Journal of Mobile Multimedia, Vol.1 No.4 (2006) 342-363
@Rinton Press

ON TRANSPORT LAYER MECHANISMS FOR REAL-TIME QoS

PANAGIOTIS PAPADIMITRIOU and VASSILIS TSAOUSSIDIS

Demokritos University of Thrace, Electrical & Computer Engineering Department

Xanthi, 67100 GREECE

E-mail: {ppapadim, vtsaousi}@ee.duth.gr

Received February 15, 2005
Revised June 24, 2005

We study transport protocol performance from the perspective of real-time applications. More precisely, we evaluate TCP and UDP supportive role in terms of real-time QoS, network stability and fairness. A new metric for the evaluation of real-time application performance is proposed to capture both bandwidth and delay requirements. Using this metric as a primary criterion in our evaluation analysis, we reach several conclusions on the specific impact of wireless links, real-time traffic friendliness, and UDP/TCP protocol efficiency. Beyond that, we also reach an unexpected result: UDP traffic has occasionally negative impact compared with TCP traffic not only for the systemwide behavior, but also for the supporting application as well.

***Key words*:** QoS, congestion control, transport protocols, performance evaluation, real-time applications
Communicated by: K Liu

1. Introduction

Quality of Service (*QoS*) is increasingly important for applications over the Internet. A main factor that complicates and even obstructs the efforts for efficient end-to-end QoS management is the heterogeneity of the Internet. However, identifying the presence of network heterogeneity is not enough. It is necessary to clarify and analyze all the heterogeneity parameters. The application domain is generally classified into non-real time (e.g. HTTP, FTP) and real-time traffic (e.g. multimedia streaming). Real-time applications are comparatively intolerant to delay and to variations of throughput and delay [7]. They are also affected by reliability parameters, such as packet loss and packet errors. Therefore, real-time applications yield satisfactory performance only under certain QoS provisions, which may vary depending on the application task and the type of media involved. Real-time applications compete with other network traffic, as they often share the same channel with corporate FTP and HTTP traffic. However, how real-time traffic affects or might be affected by other network traffic is still an open issue.

A real-time application has the option to run over *Transmission Control Protocol* (*TCP*) or *User Datagram Protocol* (*UDP*). TCP is the dominant protocol for data transmission over the Internet. Although TCP is based on the unreliable datagram service offered by IP, it manages to provide a reliable data delivery service to Internet applications. TCP uses a variety of techniques to achieve reliability. Generally, the protocol combines retransmission in conjunction with the sliding window mechanism. In standard TCP, sliding window adjustments are implemented according to the *Additive Increase Multiplicative Decrease* (*AIMD*) algorithm proposed by Chiu and Jain in [6].

TCP is designed to allocate network resources equally to each application. However, the demand of competing flows often exceeds the channel bandwidth leading to congestion. Therefore, efficient congestion control is of high importance in order to avoid undesirable

implications for the network, such as congestive collapse. A series of mechanisms have been proposed for congestion control, including Congestion Avoidance [15], Slow Start, Fast Retransmit and Fast Recovery. Congestion control is usually triggered after a single packet loss; the window is adjusted backwards and timeout is extended, which in turn degrades the protocol's capability to detect and exploit error free conditions and bandwidth availability, respectively [24]. In heterogeneous wired/wireless environments, apart from congestion, hand-offs and fading channels may trigger packet drops. Standard TCP is unable to successfully detect the nature of the errors in such a network environment. Hence, in most cases congestion control mechanisms are triggered unduly.

Although the reliable service of TCP and its congestion control are suitable for traditional network traffic, real-time applications often struggle to operate efficiently. The sliding window adjustments of TCP do not provide the regular flow required by real-time applications when transmitting data. In wireless environments, the congestion-oriented responses to wireless link errors lead to wasteful rate adjustments. The effect of these awkward conditions is long and varying delays, which damage the timely delivery of real-time data. Several TCP protocol extensions have emerged to overcome the standard TCP limitations providing more efficient bandwidth utilization and sophisticated mechanisms of congestion control.

Alternatively, some real-time implementations are based on the *User Datagram Protocol* (*UDP*). UDP is a fast, lightweight protocol without any transmission or retransmission control. UDP does not have functionality to override application characteristics, such as its transmission rates. It simply transmits at application rate and pattern. Consequently, UDP appears to be more suitable for real-time applications which tolerate some packet losses. However, the lack of a congestion control mechanism is a significant shortfall for UDP, especially as the Internetworking functionality evolves towards punishing free-transmitting protocols. Furthermore, the design principles of UDP do not guarantee fairness, thus, any applications running over UDP are not fair. So far the research community is left with the impression that UDP gains performance by *"stealing"* bandwidth from other flows. In this context, UDP is considered as a *"selfish"* protocol that results in performance gains.

Our research work is motivated by the following questions:

- How crucial is congestion control regarding real-time traffic?
- When the network load increases, does it contribute to efficiency as well?
- What is the efficiency of TCP and UDP with real-time traffic?
- When packet loss increases and UDP maintains its transmission rate, does it really maintain application efficiency as well?
- Are traditional metrics comparative enough to evaluate real-time application performance?

Initially, we refer to recent research proposals which manage QoS focusing on network and protocol design. In Section 3, we present our evaluation methodology and we define a new performance metric for real-time applications. In Section 4, we analyze the results of our experiments and finally in Section 5, we highlight our conclusions.

2. Related Work

The impact of network and protocol heterogeneity on real-time application QoS has not been studied in depth. Relevant work includes [5], where the authors discuss the impact of mobility on QoS, and [7], where the QoS of real-time traffic along with its characteristics are analyzed. Authors in [8] discuss how streaming traffic competes with other TCP traffic over low bandwidth WAN links. In [21] the characteristics of UDP packet loss are investigated through simulations of mixing UDP and TCP real-time audio traffic over WANs. Furthermore, there are remarkable

research efforts towards the efficient QoS management of real-time applications focusing on protocol design. We discuss them in the rest of this section.

2.1 TCP-Friendly Protocols

The disqualification of standard TCP to meet the requirements of real-time applications was the motive for a new family of protocols. Authors in [9, 10, 27, 28] proposed a family of TCP compatible protocols, called *TCP-Friendly*. TCP-Friendly protocols achieve smooth window adjustments, while they manage to compete fairly with TCP flows. In order to achieve smoothness, this family of protocols use a gentle backward adjustment upon congestion. However, this modification has a negative impact on the protocol responsiveness [26].

TCP-Friendly Rate Control (*TFRC*) is a representative TCP-Friendly, rate-based congestion control protocol. According to TFRC, the transmission rate is adjusted in response to the level of congestion as it is indicated by the loss rate [16]. Unlike standard TCP, the instantaneous throughput of TFRC has a much lower variation over time and consequently only smooth adjustments are needed. Furthermore, multiple packet losses in the same *Round Trip Time* (*RTT*) are considered as a single loss event by TFRC and hence, the protocol follows a more gentle congestion control strategy. TFRC eventually achieves the smoothing of the transmission gaps and therefore, is suitable for applications requiring a smooth sending rate, such as streaming media. However, this smoothness has a negative impact, as the protocol becomes less responsive to bandwidth availability [26]. TFRC has another major constraint: it is designed for applications transmitting fixed sized packets and consequently its congestion control is unsuitable for applications that use packets with variable size. In order to overcome this inconvenience, a TFRC variant, namely *TFRC-PacketSize*, has been alternatively proposed.

TCP-Real [29, 25] is a high-throughput transport protocol that incorporates congestion avoidance mechanism in order to minimize transmission-rate gaps. As a result, this protocol is suited for real-time applications, as it enables better performance and reasonable playback timers. TCP-Real employs a receiver-oriented and measurement based congestion control mechanism that significantly improves TCP performance over heterogeneous networks and asymmetric paths. In TCP-Real, the receiver decides with better accuracy about the appropriate size of the congestion window. Slow Start and timeout adjustments are present, but they are only used whenever congestion avoidance fails. However, rate and timeout adjustments are aborted whenever the receiving rate indicates sufficient availability of bandwidth [29]. In the scenario of multimedia streaming over heterogeneous networks with time-constrained traffic, wireless link errors and asymmetric paths, TCP-Real achieves improved performance over standard TCP versions.

