The Journal of Computer Networks (COMNET), 2002

TCP-Real: Receiver-oriented Congestion Control

V. Tsaoussidis and C. Zhang
College of Computer Science, Northeastern University
Boston, MA 02115, USA
{czhang, vassilis}@ccs.neu.edu

Abstract. We introduce a receiver-oriented approach to congestion control, demonstrated by an experimental protocol, TCP-Real. The protocol allows for a measurement-based transmission strategy, which complements the "blind" increase/decrease window adjustments. Owing to its design, the protocol displays an inherent property to produce comprehensive dynamics in heterogeneous environments with wired or wireless networks and delay-sensitive or -tolerant applications. TCP-Real controls congestion as standard TCP does. However, its receiver-oriented nature and its "wave" communication pattern allow for two amending mechanisms: (i) Congestion avoidance, which reduces unnecessary transmission gaps that hurt the performance of time-constrained applications, and (ii) Advanced error detection and classification, which designates recovery tactics responsive to the nature of the errors, thereby enhancing the protocol performance over wireless links or asymmetric paths.

We detail the protocol mechanisms and specification and we report extensively on the comparative fairness and efficiency evaluation of standard TCP, TCP-Real, and TCP-friendly protocols for both delay-tolerant and -sensitive applications and in both wired and wireless networks.

Keywords: Congestion Control, TCP, TCP-friendly Protocols, Real-time Protocols, Mobile/Wireless Networks

1. INTRODUCTION

Transmission control of reliable protocols, as exemplified by TCP [21], is based on somewhat "blind" increase/decrease window mechanisms that dynamically exploit the bandwidth availability of a communication channel, without relying on precise measurements of current conditions, but rather on specific events triggered by violated thresholds. An inherent characteristic of a "blind" window increase strategy is the natural cause of congestion and the subsequent need for error recovery. Congestion is detected by missing segments [1] and the increase/decrease strategy is designed precisely to maintain a dynamic balance of the protocol's transmission rate during periods of congestion and bandwidth availability. This balance is achieved by the Additive Increase/Multiplicative Decrease (AIMD) algorithm proposed in [7] through graduated adjustments upwards, when conditions permit, or downwards when a packet loss is detected.

TCP's strategy of rapid backward and graduated upward adjustments hurts the performance of TCP-based applications and damages indeed the delay-sensitive applications. Research efforts have been concentrating on two directions: (i) To avoid the damage of false congestion-oriented responses. The wireless network domain was recently the focus of attention, mainly due to its distinctive error characteristics. (ii) To avoid significant damage during congestion. Although some damage is inevitable during congestion, a smooth backward adjustment has the potential to enhance application performance. TCP-friendly protocols achieve that by trading off the parameter β, which reflects the multiplicative factor of TCP(α, β), with the parameter α, which reflects the additive increase factor. In other words, they reduce the window size less during congestion, but then, they apply a moderated increasing rate.

A combined effort that enables both smooth adjustments and network-driven error recovery has yet to be presented. However, this combination cannot be logistic. That is, a synthesis of mechanisms that target those specific problems will not necessarily produce combined dynamics but it may produce *different* dynamics. For example, a design that calls for more aggressive recovery and smoother backward adjustments could violate the established framework of fairness and, occasionally, fail to deal with congestion effectively.

The receiver-oriented research framework that we discuss here is implemented in the form of an experimental protocol, namely TCP-Real[1]. The desirable behavior for this protocol is precisely to demonstrate efficiency in heterogeneous environments with wired or wireless networks and delay-sensitive or -tolerant applications. Receiver-oriented error control incarnates the property of the receiver to determine with better accuracy the data delivery rate and the potential level of data loss. This abrogates the impact of false assessments at the sender due to lost or delayed acknowledgements. Our approach to efficiency, however, relies also on the potential of

[1] Initial results of TCP-Real are presented in [32]

measurement-based transmission control. Estimating the level of contention allows for early measures towards congestion avoidance, which, in turn, reduces the damaging transmission gaps. Accurate measurements also enable a basic error classification. For example, a drop associated with high-contention and queuing delay is potentially due to congestion. Similarly, a missing packet from a data window that is delivered with minimum delay would rather indicate a random bit corruption. Such initial assessments can be further elaborated and confirmed by matching the expected and delivered data rates during previous, current, and oncoming RTTs, targeting an enhanced error classification and hence a sophisticated and responsive protocol strategy. Our modifications reflect such a protocol strategy enhanced with new recovery tactics. However, the semantics of TCP are not violated and, indeed, the established goals of fairness and friendliness are further emphasized. We evaluate three major aspects of the protocol behavior: its efficiency, fairness and friendliness. Efficiency is considered on the basis of the application requirements and the underlying network characteristics. We compare our protocol, TCP-Real, with the standard TCP and TCP-friendly protocols.

We organized the paper as follows: In section 2 we discuss the limitations of TCP from the perspective of the application requirements and network characteristics, emphasizing on the conditions under which friendliness can be achieved. We detail our design of receiver-oriented congestion control in Section 3. There, we also describe the justification and implementation of our experimental protocol, TCP-Real. We justify our testing methodology and evaluation procedure in section 4. Results are presented and discussed in section 5 and our conclusion is summarized in section 6.

2. A FRAMEWORK FOR POTENTIAL IMPROVEMENTS

The application requirements and the limitations of congestion control circumscribe a framework for potential improvements. We identify five distinct cases of interest, which have been discussed in the recent literature.

1 Additive Increase leads naturally to congestion, which, in turn, degrades throughput for two reasons: (i) Routers need time to recover even from a transitory congestive collapse and sources need time to detect and retransmit missing packets [17] (ii) Congestion control is triggered upon congestion; the window is adjusted backwards and the timeout is extended, which in turn degrades the protocol's capability to detect and exploit error free conditions and bandwidth availability, respectively [28].

2 Additive Increase is not efficient when the network dynamics encompass rapid changes of bandwidth availability. For example, when short flows that cause congestion complete their task, bandwidth becomes available. Similarly, when a handoff is completed in a cellular network, the entire channel's bandwidth becomes available. A more rapid response is then clearly indicated. [24]

3 Multiplicative decrease causes transmission gaps that hurt the performance of real-time applications that experience jitter and degraded goodput. Furthermore, multiplicative decrease with a factor of ½ or a window adjustment to 2 packets characterizes a rather conservative strategy [9, 13, 16, 24, 30, 31].

4 Error detection lacks an appropriate classification module that would permit a responsive strategy, oriented by the nature of potential errors. That is, when errors appear to be transient due to short-lived flows or random wireless interference, congestion control mechanisms (i.e. timeout extension and multiplicative window adjustment) are triggered unduly. The insufficient error detection/classification may also lead to unfair bandwidth allocation in mixed wired/wireless networks. By default, flows that experience wireless errors do not balance their bandwidth loss with a more aggressive recovery although such behavior could be justified: flows that experienced no losses have occupied extra bandwidth at the router temporarily, when the wireless errors forced some senders to back off. This situation is discussed as an open issue in [28]; we demonstrate the validity of this argument, based on experimental results, in section 5.

5 Source-based decision on the transmission rate, based on the pace of the acknowledgements, necessarily incorporates the potentially asymmetric characteristics (e.g. ack delays and/or losses) of the reverse path [3]. Hence, the sender's transmission rate does not always reflect the capacity of the forward path. This situation has a direct impact on efficiency since available bandwidth remains unexploited.

