Journal of Mobile Multimedia, Vol. 2, No.4 (2006) 327-343

PERFORMANCE EVALUATION OF REAL-TIME TRANSPORT WITH LINK-LAYER RETRANSMISSIONS IN WIRED/WIRELESS NETWORKS

PANAGIOTIS PAPADIMITRIOU AND VASSILIS TSAOUSSIDIS

Demokritos University, Xanthi, Greece
{ppapadim, vtsaousi}@ee.duth.gr

Received May 31, 2006
Revised September 20, 2006

Real-time transport over wired/wireless networks is challenging, since wireless links exhibit distinct characteristics, such as limited bandwidth and high error rates, due to fading or interference. We focus on the efficiency of mechanisms that bind operationally wired and wireless links. In this context, local error control is attractive, due to the remarkable feasibility of wireless link protocols in terms of wide range deployment. We investigate whether local retransmissions enable TCP to efficiently utilize wireless resources under the constraint of bounded end-to-end delay. Based on an analytical approach, as well as extensive simulations, we show that local recovery prevents wasteful end-to-end retransmissions and allows the transport protocol to utilize a higher fraction of the available bandwidth. However, we uncover undesirable effects of local error control which degrade the performance of real-time delivery in several occasions. Furthermore, we investigate whether local error control compares favorably with selected transport-layer mechanisms.

Key words: wireless networks, TCP, congestion control, performance evaluation, real-time applications

1 Introduction

Towards a next-generation Internet, a variety of heterogeneous wired/wireless networks gain popularity and fall under extensive research activity. Wireless links exhibit distinct characteristics, such as limited bandwidth, varying error-rates and potential handoff operations. Consequently, *Quality of Service* (*QoS*) requirements in wireless networking are stringent and complicated, taking additionally into account the influencing mobile device characteristics and limitations.

Transmission Control Protocol (*TCP*) is basically designed to provide a reliable service for wired Internet. The *Additive Increase Multiplicative Decrease* (*AIMD*) algorithm [8], incorporated in standard TCP versions, achieves stability and converges to fairness when the demand of competing flows exceeds the channel bandwidth. TCP is further enhanced with a series of mechanisms for congestion management, including *Congestion Avoidance* [14], *Slow Start*, *Fast Retransmit* and *Fast Recovery* [23]. Despite these features, TCP demonstrates inadequate performance in heterogeneous wired/wireless environments. Authors in [24] outline three major shortfalls of TCP: (i) ineffective bandwidth utilization, (ii) unnecessary congestion-oriented responses to wireless link errors (e.g. fading channels) and operations (e.g. handoffs), and (iii) wasteful window adjustments over asymmetric, low-bandwidth reverse paths. More precisely, a suitable TCP for wired/wireless networks should be able to detect the nature of the errors that result in packet loss in order to determine the appropriate error-recovery strategy. Based on such an approach, the sender would not be obliged to

reduce its transmission rate in the event of a wireless error or handoff. A next level of enhancement for TCP would enable a more sophisticated error-recovery strategy adjusted to the error characteristics of the underlying network, device constraints and performance trade-offs.

The difficulty of the task that TCP has to perform is further enhanced, when the protocol provides services for real-time applications. Such applications are comparatively intolerant to delay and variations of throughput and delay. They are also affected by reliability factors, such as packet drops due to congestion or link errors. Hence, time-sensitive applications yield satisfactory performance only under certain QoS provisions, which may vary depending on the application task and the type of media involved. TCP occasionally introduces arbitrary delays, since it enforces reliability and in-order delivery. Furthermore, the process of probing for bandwidth and reacting to observed congestion causes oscillations to the achievable transmission rate. In response to standard TCP limitations, several TCP protocol extensions [10, 4] have emerged providing more effective bandwidth utilization and sophisticated mechanisms for congestion control. *TCP-friendly* protocols, proposed in [10, 27, 28], achieve smooth window adjustments, while they manage to compete fairly with TCP flows. In order to achieve smoothness, they use gentle backward adjustments upon congestion. However, this modification has a negative impact on responsiveness [26].

User Datagram Protocol (*UDP*) has been widely used instead of TCP in real-time applications. UDP lacks all basic mechanisms for error recovery and flow/congestion control. Thus, it allows for transmission attempts at application speed. That said, UDP can not guarantee reliability, and certainly is not able to deal with network delays either. In [19] we have shown that UDP may perform worse than TCP in several occasions. Along these lines, we do not include UDP in this study.

Although numerous research proposals have emerged towards improving transport services over wireless links [2, 3], the converged domain of real-time traffic over wireless networks has not attracted the required attention from the research community so far. Several approaches operate on transport layer, most of them pronounced as enhanced TCP versions. In addition, a series of independent mechanisms have been proposed, which normally interact with TCP and provide reliable transmission over wireless links. Most of them operate on link-layer. Addressing link errors near the site of their occurrence appears intuitively attractive for several reasons. First of all, link-layer schemes are likely to respond more quickly to changes in the error environment, and generally local error control may be significantly more efficient than end-to-end error control. Furthermore, local error control commonly operates on exactly the links that require it, rendering the deployment of new and existing wireless link protocols significantly more feasible than applying novel transport-layer solutions.

However, link-layer approaches may degrade performance, especially in the presence of highly variable error rates. Local error recovery may alter the characteristics of the network affecting the functionality of higher layer protocols. For example, local retransmission could result in packet reordering or in large fluctuations of *Round Trip Time* (*RTT*), either of which could trigger TCP timeouts and retransmissions. In addition, concurrent responses from both local and end-to-end error control may result in undesirable interactions, causing inefficiencies and potentially instability. The duplicate retransmissions generated by a link-layer scheme and TCP induce excessive bandwidth consumption or even buffer overflows. Considering real-time traffic where data packets bear information with a limited useful lifetime, retransmissions are often a wasted effort. In such conditions, unfruitful retransmissions deliver delayed packets which are either discarded, or at the worst they obstruct the proper reconstruction of oncoming packets.

Our study builds on and extends the results of [20]. We provide an in-depth assessment of the performance of link- and transport-layer mechanisms that bind operationally wired and wireless links. We specifically focus on the performance of real-time delivery applying our performance metric [19] that effectively captures the joint effect of jitter and packet loss. In this context, we demonstrate the considerable degradation of TCP performance in the face of packet loss, and we investigate whether local retransmissions overcome these implications in a variety of situations. We validate both analytically and experimentally that local error control improves bandwidth utilization, while we emphasize on the impact of queuing delays across the wireless channel on real-time delivery. We also point out certain conditions that render local retransmissions more effective. In addition, we investigate whether local error control compares favorably with selected transport-layer approaches which address the fundamental QoS provisions of real-time traffic.

We organize the rest of the paper, as follows. Section 2 summarizes related work and provides an overview of research proposals towards the improvement of real-time performance with TCP. In Section 3 we evaluate analytically the effect of local error control on flow throughput and network delay. Section 4 includes our evaluation methodology, followed by Section 5 where we analyze the results of the experiments we performed. Finally, in the last section we highlight our conclusions and refer to future work.