TCP Westwood is a sender-side-only modification of *TCP Reno* congestion control, which exploits end-to-end bandwidth estimation to properly set the values of slow-start threshold and congestion window after a congestion episode [12]. TCP Westwood significantly improves fair sharing of high-speed networks capacity. The protocol performs an end-to-end estimate of the bandwidth available along a TCP connection to adaptively set the control windows after congestion [18]. Although TCP Westwood does not incorporate any mechanisms to support error classification and the corresponding recovery tactics for wired/wireless networks, the proposed mechanism appears to be effective over wireless links due to its efficient congestion control. *TCP Westwood+* is a recent extension of TCP Westwood, based on the additive increase/adaptive decrease (*AIAD*) mechanism [13]. TCP Westwood+ exploits the acknowledgment packets stream in order to estimate the bandwidth that is available for the TCP connection.

2.2 Congestion Avoidance

A congestion episode usually has a negative impact on the performance of a real-time application, regardless of the effectiveness of the TCP congestion control mechanisms. Based on this

observation, an approach, dealing with congestion from another perspective, has been proposed. The goal of this approach, called congestion avoidance, is to estimate the level of congestion before it takes place, and hence avoid it. In this context, network traffic is constantly monitored in an effort to anticipate and avoid congestion at common network bottlenecks. 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 [20], where packets are marked rather than dropped when congestion is about to happen. The benefit is obvious: TCP functionality regarding congestion is enhanced without the need to drop and eventually retransmit packets.

A well-designed, congestion avoidance mechanism is *TCP Vegas*. Every RTT the sender calculates the throughput rate which subsequently is compared to an expected rate [24]. Depending on the outcome of this comparison the transmission rate of the sender is adjusted accordingly. Based on [3] admissions, Vegas achieves better transmission rates than TCP Reno and TCP Tahoe. Although the protocol is compliant to the rules of fairness (AIMD algorithm), according to [14], Vegas can not guarantee fairness. Furthermore, the protocol is not able to distinguish the nature of error. However, this constraint is common to the most TCP versions.

2.3 Improving TCP Performance over wireless links

Finally, several proposals have been presented to tackle the problems of TCP over wireless/mobile networks. Most of these proposals rely on some form of local retransmission at the wired/wireless boarder, and do not deal with real-time application constraints [1, 2, 24]. *TCP-Freeze* [11] distinguishes handoffs from congestion through the use of the advertised window. *WTCP* [22] implements a rate-based congestion control replacing entirely the ACK-clocking mechanism. *TCP Probing* [23] grafts a probing cycle and an *Immediate Recovery Strategy* into standard TCP in order to control effectively the throughput/overhead trade-off. Authors in [4] present and analyze the bandwidth estimation schemes implemented at the sender side of a TCP connection, including the estimation algorithms of TCP Vegas and TCP Westwood. In addition, they propose *TIBET* (*Time Intervals based Bandwidth Estimation Technique*), a new bandwidth estimation scheme implemented within the TCP congestion control procedure, which enhances TCP source performance over wireless links.

3. Evaluation Methodology

3.1 Scenarios and Parameters

The evaluation plan was implemented on the *NS-2* network simulator. In our experiments we used the typical single-bottleneck *dumbbell*, as a network topology (Fig. 1). We selected two configurations for the capacity of the bottleneck (*bw_2*), the access links to source nodes (*bw_1*) and the access links to sink nodes (*bw_3*). In the first two scenarios by setting *bw_2 = 100 Mbps* and *bw_1 = bw_3 = 10 Mbps*, we simulated a relatively high speed topology where each flow has enough fair-share to expand its window. In the rest of our experiments we used a topology suited for wireless environments where *bw_2 = 10 Mbps* and *bw_1 = bw_3 = 1 Mbps*. The link delay of *bw_1* and *bw_3* is always set at 10ms. The bottleneck link has a propagation delay of 25ms, apart from the last scenario, where several adjustements are made in order to investigate the impact of link delay on real-time application performance. The number of source and sink nodes are equal in all scenarios. We used droptail routers for both topologies. The buffer size of the router that forwards the packets to the receivers was adjusted in accordance with the *delay-bandwidth* product. Hence, in the first topology the buffer size was adjusted at 150 packets, while in the second one it was adjusted at 30 packets.

Most of our experiments focus on protocol behavior and the associated impact on real-time performance in wireless environments. Handoffs are very common events in wireless networks. The protocols experience a situation where 5 handoff events occur and each one lasts 1 sec. Furthermore, for wireless network simulations, NS-2 error models were inserted into the access links to the sink nodes. Error models were configured on both (forward and reverse) directions of the link traffic. The *Bernoulli* model was used in order to simulate link-level errors with configurable packet error rate (*PER*). In the wireless experiments we adjusted PER at 0.01. The third scenario is the only exception, where PER varies from experiment to experiment.

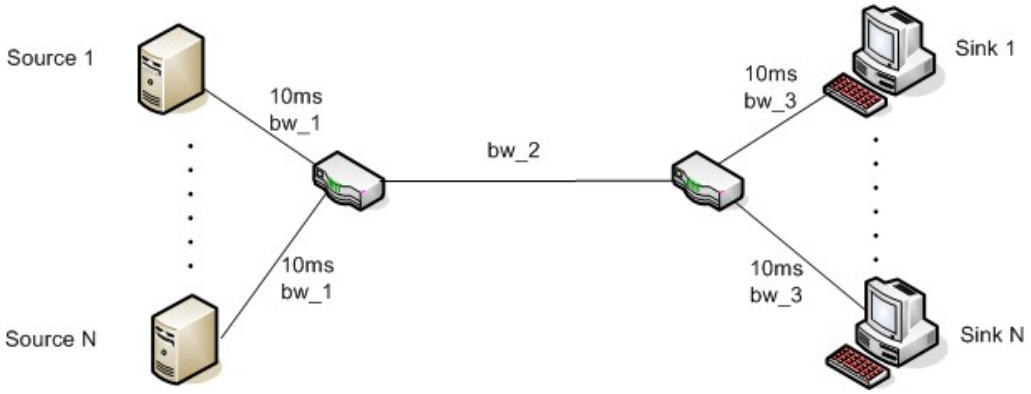

Figure 1. Simulation topology

In our experiments, we used the MPEG-4 traffic generator, proposed in [17], in order to simulate real-time traffic. The traffic generated closely matches the statistical characteristics of an original video trace. The model developed is based on *Transform Expand Sample* (*TES*) [19]. We used three separate 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).

We simulated a wide range of flows (1-80) adjusting the contention accordingly. For all the experiments, the simulation time was fixed at 60 seconds, an appropriate time-period for all the protocols to demonstrate their potential. In order to evaluate TCP performance, we used two representative TCP protocols as reference: Reno and Vegas. However, in the second scenario, we added the TCP-Friendly protocols TCP-Westwood and TCP-Westwood+ in order to evaluate their efficiency in comparison with the rest of TCP versions.

3.2 Measuring Performance

The system goodput is used to measure the overall system efficiency in bandwidth utilization. The system Goodput 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 the data delivery. In order to evaluate the efficiency of selected flows, such as an MPEG flow competing with other TCP flows, we also measure the goodput of each flow separately. Fairness is measured by the *Fairness Index*, derived from the formula given in [6], and defined as:

$$\text{Fairness Index} = \frac{(\sum_{i=1}^{n} \text{Throughput}_i)^2}{n(\sum_{i=1}^{n} \text{Throughput}_i^2)}$$

where $Throughput_i$ is the throughput of the i_{th} flow and n is the total number of flows.

In [25] Tsaoussidis and Zhang proposed a new metric for the performance evaluation of real-time traffic, called "*x% Application Success Percentage"*. Virtually, the metric captures the number of discrete time slots when the flow achieves at least *x%* of the *Targeted Receiving Rate*. The calculation of *Application Success Percentage* is based on the *AllottedGoodput(i, j)* which is the goodput of the i_{th} flow within the j_{th} second. Although this metric is more appropriate for the evaluation of real-time application performance than goodput, it exhibits certain limitations. More precisely, it calculates the percentage of application's successful attempts to read *x%* of an amount

of data from the playback buffer. However, the metric determines whether each attempt is successful or not, based only on instantaneous goodput measurements. Hence, critical parameters, such as packet inter-arrival times, are not taken into account. Furthermore, the metric has 3 parameters: *Targeted Receiving Rate*, *x%* and the amount of data that application consumes every second. Adjusting them accurately for each experiment inevitably adds complexity, since each parameter should be carefully configured, depending not only on the application type, but also on the network specifications.

