, in order to evaluate the benefit provided by rich feedbacks with window-based control on lossy connections. As previously, we consider again an explicit congestion control framework, and we discuss its ability to support connections over lossy links. We first present the protocols that allow to generate rich feedback with receiver acknowledgments. Then we describe the novel loss retransmission mechanisms that have been specifically designed to exploit this feedback information. Simulations later demonstrate the advantages offered by rich feedback in Sections VII and VIII.

A. Packet sequence number feedback

The main limitation of cumulative acknowledgments comes from the imprecise information they bring about the status of the connection. Upon reception of a duplicate ACK, the sender can not infer which exact packet has triggered the ACK, and consequently, which data segment has reached the receiver. A simple way to circumvent this limitation is to uniquely identify every packet that is sent on the network, and to force the receiver to include that unique identifier within the acknowledgments returned to the sender. In our simulator, this has been done by adding a field to the packet header that contains its packet sequence number. Remember from Section II that the packet sequence number is defined based on a counter incremented by one each time a new packet is sent. Every time the source sends a packet, it writes the state of the counter in the packet header, and increments the counter by one. This concept of packet sequence number has been introduced previously by Keshav and Morgan [15] for the design of efficient retransmission mechanisms in the context of rate-based congestion control, where the transmission of new packets and the loss recovery mechanisms are totally decoupled [15]. On the contrary, we are interested in window-based congestion control, and analyze the benefit to draw from a feedback including packet numbers when the emission of novel data segments is directly constrained by a send-out window, whose head is attached to the largest cumulative acknowledgment received by the sender.

Note that another way for the sender to learn about the data segments that have reached the receiver is the selective acknowledgment (SACK) option proposed for TCP [14]. We have not consider this mechanism here, but we expect that conclusions drawn from our implementation can be extended to SACK implementations.

B. Retransmission based on accurate loss monitoring

When informed about the data segments that have been correctly received, the sender can adopt intelligent strategies to retransmit the missing data. Based on the feedback about the packets that have been received in order and out of order, the sender updates a **loss monitoring window**. It records information about the segments that are still waiting for a cumulative ACK, and its size is limited by the largest number of out-of-sequence packets that can be buffered at the receiver (i.e., $rwnd$). We now define how the loss monitoring information is maintained, and later explain how this information is exploited to trigger data retransmissions.

Let $N < rwnd$ denote the size of the loss monitoring window, in packets. Given the largest acknowledged data sequence number ($lack$), the loss monitoring window stores the state of all segments whose data sequence number j verifies $lack < j < lack + 1 + N$. In practice, we use a circling buffer to store the state of the relevant data segments. Let $W[.]$ be an array of size N. At any time, $W[j \bmod N]$ stores the loss monitoring window state corresponding to the j^{th} data segment. Given $lack < j < lack + 1 + N$, $W[j \bmod N]$ is defined as follows:

- $W[j \bmod N] = FREE$, with $FREE$ being a constant flag value, when the j^{th} data segment has not yet been sent over the network;
- $W[j \bmod N] = RECV$, with $RECV$ being a constant flag value, when the j^{th} data segment has been received out of order by the receiver;
- $W[j \bmod N] = X$, with $X > 0$ being the packet sequence number of the latest packet sent over the network and conveying the j^{th} data segment, in any other case.

In more details, the loss monitoring window state variable is maintained as follows:

(a) First, each array position is initialized to the constant $FREE$ value, indicating that each block of the array is available to store the state of future data segments.
(b) When a packet is sent out, the loss-monitoring window is updated as follows. Let d denote the data sequence number of the data segment conveyed by the packet, and p be the packet sequence number. Then, $W[d \bmod N]$ is set to p, indicating that all packets with a packet sequence number larger than p have been sent out before the last emission of the d^{th} data segment. Recording this information is important for the retransmission mechanism proposed hereunder.
(c) Upon reception of a new ACK, the $lack$ state variable is updated. $W[j \bmod N]$ is reset to $FREE$, $\forall j$ such that $olack < j < nlack$. This indicates that the corresponding positions of the array are now available to store the state of future data segments. Note that before sending out a new data segment n on the network, the sender has to check that $W[n \bmod N] = FREE$, to ensure that the new segment will not exceed the storage capacity of the loss monitoring window.
(d) Upon reception of a duplicate ACK, $W[d \bmod N]$ is set to $RECV$, where d denotes the sequence number of the data segment conveyed by the packet that has triggered the ACK. It means that the d^{th} data segment has been correctly received. Note that packet number can be read from the header of the received ACK, and that the sender obviously knows which data segment has been sent in a given packet.

Given the state of the loss monitoring window $W[.]$, the design of novel retransmission mechanisms is driven by the following rules. First, a data segment can only be retransmitted once the sender has received $dupackthreshold$ acknowledgments, triggered by packets that have been sent out later. Second, the number of retransmissions per $dupackthreshold$ acknowledgments is limited to one. Third, a data segment should only be retransmitted once per RTT, so as to preserve the 'packet conservation' principle. To follow these rules, we define the state variable $rseqn$ to denote the sequence number of the data segment that is expected to be the best candidate for a retransmission. Among all data segments monitored in the window W, $rseqn$ is defined as the segment with the least recent (re)transmission, and for which the sender has no indication about correct delivery. Formally, given $W[.]$, $rseqn$ is defined by:

$$rseqn = \underset{lack+1<j<lack+1+N,\ W[j] \neq RECV/FREE}{\arg\ \min} W[j]. \quad (1)$$

Let $dupcount$ then denote a counter that is incremented by one every time an ACK triggered by a packet whose sequence number is larger than $W[rseqn]$ reaches the sender, without indication on the correct reception of the $rseqn^{th}$ data segment. This happens when the ACK or packet corresponding to the last retransmission of the $rseqn^{th}$ data segment has either been delayed or lost. When $dupcount$ reaches $dupackthreshold$, the $rseqn^{th}$ data segment is finally retransmitted. After a retransmission, or after the sender received an ACK indicating the correct reception of the $rseqn^{th}$ data segment, $dupcount$ is reset to zero and $rseqn$ is updated based on Equation (1). To avoid multiple retransmissions of the same data in a single RTT, $dupcount$ is only reset to zero once the packet sequence number gets larger than $W[rseqn] + cwnd$.

It is worth mentioning here that the send-out window is respectively inflated or partially deflated in response to a duplicate ACK or to a new ACK, as explained in Section III-A. The inflation/deflation process is important to keep the connection active while lost segments are retransmitted. Moreover, both the retransmission timer and the aggressive recovery timer defined in Section III-C and III-B are also used in conjunction to the loss monitoring window.

V. Explicit window-based congestion control algorithms

This section presents the two specific window-based congestion control algorithms that are considered in this work to evaluate our proposed loss-resilient mechanisms. Both protocols rely on the explicit transmission of information about the state of congestion at the routers to compute the $cwnd$ value. However, the information provided about congestion is different in both protocols. The first protocol, introduced in [1], exploits an accurate feedback from the routers. In the second protocol, we propose to rely on a coarse binary feedback to notify of congestion. In Sections VII and VIII, simulations allow to discuss the impact of the congestion feedback granularity on the robustness of the window-based transport protocols in lossy connections.