2. An Overview of Related Work and TCP Enhancements

2.1 Improving TCP Performance over Wireless Links

We hereby summarize the most remarkable proposals which target at improving the performance of TCP over wireless links. Authors in [2] provide a comparative overview of such approaches. Furthermore, open issues of TCP in mobile environments are extensively discussed in [24]. Selected end-to-end loss differentiation algorithms are applied to *TCP-friendly Rate Control* (*TFRC*) and their efficiency is analyzed in [7]. There are several techniques operating on the link layer, which attempt to ameliorate the impact of wireless errors [2, 3]. The most remarkable implementations, which provide error-correction, are *Forward Error Correction* (*FEC*) and *Automatic Repeat Request* (*ARQ*) [9]. FEC introduces added overhead to data bits in order to cope with data corruption. Corrupted packets are directly corrected, without retransmission, which is critical for lossy links exhibiting long delays. In addition, FEC does not interfere with TCP mechanisms. However, the redundant information is not exploited in the absence of link errors resulting in a waste of bandwidth. Furthermore, FEC requires additional resources in CPU processing time, memory and power consumption.

On the other hand, ARQ mechanisms are invoked when packets containing bit errors can not be corrected. In such case, the erroneous packets are discarded and a retransmission is directly triggered. Unlike FEC, ARQ allocates additional network resources only when a packet is retransmitted. The mechanism generally operates more efficiently for low bit rates. An undesirable effect of ARQ is that it may interfere with TCP [3]. Concerning the relaxed packet loss requirements of time-sensitive applications, as well as the implications that may be induced by FEC/ARQ in order to maximize reliability, we chose not to include such mechanisms in our evaluation experiments.

Snoop protocol [3, 2] provides a reliable solution by maintaining TCP end-to-end semantics while recovering the wireless errors locally. Snoop uses link level buffers at the base station (*BS*) to cache packets traversing the wireless link. It retransmits unacknowledged packets and consequently,

unnecessary timeouts are avoided. Furthermore, Snoop suppresses duplicate acknowledgments (*DACKs*) for locally retransmitted packets in order to prevent TCP from performing fast retransmissions and backward window adjustments.

Additional proposals include split connection protocols. A split connection protocol virtually splits a TCP connection into two separate connections. The first one connects the sender with the base station, while the other connection is maintained between the base station and the receiver. A well-known representative of this family of protocols is *Indirect-TCP* (*I-TCP*) [1]. However, these protocols do not handle handoff operations efficiently [6], since such procedures tend to be slow and complicated. Furthermore, due to the split scheme, end-to-end semantics of TCP is violated.

2.2 Improving Real-Time Performance with TCP

Since standard TCP is rarely chosen to transport real-time traffic over the Internet, TCP-friendly protocols constitute an elegant framework for multimedia applications. We consider as TCP-friendly any protocol whose long-term arrival rate does not exceed the one of any conformant TCP in the same circumstances [11]. TCP-friendly congestion control has the ability to maintain network stability by promptly responding to congestion and to be cooperative with other flows, while it commonly provides more efficient QoS, (i.e. a smoothed sending rate and bounded latency for playback multimedia applications). The differences between standard TCP and TCP-friendly congestion control lie mainly in the specific values of α and β, while their similarities in their AIMD based congestion control (a characteristic that enables us to include them both in the family of TCP (α, β) protocols). Standard TCP is therefore viewed as a specific case of TCP (α, β) with $\alpha = 1$ and $\beta = 0.5$.

TCP-Real [25, 29] is a high-throughput transport protocol that incorporates congestion avoidance mechanism in order to minimize transmission-rate gaps. As a result, the protocol is suited for real-time applications, since it enables better performance and reasonable playback timers. TCP-Real approximates a receiver-oriented approach beyond the balancing trade of the parameters of additive increase and multiplicative decrease. The protocol introduces another parameter, namely γ, which determines the window adjustments during congestion avoidance. More precisely, the receiver measures the data-receiving rate and attaches the result to its acknowledgments (*ACKs*), directing the transmission rate of the sender. When new data is acknowledged and congestion window (*cwnd*) is adjusted, the current data-receiving rate is compared against the previous one. If there is no receiving rate decrease, *cwnd* is increased by 1 *Maximum Segment Size* (*MSS*) every *RTT* ($\alpha = 1$). If the magnitude of the decrease is small, *cwnd* remains temporarily unaffected; otherwise, the sender reduces *cwnd* multiplicatively by γ. In [29] a default value of $\gamma = 1/8$ is suggested. However, this parameter can be adaptive to the detected conditions. Generally, TCP-Real can be viewed as a TCP (α, β, γ) protocol where γ captures the protocol's behavior prior to congestion, when congestion boosts up.

TCP Westwood [15] is a TCP-friendly protocol that emerged as a sender-side-only modification of TCP Reno congestion control. TCP Westwood exploits end-to-end bandwidth estimation in order to adjust the values of slow-start threshold (*ssthresh*) and *cwnd* after a congestion episode. The protocol incorporates a recovery mechanism which avoids the blind halving of the sending rate of TCP Reno after packet loss, and enables TCP Westwood to achieve a high link-utilization in the presence of wireless errors. However, in [17] we showed that TCP Westwood tends to overestimate the available bandwidth, due to ACKs clustering. *TCP Westwood+* is a recent extension of TCP Westwood, based on the *Additive Increase/Adaptive Decrease* (*AIAD*) mechanism. Unlike the initial version of

Westwood, TCP Westwood+ computes one sample of available bandwidth every RTT using all data acknowledged in the specific RTT, therefore obtaining more accurate estimates [12].

Congestion episodes often damage the timely delivery of packets and consequently, degrade real-time application performance. Hence, congestion avoidance mechanisms usually provide improved real-time performance. Congestion avoidance may be achieved through packet dropping (i.e. *RED*) or otherwise through bandwidth and delay estimation, which trigger transport-level adjustments prior to congestion. Alternatively, *ECN* is proposed in [21], where packets are marked rather than dropped when congestion is about to happen. A well-designed, congestion avoidance mechanism is *TCP Vegas* [5, 13]. In TCP Vegas, every RTT the sender calculates the throughput rate which subsequently is compared to an expected rate. Depending on the outcome of this comparison the transmission rate of the sender is adjusted accordingly. More precisely, let RTT_{min} denote the minimum RTT measured by the TCP source. Whenever an *ACK* is received, TCP Vegas computes the quantity:

$$\text{diff} = (\text{expected\_Rate} - \text{actual\_Rate}) * \text{RTT}_{\text{min}} \tag{1}$$

The size of *cwnd* is then increased by 1 if $diff < 1$, decreased by 1 if $diff > 3$ and left unchanged if $1 \leq diff \leq 3$. Based on [5] admissions, *Vegas* achieves higher transmission rates than *TCP Reno* and *TCP Tahoe*.