Targeting at a more efficient criterion for the evaluation of real-time traffic performance, we introduce a new metric, called *Real-Time Performance*. The design principles of this metric are in accordance with the relatively strict requirements of real-time applications. The key to evaluate the performance of a real-time application is to measure the packets that arrive at the receiver(s) on time, so that they can be effectively used. The metric achieves the efficient performance evaluation of real-time traffic and can be easily configured according to individual application requirements. More precisely, *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:

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

Real-Time Performance monitors packet inter-arrival times and distinguishes packets arriving on time from excessively delayed packets (according to a configurable inter-arrival threshold). Practically, delayed packets are either discarded and considered lost, or at the worst they obstruct the proper reconstruction of oncoming packets. The proportion of the number of delayed packets is expressed by *Delayed Packets Rate*. 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 protocol. In order to measure the number of received packets that can be effectively used by the real-time application, an extension has been made to the functionality of the receiver. Each recipient, receiving packets from the MPEG streaming application, calculates the number of *timely received packets* based on the following algorithm:

Algorithm 1. Timely Received Packets

```
# For each packet received with sequence number i, determine
# whether it is a timely received packet or a delayed packet

if threshold > 0 then
   set packetTime[i] = currentTime
   increase packetsReceived
   if i = 0 then
      increase timelyPackets
   else
      if packetTime[i] - PacketTime[i - 1] > threshold then
         increase delayedPackets
      end if
   end if
end if
set timelyPackets = packetsReceived - delayedPackets
```

Several notations used in the pseudocode algorithms are as follows:

1. *threshold* : packet inter-arrival time threshold
2. *timelyPackets* : number of packets with inter-arrival times within the threshold
3. *delayedPackets* : number of packets with inter-arrival times exceeding the threshold
4. *packetTime* : packet arrival time
5. *packetsReceived* : number of packets reaching the receiver

The algorithm basically uses packet inter-arrival times in order to identify whether each received packet can be effectively used by the client application. Each packet with inter-arrival time greater than the specific threshold is considered as *delayed*. We set here an inter-arrival threshold at 100 ms. Certainly, this can be adjusted depending on the application, user requirements and network delay. Diverse inter-arrival threshold adjustments for a specific scenario should normally produce variable real-time performance results. Since most of our experiments were performed on several MPEG flows, we present the average of the real-time performance of each MPEG flow.

4. Results and Discussion

In order to evaluate the impact of the protocols and the network characteristics on real-time application performance, we conducted several experiments based on 5 distinct scenarios. The basic parameters of each scenario are as described in the previous section. In the first and second scenario, we carried out our experiments simulating both a wired and wireless topology. The rest of the experiments were conducted exclusively over a wireless network (i.e. including handoffs and packet errors).

4.1 MPEG Traffic Friendliness

In the first scenario, we simulated (i) 1 FTP flow (over TCP), (ii) 1 MPEG flow over TCP, and (iii) 1 MPEG flow over UDP, each one of them competing with a number of other FTP flows (1, 10, 20, 40 and 80 flows). We measured the aggregated goodput of all flows in the system except from the first FTP, MPEG over TCP and MPEG over UDP flow, respectively. The data rate of each MPEG flow is adjusted appropriately, so that it can compete fairly with an FTP flow.

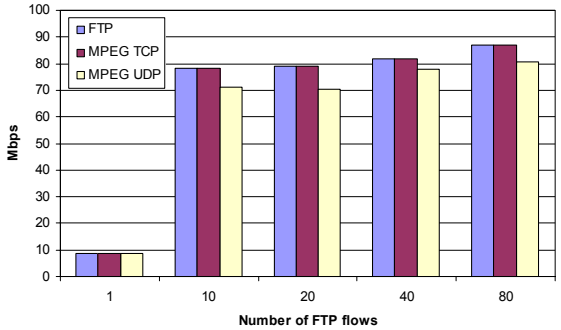

Figure 2. Goodput of remaining FTP flows (Reno/Wired)

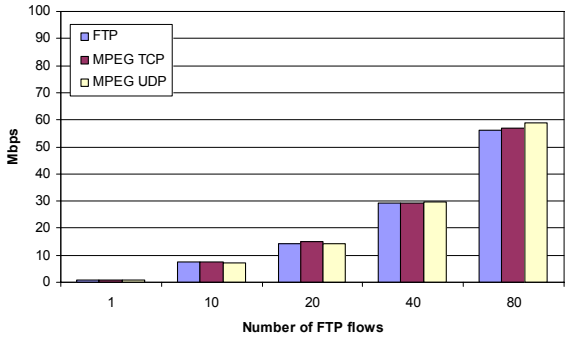

Figure 3. Goodput of remaining FTP flows (Reno/Wireless)

Our purpose here is to demonstrate the impact of the MPEG application on the other applications when they all share the same channel. We conducted the experiments simulating both a wired (Figs. 2, 4) and wireless topology (Figs. 3, 5). We used the TCP variants Reno (Figs. 2, 3) and Vegas (Figs. 4, 5) for the FTP flows and the MPEG TCP flow. The results illustrate the impact of the MPEG application on TCP network traffic. Using the results of the 1st FTP flow as reference, we reach the conclusion that the MPEG flow over TCP does not affect the efficiency of the remaining FTP flows. However, we observe that the MPEG flow over UDP has a negative impact on the other flows in each case, as their aggregated goodput is decreased. This observation is inline with our expectations, considering the lack of any back-up policies of the UDP protocol. While TCP backs off as contention increases, UDP keeps almost a steady transmission rate. Consequently, *"greedy"* UDP commonly allocates excessive network resources and prevents interfering TCP flows from obtaining a fair share of the link. Therefore, any application sharing the same channel with UDP flows is in an unfavorable situation. This phenomenon is more intense in situations of excessive contention (Figs. 2, 4).

In the context of wireless network (Figs. 3, 5), both TCP protocols tested follow a rather conservative policy, because of the unnecessary congestion-oriented responses to the handoff events and packet errors. As a result, TCP protocols have limited efficiency in such environments. However, Vegas achieves higher transmission rates than Reno, due to its sophisticated congestion avoidance mechanism.

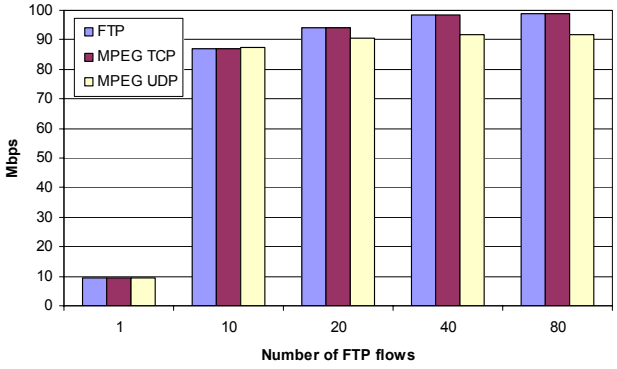

Figure 4. Goodput of remaining FTP flows (Vegas/Wired)

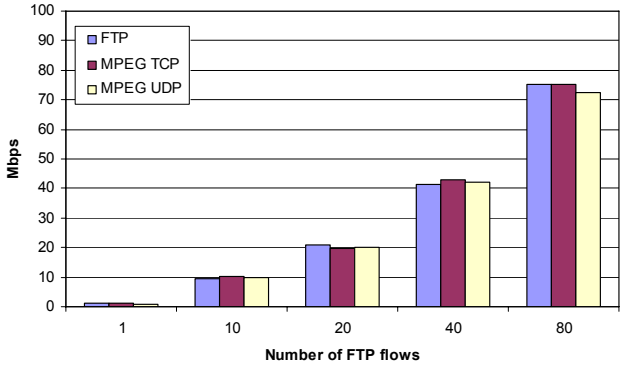

Figure 5. Goodput of remaining FTP flows (Vegas/Wireless)

4.2 TCP vs. UDP

This scenario includes the simulation of (i) N MPEG flows over TCP, and (ii) N MPEG flows over UDP, where N is set to 1, 10, 20, 40 and 80, successively. Here, apart from the two referenced TCP versions (i.e. Reno and Vegas), we additionally used TCP Westwood and TCP Westwood+. Our primary goal is to evaluate the impact of these protocols on real-time application QoS, when there are exclusively MPEG flows in the system. Our metrics consist of system goodput, the average of the Real-time Performance of each MPEG flow and Fairness Index.

We carried out the experiments using both the wired and wireless topology. However, the data rate of the MPEG flows was adjusted separately on the wired and wireless network, so that one MPEG flow over TCP should have approximately the same goodput with a single MPEG flow over UDP. Since the performance of TCP varies between wired and wireless environments, we finally used two different adjustments for the MPEG rate for each simulated topology.