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 (either directly or indirectly) with real-time application constraints (e.g. [2], [4] – see [28] for a detailed description). Some recent protocols restrict the modifications at the transport level. TCP-Freeze [15] distinguishes handoffs from congestion through the use of the Advertised Window. WTCP [23] implements a rate-based congestion control replacing entirely the ACK-clocking mechanism. TCP-Probing [25] grafts a probing cycle and an Immediate Recovery Strategy into standard TCP, in order to control effectively the throughput/overhead tradeoff. Although TCP-Probing deals effectively with both throughput and energy performance in heterogeneous networks, due to its probing mechanism, it may not satisfy the requirements of

delay-sensitive applications for reduced transmission gaps. TCP-Westwood [6] relies on bandwidth estimation to set the slow start threshold and the congestion window upon three duplicate acknowledgments or timeout. No specific mechanism exists to support error classification and the corresponding recovery tactics for wired/wireless networks, albeit the proposed mechanism appears to be relatively effective over symmetric wireless links due to its efficient congestion control. Naturally, a sender-based strategy does not cancel the TCP deficiency over asymmetric channels. A well-designed, measurement-based version of TCP is TCP Vegas [5]. Vegas defines a BaseRTT to be the minimum of all measured RTTs, and ExpectedRate to be the ratio of congestion window and BaseRTT. The sender measures the ActualRate based on the sample RTTs. If the difference between the ExpectedRate and ActualRate is below a threshold[2] α, the congestion window increases linearly during the next RTT; if the difference exceeds another threshold β, TCP Vegas decreases the congestion window linearly during the next RTT. Vegas does not tackle the problems of wireless errors and asymmetric paths, listed as cases 4 and 5 above. However, from the research perspective of the present work it is important to consider that the authors of Vegas demonstrated effectively that measurement-based window adjustment is a viable mechanism.

A new family of protocols has recently emerged, namely the TCP-friendly protocols [10, 13, 16, 22, 24, 30]. Congestion control was designed based on two fundamental requirements: (i) to achieve smooth window adjustments; this is done by reducing the window decrease factor during congestion, and (ii) to compete fairly with TCP flows, which is achieved by adjusting the increase rate, calculated from a TCP throughput equation. TCP Friendly Rate Control (TFRC) is an equation-based TCP-Friendly congestion control protocol [13]. The sender explicitly adjusts its sending rate as a function of the measured rate of loss events, to compete fairly with TCP. A loss event consists of one or more packet drops within a single round-trip time. The receiver calculates the loss event rate and reports feedback to the sender. The benefit of TFRC is its "gentle" rate regression upon congestion. GAIMD [30] generalizes TCP by parameterizing the congestion window increase value α and decrease ratio β. It provides a balancing relationship between α and β to guarantee friendliness:

$$\alpha = 4(1-\beta^2)/3 \qquad (1)$$

Based on experiments, the authors in [30] have chosen $\beta = 7/8$ as the appropriate value[3] for the reduced the window (i.e. less rapidly than TCP does). For $\beta = 7/8$, (1) gives an increasing value $\alpha = 0.31$.

Obviously, the choice of parameters α and β has a direct impact on the "responsiveness" of the protocols to conditions of increasing contention or bandwidth availability. Indeed, the combined dynamics of α and β from equation (1) exploit an interesting tradeoff: a choice of α that allows for rapid bandwidth consumption (additive increase) is counterbalanced by a friendliness-driven response with multiplicative decrease, rendering the protocol inappropriate for real-time applications.

The differences between TCP, TFRC [4] and GAIMD congestion control lie mainly in the specific values of α and β; their similarities lie 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 here as a specific case of TCP (α,β) with α=1, β=0.5. From the perspective of our classification, TCP-friendly protocols are designed to satisfy specific application requirements such as those outlined in cases 1 and 3 outlined in page 2; however, as we show here, they may exhibit further weakness when bandwidth becomes available rapidly (case 2, page 2). Apparently, the tradeoff between responsiveness and smoothness can be controlled to favor some applications, but it will cause some other damages. Considering the variability of network conditions and the duration of flows, the equation-based recovery may provide weak guarantees for friendliness. We briefly exploit this situation below.

According to [20] and extended by authors of [30], TCP (α,β) throughput can be modeled as:

$$T_{\alpha,\beta}(p, RTT, T_0, b) = \frac{1}{RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}p} + T_0 \min\left(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}p}\right) p(1+32p^2)} \qquad (2)$$

where p is the loss rate; T_0 is the retransmission timeout value; b is the number of packets acknowledged by each ACK; α and β are the congestion control parameters: the sender's window size is increased by α if there is no packet loss in a round-trip time, and the window is decreased by β of current value if there is a loss indication. Observations of the window dynamics and event losses are frequently assumed within a time period of a *congestion epoch.* A congestion epoch is defined in [12] as the time

[2] The thresholds' notation in Vegas match coincidentally the notation of additive increase/multiplicative decrease parameters.

[3] Note that although β was so far called the multiplicative factor, its value determines the size of the window. That is, 7/8 means that the window was reduced by 1/8th.

[4] In fact, TFRC has several other mechanisms that differ from TCP

period that reflects the *uninterrupted growing lifetime of a window*[5]. More precisely, a congestion epoch begins with βW packets, increased by α packets per RTT and reaching a congestion window of W packets, when a packet is dropped. The congestion window is then decreased to βW. Hence, a congestion epoch involves

$$n = \beta/\alpha * W + 1 \quad RTTs \tag{3}$$

TCP-Friendly (α, β) protocols approximate the throughput of standard TCP $(\alpha = 1, \beta = 0.5)$, which means that equation (4) which is derived from (2) (see [13, 30]) provides a rough guide to achieve friendliness.

$$T_{\alpha,\beta}(p, RTT, T_0, b) = T_{1,0.5}(p, RTT, T_0, b) \tag{4}$$

However, having the network or application conditions changing rapidly, friendliness might not be attained. More precisely, based on (3) we conclude that (4) can be achieved at a time tn or later since multiple drops will extend further the time of convergence. Based on (3) we also conclude that the time period required for (4) to hold is in reverse proportion to the number of flows within a fixed bandwidth channel; the smaller the number, the larger the window. Finally, the propagation delay has a direct impact on the time required for TCP(α, β) to reach a full-window size. Practically (and deterministically) this means that for a window of 64KB and an RTT of 100ms, TCP(1, ½) needs at least 3.2 seconds to reach the max window size. Any interrupting event prior to completeness of the lifecycle would impact the dynamics of friendliness.

Our receiver-oriented approach intends to elude the balancing trade of the parameters of additive increase and multiplicative decrease by introducing another parameter, namely γ, which determines the window adjustments during congestion *avoidance*. Congestion avoidance achieves the objective of smoothness by eliminating the frequency of congestive drops; it maintains responsiveness through the *unchanged* additive increase rate. It can be assumed initially that since during congestion the protocol behavior is not exhibiting any differences from standard TCP, the established standards of protocol behavior are not violated. We verify this hypothesis experimentally, in section 5.

3. TCP-REAL: RECEIVER-ORIENTED CONGESTION CONTROL

3.1 Protocol Strategy and Justification

In order for the receiver to observe accurately the level of contention and/or packet loss, a limpid communication pattern with the sender is needed. We call this pattern a "wave" and it was introduced in [26, 27]. A wave consists of a number of fixed-sized data segments sent back-to-back, matching the inherent characteristic of TCP to send packets back-to-back. By default, a TCP wave is effectively the congestion window with an additional attribute: its size is published to both peers. However, since the transmission pattern of TCP is also constrained by its ACK-clocking mechanism, when packets or acknowledgements are delayed or lost, the congestion window may be partitioned into smaller groups of back-to-back packets; these groups correspond to distinct waves. In effect, the wave pattern enables the marking of those parts of the congestion window that are sent back-to-back. The pattern in its own right has the potential to cancel the impact of the hidden assumption that packets within a congestion window are actually sent back-to-back. The assumption is made implicitly each time the receiver attempts to monitor the network dynamics based on packet dispersion, without shaping the traffic at the sender prior to transmission.