Although we explicitly study the behavior of TCP mechanisms in heterogeneous wired/wireless networks, we briefly refer to selected rate-based protocols, which compose an elegant framework for time-sensitive applications. TFRC [10] is a representative TCP-friendly protocol, which adjusts its transmission rate in response to the level of congestion, as estimated based on the calculated loss rate. Multiple packet drops in the same RTT are considered as a single loss event by TFRC and hence, the protocol follows a more gentle congestion control strategy. More precisely, the TFRC sender uses the following TCP response function:

$$\text{T}(\text{p}, \text{RTT}, \text{RTO}) = \frac{1}{\text{RTT}\sqrt{\frac{2\text{p}}{3}} + \text{RTO}\,(3\sqrt{\frac{3\text{p}}{8}})\,\text{p}\,(1+32\text{p}^2)} \tag{2}$$

where p is the steady-state loss event rate and *RTO* is the retransmission timeout value. Equation (2) enforces an upper bound on the sending rate T. However, the throughput model is quite sensitive to parameters (e.g. p, RTT), which are often difficult to measure efficiently and to predict accurately. Also, the long-term TCP throughput equation does not capture the transit and short-lived TCP behaviors, and it is less responsive to short-term network and session dynamics. According to [10], TFRC's increase rate, which is solely determined by the value of α, never exceeds 0.14 packets per RTT (or 0.28 packets per RTT when history discounting has been invoked). In addition, the protocol requires 5 RTTs in order to halve its sending rate. Consequently, the instantaneous throughput of TFRC has a much lower variation over time. TFRC eventually achieves the smoothing of the transmission gaps and therefore, is suitable for applications requiring a smooth sending rate.

Scalable Streaming Video Protocol (*SSVP*) [18] is a new congestion control scheme, which operates on top of UDP and is optimized for unicast video streaming applications. The transmission rate is controlled in a TCP-friendly fashion by properly adjusting the inter-packet-gap, spacing outgoing packets evenly to produce a smoothed flow. SSVP eventually adapts to the vagaries of the network and provides efficient QoS provisioning for video streaming applications.

3. Throughput and Delay Analysis in Wired/Wireless Networks

We consider a typical scenario for real-time transmission over a network that includes a wireless link with capacity B_W. As depicted in Fig. 1, a streaming server transmits data in real-time to a receiver located in the wireless network. For sake of simplicity, we assume a constant packet loss rate p_W across the wireless channel, due to fading or interference. Consequently, the maximum throughput over the wireless link is $B_w(1 - p_W)$. We also consider the probability of packet loss at nodes A and B, due to congestion. Therefore, p_A and p_B denote the packet loss rates at nodes A and B when real-time traffic and/or interfering traffic overflow the corresponding buffers.

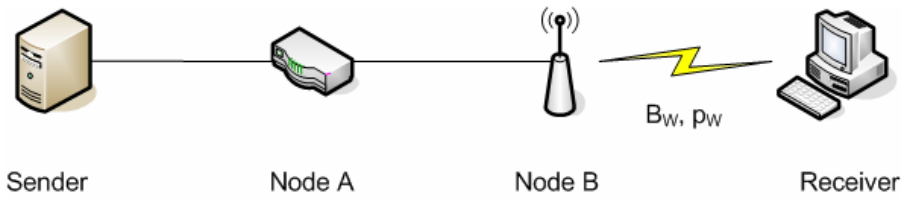


Figure 1 Typical Wireless Scenario

We model throughput rate based on a simple TCP throughput equation [16], which assumes that *DACKs* is the only congestion signal during congestion avoidance:

$$T = \frac{C_0 \, MSS}{RTT\sqrt{p}} \tag{3}$$

where T is the calculating sending rate, *MSS* represents the Maximum Segment Size, p the measured packet loss rate, and $C_0 = \sqrt{3/2}$. The end-to-end packet loss rate p_S, as observed by the receiver, can be expressed as a function of p_A, p_B and p_w, as follows:

$$p_S = p_A + (1 - p_A)\, p_B + (1 - p_A)\,(1 - p_B)\, p_W \tag{4}$$

With respect to equations (3), (4) and assuming that a single real-time flow with throughput T_S does not fully utilize wireless link capacity B_W, we obtain:

$$T_s = \frac{C_0 \, MSS}{RTT\sqrt{p_s}}$$

$$T_S\,(1 - p_S) < B_W\,(1 - p_W) \tag{5}$$

Consequently, if a TCP flow underutilizes the wireless link, equation (5) is satisfied. If interfering traffic causes no congestion and assuming that B_W does not exceed the capacity B of the wired links ($B_W \le B$), we have $p_S = p_W$, since $p_A = p_B = 0$. Therefore, condition in (5) is simplified to $T_S < B_W$.

In the situation of local error control, such as TCP-aware Snoop protocol, p_W becomes effectively zero through local retransmissions. Hence, perceived end-to-end packet loss rate p_S' is represented as:

$$p_s' = \lim_{p_w \to 0} [p_A + (1 - p_A)\, p_B + (1 - p_A)\,(1 - p_B)\, p_w]$$

$$p_s' \approx p_A + (1 - p_A)\, p_B \tag{6}$$

With respect to equation (6), TCP throughput T_S' under local-error control becomes independent of the wireless loss rate p_w:

$$T_s' = \frac{C_0 \, MSS}{RTT\sqrt{p_s'}}$$

and condition in (5) is independent of the wireless channel errors. Furthermore, $p_S' < p_S \Rightarrow T_S' > T_S$. Thus, we validated the effectiveness of local retransmissions in terms of throughput performance. Enabling local error control, the impact of the wireless channel on traffic is expected to be similar to that of a regular wired connection from the perspective of bandwidth utilization.

The task of specifying the effects of network QoS parameters on real-time delivery is challenging. Transmission rate fluctuations, increased delays, jitter and packet loss commonly deteriorate the perceived quality or fidelity of the received video content. We note that these network QoS parameters do not affect quality in an independent manner; they rather act in combination or cumulatively, and ultimately, only this joint effect is detected by the end-user. Considering real-time video or voice transmission, end-to-end delay consists of the delay incurred by the signal from the instant it is produced by the source until its playback at the recipient. Initially, the analog signal is encoded, followed by the packetization phase, incurring an encoding (D_{enc}) and packetization (D_{pack}) delay, respectively. Video/voice packets are then transmitted on the network. Network delay is expressed by the summation of propagation (P_h), transmission (T_h), and the variable queuing and processing delays (Q_h) for each hop h in the path from the source to the destination. If we include a playback delay (D_{play}) and we ignore the processing delays at both sender and receiver, the end-to-end delay D for a packet is expressed, as:

$$D = D_{enc} + D_{pack} + \sum_{h \in Path} (T_h + Q_h + P_h) + D_{play} \quad (7)$$

We observe than for a certain codec and connection the only random component in equation (7) is queuing delay. With concern to the significance of queuing delays, we discuss whether TCP benefits from local retransmissions in terms of real-time delivery.

Snoop protocol recovers wireless losses locally by cashing incoming packets at a buffer located at the BS. If we presume that the wireless link is the bottleneck, the queuing delay over the wireless channel is dominant for the network, and with respect to equation (7), critical for timely delivery. If error conditions degrade, local error control will need to retransmit harder in order to transfer packets across the wireless link. As a result, the queue will drain more slowly and will eventually overflow, resulting in packet drops. Apart from the increased delays, failing to recover from a number of wireless losses will cause TCP to back off and retransmit the lost packets. Furthermore, RTTs may become highly variable resulting in wasteful timeouts and retransmissions or, alternatively, excessive back-offs that would unnecessarily delay later retransmissions.