Figs. 6, 7 illustrate that TCP protocols occasionally achieve higher goodput rates than UDP (mainly in the wired experiments). Generally, this difference increases along with the number of flows. Either over TCP or UDP congestion episodes occur. Since TCP protocols incorporate congestion control mechanisms, they deal with congestion more effectively. UDP with the absence of such mechanisms is unable to control the congestion. The MPEG streaming application transmits packets over UDP ignoring the prevailing network conditions and consequently induces a large number of packets drops (Figs. 12, 13). This phenomenon is reflected in the real-time performance results (Figs. 8, 9), where UDP has inadequate performance, especially when the number of flows exceeds 40. However, in the wireless experiments UDP is generally more efficient. This phenomenon is not actually the outcome of the improved performance of UDP rather than the limited efficiency of TCP due to its back-off policies, as discussed in the previous scenario.

Furthermore, UDP does not compete fairly with the rest of UDP flows, since UDP is not designed to anticipate fairness. This observation is more apparent in the wired results (Fig. 10), since the data rate of each MPEG flow is higher in the wired than in the wireless experiments. Generally, a series of similar experiments with diverse MPEG data rates indicate that UDP exhibits less fair behavior, as the MPEG rate increases.

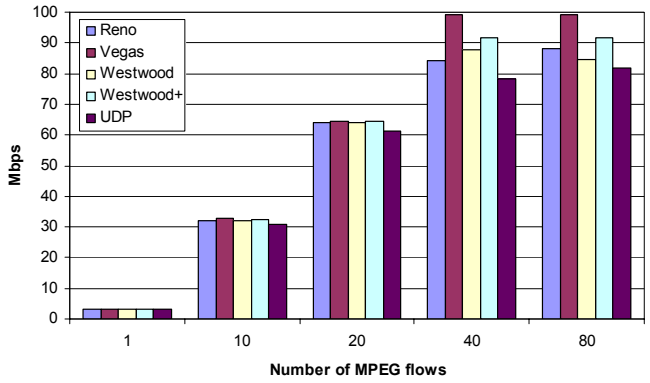

Figure 6. System goodput (Wired)

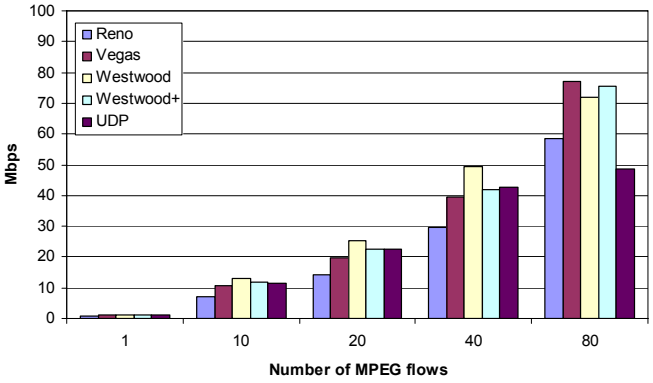

Figure 7. System goodput (Wireless)

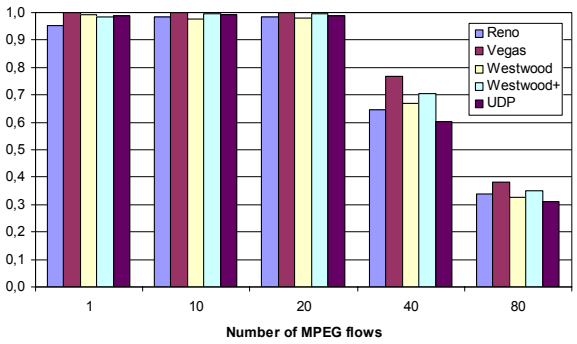

Figure 8. Average Real-Time Performance (Wired)

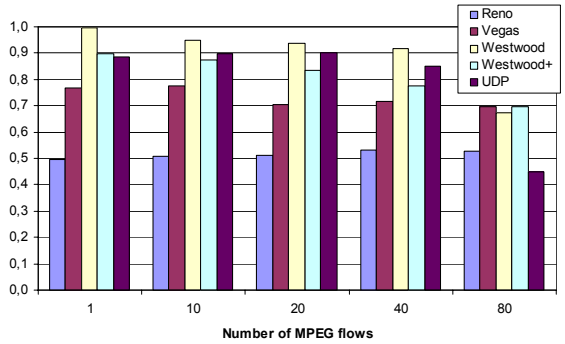

Figure 9. Average Real-Time Performance (Wireless)

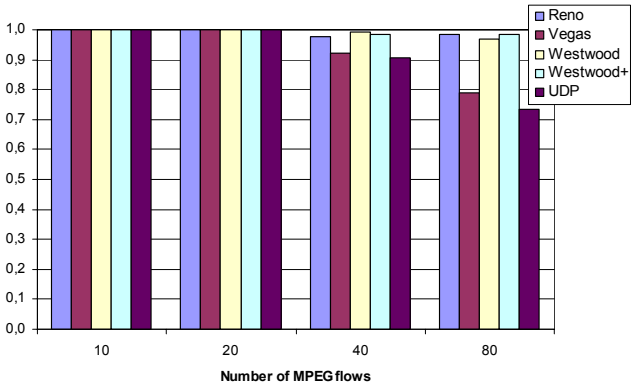

Figure 10. Fairness Index (Wired)

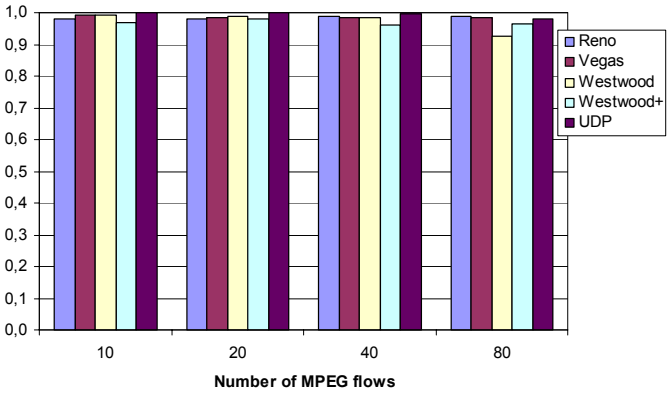

Figure 11. Fairness Index (Wireless)

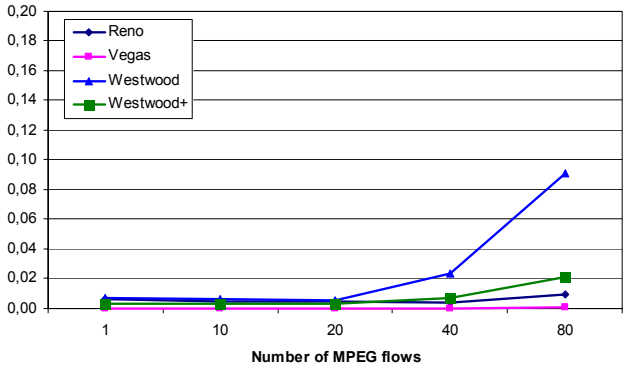

Figure 12. Packet Loss Rate (Wired)

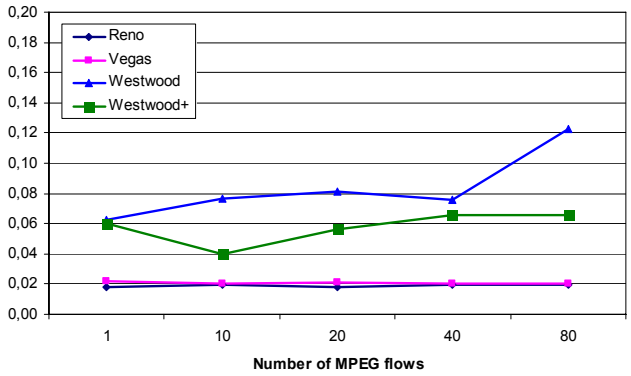

Figure 13. Packet Loss Rate (Wireless)

A very interesting outcome of the associated results is the performance of TCP-Friendly protocols Westwood and Westwood+. This family of protocols use smooth window adjustments which appear to be inline with real-time application requirements. However, during excessive congestion the conservative policy of TCP-Friendly protocols results in higher packet loss rates (Figs. 12, 13). Westwood and especially Westwood+ are more effective in a wireless environment, due to their slight backward window adjustments to handoffs and packet errors (Figs. 7, 9). Authors in [4] also showed that TCP Westwood obtains a higher goodput that TCP Reno and Vegas through simulations over wireless links. However, both Westwood and Westwood+ are outperformed by TCP Vegas and occasionally by TCP Reno in the wired experiments (Figs. 6, 8). Furthermore, Fig. 10 illustrates that TCP Vegas trades fairness for a remarkable performance; its congestion avoidance mechanism can not handle bandwidths sharing efficiently. Although in Fig. 11 Vegas seems to exhibit a fair behavior, similar experiments over wireless links with higher MPEG data rates revealed that the protocol does not achieve fairness, when contention is increased and bandwidth availability is limited.