A. The eXplicit Control Protocol (XCP)

The first protocol that we consider here is the eXplicit Control Protocol (XCP), proposed in [1]. XCP is window-based, and controls the size of the congestion window based on explicit and accurate feedback from routers. In short, XCP is based on a few bytes of control information conveyed in the packet headers. To control the link utilization, routers inform the senders about the degree of congestion in bottleneck links. In a router, the congestion information is computed based on the mismatch between the aggregate traffic rate and the link capacity, and is adjusted according to the delay expected for the feedback packet. Fairness is achieved by reallocation of bandwidth between individual flows. Extensive simulations demonstrate that XCP maintains good utilization and fairness among lossless connections, while maintaining small standing queue sizes [1]. In particular, it has been shown that XCP outperforms TCP when the per-flow delay-bandwidth product becomes large. In Section VII, we consider the combination of XCP with the proposed loss recovery mechanisms.

B. An eXplicit TCP (XTCP)

As an alternative to XCP, we propose a new explicit congestion control protocol named eXplicit TCP (XTCP). Overall, the only difference between the proposed eXplicit TCP and conventional TCP is the way they infer congestion from the network feedback. The TCP sender implicitly infers congestion from a lost data [17]. On the contrary, XTCP requires an explicit feedback from the routers to infer that congestion occurs and decrease its congestion window. In order to decouple the gain provided by an explicit framework, from the congestion control mechanism itself, XTCP mimics the TCP behavior. It relies on a minimalist binary feedback from the routers (just as TCP relies on the binary congestion signal inferred from losses), and adopts the exact same additive increase - multiplicative decrease behavior as TCP. Hence, the XTCP sender just probes the network to the point of congestion before backing off, just as TCP would proceed. Beyond the interest it provides regarding the comparison between explicit and implicit congestion controls, a TCP-like explicit protocol obviously presents also advantages in terms of deployment issues (see Section VIII-A).

In more details, the binary feedback is provided by a *congestion flag* contained in the XTCP packet header. The flag is initialized to zero by the sender, and is set to one when the packet encounters a congested router. When the packet reaches the receiver, the flag is copied in the ACK header, and returned to the sender. Upon ACK reception, the sender decides whether the congestion window should be decreased or increased based on the congestion flag. Similarly to TCP, the congestion window is incremented each time an ACK with a null congestion flag is received, and divided by two when the sender infers a congestion event based on the returned congestion flags. By definition, a congestion event occurs when an ACK with a congestion flag set to one is received, and when the latest congestion event is older than one RTT. This is to avoid multiple backoffs during one RTT. In practice, an exponential weighted average of the RTT samples is used to estimate RTT.

We now explain how routers define the congestion flag. Formally, a *congestion counter* is associated with every queue in the network. Each time a queue drops a packet due to congestion, the congestion counter associated to the queue is incremented by one. When an XTCP packet leaves the queue to be sent out to the output link, if the counter is positive, the congestion flag of the XTCP packet is set to one, and the counter is decremented by one[3]. The packet whose congestion flag is set to one does not necessarily belong to the same flow as the packet whose drop is responsible for incrementing the congestion counter. There is no need to maintain per flow congestion states in the router. Note finally that XTCP does not make any assumption about the queue management strategy used in routers. In our simulations, XTCP has been tested both with Droptail or RED policies [18].

VI. Simulation setup and performance evaluation

Sections VII and VIII analyze through simulation the advantages and limitations of the retransmission mechanisms presented in Sections III and IV, in the context of explicit congestion control. We present results based on NS simulations. For simplicity, a single topology is considered throughout the simulations. This topology is represented in Fig. 1, where n sources share a bottleneck link. Half of the flows ends up in node N2, through a loss-free link, while the other half ends up in node N3, across the lossy link. We refer to the flows that go through the lossy (loss free) link as lossy (loss free) flows. All links have a bandwidth equal to 5 Mbps, and are characterized by the same delay of 20ms (except when explicitly mentioned). In the rest of the paper, the number of sources n is equal to 10, and losses generated on the lossy link are either randomly distributed (default case), or follow a bursty process.

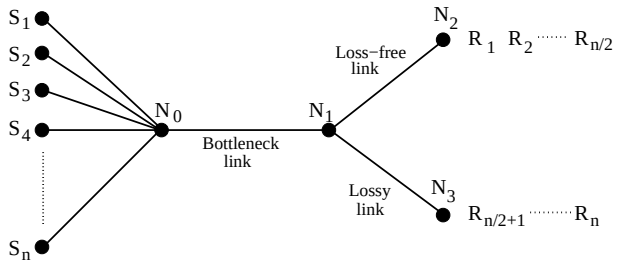
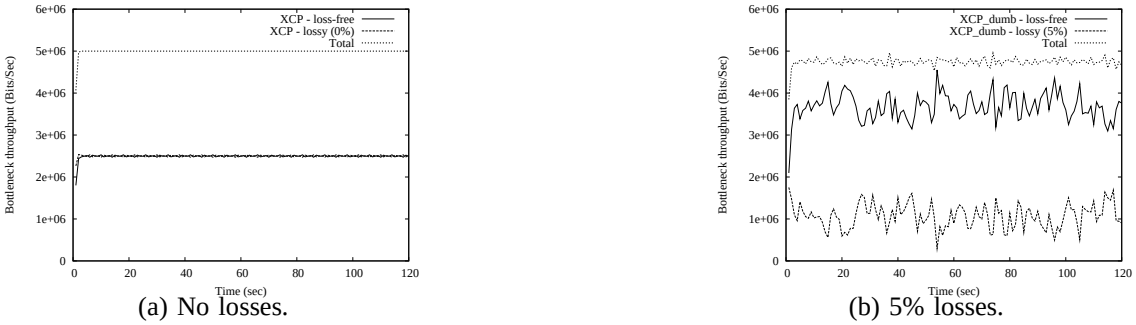


Fig. 1. Network topology reflecting different users accessing a bottleneck through links with different loss characteristics, e.g. wired and wireless.

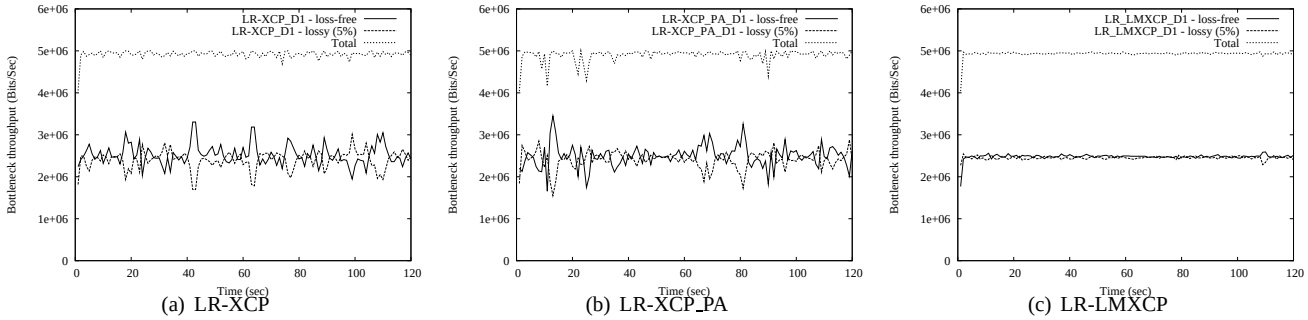
In order to evaluate the performance of the proposed loss resilience mechanisms, we measure the bottleneck link utilization and analyze how the bottleneck bandwidth is partitioned between loss-free and lossy links. A desirable solution ensures full-link utilization and a fair partition of the bottleneck between the connections, independently of whether they are affected by losses or not. Hence, for the above topology, the performance of loss-resilient protocols is estimated by comparing the sum of the bottleneck throughputs measured for lossy and loss-free flows. An equal usage of the bottleneck by lossy and loss-free flows reflects good performance of the loss-resilience mechanisms.