Error control strategies with different degrees of effectiveness and persistence can be implemented. However, considering the stringent requirements of time-sensitive traffic, persistent local retransmission may delay packet delivery significantly. In summary, we identified that link-layer schemes, and particularly Snoop, improve link utilization by absorbing the unpleasant effects of wireless losses (i.e. DACKs). However, real-time delivery may suffer, due to the additional and

variable queuing delays at link-level buffers. In situations of increased or varying error rates local error control tends to interfere with TCP, degrading performance.

4. Experimental Methodology

4.1 Experimental Settings

The evaluation plan was implemented on the *NS-2* network simulator. In our experiments we used a wired-cum-wireless topology (Fig. 2), where two LANs are connected with a wireless link (5 Mbps). We simulated local retransmissions based on snooping at the wireless base station in order to study the interactions between TCP and the Snoop protocol. Error models were configured on both (forward and reverse) directions of the wireless bottleneck link with configurable packet error rates (*PER*). PER is adjusted at 0.01, unless otherwise explicitly stated. The number of source and sink nodes are always equal. In all experiments, we used droptail routers with buffer size adjusted in accordance with the *bandwidth-delay* product.

In order to simulate real-time traffic, we developed an *MPEG-4 Traffic Generator*. The traffic generated closely matches the statistical characteristics of an original video trace. We used three separate *Transform Expand Sample* (*TES*) models for modeling I, P and B frames respectively. The resulting MPEG-4 stream is generated by interleaving data obtained by the three models. The MPEG traffic generator was integrated into NS-2 and provides the adjustment of the data rate of the MPEG stream, as well as useful statistical data (e.g. average bit-rate, bit-rate variance).

Although we performed a series of experiments over various TCP protocols, we only comment on results from TCP variants Reno and Vegas, as well as TCP-friendly protocols Westwood+ (TCPW+) and TCP-Real, which let us reach the most thorough conclusions. We set the packet size to 1000 bytes and the maximum congestion window for all TCP connections to 64 KB. Each simulation lasts for 60 seconds, and diverse randomization seeds were used in order to reduce simulation dynamics. All the results are collected after 2 sec in order to avoid the skew introduced by the warming up effect.

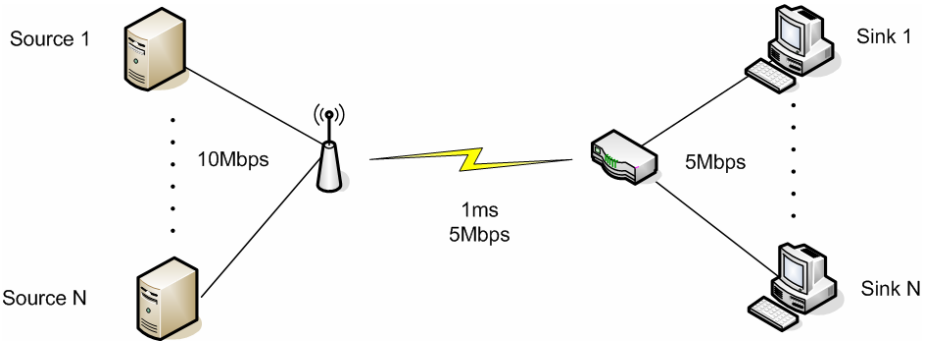


Figure 2 Simulation topology

4.2 Measuring Performance

System goodput is used in order to measure the overall system efficiency in bandwidth utilization and is defined as *Goodput = Original_Data / Connection_time*, where *Original_Data* is the number of bytes delivered to the high-level protocol at the receiver (i.e. excluding retransmitted packets and

overhead) and *Connection_Time* is the amount of time required for data delivery. Long-term fairness is measured by *Fairness Index*, derived from the formula given in [8], and defined as $(\sum_{i=1}^{n} \text{Throughput}_i)^2 / (n\sum_{i=1}^{n} \text{Throughput}_i^2)$, where $Throughput_i$ is the throughput of the i^{th} flow and n is the total number of flows.

Jitter composes a critical factor in the performance of video delivery. According to [22] packet jitter is the value of packet spacing at the receiver compared with packet spacing at the sender for a pair of packets. Therefore, the value of jitter reflects packet-by-packet delay. Let S_i and R_i denote the sending and receiving time for packet i respectively; then for two packets i and j, packet jitter can be expressed as:

$$J(i, j) = (R_j - R_i) - (S_j - S_i) = (R_j - S_j) - (R_i - S_i) \quad (8)$$

According to equation (8), there is no delay between packets i and j, if the value of $J(i, j)$ is equal to zero. Let RTT_{min} denote the RTT excluding the queuing delays and the local retransmissions. Considering our simulation topology, RTT_{min} can be approximately constant, since the wireless link is the bottleneck, and as a result queuing delay over the wireless link is dominant and capable of absorbing the jitter in the wired domain.

Along these lines, in [19] we proposed a new metric for the performance evaluation of time-sensitive traffic, called *Real-Time Performance*, which captures the joint effect of jitter and packet loss on perceived quality. The metric monitors packet inter-arrival times and distinguishes the packets that can be effectively used by the client application from delayed packets (according to a configurable inter-arrival threshold). The proportion of the delayed packets is reflected in *Delayed Packets Rate*. Hence, Real-Time Performance Index is defined as the ratio of the number of *"timely received packets"* over the total number of packets sent by the application:

$$\text{Real} - \text{Time Performance Index} = \frac{\#\,\text{timely received packets}}{\#\,\text{sent packets}} \leq 1$$

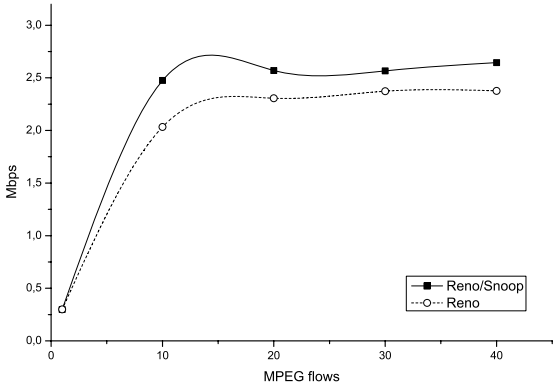
In our experiments, the inter-arrival threshold is adjusted at 75 ms. Since real-time traffic is sensitive to packet losses, we additionally define *Packet Loss Rate*, as the ratio of the number of lost packets over the number of packets sent by the application. For a system with multiple flows, we present the average of the real-time performance of each MPEG flow.

5. Results and Discussion

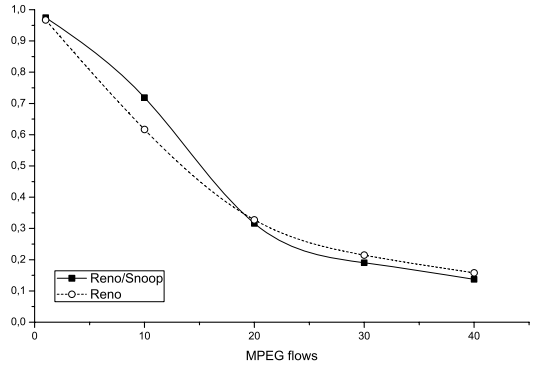
In the sequel, we demonstrate and analyze the most prominent results from the experiments we performed based on three distinct scenarios. The basic parameters of each simulation scenario are as described in the previous section.