4.3 MPEG Performance vs. Packet Errors

In this scenario we simulated (i) N MPEG flows over TCP, and (ii) N MPEG flows over UDP, where N is set to 10, 20 and 40 successively. We used TCP Reno and TCP Vegas in conjunction with UDP. Our purpose here is to evaluate the impact of variable packet error rates on real-time application performance. Therefore, the experiments were conducted on a wireless topology featuring exclusively packet errors and not handoffs. We measured the system goodput and the average real-time performance of all the MPEG flows. Apart from the variable packet error rate adjustments (PER: 0.01 - 0.05), we carried out an experiment without packet errors and used it as a reference.

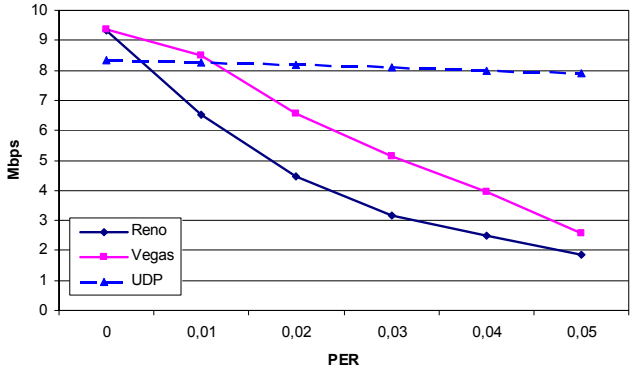

Figure 14. System goodput (10 flows)

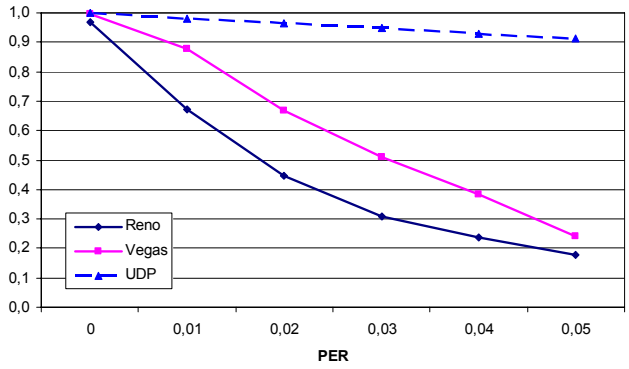

Figure 15. Average Real-Time Performance (10 flows)

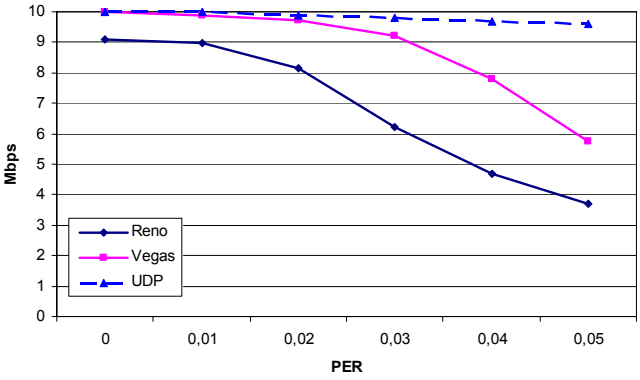

Figure 16. System goodput (20 flows)

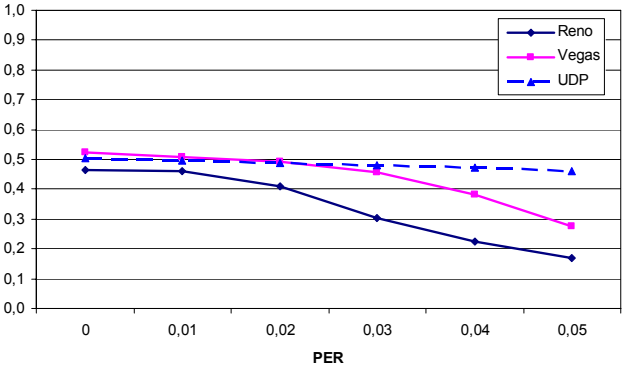

Figure 17. Average Real-Time Performance (20 flows)

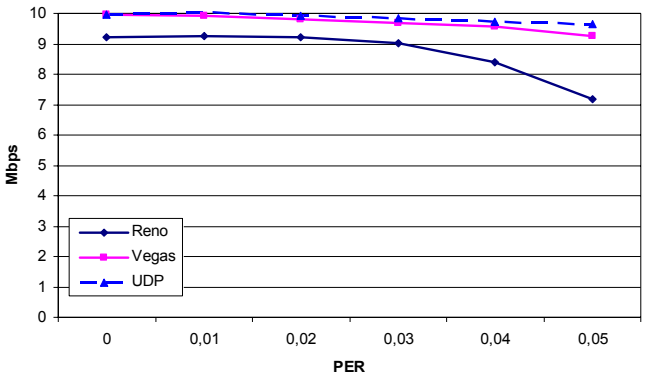

Figure 18. System goodput (40 flows)

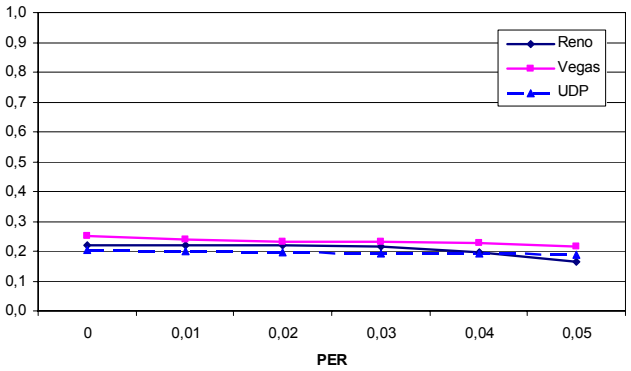

Figure 19. Average Real-Time Performance (40 flows)

Based on our results we reach some useful conclusions about the behavior of these protocols on the specific network environment. Both TCP protocols are quite sensitive to diverse PER adjustments, since they do not have an error classification mechanism and hence, recovery can not be responsive to the nature of error. Consequently, Vegas and especially Reno invoke congestion-oriented responses to the wireless errors introduced in our scenario. As a result, their congestion control mechanisms unnecessarily reduce the congestion window. As the error rate increases, the system goodput naturally decreases. Along with goodput decrease, application performance is degraded (Figs. 15, 17, 19). Goodput and real-time performance degradation is more intense in the situation of 10 and 20 flows, since the relatively high transmission rate (due to low contention) is diminished by the reduction of the congestion window. In all experiments, we observe that Vegas is more efficient that Reno, due to its effective bandwidth estimation algorithm. Similar findings have been reported in [4], where Vegas outperforms Reno in experimental scenarios over wireless links with diverse packet error rates. On the other hand, UDP is only slightly affected by PER adjustments, as the protocol transmits at a steady rate and does not account for any type of error.

4.4 MPEG Performance vs. Buffer Size

Based on the experiments of this scenario we investigate the impact of diverse buffer size adjustments on the protocol behavior and consequently on real-time application performance. Each experiment simulates (i) N MPEG flows over TCP, and (ii) N MPEG flows over UDP, where N is set to 20 and 40, successively. We used the wireless topology with 5 handoffs and packet error rate set to 0.01. We repeated each experiment adjusting the size of the bottleneck buffer at 20, 30, 50 and 80 packets successively. Apart from the corresponding goodput and real-time performance results (Figs. 20-23), we present traces of queue-length for both TCP protocols and UDP in the situation of 40 flows (Figs. 24-26). Each figure includes 3 traces for buffer sizes of 30, 50 and 80 packets.