[3]To make sure that we do not run in a situation where the counter is positive, and the queue is empty, we only increment the counter if its current value is smaller than the number of packets present in the queue.



(a) No losses. (b) 5% losses.

Fig. 2. Sums of throughputs measured respectively for loss-free and lossy flows. Throughputs are measured on the bottleneck link as a function of time.



(a) LR-XCP (b) LR-XCP_PA (c) LR-LMXCP

Fig. 3. Sums of throughputs measured respectively for loss-free and lossy flows. Sums of throughputs are plotted as a function of time. In all graphs, the $dupackthreshold$ parameter is set to 1 (as indicated by D1).

VII. Loss-resilient XCP validation

We now introduce the notation terminology used to denote the transport protocols obtained when combining XCP [1] with several loss retransmission mechanisms. In Table I, XCP_dumb refers to the implementation proposed in [1]. It simply relies on the mechanisms implemented by TCP Reno to recover from losses, without exploiting the advantages provided by the explicit congestion framework. In contrast, XCP does not halve $cwnd$ upon reception of a duplicate ACK, and only partially deflates $dupwnd$ upon new ACK reception. We then use the prefix LR to denote the implementations of retransmission timer and aggressive reset of the recovery timer (see Section III-C and II-B.c). The suffix PA then indicates that partial acknowledgments are used to trigger fast retransmissions (see Section II-B.b). Eventually, LMXCP assumes that the packet sequence number is conveyed by the packet header to allow for accurate loss monitoring, as described in Section IV-B.

Acronym	Definition
XCP_dumb	eXplicit Control Protocol with retransmission and recovery mechanism implemented as in TCP Reno, similar to [1]
XCP	eXplicit Control Protocol with partial deflation of $dupwnd$ and preservation of $cwnd$ upon loss observation (see Section III-A)
XCP_PA	XCP + retransmissions based on partial ACKs (see Section II-B.b)
LR-XCP	XCP + retransmission timer (see Section III-C)
LR-XCP_PA	XCP_PA + retransmission timer
LR-LMXCP	XCP + retransmissions based on accurate loss-monitoring (see Section IV)

TABLE I

Acronyms for loss resilient XCP protocols.

A. Performance of loss resilient mechanisms

As explained above, the performance of the proposed loss recovery mechanisms can be evaluated in terms of the fairness between the loss free and lossy connections sharing a common bottleneck link.

We thus first analyze the throughputs offered to lossy, and lossless flows in the network topology presented in Fig. 1. On the one hand, Fig. 2 illustrates the problem encountered by the reference implementation of XCP [1] in lossy environments. We observe that the presence of losses causes starvation of the lossy flows. This is because, by default, the XCP loss recovery mechanisms are the ones optimized for TCP, and do not exploit the explicit framework specificities. On the other hand, we show in Fig. 3 and 4, that the presence of loss-resilient mechanisms dedicated to the explicit control framework allows for a better distribution of resources between heterogeneous connections. In particular, we observe in Fig. 3 that all loss-resilient XCP protocols significantly improve the fairness in comparison with Fig. 2(b). We also observe that the accurate monitoring of losses performed by LR-LMXCP mitigates the throughput fluctuations. Fig. 4 compares the sum of bottleneck throughputs corresponding to all lossy flows with the one corresponding to all lossless flows, as a function of the loss rate. It shows that, among the protocols defined in Table I, the fairest pair of troughputs is given by the loss resilient XCP with accurate loss monitoring, as expected. The worst case in terms of fair bottleneck bandwidth allocation is given by the XCP_dumb algorithm.

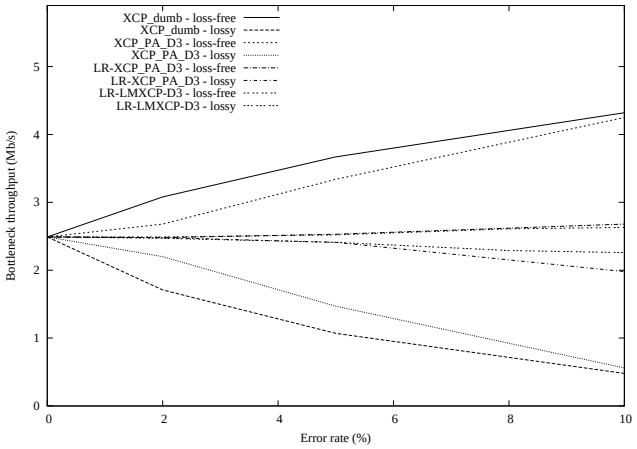


Fig. 4. Sums of average throughputs computed respectively for loss-free and lossy flows. Throughputs are averaged over a 120s period. Sums of average throughputs are then plotted as a function of the rate of losses generated on the lossy link. $dupackthreshold = 3$

We now analyze in more details the performances of the proposed algorithms. Fig. 5 presents the fairness ratio measured between lossy and loss-free flows as a function of the loss rate, for two values of the $dupackthreshold$ parameter. The fairness ratio between lossy and loss-free flows is defined as the ratio between the sums of throughputs measured respectively for all lossy and loss-free flows on the bottleneck link. We observe on both figures that LR-XCP_PA performs better than XCP_PA. **We conclude that the presence of a retransmission mechanism based on a timer brings a significant benefit.** Fig. 5 also quantifies the benefit obtained when partial acknowledgments (PA) are used in addition to duplicate ACKs to trigger retransmissions. By comparing (LR-)XCP with (LR-)XCP_PA, we observe that partial acknowledgments mainly help at high loss rates, i.e., when more than one packet is likely to be lost in a single RTT. We also note that, in the absence of retransmission timer (i.e., for XCP and XCP_PA), and when $dupackthreshold = 3$, partial ACKs do not help. In that case, the sender has little chance to enter the fast recovery mode and reset the $recover$ parameter (see Section II-B.b).