5.1 Impact of Link-layer Retransmissions

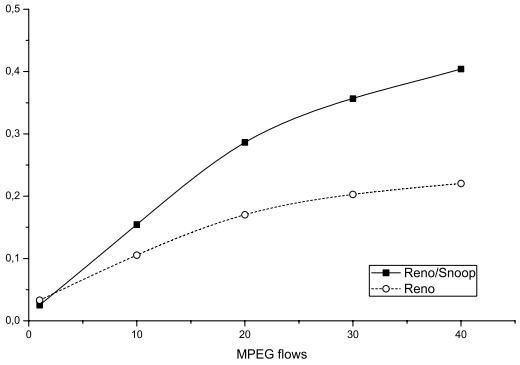
Initially, we assess the efficiency of local error control with the intent of understanding its behavior. More precisely, we evaluate the performance of real-time delivery by investigating the interactions between TCP and the Snoop protocol. We simulated a diverse range of MPEG flows (1-40) and demonstrate the most conclusive results from TCP Reno (Fig. 3) and TCP Vegas (Fig. 4). We performed our experiments for each TCP protocol with and without Snoop.



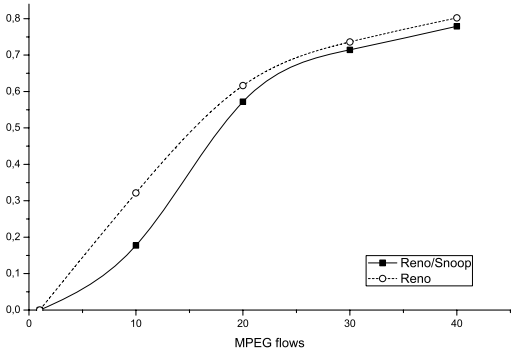
(a) System Goodput



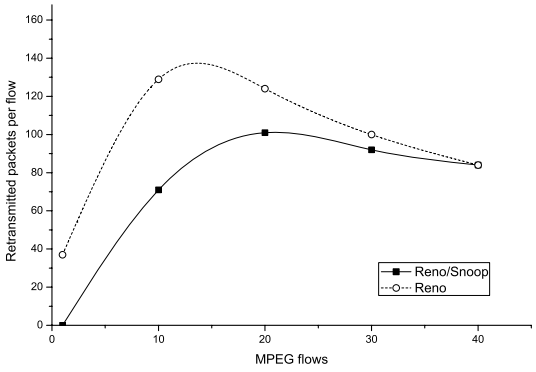
(b) Average Real-Time Performance



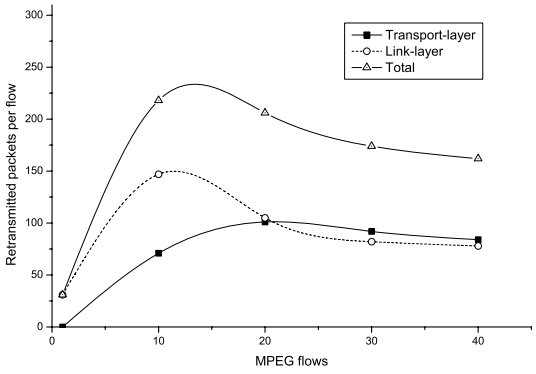
(c) Delayed Packets Rate



(d) Packet Loss Rate

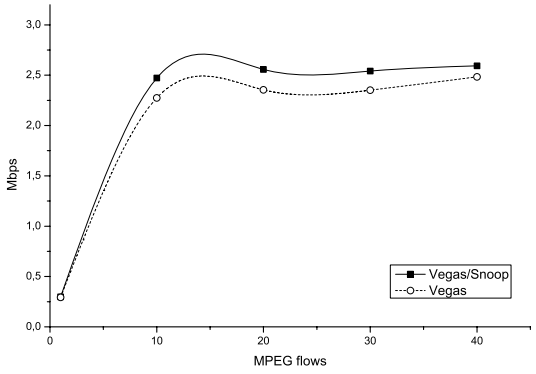


(e) Number of retransmitted packets

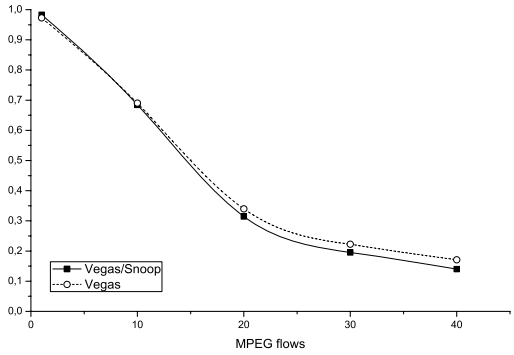


(f) Link- vs. transport-layer retransmissions

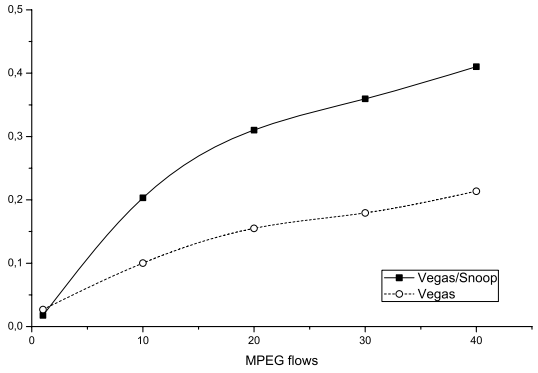
Figure 3 Impact of link-layer retransmissions on TCP Reno