Having a well-known pattern of data exchange that permits both peers to know the size of the wave enables the receiver to estimate the level of contention/congestion. The inter-packet gap and the wave delivery time can be sources of information that indicate the presence of contention within the network, which in turn allows for error classification and appropriate recovery. More precisely, the receiver computes the data-receiving rate of a wave (see section 3.3 for details), which is determined by the interleaving patterns of packets from different flows at the bottleneck router. The lower the perceived rate, the higher the multiplexing level at the bottleneck and the smaller the congestion window suggested, and vice versa. During congestion, the data-receiving rate might fluctuate dramatically due to packet drops that occur at the bottleneck router. However, if a packet drop is due to a wireless error, the data-receiving rate shall not be affected, and the corresponding gap of the missing packet can be both observed and estimated since the wave size is known to the receiver. These observations are used herein to implement an *ad interim* tactic to distinguish transient random errors from congestion; the overall strategy is verified at the next RTT by comparing the perceived rate and the previous rate. The reasoning behind the *interim* nature of the recovery tactic is the associated uncertainty of the error detection and rate estimation. Although a large window of data provides a good sample for measurements of the receiving rate and a better possibility to observe at least one inter-packet gap, a ruling based on a small window is exposed to potential errors and coincidences. The protocol's default behavior in the presence of uncertain detection is dominated by the standard AIMD. Presently, we are investigating the potential of a recovery strategy in association with the estimated error margin of detection and the specific

[5] This is the interpretation of the notion of congestion epoch of the present authors.

```
01    Whenever the sender receives an ACK:
02
03    if  (the ACK contains data-receiving rate of the next wave)
04          if (the wave size is larger than 4 packets){
05                previous_rate  =  current_rate;
06                current_rate  =  the data-receiving rate in the ACK
07                numberOfRateUpdates ++;
08                gamma_applied = 0;
10          }
11          else {
12                /*  ignore the measured rate since the wave size is too small and  the rate
13                    measurement can be noisy. Apply Standard Additive Increase.
14                */
15                previous_rate  =  current_rate;
16          }
17
18    if  (the ACK acknowledges new data) {
19          rate_ratio =  current_rate  / previous_rate;
20          if (rate_ratio >= 1.0)
21                cwnd +=  1 / cwnd;          // additive increase
22          else   if (rate_ratio >= 0.8)
23                cwnd +=  0;
24          else                              // measurement-based congestion avoidance
25          if (gamma_applied == 0){
26                cwnd  -= 1 / 8;             // apply multiplicative decrease with γ
27                gamma_applied = 1;          // apply γ at most once every RTT
28          }
29    }
```

where the cwnd is the congestion window in number of packets, and *numberOfRateUpdates* is the number of data-receiving rate updates since the last congestion window backoff.

Figure 1. TCP Real: Congestion Avoidance Algorithm

characteristics of the error pattern (i.e. frequency, duration etc.).

The improved error classification and contention/congestion estimation enables an error recovery strategy based on the nature of the error. The receiver can decouple the packet losses from the window adjustments: whenever the receiver observes data delivery with low jitter and high receiving rate, missing packet(s) indicate a random (perhaps wireless) error. Therefore, unnecessary congestion-oriented back-offs are avoided: whenever contention boosts up, the receiver instructs the sender to smoothly adjust its rate backwards, prior to congestion. Hence, congestion avoidance could eliminate the number of drops and the subsequent actions of congestion control. Clearly, receiver-oriented control could also overcome the problems of asymmetric paths – the receiver could rule on the need for rate adjustment regardless of the reverse-path characteristics.

3.2 Congestion Avoidance and Control Algorithms of TCP-Real

The congestion avoidance algorithm of TCP-Real is shown in Figure 1. The receiver measures the data-receiving rate of each wave and attaches the result to its ACKs, directing the transmission rate of the sender. When new data is acknowledged and the congestion window is adjusted, the current data-receiving rate is compared against the previous one. If there is no receiving rate decrease, the congestion window is increased by 1 MSS every RTT (α=1). If the magnitude of the decrease is small, the congestion window remains temporarily unaffected. If the magnitude of the rate decrease is high, the sender reduces the congestion window multiplicatively by γ where γ selected 1/8 for our experiments, although it can be adaptive to the detected conditions. The congestion avoidance algorithm of TCP-Real is shown in Figure 1. Note that here the congestion window size is decreased prior to congestion and therefore the value of γ corresponds to an additional parameter. When the wave size is too small (less than 4 segments[6]), the measured data-receiving rate can only reflect transient behavior and is ignored: the congestion window is increased by 1 every RTT.

[6] We determined the threshold of 4 empirically.

```
01      Whenever 3-dacks are received, or a timeout occurs, do the following:
02
03      congestion = 0;  // a boolean indicating whether or not congestion occurs
04
05      if (( highest_rate  > lowest_rate * c ) AND
06         ( current_rate  - lowest_rate <  highest_rate  - current_rate )) {
07              congestion = 1;
08      }
09
10      if  (numberOfRateUpdates < threshold)
11              congestion = 1;
12
13     /*  if  the variable congestion  equals to 0,  then this is a  wireless error (do nothing).
14         Otherwise congestion occurs. */
15      if (congestion)
16              slowdown();          // backoff. Slow start (if timeout) or fast recovery (if 3dacks)
```

where *numberOfRateUpdates* is the number of data-receiving rate updates since last congestion window backoff. Note *numberOfRateUpdates* is not incremented if the wave size is not larger than 4 packets, shown in Figure 1.

Figure 2. TCP Real: Congestion Control Algorithm

In each epoch, the sender keeps a record of the lowest and highest data-receiving rate. When a timeout or 3-dack event occurs, the sender evaluates two conditions shown in lines 5 and 6 of Figure 2. If both are satisfied, i.e. if the magnitude of receiving-rate fluctuation is high (higher than *c*, where *c* is a constant set at 3 in our experiments) and the current data-receiving rate is relatively low (i.e. the multiplexing level at the bottleneck is high), congestion control is activated. The sender reduces the congestion window as TCP Reno does (β=1/2). This mechanism acts as an inspection point of the receiver's judgment during congestion, when it is likely that the window size is small and hence the wave-based rate-detection capability might be weak. Otherwise, the packet loss is assumed to be due to a transient (perhaps wireless random) error and the congestion window size is not adjusted backward. That is a reasonable action; a somewhat significant queue build-up or congestion most likely will be reflected at the data-receiving rate, since packet interleaving will affect the data-delivery pattern. We note, however, that if the time of a congestion epoch is too short (determined by a threshold, see lines 10-11 in Figure 2), neither the sender nor the receiver have enough information to assess the network conditions, and the congestion window is reduced as in standard TCP. Hence, TCP-Real enhances standard TCP with a congestion avoidance tactic, taking advantage of its error/contention detection capability[7]. It can therefore be viewed as a TCP (α, β, γ) protocol where γ captures the protocol's behavior prior to congestion, when contention boosts up.

In order to avoid the wasteful window adjustments downward over asymmetric links, the sender needs to decouple the timeout mechanism and the RTT from the window size. That is, in standard TCP, acknowledgement losses may cause timeout to be extended and congestion window to be reduced. In TCP-Real, the timeout can be extended, but the window size could remain the same or even increase. The reasoning behind this strategic modification is that the sender needs to extend the timeout based on the RTT measurements, in order to accommodate the potential delays of the reverse path and avoid an early timeout. However, only the perceived congestion level of the forward path will determine the sender's congestion window size.

3.3 Implementation Notes

The sender sends data in waves and piggybacks the wave sequence number and wave size information using a TCP header option: TCP_REAL. TCP_REAL option is four bytes long. The third byte contains the current wave sequence number, while the fourth byte contains the wave size (number of segments) of the wave. The wave size attached is crucial for the measurements of

[7] It is worth mentioning that error detection and capacity measurement is a research topic in its own right. Precision of measurements and accuracy of estimation would have a direct impact on protocol performance. Presently, we have not exhausted the error detection capabilities of the wave pattern and therefore we did not incorporate a high degree of sophistication in the protocol's recovery strategy. Whenever such sophistication is required due to dubious measurements or incomplete information the protocol behaves as standard TCP. Practically, this observation is in favor of our results, which demonstrate notable improvement without exhausting the mechanisms' potential.

the receiver, since the number of packets sent side by side may be less than the congestion window size.