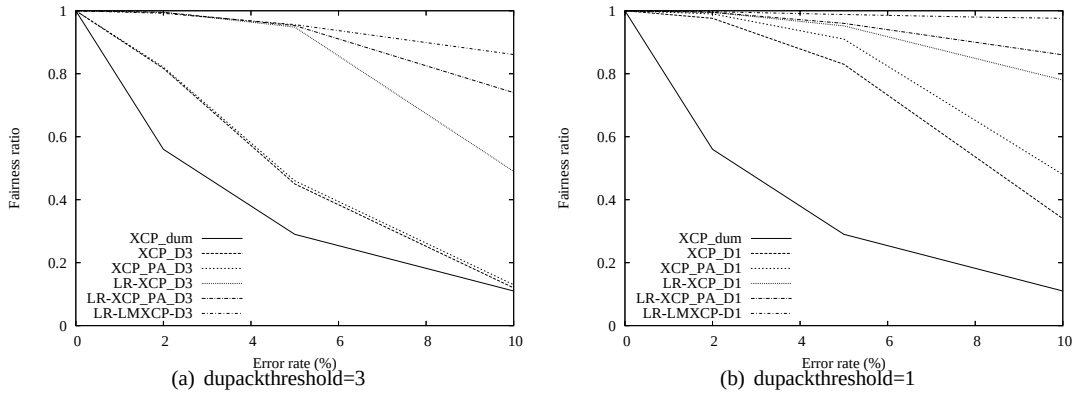
In addition, Fig. 4 and 5 confirm that a precise monitoring of losses, such as performed by LR-LMXCP, is beneficial. Comparing graphs (a) and (b) in Fig. 5 shows that this is exacerbated when the $dupackthreshold$ parameter is small. In that case, retransmissions are rapidly triggered by LR-LMXTCP, and losses are rapidly recovered. If needed, several different data segments might be retransmitted in a single round trip time. On the contrary, even with a small $dupacktheshold$, LR-XTCP can only consider the retransmission of the $(lack+1)^{th}$ segment, and is therefore limited to a maximum of one retransmission per-round trip time.

Fig. 6 further analyses the impact of the $dupackthreshold$ parameter. It shows that increasing the $dupackthreshold$ from one to three only reduces the performance of loss retransmission mechanisms for high loss rates. Hence, for reasonable loss rates, a value of three is recommended, as it ensures some independency between retransmissions and potential packet reordering.

B. Bottleneck link utilization

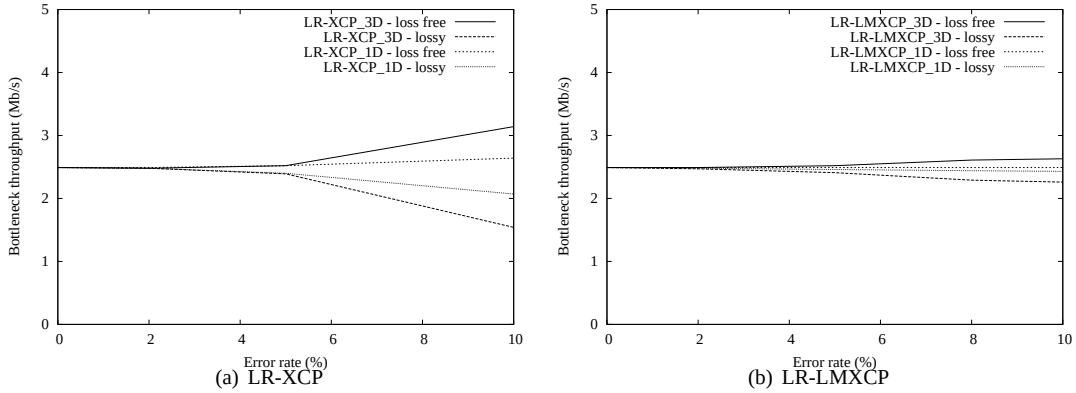
In Fig. 2 and 3, we observe that the total throughput traces do not saturate at 5 Mbits/sec. The (loss-resilient) XCP protocols therefore fail to achieve full link utilization. This is confirmed by a detailed analysis of Fig. 4. When the loss rate increases, the sum of the loss-free and lossy throughputs corresponding to a given protocol becomes smaller than 5 Mbits/sec. Hence, the total bottleneck link utilization decreases as the loss rate increases. We explain this link utilization deficiency by the small queue sizes maintained by XCP routers [1], which makes them unable to absorb rate fluctuations that can be due to a recovery phase caused by packet losses.

In order to improve the link utilization, we propose to maintain non-zero persistent queues in XCP routers. Therefore, we have modified equation (1) in [1] so that the efficiency controller targets both maximal link utilization and a non-zero persistent



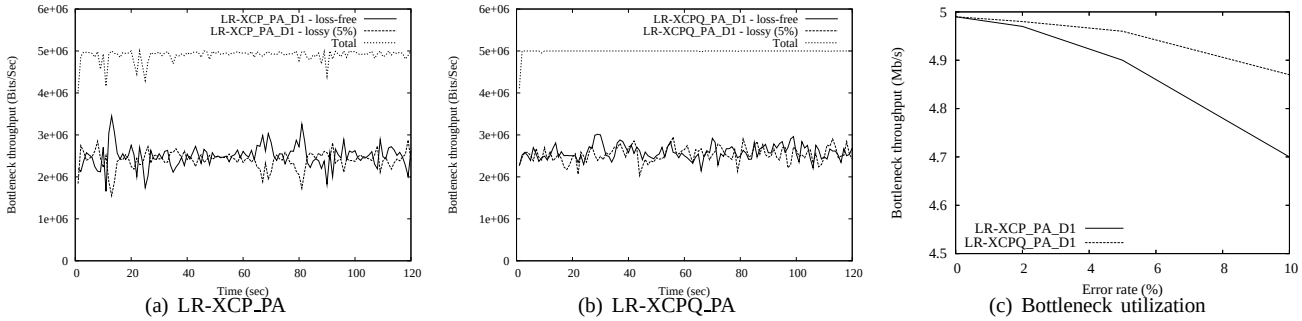
(a) dupackthreshold=3 (b) dupackthreshold=1

Fig. 5. Fairness ratio measured between lossy and loss-free flows as a function of the loss rate, and $dupackthreshold$ parameter.



(a) LR-XCP (b) LR-LMXCP

Fig. 6. Impact of the $dupackthreshold$ parameter on the fairness achieved between lossy and loss-free flows. Traces are defined as in Fig. 4.



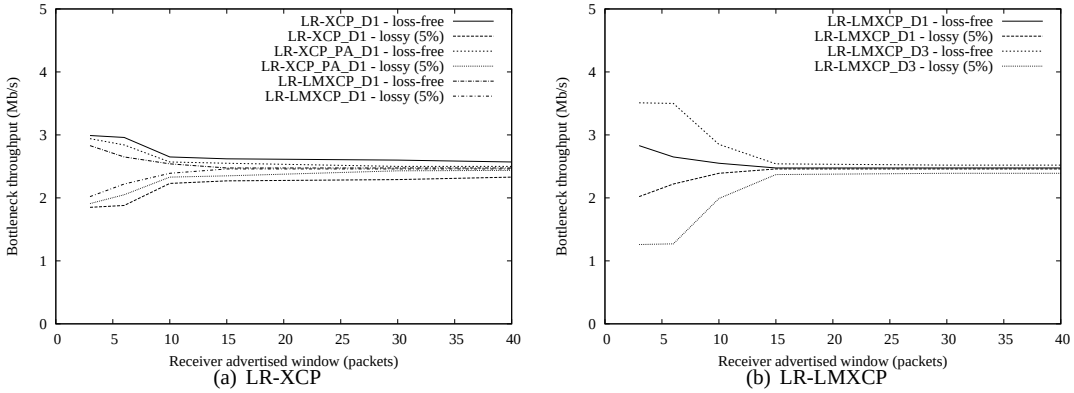
(a) LR-XCP_PA (b) LR-XCPQ_PA (c) Bottleneck utilization