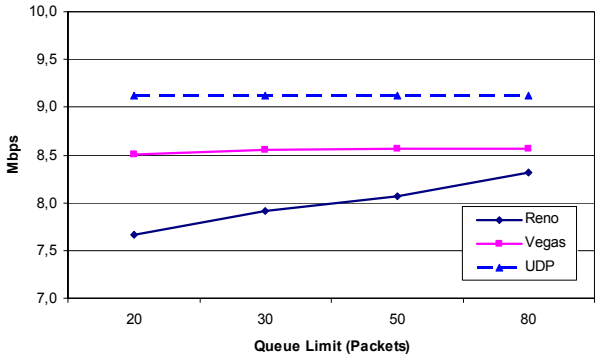

Figure 20. System goodput (20 flows)

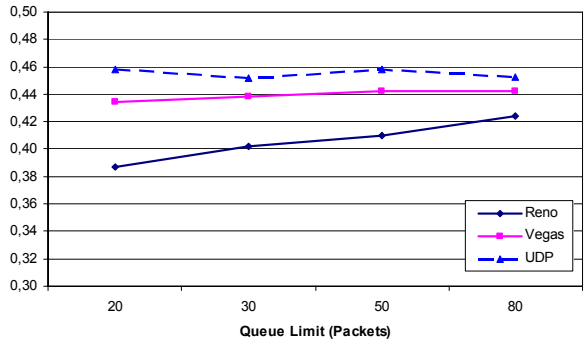

Figure 21. Average Real-Time Performance (20 flows)

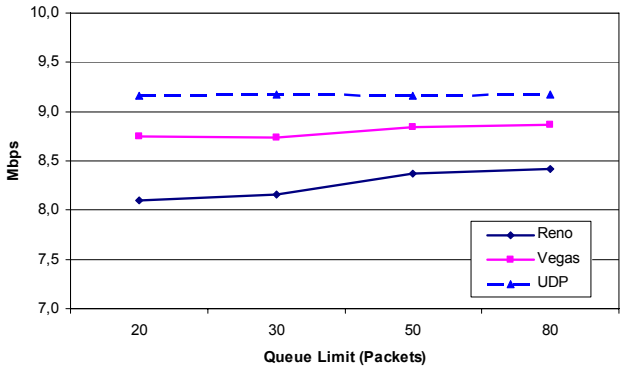

Figure 22. System goodput (40 flows)

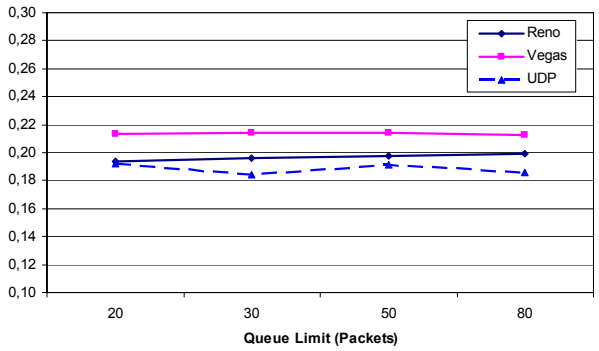

Figure 23. Average Real-Time Performance (40 flows)

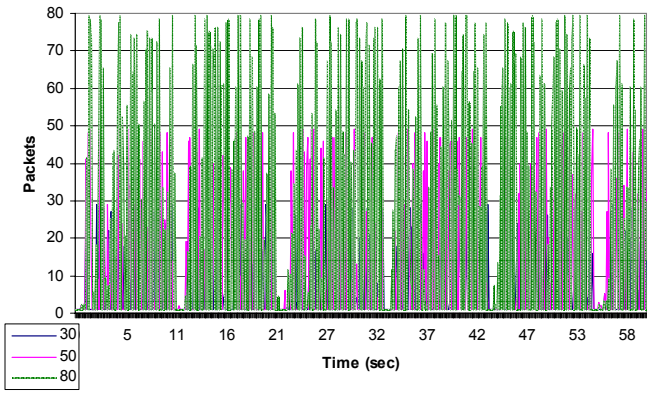

Figure 24. Queue Length (Reno, 40 flows)

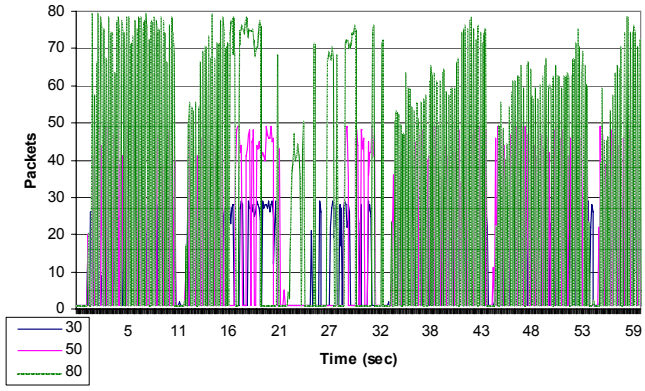

Figure 25. Queue Length (Vegas, 40 flows)

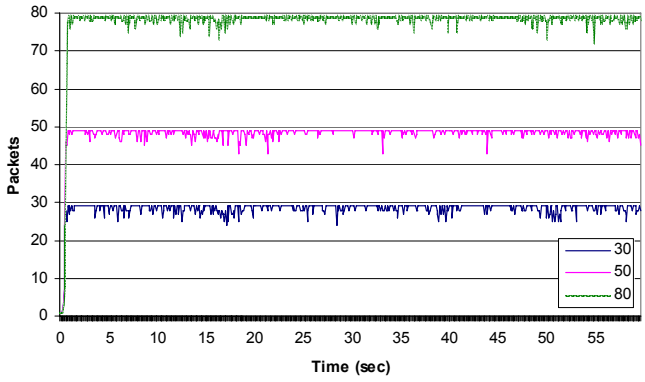

Figure 26. Queue Length (UDP, 40 flows)

TCP Reno is the most sensitive protocol to buffer size adjustments achieving notable gains from a relatively large buffer (Figs. 20, 22). More precisely, in the situation of 20 flows the average queue-length of TCP-Reno increases from 8.8 to 16.1 packets, when the queue limit is adjusted from 50 to 80 packets (Table 1). On the contrary, TCP Vegas does not achieve any goodput gains, since the average queue-length is 7.6-7.7 packets for buffer adjustments of 30 packets and above. However, in the situation of 40 flows both TCP protocols take advantage of larger buffers, improving their goodput performance (Figs. 22, 24, 25). Table 2 indicates that the average queue-length increases along with the size of buffer for both Reno and Vegas. Despite the reported gains in goodput rates, Figs. 21 and 23 indicate minor or no improvements in the real-time performance by increasing the buffer size. The only exception is the situation of 20 Reno connections, where

we observe notable real-time performance gains (Fig. 21). Indeed, increasing the size of the bottleneck buffer usually improves goodput performance; however, from the perspective of real-time performance the reported gains are occasionally diminished by the increasing queuing delays. Figs. 27, 28 illustrate that large buffers result in an increased number of delayed packets, especially in the situation of 40 flows.

Queue Limit (Packets)	**20**	**30**	**50**	**80**
TCP Reno	3.0	5.4	8.8	16.1
TCP Vegas	5.5	7.6	7.7	7.7
UDP	16.9	27.2	46.6	75.7

Table 1. Average Queue Length in packets (20 flows)

Queue Limit (Packets)	**20**	**30**	**50**	**80**
TCP Reno	4.7	7.4	14.3	24.4
TCP Vegas	7.1	11.3	19.0	29.2
UDP	18.5	28.3	48.1	77.8

Table 2. Average Queue Length in packets (40 flows)

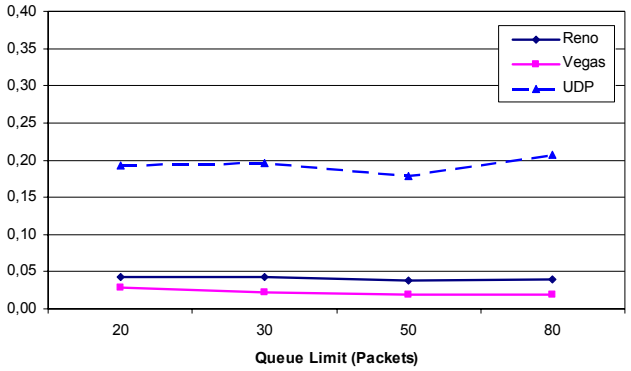

Figure 27. Delayed Packets Rate (20 Flows)

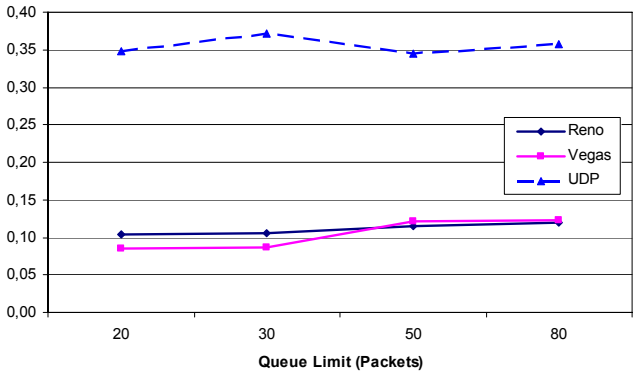

Figure 28. Delayed Packets Rate (40 Flows)