The receiver computes the data transmission rate by collecting the data packets corresponding to the current wave. The receiver also records t_f and t_l, the arriving time of the first and last segment in the wave, respectively. The data-receiving rate can therefore be computed as the ratio of the wave size to wave receiving time. The wave receiving time is the difference between t_f and t_l, and could be much smaller than the RTT. Thus, the data receiving rate captures the packet interleaving pattern at the bottleneck, rather than the available bandwidth or the achieved throughput. The wave size used in calculations is the "expected" wave size contained in the header, not the actual "received" wave size, since loss of corrupted segments in a wave needs to be counted. Packet loss due to random transient errors shall not affect the computed rate, which is used to measure the level of multiplexing/contention; packet loss due to congestion could be detected by the change in the receiving rate anyway.

The receiver reports the measured data-receiving rate and the corresponding wave sequence number back to the sender within the returned ACKs, using a TCP header option: TCP_REAL_ACK. TCP_REAL_ACK option is four bytes long. The third byte contains the wave sequence number of the last wave received completely, while the fourth byte contains the corresponding data-receiving rate. The data-receiving rate is re-scaled to the range of one byte [0, 255], where 255 in the TCP_REAL_ACK option corresponds to the highest data-receiving rate observed by the receiver so far. Since the identical TCP_REAL_ACK option is reported in every ACK until the wave sequence number changes, the probability not to deliver the wave-level information to the sender is quite low.

4. EXPERIMENTAL METHODOLOGY

4.1 Testing Environment

TCP-Real was implemented on both the xkernel protocol platform [29] and the ns-2 network simulator [19]. Results from an earlier stage of the protocol with the xkernel and single-flow measurements can be found in [32].

The environment of ns-2 enabled simulations of multiplexed wired/wireless channels and hence it was deemed appropriate for implementing our major testing plan. Simulations were conducted for both simple and complex network topologies. The simple topology used as a test-bed is the typical single-bottleneck *dumbbell*, as shown in Figure 3. The capacity of the bottleneck link (`bw_bottleneck`), access links to source nodes (`bw_src`), and access links to sink nodes (`bw_dst`) was occasionally re-configured for the different evaluation scenarios. In most cases, however, *bw_bottleneck* = *bw_src* = *bw_dst* unless it is pointed out explicitly otherwise. By default, all access links have a delay of 5ms while the bottleneck link has a delay of 25ms. However, the delays of access links to source nodes (`delay_src`) and access links to sink nodes (`delay_dst`) were re-configured in order to evaluate the protocol behavior when the flows have different RTTs (see section 5.1). For heterogeneous (wired and wireless) network simulations, ns-2 error models were inserted into the access links to the sink nodes. The Bernoulli model was used to simulate link-level errors with configurable packet error rate (PER). Error models were configured on both (forward and reverse) directions of the link traffic, except for the link asymmetry test (see section 5.4), where the error is configured in the reverse direction only. The number of flows (or the number of source-sink pairs) *n*, varied from experiment to experiment.

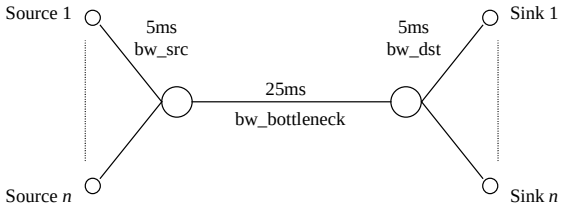

Figure 3. Dumbbell Network Topology

Protocol performance and fairness were also tested with multiple bottlenecks and cross traffic, using the scenario of Figure 4. The router R1 is the bottleneck for the main traffic (flows between source nodes to sink nodes), while the router R3 is another bottleneck for the competing main traffic and cross traffic (flows between peripheral source nodes and peripheral sink nodes).

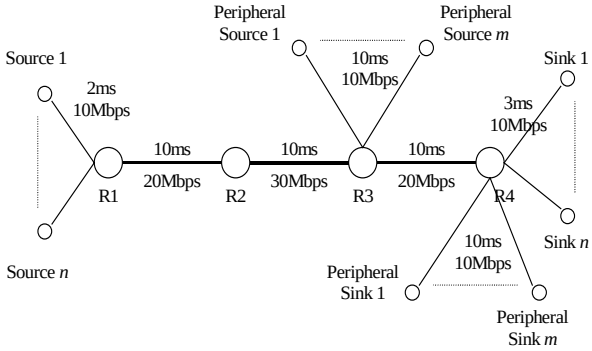

Figure 4. Network Topology with Multiple Bottlenecks and Cross Traffic

4.2 Testing Plan

The purpose of evaluation was to demonstrate the capability of TCP-Real to combine the advantages of standard TCP and TCP-friendly protocols to control congestion and increase smoothness, respectively. According to expectations, TCP-Real should improve

throughput and fairness of standard TCP in heterogeneous wired/wireless networks due to its enhanced error detection/classification, and achieve better performance with real-time traffic due to its congestion avoidance tactics. Being less aggressive than TCP-friendly protocols during congestion and more aggressive when bandwidth becomes available, Real should also demonstrate better friendliness. Accordingly, we have selected five protocols for comparative evaluation: TCP-Real, Reno, SACK, GAIMD and TFRC. TCP Reno introduces Fast Recovery [1] in conjunction with Fast Retransmit. Fast Recovery sets the congestion window to half its previous value, in the event of 3 dacks, after the retransmitted segment gets acknowledged. TCP-SACK [8, 18] is the newer version of TCP, and for that reason it was also included in the experiments [8]. However, TCP-SACK differs in the acknowledgment and retransmission strategy, enabling multiple segment retransmission within one RTT. We note that its mechanisms can be incorporated in other protocols as well. All the other protocols selected here have similar constraints with regard to retransmission during congestion (number of packets and timeouts), with the exception of TFRC, which is selected to enable a rough comparative assessment on the performance of rate-based TCP-friendly protocols under specific scenarios. Indeed, TFRC is significantly more aggressive than the other four protocols and it was deemed appropriate to demonstrate the conflicting behavior of aggressive, equation-based protocols when the situation calls for rapid bandwidth consumption.

Evaluation is conditional according to the testing subject and the protocols' and application characteristics. To honor a focused discussion and a reasonably tangible assessment we demonstrate the results selectively and in steps. We normally begin each scenario with a single-flow report, which is used as a reference point. Results are presented with `ftp` applications, over heterogeneous (wired and wireless) networks, asymmetric transient errors, and handoff conditions. To investigate the protocols' performance with delay-sensitive applications, we used the ns-2 CBR (Constant Bit Rate) agent to simulate a playback-enabled application with data rate 1Mbps. The bottleneck link bandwidth satisfies the condition

`1Mbps * n = bw_bottleneck`

in order to provision *just enough* bandwidth to all flows.

We also conducted experiments with diverse RTTs, multiple bottlenecks, and cross traffic. To further investigate the protocols' performance when the reverse path is congested by cross traffic, we configured the ns-2 exponential On/Off traffic generator on the reverse path.

We evaluate TCP-friendly protocols in line with the issues outlined in the introduction. Our major issue here is the relative friendliness of TCP-Real. We present a scenario where multiple flows of protocol *pairs* compete for the channel's bandwidth and their "friendliness" can be adequately demonstrated. Finally, as pointed out in section 2, a design tradeoff of TCP-friendly protocols is that a smooth rate reduction comes at the expense of a slow response to available bandwidth. In order to evaluate this tradeoff we created a scenario of temporary "blackouts" due to handoffs, during which all transmitted packets were lost and the channel's bandwidth was becoming available immediately afterwards.