Fig. 7. Comparison between an XCP router that minimizes the persistent queue, and an XCP router that targets a persistent queue of 10 packets (LR-XCPQ_PA). (a) and (b) compare for both routers the sums of throughputs measured respectively for loss-free and lossy flows. Losses are randomly generated with a 5% rate. Sums of throughputs are plotted as a function of time. In all graphs, the $dupackthreshold$ parameter is set to 1 (as indicated by D1). (c) plots the bottleneck link utilization as a function of the loss rate for both systems.

queue. Specifically, Q is replaced by $(\overline{Q}-\gamma)$ in equation (1) of [1], where γ denotes the size in packets of the targeted persistent queue. $\overline{Q}$ is defined based on the Q samples as follows. In [1], a Q sample corresponds to the minimum queue seen by an arriving packet during the last propagation delay. This definition results in large fluctuations of Q along the time. To derive a stable signal from Q, we define $\overline{Q}$ as the exponential weighted average of the Q samples, i.e., each time a new Q sample is generated, $\overline{Q}$ is set to $\beta\ \overline{Q} + (1-\beta)\ Q$. In our simulation, β has been set to 0.9, while the persistent queue γ has been set to 10 packets. The thresholds defining the RED queue policy [18] have been increased to take the persistent queue into account. Fig. 7 presents the results obtained with and without persistent queues in XCP routers for the LR-XCP_PA protocol. We observe that the presence of persistent queues in routers preserves the bottleneck link utilization. We conclude that, even when accurate explicit feedback about congestion is available, it is relevant to maintain persistent queues in routers. They can absorb the unpredictable throughput fluctuations resulting from packet losses, which may cause the expiration of the recovery timer, for example.

C. Receiver buffers

Fig. 8 illustrates the impact of the constraint imposed on the sender by the receiver advertised window, denoted $rwnd$. This window reflects the receiver buffer capacity, and corresponds to the largest number of out-of-sequence packets that can be buffered at the receiver [17]. It constrains the send-out window of the sender and limits the number of packets the sender can send in advance, while waiting for the recovery of a lost packet. Fig. 8 shows that the performance of the loss-resilience mechanisms only significantly degrades when the constraint on the send-out window becomes of the same order of magnitude as the congestion window. This observation is important because it demonstrates that efficient loss resilient mechanisms do not require large buffering capabilities from the end-hosts.



(a) LR-XCP (b) LR-LMXCP

Fig. 8. Impact on fairness of the constraint imposed on the send-out window by the receiver advertised window. $dupackthreshold$ is set to one, and loss rate is equal to 5%.

D. Bursty losses

Finally, we discuss the effects of bursty loss processes on the congestion control performance. Fig. 9 analyzes how the correlation of losses affects the retransmission mechanisms. In these simulations, each loss event on the lossy link causes the loss of X consecutive packets, with X ranging from 1 to 4. As expected, we observe that LR-XCP mechanisms are more sensitive to bursts of losses than LR-LMXCP. This is because LR-XCP mechanisms retransmit at most one packet per round trip time, and are thus less efficient than LR-LMXCP when multiple losses occur in the same window of data. Interestingly, we finally observe that the retransmission mechanism based on partial ACK is already beneficial at low loss rates when losses are bursty.

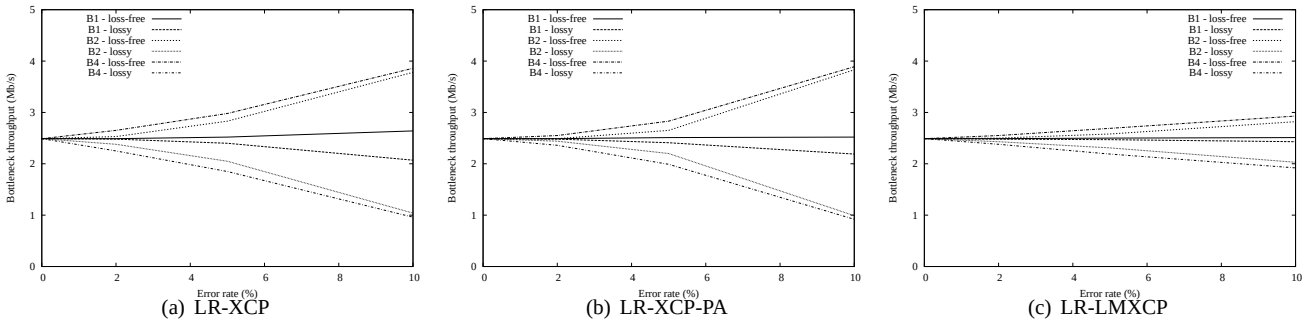
VIII. Loss-resilient XTCP analysis

In this section, we explore the behavior of the proposed eXplicit TCP (XTCP) in presence of losses, and consider its gradual deployment. We use the simulation setup described in Section VI, and we analyze in details the benefits of each loss resilient mechanisms. Table II presents the acronyms that are used to denote the combination of XTCP with different loss-resilience mechanisms. The last acronym, namely LR-LMXTCP_TCPfriend, refers to a TCP-friendly version of LR-LMXTCP, which is defined in the next section.

Acronym	Definition
LR-XTCP	eXplicit TCP with retransmission timer and recovery mechanisms adapted to the explicit congestion control framework (see Section III-D)
LR-XTCP_PA	LR-XTCP + retransmissions based on partial ACKs (see Section II-B.b)
LR-LMXTCP	eXplicit TCP + retransmissions based on accurate loss-monitoring (see Section IV)
LR-LMXTCP -TCPfriend	LR-LMXTCP + mechanism to ensure TCP friendliness (see Section VIII-A)

TABLE II

Acronyms for loss resilient XTCP protocols.

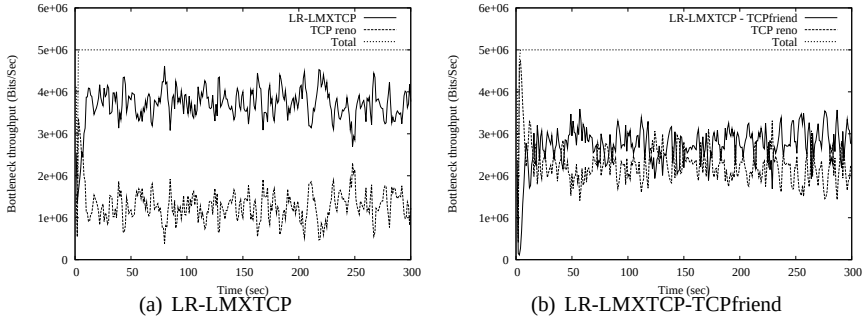


(a) LR-XCP (b) LR-XCP-PA (c) LR-LMXCP

Fig. 9. Impact on fairness of the bursty nature of losses appearing on the lossy link. The acronym BX, with X = 1, 2 or 4, means that each time a loss event happens, X consecutive packets are dropped on the link. The loss rate refers to the product of loss event with the X parameter.