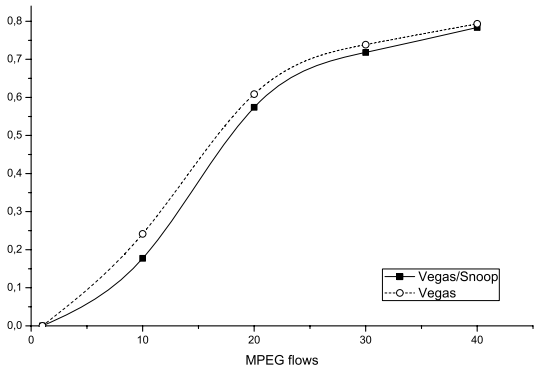
(a) System Goodput



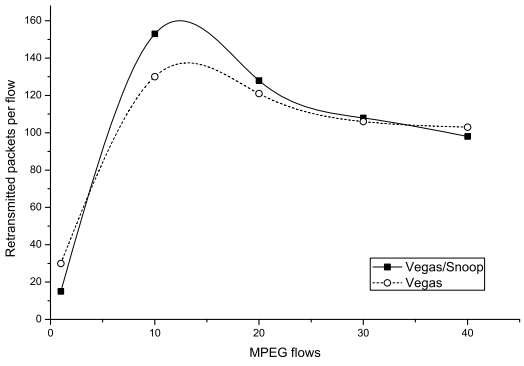
(b) Average Real-Time Performance



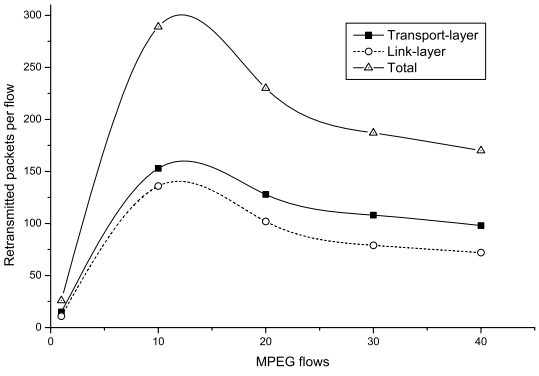
(c) Delayed Packets Rate



(d) Packet Loss Rate



(e) Number of retransmitted packets



(f) Link- vs. transport-layer retransmissions

Figure 4 Impact of link-layer retransmissions on TCP Vegas

Fig. 3a illustrates that Snoop enables TCP Reno to achieve a more efficient performance in terms of bandwidth utilization. Snoop, running at the link layer, responds to packet losses faster than Reno. Most wireless losses are recovered locally (within TCP's timeout) and consequently, TCP retransmissions along with wasteful backward window adjustments are eventually prevented. This observation is also depicted in Fig. 3d, where the combination of Reno and Snoop exhibits fewer packet drops than Reno alone. Figs. 3e, 3f demonstrate statistics from retransmitted packets at the transport layer (per flow), and a comparison between TCP and local error retransmissions (per flow), respectively. Local error control in part confines end-to-end retransmissions, allowing the sending window to inflate. Therefore, higher transmission rates can be achieved. Fig. 3f reveals that Snoop interacts efficiently with Reno, since the link-layer mechanism manages to recover a considerable amount of wireless losses locally and prevent several undesirable implications on TCP's operation.

However, from the perspective of real-time delivery, Snoop's supportive role is not evident, since minimal performance gains are occasionally achieved (Fig. 3b). In the situation of high link-multiplexing where congestion is the primary cause for packet loss, local error control slightly degrades performance, as the effect of an increased number of delayed packets (Fig. 3c). As we identified in Section 3, the queuing delays across the wireless channel compose a critical factor for path delay, while delay variations in the link-layer buffers usually introduce jitter. Consequently, although Reno benefits from Snoop in terms of bandwidth allocation, the fluctuations in RTTs compromise real-time performance, and occasionally cause data unavailability at the receiver.

A comparative overview between the interactions of Snoop with Reno (Fig. 3) and Vegas (Fig. 4) respectively, leads to the overall conclusion that the combination of Reno and Snoop is more effective. In the situation of Vegas, gains in bandwidth utilization are also noticeable (Fig. 4a), but not as profound as with Reno (Fig. 3a). In our homogeneous scenario, the competing Vegas connections may converge to different *cwnd* values causing variations in flow throughputs. Furthermore, the protocol measures RTT_{min} which approximates a constant value in the simulated topology, as reported in Section 4. Therefore, Vegas does not respond effectively to the fluctuations of RTTs caused by the varying queuing delays at the BS buffer. In this context, protocol efficiency would be seriously affected by a persistent local error control scheme. Fig. 4e also validates our conclusion that Snoop does not interact effectively with Vegas: end-to-end retransmissions are occasionally increased in the event of local recovery. Therefore, Snoop does not react to packet drops fast enough in order to prevent retransmissions from the Vegas source. In terms of real-time delivery, Snoop degrades the performance of Vegas (Fig. 4b), as the proportion of delayed packets (Fig. 4c) is remarkably increased.

Apart from Reno and Vegas, we investigated the interactions of Snoop with other TCP variants, such as TCPW+ and TCP-Real, and we evaluated the associated impact on real-time application performance. The overall conclusion of these efforts is that Reno interacts with Snoop more efficiently among all the TCP versions tested. Inline with our analysis in Section 3, we validated that local error control commonly improves bandwidth utilization regardless of the TCP protocol. We identify that despite these gains, the wireless bottleneck link remains underutilized, i.e. $T_S (1 - p_S) < B_W (1 - p_W)$ holds. However, Snoop's supportive role in terms of real-time performance is not evident. The interaction of Snoop with end-to-end solutions that address time-sensitive traffic (i.e. TCP Real) is discouraging. An important issue that enables local error control to achieve adequate performance is the proper adjustment of retransmission timeout (*RTO*). That is, RTO should be substantially longer that the wireless-hop RTT. Furthermore, local retransmission should not be persistent, since packets of real-time traffic ought to reach the receiver after a short time interval. In the following scenarios the

experimental study of Snoop is limited only under TCP Reno (i.e. where Snoop yields most of its effectiveness).

5.2 Link- vs. Transport-Layer Efficiency

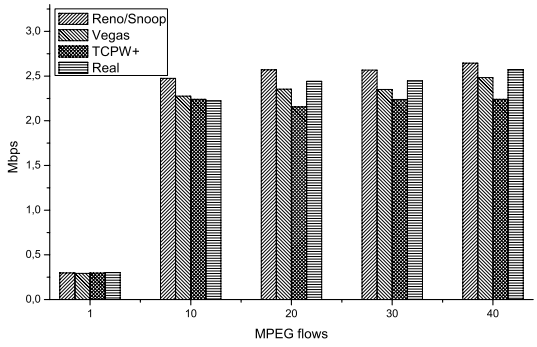
Departing from the analysis of Snoop's supportive role, we investigate whether local error control compares favorably with selected transport-layer mechanisms. In the sequel, we present conclusive results from TCP Reno interacting with Snoop versus TCP protocols Vegas, TCPW+ and Real (Fig. 5), all of which are able to change the sending rate adaptively, although in a different fashion. Both TCPW+ and TCP-Real address the distinct characteristics of time-sensitive traffic and are implied to yield remarkable efficiency on real-time delivery over a wide range of network and session dynamics.

Fig. 5a illustrates that the combination of TCP Reno and Snoop achieves the highest bandwidth utilization. In comparison with the rest of the protocols, Reno and Snoop exhibit the most effective responses to wireless errors, preventing TCP from unnecessary fast retransmissions and congestion control invocations. Consequently, local error control composes the most prominent approach towards bandwidth allocation over wireless links, since it manages to alleviate most of the impairments induced by wireless errors (Fig. 5d). However, the bottleneck link still remains underutilized, since the transient errors do not let the sending window inflate to higher values.

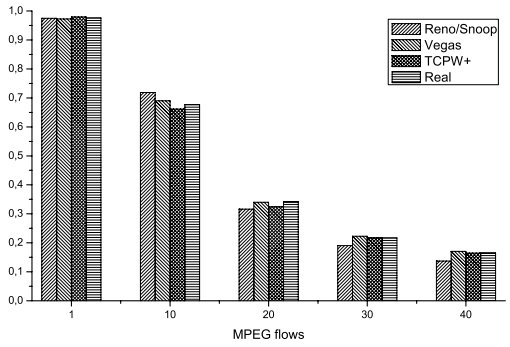
Both TCP Vegas and TCP-Real achieve remarkable goodput rates, although Vegas is not designed for wireless environments, since it is not able to detect the nature of error. On the contrary, Fig. 5a depicts a deficiency of TCPW+, especially during increased contention. The protocol does not inherently support error classification invoking congestion-oriented responses to wireless errors. In addition, despite the improvements over the initial version of Westwood, TCPW+'s algorithm still does not obtain accurate estimates in heterogeneous environments, failing to achieve adequate utilization of the available bandwidth.