4.3 Performance Metrics

Our evaluation plan calls for common as well as non-traditional metrics. The *System Goodput* is used to measure the overall system efficiency in bandwidth utilization with stationary environment. The *Goodput* for each flow 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 `Connection_Time` is the amount of time required for the data delivery. Consequently, the *System Goodput* is the sum of the Goodput of all flows, defined as

$$\texttt{System\_Goodput} = \sum_i g_i$$

where g_i is the goodput for the i^{th} flow. Similarly, we define *Aggregated Protocol Goodput*, as the goodput sum of all the flows that correspond to a particular protocol. The metric is used in protocol-pair tests to enable comparison of protocol *friendliness*.

Fairness is measured by the *Goodput Fairness Index*, derived from the formula given in [7] and defined as:

$$GFI = \frac{\left(\sum_i g_i\right)^2}{n\left(\sum_i g_i^2\right)}$$

In order to characterize the behavior of different traffic sources in the multi-bottleneck environment shown in Figure 4, we define *Average Traffic Goodput* to be the average goodput of all the flows belonging to the same traffic, either the main or the cross traffic. The system fairness is thus captured by the *Average Traffic Goodput Ratio*:

[8] The performance of TCP-NewReno [11], another version of TCP, was similar to either Reno's or SACK's and, hence, was not included in the presentation.

$$ATGR = \frac{\text{Average Traffic Goodput of Main Traffic}}{\text{Average Traffic Goodput of Cross Traffic}} = \frac{\frac{1}{n}\sum_i g\_main_i}{\frac{1}{m}\sum_j g\_cross_j}$$

where g_main_i is the goodput for the i^{th} flow of the main traffic and g_cross_j is the goodput for the j^{th} flow of the cross traffic; n and m are the number of flows belonging to the main traffic and the cross traffic, respectively. A value of *ATGR* close to 1 is desired: the system is unfair to the main traffic when the *ATGR* is smaller than 1; the system is unfair to the cross traffic when the *ATGR* is larger than 1.

In the experiments with real time traffic, the application attempts to read and consume up to 125KB every second, (assuming the playback buffer is exactly 125 KB). Because of the sending window fluctuation and transmission gaps of TCP, there are instances when the data is unavailable to the application. The percentage of application's successful attempts to read x% of 125KB data from the playback buffer, namely *x% Application Success Percentage*, is used to measure the protocol's real-time performance:

$$\text{Application Success Percentage} = 100 * \left[\sum_{j=1}^{T}\sum_{i=1}^{n} Success(i,j) / (nT)\right] \%$$

Where, n is the total number of flows; T is the connection time (in seconds); $Success(i,j)$ is defined as:

$$Success(i,j) = \begin{cases} 1 & \text{if } (AllotedGoodput(i,j) / TargetedReceivingRate) > x\% \\ 0 & \text{otherwise} \end{cases}$$

where $AllottedGoodput(i,j)$ is the goodput of the i^{th} flow within the j^{th} second. In our experimental configuration, $Targeted\ Receiving\ Rate$ is 1Mbps. From another perspective, the metric *x% Application Success Percentage* captures the *number* of discrete time slots when the flow achieves at least x% of 1 Mbps data receiving rate.

The *Sending rate* is used to capture a protocol's aggressiveness to exploit available bandwidth. We use this metric in scenarios of rapid bandwidth increase when bandwidth becomes immediately available after the handoff.

5. RESULTS AND DISCUSSION

5.1 Performance with time-tolerant applications

Figures 5 and 6 show that in a wired network with a 10Mbps bottleneck link, TCP-Real's goodput and fairness performance is as good as Reno's and SACK's, with the number of flows ranging from 10 to 100 in the experiments.

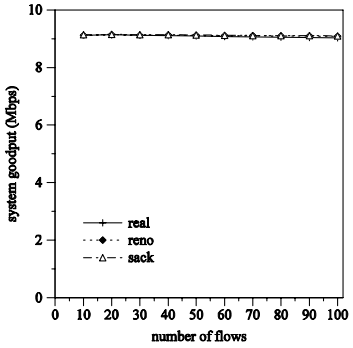

Figure 5. Goodput over Wired Network (10Mbps bottleneck link)

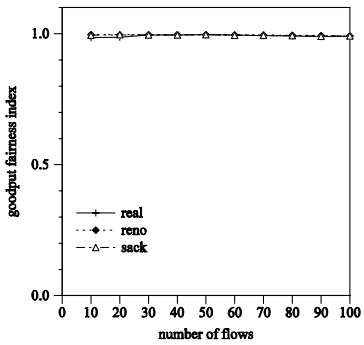

Figure 6. Fairness over Wired Network (10Mbps bottleneck link)

We repeated the experiments with diverse RTTs. The minimum RTT is fixed at 30ms, while the maximum flow RTT varies from experiment to experiment. The total number of flows is 50, and the i^{th} flow's RTT is given by the following equation:

$$RTT_i = 30 + i * (maxRTT - minRTT) / (50 - 1) \quad (ms)$$

Figures 7 and 8 show that in most cases, Real's fairness is not worse than Reno's or Sack's. Especially when the RTT dynamic range is high (> 400ms) and the network capacity is limited (10 Mbps), TCP-Real achieves better fairness.

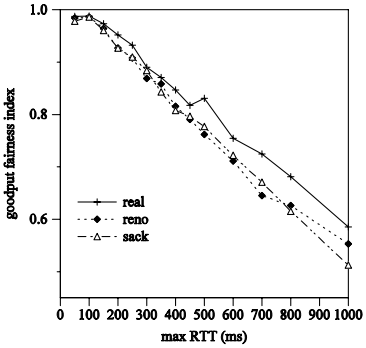

Figure 7. Fairness over Wired Network (10Mbps bottleneck link, 50 flows. RTTs are uniformly distributed between 30ms and max RTT)

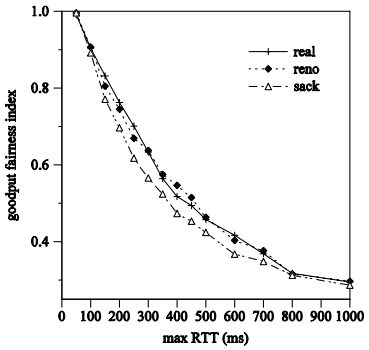

Figure 8. Fairness over Wired Network (100Mbps bottleneck link, 50 flows. RTTs are uniformly distributed between 30ms and max RTT)

Protocol performance was also tested with multiple bottlenecks and cross traffic (see Figure 4). Half of the flows form the main traffic, while the other half form the cross traffic. Our first simulation was conducted with drop tail routers. The result shown in Figure 9 appears initially surprising: although the propagation delay of the cross traffic was smaller than the delay of the main traffic and the bandwidth provisioned to the cross traffic was higher, the main traffic consumed more bandwidth. However, we note that the flows of the main traffic were aggregated in a 20Mbps link (R1 – R2) before entering the queue of the bottleneck R3, where they compete with the cross traffic. A detailed examination of the trace files has shown that packets aggregated before entering the bottleneck R3 were more uniformly distributed in the time domain, therefore having smaller probability to get dropped, compared to the bursty traffic of non-aggregated cross-traffic flows (see *Source 1 … Source n* and *Peripheral Source 1 … Peripheral Source m* in Figure 4). Notably, TCP-Real achieves relatively better fairness. We repeated this experiment with RED gateways [14]. The results show (Figure 10) that better system fairness is achieved, with TCP-Real maintaining a slight comparative advantage. The experiment also indicates that TCP-Real does respond appropriately to RED drops, which happen only when congestion boosts up.