A. Gradual deployment: joint TCP and XTCP queuing

We consider the coexistence of TCP and XTCP traffic, and we describe a mechanism that allows end-to-end loss resilient XTCP flows to compete fairly with TCP flows. This mechanism provides a possible path for incremental XTCP deployment. An XTCP-enabled router queues both TCP and XTCP traffics together in a single buffer, but only increments or decrements the XTCP congestion counter when it deals with XTCP packets. TCP and XTCP routers are therefore very similar, and only minor changes are necessary to enable XTCP in a router, which surely facilitates the deployment of XTCP. To start a loss-resilient XTCP connection, the sender then has to check whether the receiver and the routers along the path are XTCP enabled. As mentioned in [1], this kind of verification can be done using TCP and IP options. If routers are not compliant with XTCP, the sender reverts to TCP.



(a) LR-LMXTCP (b) LR-LMXTCP-TCPfriend

Fig. 10. Throughput traces showing how loss resilient XTCP competes with TCP. (a) LR-LMXTCP is unfair to TCP, (b) LR-LMXTCP with artificial backoff simulations achieves improved fairness.

In order to allow for such a gradual deployment of XTCP, we now extend the design of loss-resilient XTCP into a TCP-friendly version, which leads to a fair allocation of resources between TCP and XTCP flows. In order to illustrate the unfairness between TCP and XTCP, Fig. 10 first shows the sum of throughput measured on the bottleneck link, for TCP and LR-LMXTCP flows respectively. The topology considered for this simulation is the one described in Fig. 1. Routers obey a drop tail policy, and are XTCP-enabled. All links are loss-free. Half of the flows are TCP flows. The others are LR_LMXTCP. In Fig. 10(a), we observe that LR-LMXTCP flows achieve higher throughput than TCP flows. This unfairness is mainly due to the different behavior of LR-LMXTCP and TCP when they face (congestion) losses. LR-LMXTCP handles losses in an efficient way, while TCP generally resorts to a recovery phase when more than one loss occur in a single flight of packets [3].

To increase fairness between XTCP and TCP, we propose a simple change to the design of the loss-resilient XTCP sender, so that it triggers an artificial connection backoff when it detects conditions for which TCP is expected to experience a timeout. We use the LR-LMXTCP_TCPfriend acronym to refer to this version of LR-LMXTCP. The artificial backoff emulates the TCP recovery process described in Section II-B.c. It consists in resetting the congestion window to one, but without resetting the $nextseq$ state variable to $lack + 1$. To mimic the TCP recovery phase, a state variable, denoted $recphase$, is updated to the largest data sequence number ever sent out by the sender, and the congestion window is not incremented based on received acknowledgments as long as the $lack$ state variable remains smaller than $recphase$. The artificial backoff is triggered when (i) a congestion flag is received and (ii) either the congestion window is smaller than $dupackthreshold$, or two congestion flags are received in less than one RTT. These conditions reflect the fact that a loss ends up in a recovery phase for TCP either when the connection can not enter a fast recovery phase, or when two packets are lost in a single RTT. We observe in Fig. 10(b) that this small modification is efficient, since LR-LMXTCP_TCPfriend now competes rather fairly with TCP.

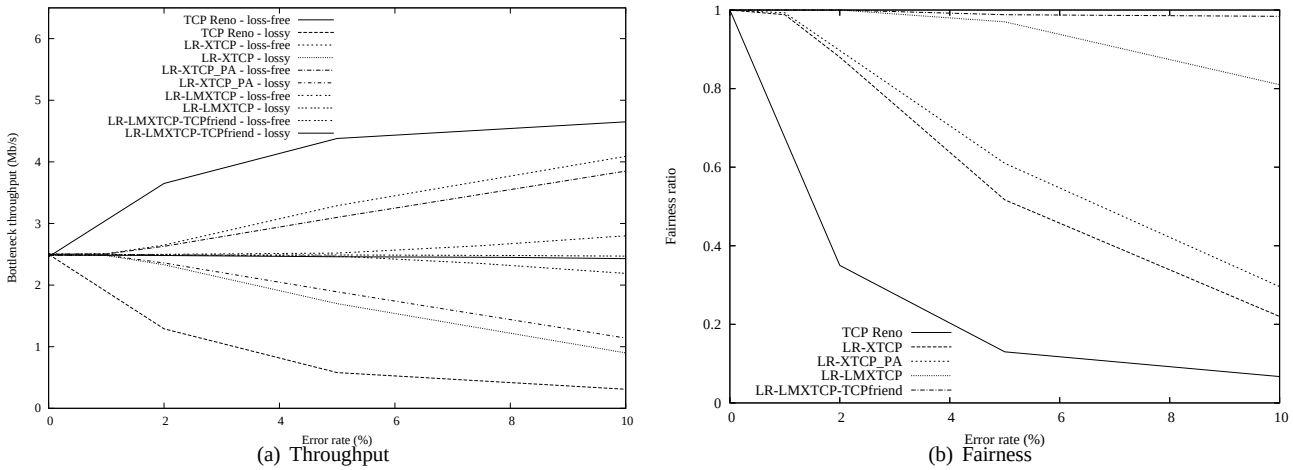


Fig. 11. (a) Sum of throughputs measured for lossy and loss-free flows over the bottleneck as a function of the loss rate. Losses are randomly distributed. The $dupackthreshold$ parameter is set to 3. (b) Fairness ratio measured between lossy and loss-free flows.