A comparative view in the results of Figs. 5a and 5b reveals that high goodput rates do not necessitate improved performance on real-time delivery. Hence, the superior performance of Reno and Snoop in terms of bandwidth utilization is not met in the real-time performance results (Fig. 5b). Although Snoop appears to deal effectively with link errors, it is responsible for excessive delays (Fig. 5c), which degrade performance. Local error control has the disadvantage of adding extra queuing and processing delays, which eventually impair the perceived quality of the real-time stream. Furthermore, local retransmissions introduce variation in RTTs and are responsible for varying gaps in the receiving rate.

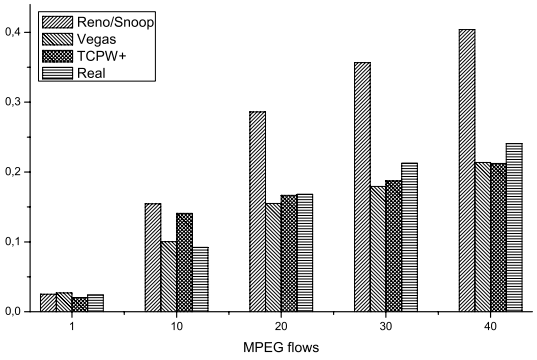
TCP Vegas and TCP-Real combine effective bandwidth utilization with an acceptable amount of delayed packets (Fig. 5c) achieving more efficient QoS provisioning for time-sensitive applications (Fig. 5b). Their congestion avoidance mechanisms enable both protocols to adapt to the vagaries of the network and eventually deliver a smooth real-time flow. TCP-Real in particular exploits its mechanisms which smooth transmission gaps and confine short-term oscillations in the sending rate. On the contrary, as we reported TCPW+ suffers from bandwidth utilization problems with a direct impact on real-time delivery (Fig. 5b). However, for high link-multiplexing each connection has limited bandwidth to allocate and thus, TCPW+'s bandwidth underutilization issue is not critical. In this situation, its real-time performance is not confined. Finally, Fig. 5e depicts that TCPW+ and in part Vegas can not handle bandwidths sharing efficiently; for both protocols the competing connections converge to different *cwnd* values, and therefore achieve different flow throughput rates.



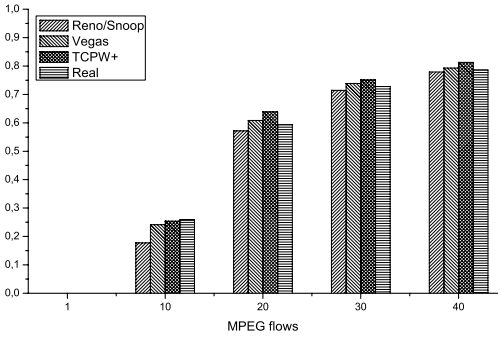
(a) System Goodput



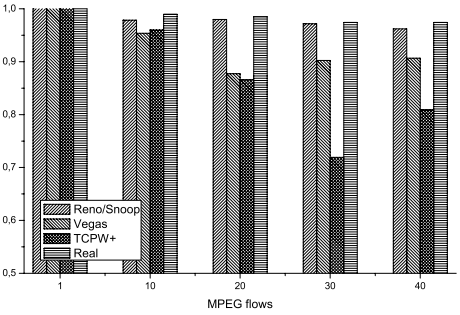
(b) Average Real-Time Performance



(c) Delayed Packets Rate



(d) Packet Loss Rate



(e) Fairness Index

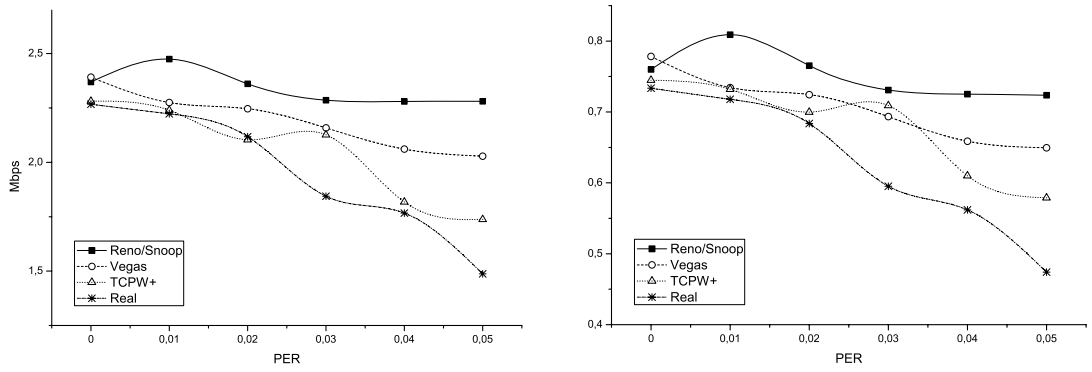
Figure 5 Protocol and Real-Time Performance

5.3 Real-Time Delivery vs. Packet Error Rates

In the last scenario, we performed the experiments using diverse packet error rates (PER: 0.01 - 0.05). We also carried out the same experiment without link errors and used it as a reference. Our

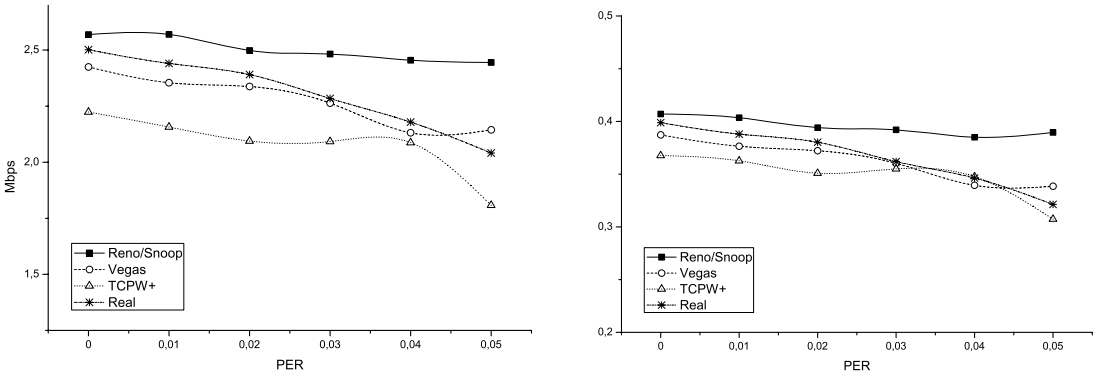
objective is to demonstrate the impact of diverse packet error rates on bandwidth utilization and primarily on real-time delivery (Figs. 6, 7).

According to our expectations, the corresponding results illustrate TCP's performance degradation in the event of increasing link errors. Reno in conjunction with Snoop is less sensitive to the diverse packet error rates, due to the extended reliability provided by Snoop. Local error control is able to mask the worst effects of the high and dynamic error rates often found in wireless networks. The supportive role of Snoop is more effective, as link errors increase across the wireless channel. Similar to Reno/Snoop, Vegas exhibits minor implications in the event of increasing wireless errors. On the contrary, TCP-Real and TCPW+ demonstrate limited efficiency at relatively high packet error rates (PER: 0.04, 0.05). However, the performance of TCP-Real is notably improved when contention is increased (i.e. 20 flows).