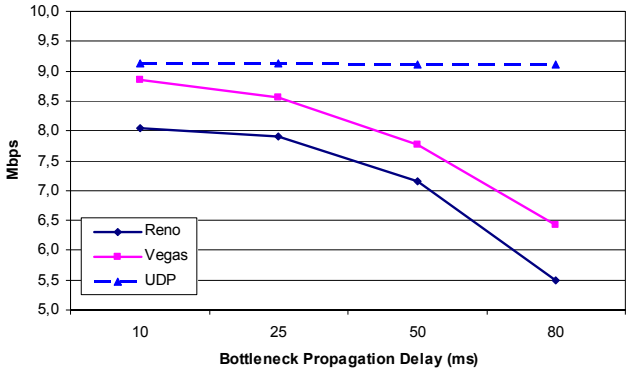

Figure 29. System goodput (20 flows)

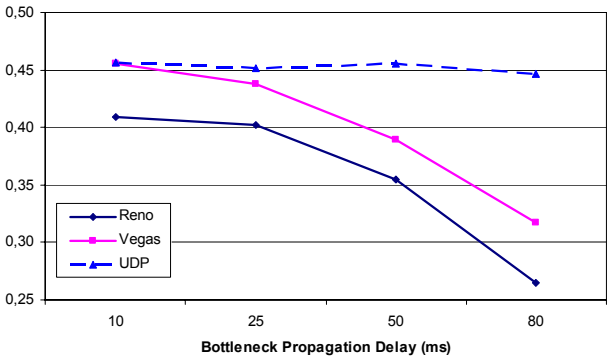

Figure 30. Average Real-Time Performance (20 flows)

What may appear as a paradox comes from the UDP behavior in this scenario. The efficiency of UDP remains unaffected despite the larger buffer size. In periods of congestion, UDP results in rapidly growing queues and eventually in bottleneck buffer overflows, since the protocol never reduces its transmission rate. This phenomenon is clearly reflected in the corresponding queue-length trace (Fig. 26), as well as in the average queue-length measurements (Tables 1, 2). On the contrary, a TCP flow adjusts the sending rate according to the prevailing network conditions. During a congestion episode, once the buffer is overflowed, congestion control is immediately triggered and the sending window is reduced according to the protocol backward policy. Hence, the buffer starts to drain. To sum up, heavy TCP traffic often results in variable buffer conditions (Figs. 24, 25), while heavy UDP traffic is commonly associated with overflowed buffers (Fig. 26). Therefore, in the situation of UDP, a large buffer does not improve performance, neither prevents congestion episodes. Furthermore, a buffer of considerable size may cause long queuing delays, which increase the number of delayed packets (Figs. 27, 28), and consequently degrade real-time performance (Figs. 21, 23).

4.5 MPEG Performance vs. Link Delay

The last scenario features a class of experiments tackling the impact of propagation delay on real-time application performance. We simulated 20, 40 and 80 MPEG flows over TCP Reno, TCP Vegas and UDP, successively. All the experiments were carried out on the wireless topology (i.e. with 5 handoffs and PER adjusted at 0.01). For each experiment we reconfigured the propagation delay of the bottleneck link varying from 10 ms to 80 ms.

TCP Reno and TCP Vegas are sensitive to propagation delay adjustments, while UDP remains almost unaffected. Our results illustrate that adjusting propagation delay above 25ms significantly degrades real-time performance for both TCP protocols. Increasing delay adjustments induce a growing number of delayed packets for Reno and Vegas (Figs. 35-37).

UDP is slightly affected by propagation delay adjustments, since the protocol delivers a relatively regular number of delayed packets. In all situations, the delayed packets rate of UDP is

remarkably higher than the corresponding rate of the TCP protocols. This observation is more intense for a large number of flows (Figs. 36, 37). Consequently, the performance of a real-time application running over UDP is degraded (Figs. 30, 32, 34).

Furthermore, Figs. 29-34 indicate that the goodput rates of the three protocols tested are not inline with the real-time application performance. Although UDP achieves higher goodput than TCP protocols, in the situation of 40 and 80 flows both TCP versions outperform UDP in terms of real-time performance. The same conclusion is derived by the corresponding goodput and real-time performance results from the previous scenarios. Therefore, goodput is not an accurate metric for the evaluation of real-time application performance.

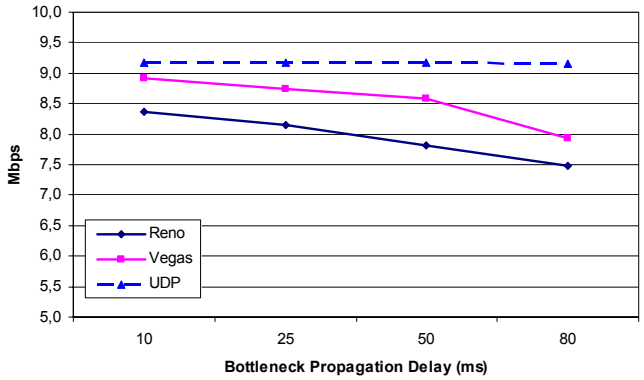

Figure 31. System goodput (40 flows)

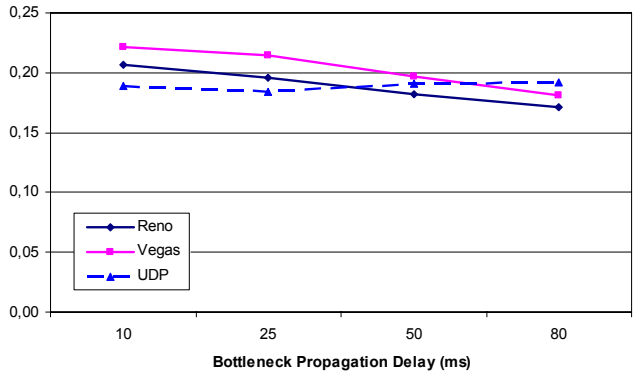

Figure 32. Average Real-Time Performance (40 flows)

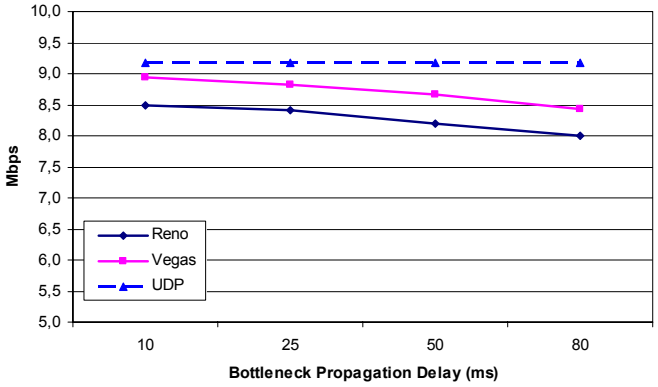

Figure 33. System goodput (80 flows)

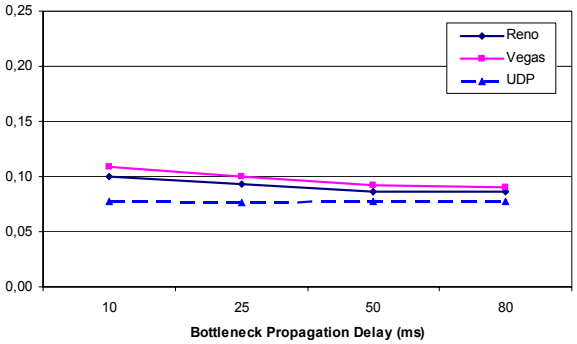

Figure 34. Average Real-Time Performance (80 flows)

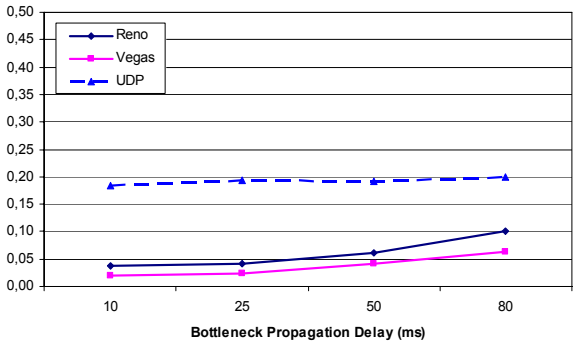

Figure 35. Delayed Packets Rate (20 flows)

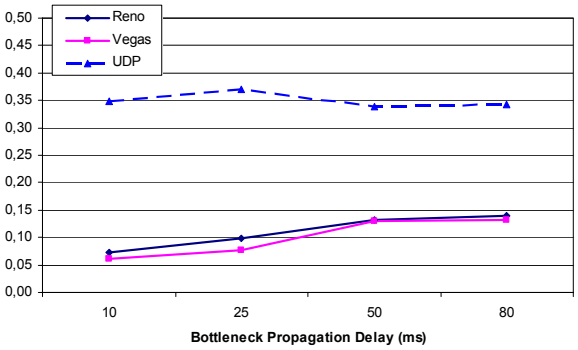

Figure 36. Delayed Packets Rate (40 flows)