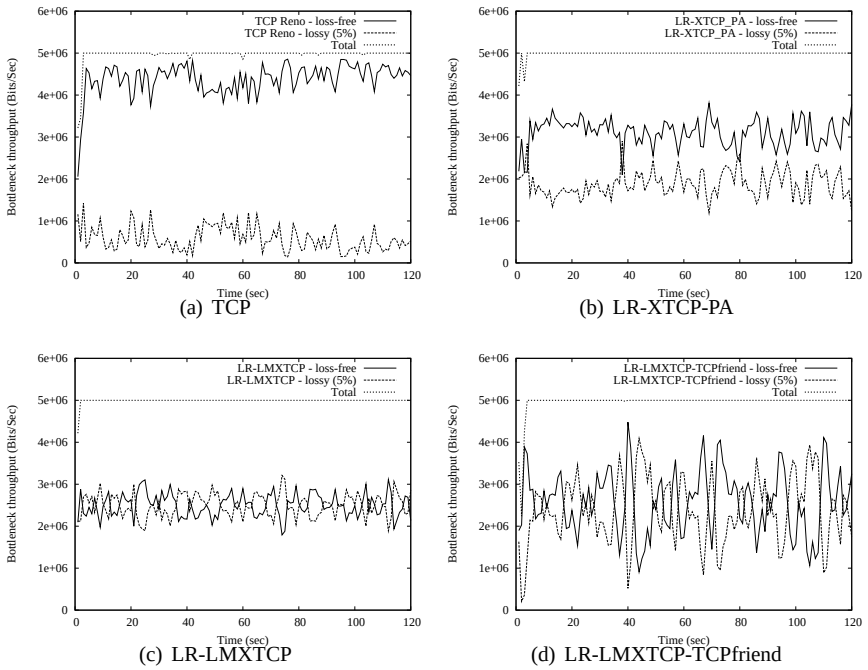
B. Loss-resilient XTCP performances

We now use simulations to analyze the performance of loss resilient XTCP protocols. Fig. 11(a) first plots the throughputs measured over the bottleneck link, respectively for lossless and lossy flows, as a function of the loss rate experienced on the lossy link. Fig. 11(b) presents the same results in terms of fairness ratio between lossy and lossless flows. We observe that TCP rapidly starves the lossy flows. We also see that the retransmission based on partial ACKs only improves LR-XTCP beyond a sufficient loss rate. Moreover, we note that an accurate feedback, as explored by LR-LMXTCP, brings a significant improvement over approaches that are based on cumulative ACKs. Because LR-LMXTCP preserves high efficiency at high loss rates, we conclude that window-based congestion control protocols, when coupled with a precise feedback from the receiver, are able to support lossy environments. In addition to these general conclusions, we also observe that LR-LMXTCP-TCPfriend performs even better than LR-LMXTCP, since the artificial backoff introduced in this TCP friendly version improves the fairness between lossless and lossy flows.

Interestingly, in-depth comparisons between Fig. 11(b) and Fig. 5(a) reveal that LR-XTCP performs worse than LR-XCP for loss rates larger than 1%. We explain that observation by the fact that a connection controlled based on a binary congestion feedback is severely impaired when simultaneously affected by losses and congestion notifications. Indeed, upon congestion notification, $cwnd$ is halved by two. In presence of losses, the head of the send-out-window stays blocked at the last acknowledged data segment, and the amount of transmitted data per RTT gets directly penalized by the sharp reduction of $cwnd$. We conclude that the smooth regulation of $cwnd$ supported by XCP is helpful in presence of heavy loss rates, at least when the sender infers losses based on cumulative acknowledgments. In contrast, when the sender receives accurate information about received data segments from the receiver, we observe that similar loss resilience is achieved whatever the smoothness of the congestion control mechanism, and LR-LMXCP and LR-LMXTCP perform equally well. In presence of accurate feedback about packet arrival, each connection indeed triggers the retransmissions faster. **We conclude that either a smoothed and finely-tuned congestion control or an accurate loss monitoring is required to face large error rates.** In other words, the combination of a coarse $cwnd$ adjustment mechanism, with cumulative acknowledgments, results in a lack of aggressiveness of lossy connections compared to the connections that are not subject to losses.

We now observe in more details the temporal behavior of the congestion control algorithms. Fig. 12 traces the sum of lossy and loss-free throughputs as a function of time. We make two observations. First, from the total aggregate throughput value, we observe that all loss-resilient XTCP protocols achieve full utilization of the bottleneck link. This is confirmed by Fig. 11(a), where the sums of corresponding loss-free and lossy throughputs equal the bottleneck link bandwidth, i.e. 5 Mb/s. Second, based on graphs (c) and (d), we note that the TCP friendly throughput fluctuates more than the non friendly one. This is in accordance with what we expect from TCP-like connection backoffs.

Finally, we discuss TCP friendliness of the proposed scheme, with various connection parameters. Fig. 13 shows the bottleneck link utilization as a function of the link delay parameter, when the droptail router queue size is fixed. As expected, the link utilization degrades when the bandwidth-delay product of the connection increases in comparison with the queue size. The queue becomes too small to absorb the reduction of rate due to connection backoffs. We observe in Fig. 13 that LR-LMXTCP preserves higher utilization than TCP or LR-LMXTCP_TCPfriend. This was foreseeable as connection backoffs are more frequent with TCP than with LR-LMXTCP, which is able to face losses without resetting the connection.



(a) TCP (b) LR-XTCP-PA

(c) LR-LMXTCP (d) LR-LMXTCP-TCPfriend

Fig. 12. Sum of loss-free and lossy flow throughputs as a function of time. Losses are randomly distributed with a loss rate of 5%.

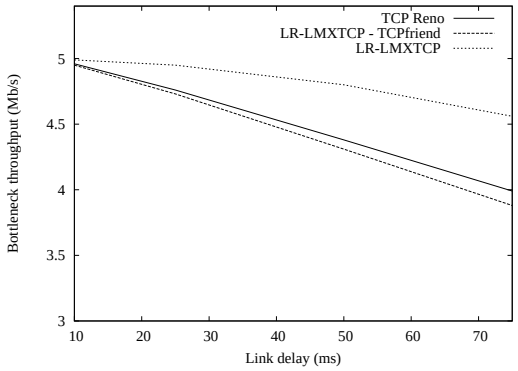


Fig. 13. Bottleneck link utilization as a function of the link delay parameter. Queue size is fixed to 50 packets.

IX. Related works

The first paragraph of this section surveys the numerous research efforts that have been made in order to improve TCP performance over lossy links. Because TCP interprets any kind of losses as a congestion notification, without any precaution, TCP results in flow starvation over lossy links. The problem is well-known, especially in wireless environments. In the past decade, three main approaches have been considered to circumvent that problem [4], [11], [12], [13]. The first one consists in increasing link-layer reliability to hide link-related losses from TCP sender [19], [20], [21]. The second one splits the end-to-end connection, and terminate the TCP connection at the base station to hide the lossy link from the sender [22], [23]. The third approach works end-to-end, and attempts to give the TCP sender the capability to handle appropriately losses that are not related to congestion. Some of those schemes builds on refined TCP acknowledgments to allow the TCP sender to recover from multiple losses, without resorting to a coarse timeout [14], [24], [25]. Others methods distinguish between congestion-related losses and other forms of losses, either based on explicit loss notification [4], or on end-to-end bandwidth estimation [26]. A last strategy proposed to handle non-congestion related losses consists in delaying or in freezing the congestion response algorithms to allow the recovery of the losses caused by channel errors [13] or hand-offs [12], [27]. In a sense, our work is related to the third approach, since we do not attempt to hide losses to the sender, but we rather give the sender the capability to handle them. However, our work is quite different from the approaches described in the above references, essentially because our goal is not to improve TCP performance, but rather to explore the limitations of the window-based congestion control paradigm in lossy environments.

In parallel to solutions for circumventing the fragility of TCP to non-congestion related losses, a number of works have proposed to decouple congestion control from the observation or inference of losses, similarly to our work. In [10], the size

of the congestion window is adapted based on the explicit rate feedback provided by the bottleneck. The authors analyze the performance improvement resulting from the explicit and fine-granular participation of the network in the congestion control. They do not specifically address the loss resilience problem, and in consequence, they do not encourage the use of aggressive retransmission mechanisms. In [11], the authors exploit the fact that the packet error-rate on wireless links is proportional to the packet size, and propose to control congestion based on the loss patterns observed for tiny TCP/IP header packets. Doing so, they decouple the congestion control from the wireless losses affecting the large data packets. While discussing the performance of their proposed decoupling strategy, the authors in [11] mention that it is important to be aggressive in retransmitting lost data, as long as their transmission is allowed by the congestion window defined based on the small control packets. In that, they stick to one of the main conclusions of our investigations. We go beyond by understanding and alleviating the limits of conventional loss recovery mechanisms when used in conjunction with explicit congestion control mechanisms. Another work of interest is described in [28], where the authors demonstrate that explicit congestion notification (ECN) is unable to distinguish between congestion and wireless losses. They propose to control the flow only based on ECN bits, i.e., without adjusting the congestion window in response to loss packets. In that, the approach is similar to our proposed XTCP. However, the authors in [28] do not discuss the need and relevance for aggressive loss retransmission mechanisms, which is the central component of our contribution.