(a) System Goodput (b) Average Real-Time Performance

Figure 6 Impact of diverse error rates (10 flows)



(a) System Goodput (b) Average Real-Time Performance

Figure 7 Impact of diverse error rates (20 flows)

6. Conclusions and Future Work

We investigated the effect of local error control on real-time transport, motivating by the remarkable feasibility of wireless link protocols in terms of wide range deployment. We showed both analytically and experimentally that local retransmissions can deliver considerable improvements on TCP performance over lossy wireless links, by allowing the protocol to use a higher fraction of the available bandwidth. However, cashing incoming packets at link-layer buffers may introduce arbitrary delays that cause perceptible variations in RTTs and disturbing fluctuations in the receiving rate. We demonstrated that local error control degrades the performance on real-time delivery in a wide range of network dynamics and regardless of the TCP variant. Although local retransmissions should be persistent enough to virtually eliminate non-congestive losses, delay-sensitive traffic certainly requires local recovery of reduced-persistence. Furthermore, RTO should be substantially longer that the wireless-hop RTT in order to avoid unnecessary timeouts and false triggering of end-to-end congestion control invocations.

The comparison of Reno/Snoop with selected end-to-end proposals reveals the disability of local error control to achieve perceptible gains in terms of real-time delivery. Our results show that real-time performance gains are eventually attained by end-to-end mechanisms that explicitly address time-sensitive traffic, such as TCP-Real. The specific protocol achieves remarkable performance for highly multiplexed heterogeneous networks. We also identified the efficiency of TCP Vegas, although the protocol does not achieve a fair behavior. In summary, most of the end-to-end solutions we tested achieve effective QoS provisioning for time-sensitive applications. Their performance deteriorates only over highly error-prone links (i.e. PER > 0.03), where flow characteristics do not follow a prescribed and static behavior. In such error environments, local error control composes a more attractive approach. Future work includes the design of a cross-layer scalable scheme that will exploit information from the link-layer in order to utilize wireless resources more effectively. Such an approach is intended to support a variety of multimedia applications with a broad range of QoS requirements over heterogeneous wired/wireless networks.

References

1. A. Bakre and B. R. Badrinath, I-TCP: Indirect TCP for Mobile Hosts, In Proc. of 15th Int/nal Conference on Distributed Computing Systems (IDCS), Vancouver, Canada, May 1995
2. H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, A Comparison of Mechanisms for Improving TCP Performance over Wireless Links, ACM/IEEE Transactions on Networking, 5 (6), 1997, 756-769
3. H. Balakrishnan, S. Seshan and R. Katz, Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks, ACM Wireless Networks, 1(4), December 1995, 469-482
4. D. Bansal and H. Balakrishnan, Binomial Congestion Control Algorithms, In Proc. of IEEE INFOCOM 2001, Anchorage, Alaska, USA, April 2001
5. L. Brakmo and L. Peterson, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal on Selected Areas of Communications, 13 (8), October 1995, 1465-1480
6. R. Cáceres and L. Iftode, Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments, IEEE Journal on Selected Areas in Communications (JSAC), 13 (5), June 1995, 850-857
7. S. Cen, P. C. Cosman and G. M. Voelker, End-to-end Differentiation of Congestion and Wireless Losses, IEEE/ACM Transactions on Networking, 11 (5), October 2003, 703-717

8. D. Chiu and R. Jain, Analysis of the increase/decrease algorithms for congestion avoidance in computer networks, Journal of Computer Networks, 17 (1), June 1989, 1-14
9. A. Chockalingam, M. Zorzi and V. Tralli, Wireless TCP performance with link layer FEC/ARQ, In Proc. of IEEE ICC '99, Vancouver, Canada, June 1999
10. S. Floyd, M. Handley, J. Padhye and J. Widmer, Equation-Based Congestion Control for Unicast Applications, In Proc. of ACM SIGCOMM 2000, Stockholm, Sweden, August 2000
11. S. Floyd and K. Fall, Promoting the use of end-to-end congestion control in the Internet, IEEE/ACM Transactions on Networking, 7 (4), August 1999, 458-472
12. L. Grieco and S. Mascolo, Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control, ACM Computer Communication Review, 34 (2), April 2004, 25-38
13. U. Hengartner, J. Bolliger and T. Cross, TCP Vegas Revisited, In Proc. of IEEE INFOCOM 2000, Tel-Aviv, Israel, March 2000
14. V. Jacobson, Congestion avoidance and control, In Proc. of ACM SIGCOMM '88, Stanford, USA, August 1988
15. S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi and R. Wang, TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links, In Proc. of ACM MobiCom 2001, Rome, Italy, July 2001
16. M. Mathis, J. Semke, J. Mahdavi and T. Ott, The macroscopic behavior of the TCP congestion avoidance algorithm, ACM Comp. Communications Review, 27 (3), July 1997, 67-82
17. P. Papadimitriou and V. Tsaoussidis, Assessment of Internet Voice Transport with TCP, Int/nal Journal of Communication Systems (IJCS), 19 (4), May 2006, 381-405
18. P. Papadimitriou and V. Tsaoussidis, End-to-end Congestion Management for Real-Time Streaming Video over the Internet, In Proc. of 49th IEEE GLOBECOM, San Francisco, USA, November 2006
19. P. Papadimitriou and V. Tsaoussidis, On Transport Layer Mechanisms for Real-Time QoS, Journal of Mobile Multimedia, Rinton Press, 1 (4), January 2006, 342-363
20. P. Papadimitriou, V. Tsaoussidis and A. Tsioliaridou, The Impact of End-to-end vs. Link-layer Mechanisms on Real-Time Performance over Wireless Links, In Proc. of 20th Int/nal Conference on Advanced Information Networking and Applications (AINA 2006), Vienna, Austria, April 2006
21. K. Ramakrishnan and S. Floyd, A proposal to add explicit congestion notification (ECN) to IP, RFC 2481, January 1999
22. H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, RTP: A transport protocol for real-time applications, RFC 1889, IETF, January 1996
23. W. Stevens, TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms, RFC 2001, Janary 1997
24. V. Tsaoussidis and I. Matta, Open issues on TCP for Mobile Computing, Journal of Wireless Communications and Mobile Computing, 2 (2), February 2002, 3-20
25. V. Tsaoussidis and C. Zhang, TCP Real: Receiver-oriented congestion control, Journal of Computer Networks, 40 (4), November 2002, 477-497
26. V. Tsaoussidis and C. Zhang, The Dynamics of Responsiveness and Smoothness in Heterogeneous Networks, IEEE Journal on Selected Areas in Communications (JSAC), 23 (6), June 2005, 1178-1189
27. Y. R. Yang, M.S. Kim and S.S. Lam, Transient Behaviors of TCP-friendly Congestion Control Protocols, In Proc. of IEEE INFOCOM 2001, Anchorage, Alaska, USA, April 2001
28. Y. R. Yang and S.S. Lam, General AIMD Congestion Control, In Proc. of 8th Int/nal Conference on Network Protocols (ICNP), Osaka, Japan, November 2000
29. C. Zhang and V. Tsaoussidis, TCP Real: Improving Real-time Capabilities of TCP over Heterogeneous Networks, In Proc. of 11th ACM NOSSDAV, New York, USA June 2001