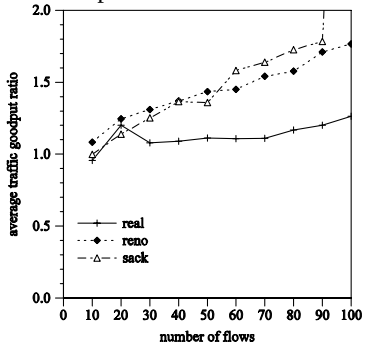

Figure 9. Traffic Goodput Ratio over Wired Network (multiple bottlenecks, with drop tail gateways)

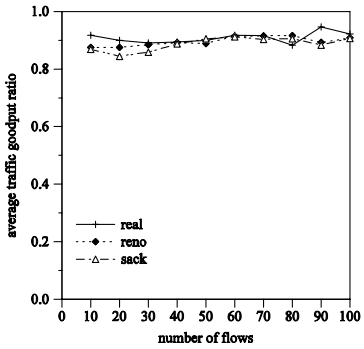

Figure 10. Traffic Goodput Ratio over Wired Network (multiple bottlenecks, with RED gateways)

5.2 Performance with Heterogeneous Networks

The relative system goodput of TCP-Real is further improved over heterogeneous networks, shown in Figure 11. Unlike TCP-Real, the other protocols do not have an error classification mechanism and hence recovery cannot be responsive to the error type.

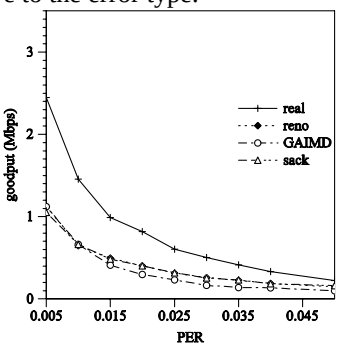

Figure 11. Goodput over Heterogeneous Network (10Mbps bottleneck link, single flow)

To further confirm that assertion, we provide a framework for characterizing the efficiency of a protocol's aggressiveness. For that purpose we exceptionally include TFRC in the comparison. The 10Mbps throughput capacity of the wireless links was interrupted by a handoff every 5 seconds; the duration of the handoff was exponentially distributed with a mean of 500ms. A protocol that is relatively aggressive is expected to behave similarly or even more aggressively when bandwidth becomes available rapidly. Otherwise its behavior is not reasonable but rather conflicting. In other words, since bandwidth becomes available immediately after the handoff, a high sending rate reflects a desirable behavior; the protocols need not adjust the rate due to congestion.

Figure 12 plots the protocol's sending rate. Note that the available capacity for each protocol is 10Mbps since each protocol was tested separately. Also note that TFRC is designed to respond to a loss event[9] instead of a packet loss. Besides its aggressiveness during congestion

[9] Recall that a loss event may include several packet losses

and the reduced timeout adjustments due to the notion of a "loss event", TFRC pays off the cost of equation-based recovery by misinterpreting the present situation. For this selected experiment the dominant parameter is α. After the handoff is over, the receiver of TCP-Real observes the lack of flow multiplexing and since contention has not been detected it adjusts its congestion window faster than GAIMD, Reno and SACK, and, in most cases, than TFRC as well. Although the handoff event had indeed triggered a timeout, the error pattern did not justify a congestion-oriented response for TCP-Real.

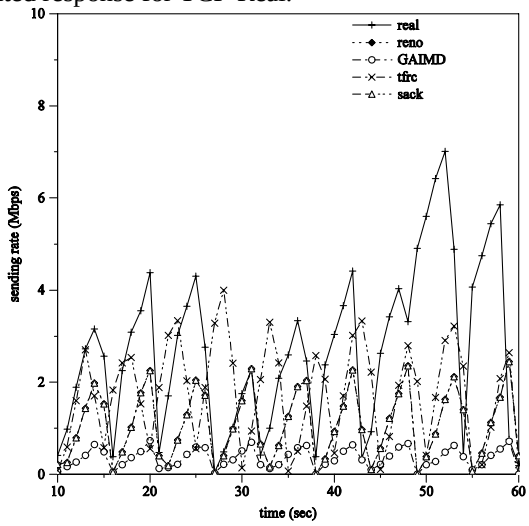

Figure 12. Sending Rate with Handoff (10Mbps bottleneck link, single flow)

It can be depicted from figures 13 and 14 below, that TCP-Real outperforms both Reno and SACK also in a multiple-flow channel with 10Mbps and 100Mbps link, respectively. With 10 flows and random transient errors varying from 0 to 0,05 PER, Reno and SACK's congestion control mechanisms unnecessarily reduce the congestion window. TCP Real avoids backward adjustments whenever the receiving rate does not justify a congestion-oriented response. However, as the error rate increases, the System Goodput naturally decreases for all protocols and the advantage of TCP-Real appears to be reasonably diminishing since the error rate increase determines the actual decreased availability of bandwidth. Figures 13 and 14 demonstrate the weakness of the (α, β) trading in wireless environments. Errors of some density force all protocols to mistakenly back off; the impact is dual for GAIMD since not only does it not avoid the unnecessary adjustment but, indeed, it delays the recovery.

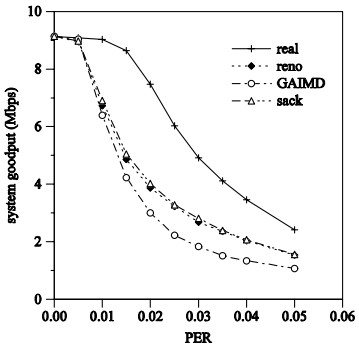

Figure 13. Goodput over Heterogeneous Network (10Mbps bottleneck link, 10 flows)

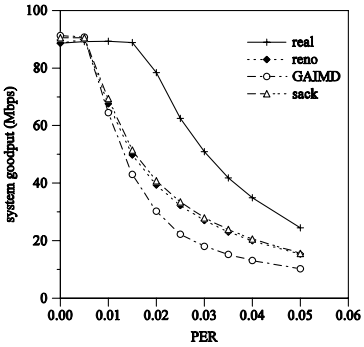

Figure 14. Goodput over Heterogeneous Network (100Mbps bottleneck, 10Mbps wireless link, 100 flows)

In order to demonstrate further the impact of TCP Real's capability to distinguish congestion and wireless errors, a *heterogeneous-flow* simulation was conducted over a 10Mbps bottleneck link. That is, the number of flows was fixed at 10, while the number of *wireless* access links (with PER = 0.01) to sink nodes varied from 0 to 10, in 11 distinct experiments. The results of the experiment are shown in Figures 15 and 16. An interesting conclusion can be drawn by those results. Reno, SACK and GAIMD appear to yield similar performance (i.e. system goodput) with that of TCP-Real, up to the point where at least one receiver is wired. The situation changes dramatically for Reno, SACK and GAIMD when all receivers are wireless (see Figure 15). We conjecture the following justification: Reno, SACK and GAIMD are capable of exploiting the channels bandwidth even with a single wired receiver; that is, a single receiver can consume all the available bandwidth, at the expense of fairness to the other (wireless) flows that experience losses due to wireless errors and hence unnecessarily reduce their transmission rate. Our hypothesis is confirmed by the results outlined in Figure 16. As the number of wireless receivers increases, fairness of TCP-Reno, SACK and GAIMD drops rapidly. When all 10 receivers are wireless, Reno, SACK and GAIMD are as fair as Real, but the bottleneck link is under-utilized (observe in Figure 15 the goodput achieved by 10 wireless Reno/SACK/GAIMD flows).