Hence, to the best of our knowledge, none of the earlier works has provided a detailed investigation and a precise description of retransmission mechanisms dedicated to an explicit window-based congestion control framework.

X. Conclusions

We have studied packet retransmission mechanisms for explicit window-based congestion control protocols, where retransmissions can be decoupled from the congestion control process. This interestingly offers increased flexibility to implement retransmission mechanisms. We have considered the design of retransmission strategies dedicated to an explicit congestion control framework in the context of two different receiver feedback mechanisms. With conventional cumulative acknowledgment mechanisms, preservation of the $cwnd$ upon reception of a duplicate ACK, partial deflation of the send-out window upon reception of a new ACK, and reset of the recovery timer upon reception of both new and duplicate ACKs appeared to significantly contribute to maintain the connection efficiency in presence of losses. An original retransmission timer has then been proposed as a complement to the recovery timer to handle multiple losses of the same data segment. In the presence of richer acknowledgments, we have designed loss monitoring and recovery mechanisms capable to exploit precise knowledge about packet reception, so as to increase retransmission aggressiveness.

The proposed approaches have been validated through simulations, both for the XCP protocol introduced in [1], and for a new eXplicit TCP (XTCP) protocol. XCP is characterized by a smooth and fine adjustment of the congestion window in response to accurate feedback computed by XCP routers, while XTCP relies on binary notification about congestion to coarsely adjust the congestion window. Our simulation results have shown that, when the sender is directly notified about packet arrival at the receiver, both XCP and XTCP achieve full utilization and fair partition of the bottleneck resources between lossy and lossless connections. Whilst being always advantageous, a precise notification of received packets to the sender however becomes almost mandatory when the congestion window is controlled based on coarse binary feedback. Eventually, we conclude that accurate feedback is required either from the router (to support a finely tuned congestion control) or from the receivers (to monitor losses accurately) in order to preserve the connection efficiency in presence of high loss rates.

The loss-resilience mechanisms proposed in this paper have been shown to maintain close to optimal link utilization, and fair allocation of bottleneck resources among lossy and lossless connections. The combination of explicit control with dedicated retransmission mechanisms provides thus an interesting solution to establish reliable and controlled window-based connections in a lossy network.

References

[1] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for high bandwidth-delay product environments," *ACM SIGCOMM*, pp. 89–102, Pittsburgh, August 2002.

[2] J. V., "Congestion avoidance and control," in *ACM SIGCOMM*, Stanford, CA, September 1988, pp. 314–329.

[3] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," in *RFC 2581, http://www.rfc-editor.org/rfc/rfc2581.txt*, April 1999.

[4] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, December 1997.

[5] C. Partridge and T. Shepard, "Tcp performance over satellite links," *IEEE Network*, vol. 11, no. 5, September/October 1997.

[6] S. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of tcp/red and a scalable control," in *IEEE INFOCOM*, New-York, USA, June 2002.

[7] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanisms," in *RFC 2488, http://www.rfc-editor.org/rfc/rfc2488.txt*, January 1999.

[8] M. G., S. Dawkins, M. Kojo, V. Magret, and N. Vaidya, "Long thin networks," in *RFC 2757, http://www.rfc-editor.org/rfc/rfc2757.txt*, January 2000.

[9] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification to IP," in *RFC 2481*, January 1999.

[10] A. Karnik and A. Kumar, "Performance of TCP congestion control with explicit rate feedback," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, pp. 108–120, February 2005.

[11] S. Wang and H. Kung, "Use of TCP decoupling in improving TCP performance over wireless networks," *ACM Wireless Networks*, vol. 7, no. 3, pp. 221–236, 2001.

[12] T. Goff, J. Moronski, D. Phatak, and V. Gupta, “Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments,” in *IEEE INFOCOM*, Tel-Aviv, Israel, March 2000, pp. 1537–1545.
[13] S. Bhandarkar, N. Sadry, A. Narasimha Reddy, and N. Vaidya, “TCP-DCR: a novel protocol for tolerating wireless channel errors,” *IEEE Transactions on Mobile Computing*, vol. 4, no. 5, pp. 517–529, September/October 2005.
[14] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” in *RFC 2018*, October 1996.
[15] S. Keshav and S. Morgan, “SMART retransmission: performance with overload and random losses,” *INFOCOM’97*, vol. 3, pp. 1131–1138.
[16] S. Floyd and T. Henderson, “The Newreno modification to TCP’s fast recovery algorithm,” in *RFC 2582*, April 1999.
[17] J. F. Kurose and K. W. Ross, “Computer Networking: a top-down approach featuring the Internet,” Addison Wesley, 2001.
[18] S. Floyd and V. Jacobson, “Random Early Detection gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.
[19] H. Balakrishnan, S. Seshan, and R. Katz, “Improving reliable transport and handoff performance in cellular wireless networks,” *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, February 1995.
[20] E. Amir, H. Balakrishnan, S. Seshan, and R. Katz, “Efficient TCP over networks with wireless links,” in *Fifth workshop on Hot Topics in Operating Systems*, May 1995, pp. 35–40.
[21] C. Parsa and J. Garcia-Luna-Aceves, “Improving TCP performance over wireless networks at the link layer,” *Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, March 2000.
[22] A. Bakre and B. Badrinath, “I-TCP: Indirect TCP for mobile hosts,” in *Proceedings of ICDCS*, May 1995, pp. 136–143.
[23] K. Brown and S. Singh, “M-TCP: TCP for mobile cellular networks,” *IEEE/ACM SIGCOMM Computer Comm. Review*, vol. 27, no. 5, pp. 19–43, Oct. 97.
[24] K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno, and SACK TCP,” *SIGCOMM Comp. Comm. Rev.*, vol. 26, no. 3, pp. 5–21, July 96.
[25] D. Lin and H. Kung, “TCP fast recovery strategies: analysis and improvements,” in *INFOCOM*, vol. 1, March 1998, pp. 263–271.
[26] C. Casetti, M. Gerla, S. Mascolo, M. Sanadidi, and R. Wang, “TCP Westwood: end-to-end bandwidth estimation for enhanced transport over wireless links,” *Wireless Networks*, vol. 8, no. 5, pp. 467–479, September 2002.
[27] W. Lioa, C.-J. Kao, and C.-H. Chien, “Improving TCP performance in mobile networks,” *IEEE Transactions on Communications*, vol. 53, no. 4, pp. 569–571, April 2005.
[28] S. Biaz and X. Wang, “Red for improving tcp over wireless networks.” in *International Conference on Wireless Networks*, Las Vegas, Nevada, USA, June 2004, pp. 628–636.