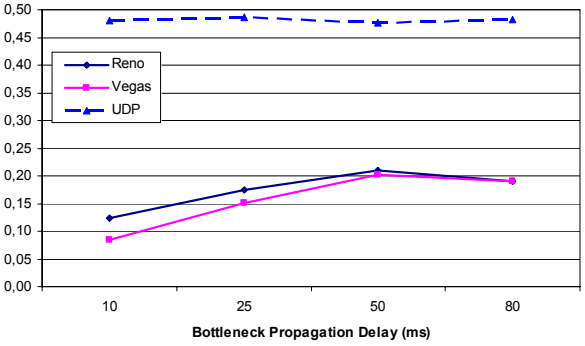

Figure 37. Delayed Packets Rate (80 flows)

6. Conclusions and Future Work

We clarified the network and protocol heterogeneity parameters and we investigated their impact on real-time application QoS. Since goodput is not a valid criterion for the evaluation of real-time application performance, our analysis is mostly based on real-time performance results. However, additional measurements, such as delayed packets and packet loss rate, contributed in the evaluation of the application performance. Along these lines, we outline the most conclusive remarks derived directly from our experiments.

Congestion control is mandatory and protocols which do not incorporate such mechanisms (e.g. UDP) have limited efficiency and potential. UDP is inefficient at periods of congestion and its high packet loss rate degrades the application performance. Because of its free-transmitting mechanism, UDP is not fair to other flows sharing the same channel, no matter if these flows run over TCP or UDP. We also showed that UDP is slightly affected by wireless link errors and adjustments of buffer size and network delay. Since UDP mostly suffers from congestion episodes, it performs efficiently only at conditions of bandwidth availability and low contention.

On the contrary, TCP generally anticipates fairness and is able to control a potential congestion episode adequately. However, not all TCP versions achieve equivalent performance. We conducted most of our experiments over TCP Reno, as a reference for congestion control and over TCP Vegas, as a reference for congestion avoidance. TCP Vegas takes advantage of its sophisticated mechanism estimating the available bandwidth and hence adjusting its transmission rate accordingly. Consequently, Vegas often avoids congestion successfully and eventually achieves higher transmission rates than standard TCP versions, especially during congestion episodes. However, inline with most TCP protocols, Vegas is not able to distinguish the nature of error and its congestion-oriented responses to wireless errors and operations (e.g. handoffs) result in wasteful backward window adjustments. Furthermore, the efficiency of Vegas comes at a cost: the protocol does not achieve a fair behavior.

On the other hand, TCP Reno is based on *"blind"* increase/decrease windows mechanisms that dynamically exploit bandwidth availability, without relying on precise measurements of current conditions. Hence, it generally demonstrates limited efficiency in the context of real-time application performance. More precisely, we showed that the efficiency of Reno and the associated real-time performance are drastically affected by unfavourable network conditions, such as lossy links, small bottleneck buffers and long network delays. On the contrary, TCP Vegas exhibits less sensitivity to diverse network conditions, providing a relatively balanced performance which favors real-time traffic.

We also noted that TCP-Friendly protocols do not always deliver the improved real-time performance their designers imply, due to their smooth window adjustments at periods of congestion. They are unable to effectively recover from excessive congestion events resulting in increased packet drops. However, they invoke more accurate responses to wireless errors by reducing the congestion window more gently than standard TCP. Consequently, TCP-Friendly protocols are generally more efficient in wireless environments. The bandwidth estimation algorithms of TCP Vegas and TCP Westwood/Westwood+ do not always obtain accurate estimates, especially over wireless links [4]; yet they are more effective than "blind" increase/decrease windows mechanisms (e.g. TCP Reno) which rely on specific events triggered by violated thresholds.

Reviewing our discussion it is obvious that current network and protocol support for real-time applications is inadequate. Most existing protocols are efficient only under certain network conditions. A combined effort that guarantees both real-time performance and TCP traffic friendliness has yet to be presented. Along these lines, future work will be focused on the design of a protocol which will provide improved support for this type of applications. Specifically, we intend to design a partially reliable protocol, as an extension of current TCP-Real.

References

1. A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", In Proc. of *15th International Conference on Distributed Computing Systems*, Vancouver, Canada, June 1995

2. H. Balakrishnan, S. Seshan, E. Amir and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks", In Proc. of *ACM MOBICOM '95*, Berkeley, California, USA, November 1995
3. 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), pp. 1465-1480, October 1995
4. A. Capone, L. Fratta, F. Martignon, "Bandwidth Estimation Schemes for TCP over Wireless Networks", *IEEE Transactions on Mobile Computing*, 3(2), pp. 129-143, 2004
5. D. Chalmers, M. Sloman, "A Survey of Quality of Service in Mobile Computing Environments", *IEEE Communication Surveys*, 2(2), pp. 2-10, 1999
6. D. Chiu, R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks", *Journal of Computer Networks*, 17(1), pp. 1-14, June 1989
7. D. D. Clark, S. Shenker, L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism", In Proc. of *SIGCOMM '92*, pp. 14-26, August 1992
8. R. Doshi and P. Cao, "Streaming Traffic Fairness over Low Bandwidth WAN Links", In Proc. of *3rd IEEE Int/nal Workshop on Internet Applications*, San Jose, California, June 2003
9. S. Floyd, M. Handley and J. Padhye, "A Comparison of Equation-based and AIMD Congestion Control", May 2000, URL: http://www.aciri.org/tfrc/
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. T. Goff, J. Moronski, D. Phatak and V. Vipul Gupta, "Freeze-TCP: A true end-to-end Enhancement Mechanism for Mobile Environments", In Proc. of *IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000
12. L. A. Grieco and S. Mascolo, "TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks", In Proc. of *7th Int/nal Workshop on Protocols for High-Speed Networks*, Berlin, Germany, April 2002
13. A. Grieco, S. Mascolo and R. Ferorelli, "Additive-Increase/Adaptive-Decrease Congestion Control: a Mathematical Model and Its Experimental Validation", In Proc. of *IEEE ISCC*, Taormina, Italy, July 2002
14. U. Hengartner, J. Bolliger, and T. Cross, "TCP Vegas Revisited", In Proc. of *IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 2000
15. V. Jacobson, "Congestion avoidance and control", In Proc. of *ACM SIGCOMM '88*, Stanford, USA, August 1988
16. L. Mamatas and V. Tsaoussidis, "Protocol Behavior: More Effort, More Gains?", In Proc. of *15th IEEE PIMRC*, Barcelona, Spain, September 2004
17. A. Matrawy, I. Lambadaris and C. Huang, "MPEG4 Traffic Modeling using the Transform Expand Sample Methodology", In Proc. of *4th IEEE Int/nal Workshop on Network Appliances*, Gaithersburg, Maryland, USA, 2002
18. S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", In Proc. of *MOBICOM '01*, Rome, Italy, July 2001
19. B. Melamed, "An Overview of TES Processed and Modeling Methodology", *Performance Evaluation of Computer and Communication Systems*, pp. 359-393, 1993
20. K. Ramakrishnan, S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP", *RFC 2481*, January 1999
21. H. Sawashima, Y. Hori, H. Sunahara, and Y. Oie, "Characteristics of UDP Packet Loss: Effect of TCP Traffic", In Proc. of *INET '97*, Malaysia, June 1997
22. P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks", In Proc. of *ACM MOBICOM '99*, Seattle, Washington, USA, August 1999
23. V. Tsaoussidis, H. Badr, "TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains", In Proc. of *8th Int/nal Conference on Network Protocols*, Osaka, Japan, November 2000
24. V. Tsaoussidis and I. Matta, "Open issues on TCP for Mobile Computing", *Journal of Wireless Communications and Mobile Computing*, Wiley Academic Publishers, 2(1), pp. 3-20, Feb. 2002

25. V. Tsaoussidis and C. Zhang, "TCP Real: Receiver-oriented congestion control", *Journal of Computer Networks*, 40(4), pp. 477-497, November 2002
26. V. Tsaoussidis and C. Zhang, "The dynamics of responsiveness and smoothness in heterogeneous networks", *IEEE Journal on Selected Areas in Communications*, 23(6), pp. 1178-1189, June 2005
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*, Osaka, Japan, November 2000
29. C. Zhang and V. Tsaoussidis, "TCP Real: Improving Real-time Capabilities of TCP over Heterogeneous Networks", In Proc. of the *11th IEEE/ACM NOSSDAV*, Port Jefferson, New York, USA, June 2001