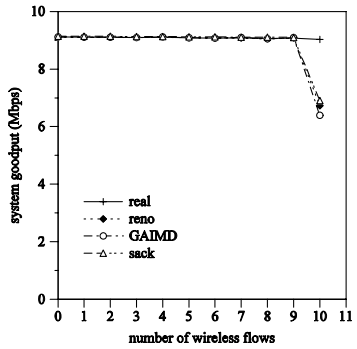

Figure 15. Goodput with Wired and Wireless Flows (10Mbps bottleneck, PER = 0.01, 10 flows)

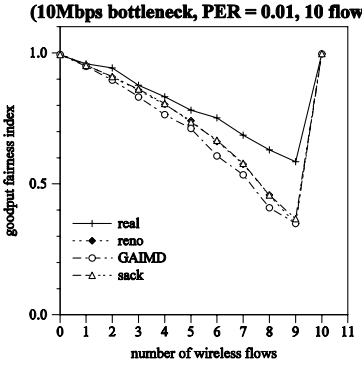

Figure 16. Fairness with Wired and Wireless Flows (10Mbps bottleneck, PER=0.01, 10 flows)

We repeated the experiments in the multi-bottleneck environment of Figure 4. The sink access links of the main traffic flows were wireless, while all cross-traffic flows were wired flows. Gateways were configured with the RED active queue management scheme. The results plotted in Figure 17 show that TCP-Real's performance degrades more gracefully.

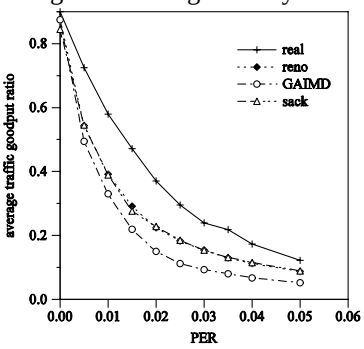

Figure 17. Traffic Goodput Ratio over Heterogeneous Networks (multiple bottlenecks, 20 flows)

5.3 Real-time Performance

The application-centered comparison of the protocols under different scenarios is outlined in Figures 18 and 19. Figure 18 depicts the successful attempts of the application to read at least 90% of the data sent. The application runs on top of TCP-SACK, Reno, GAIMD and Real, separately. We can observe that TCP Real is, in general, superior with regard to System Goodput. Furthermore, the TCP-Real-based time-constrained application experiences significantly better performance than the SACK-, Reno-, and GAIMD-based application. The dominant mechanism of TCP-Real is here the wave-based error detection capability that enables error classification (wired vs. wireless errors). This mechanism is complemented by the window manipulation prior to congestion, which is activated during high contention and is operated by parameter γ. Seasonable and moderated window adjustments cancel the deficiency of unnecessary transmission gaps due to the sharp, multiplicative window decrease and the timeout extension. The results of GAIMD with zero or minor error rate demonstrate the effectiveness of smooth backward adjustments under such conditions.

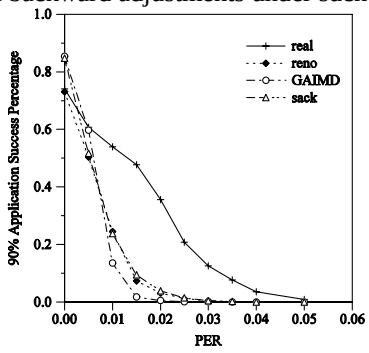

Figure 18. 90% Application Success Percentage (10Mbps bottleneck link, 10 flows)

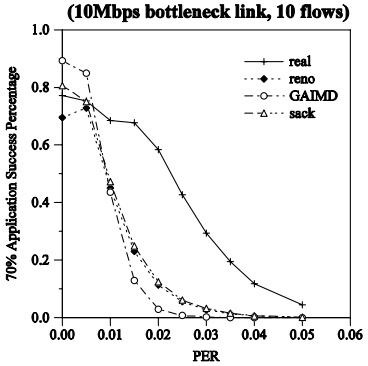

Figure 19. 70% Application Success Percentage (100Mbps bottleneck, 10Mbps wireless link, 100 flows)

5.4 Impact of Asymmetry

Our first scenario involves asymmetric error rates. A high packet-dropping rate is configured at the reverse path only. The PER of the forward path is 0.005, while PER on the reverse path ranges from 0.05 to 0.4. As outlined in Figure 20 where a single flow behavior is reported, TCP Real prevails, owing to the receiver-oriented congestion control mechanism. By having the receiver reporting the data-receiving rate, TCP-Real can protect itself from an unnecessary regression of the sender's aggressiveness due to RTT-based decisions. The sending rate of the forward path remains high since the data packets

arrive at the receiver timely and correctly. Similar measurements are taken with multiple competing flows (see figure 21).

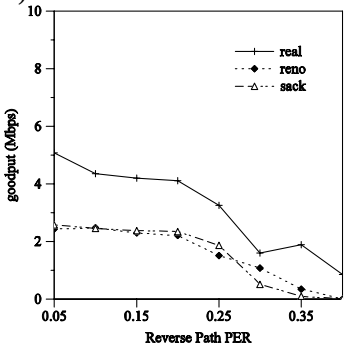

Figure 20. Goodput with Asymmetric Link (10Mbps bottleneck link, single flow forward path PER 0.001)

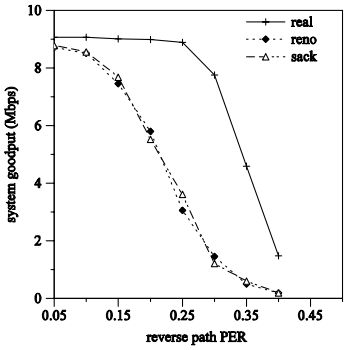

Figure 21. Goodput over Asymmertric Path (10 Mbps bottleneck link, 10 flows 0.005 forward path PER)

Our second scenario involves a multi-bottleneck network with a congested reverse path. The cross traffic of the reverse path was generated by an exponential On/Off traffic generator. The On phase sojourn time varied from 600ms to 1000ms, while the sum of the Off phase sojourn time and the On phase sojourn time was fixed at 1000ms. The On phase transmission speed was set to 20 Mbps, i.e. the capacity of the reverse bottleneck link (R4 – R3 in Figure 4), and the access link capacity of the cross traffic was adjusted correspondingly to 20Mbps to accommodate the transmission speed. Although the bandwidth required for the main traffic on the reverse path was pretty low, the bandwidth consumption on the forward path was significantly affected by the ACK losses/delays on the congested reverse path. The goodput and fairness results for the main traffic are plotted in Figures 22 and 23 and demonstrate the particular strength of receiver-oriented congestion control in dealing with asymmetry. When the On phase lasts long, the goodput of TCP-Reno/SACK is reduced to almost zero. However, TCP-Real's goodput is relatively unaffected and its fairness index is twice better.

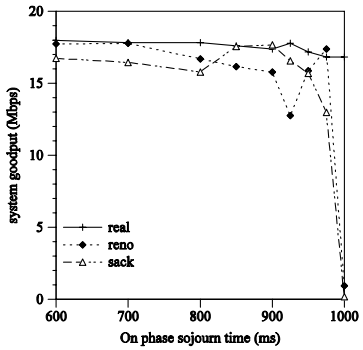

Figure 22. Goodput over Wired Network (multiple bottlenecks, 5 flows of main traffic reverse path congested by cross traffic)

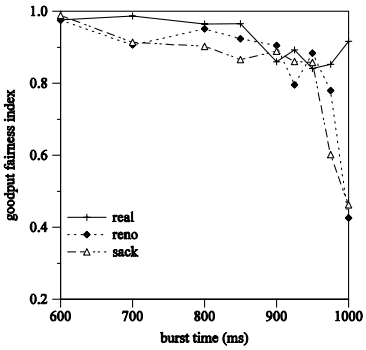

Figure 23. Fairness over Wired Network (multiple bottlenecks, 5 flows of main traffic reverse path congested by cross traffic)

5.5 Friendliness

Friendliness is defined here based on the original objective of TCP-friendly protocols to compete fairly with standard TCP; not in terms of equation-based protocols. That is, our question is not whether the equation-based adjustments are satisfied but rather whether the protocols compete fairly with standard TCP. Based on this definition, equation-based protocols are also subject to evaluation rather than reference points; the role of the reference point is held by TCP-Reno. Tests were conducted over a 10Mbps [10] link. The number of participating flows ranges from 10 to 100. Flows are divided into two groups per experiment. Half of the flows are instances of the same protocol: TCP-Real, TFRC, or GAIMD. The other half is a group of Reno flows, which serves as the reference for comparison. Ideally, each group of flows should consume exactly half of the bottleneck link capacity. Exceeding their fair-share at the expense of the other group's capacity would mean that the specific protocol is too aggressive. From the results shown in Figures 24, 25 and 26, we can conclude that TCP Real attains better performance not by simply being more

[10] Tests carried out also with a 100Mbps link; the results are similar

aggressive. It competes fairly with co-existing TCP flows. *Pace* [13], TFRC is not shown here to achieve its friendliness objective[11]. GAIMD on the other hand appears to be relatively conservative in this context, allowing TCP-Reno to consume bandwidth more aggressively, and to exceed its fair share.

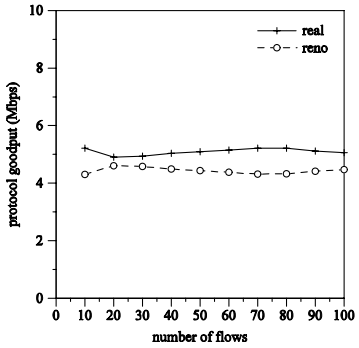

Figure 24. Real Competing with and Reno Flows (10Mbps bottleneck link)

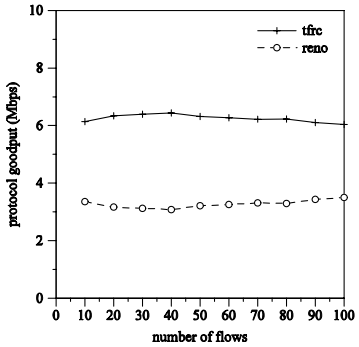

Figure 25. TFRC Competing with and Reno Flows (10Mbps bottleneck link)

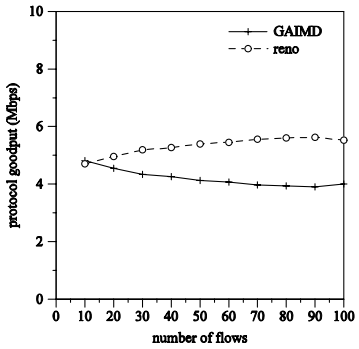

Figure 26. GAIMD Competing with Reno Flows (10Mbps bottleneck link)

6. CONCLUSIONS AND FUTURE WORK

We have evaluated the possibility of using an alternative mechanism for congestion control, using the receiver as the key peer who makes the decision about the transmission rate adjustments. We have enabled the receiver with an additional property, owing to the wave pattern, to call for window adjustments prior to congestion. The wave-based communication appears to be a useful mechanism for error classification as well. Monitoring the level of contention permits the receiver to rule on the cause of packet drops. TCP-Real was used as an experimental protocol to demonstrate the potential of this mechanism in multiplexed wired/wireless channels and with delay-tolerant and –intolerant applications. In an experimental confrontation, it was shown that TCP-friendly protocols occasionally fail to achieve both their objectives: friendliness and efficiency.

Some of the parameters of TCP-Real were empirically validated. The present experimental form of the protocol's mechanisms allows for improvements in error detection, error recovery and congestion avoidance strategy.

7. REFERENCES

1. M. Allman, V. Paxson and W. Stevens, "TCP Congestion Control", RFC2581, April 1999.
2. A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts", In Proceedings of the 15th International Conference on Distributed Computing Systems, May 1995.
3. H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effects of Asymmetry in TCP Performance", In Proceedings of ACM Mobicom '97, September 1997.
4. H. Balakrishnan, S. Seshan, E. Amir and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks", In Proceedings of ACM Mobicom '95, November 1995.
5. L.S. Brakmo and L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications, 13(8):1465-1480, Oct 1995
6. C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links", In Proceedings of ACM Mobicom 2001, July 2001
7. D.-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", Computer Networks and ISDN Systems, 17(1):1-14, June 1989.
8. K. Fall, and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Computer Communication Review, 26(3):5-21, July 1996.
9. W. Feng, D. Kandlur, S. Saha and K. Shin, "Understanding TCP Dynamics in an Integrated Service Internet", In Proceeding of ACM NOSSDAV '97, May 1997.
10. S. Floyd, "Congestion Control Principles", RFC 2914, September 2000.

[11] Note that experiments in [13] compared TFRC vs. TCP SACK

11. S. Floyd and M. Handley, “The NewReno Modification to TCP's Fast Recovery Algorithm”, RFC 2582, April 1999.
12. S. Floyd, M. Handley and J. Padhye, “A Comparison of Equation-Based and AIMD Congestion Control”, May 2000. Available from http://www.aciri.org/tfrc/.
13. S. Floyd, M. Handley, J. Padhye, and J. Widmer, “Equation-Based Congestion Control for Unicast Applications”, In Proceedings of ACM SIGCOMM 2000, August 2000.
14. S. Floyd, and V. Jacobson, “Random Early Detection gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, 1(4):397-413, August 1993.
15. T. Goff, J. Moronski, D. Phatak and V. Vipul Gupta, "Freeze-TCP: A true end-to-end Enhancement Mechanism for Mobile Environments", In Proceedings of IEEE INFOCOM 2000, March 2000.
16. M. Handley, J. Pahdye, S. Floyd, and J. Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification”, Internet draft draft-ietf-tsvwg-tfrc-02.txt, work in progress, May 2001.
17. V. Jacobson, “Congestion Avoidance and Control”, In Proceedings of ACM SIGCOMM ’88, August 1988.
18. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP Selective Acknowledgement Options”, RFC 2018, April 1996.
19. ns-2 Network Simulator, http://www.isi.edu/nsnam/ns/, 2001.
20. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", In Proceedings of ACM SIGCOMM ’98, August 1998.
21. J. Postel, “Transmission Control Protocol”, RFC 793, September 1981.
22. R. Rejaie, M. Handely and D. Estrin, “RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet”, In Proceedings of IEEE INFOCOM ’99, April 1999.
23. P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A Reliable Transport Protocol for Wireless Wide-Area Networks", In Proceedings of ACM Mobicom ’99, August 1999.
24. D. Sisalem and H. Schulzrinne, “The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme”, In Proceedings of ACM NOSSDAV ’98, July 1998.
25. V. Tsaoussidis, H. Badr, “TCP-Probing: Towards an Error Control Schema with Energy and Throughput Performance Gains”, In Proceedings of the 8th International Conference on Network Protocols, November 2000.
26. V. Tsaoussidis, H. Badr and R. Verma, “Wave and Wait Protocol: An energy-saving Transport Protocol for Mobile IP-Devices”, In Proceedings of the 7th International Conference on Network Protocols, October 1999.
27. V. Tsaoussidis, A. Lahanas and C. Zhang, “The Wave & Probe Communication Mechanisms”, Journal of Supercomputing, Kluwer Academic Publishers, 20(2):115-135, September 2001.
28. V. Tsaoussidis and I. Matta, “Open issues on TCP for Mobile Computing”, Journal of Wireless Communications and Mobile Computing (WCMC), John Wiley & Sons, 2(1):3-20, February 2002.
29. The X-Kernel: http://www.princeton.edu/xkernel
30. Y.R. Yang and S.S. Lam, “General AIMD Congestion Control”, In Proceedings of the 8th International Conference on Network Protocols, November 2000.
31. Y.R. Yang, M.S. Kim and S.S. Lam, “Transient Behaviors of TCP-friendly Congestion Control Protocols”, In Proceedings of IEEE INFOCOM 2001, April 2001.
32. C. Zhang and V. Tsaoussidis, “TCP-Real Improving Real-time Capabilities of TCP over Heterogeneous Networks”, In Proceedings of ACM NOSSDAV 2001, June 2001.