Preprint version

Adaptive rate control for streaming flows over the Internet

Luigi A. Grieco, Saverio Mascolo

Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Orabona 4, Bari, Italy (e-mail: {a.grieco,mascolo}@poliba.it)

Abstract:

The existing end-to-end TCP congestion control algorithm is well suited for applications that are not sensitive to delay jitter and abrupt changes of the transmission rate, such as FTP data transfer, but it is not recommended for delivering video data, whose perceived quality is sensitive to delay jitter and changes in the sending rate. In particular, the window-based control of Reno TCP congestion control causes burstiness in data transmission, which not only requires large buffers at the client side to provide a smooth playout but also may provoke bursts of lost packets difficult to recover via forward error correction techniques. This paper proposes an adaptive rate-based control (ARC) algorithm that stricly mimics the real-time dynamics of TCP and is based on an end-to-end mechanism to estimate the connection available bandwidth. Computer simulations using *ns*-2 have been developed to compare the ARC with the Reno TCP and with the TCP-Friendly Rate Control (TFRC) algorithm. Single- and multibottleneck scenarios in the presence of homogeneous and heterogeneous traffic sources have been considered. Simulations have shown that the ARC algorithm improves fairness and is friendly toward Reno. On the other hand, TFRC revealed itself not to be friendly toward Reno since it mimics only the long term behaviour of Reno TCP. Finally, simulations have shown that ARC remarkably improves the goodput with respect to TFRC and Reno in the presence of lossy links.

Categories and Subject Descriptors:

C.2.2 [Computer Communication Networks]: Network Protocols

General Terms:

Algorithms, Design

Keywords:

Congestion control design – Rate-based control – RTP/UDP

1 Introduction

The use of the Internet for carrying potentially high-quality video is continuously growing [34]. Integration of quality adaptive encoding schemes, forward error correction techniques, and congestion control algorithms is crucial to providing an effective video delivery system [28]. Encoding drastically reduces the number of bits used to transmit the video, error correction techniques ensure loss resilience by adding redundancy data [28,31], and congestion control algorithms allow senders to match the network available bandwidth [28].

This paper focuses on the design of an end-to-end rate-based congestion control algorithm for streaming flows over the Internet.

The proposed adaptive rate control (ARC) algorithm is based on a mechanism to estimate both the used bandwidth and the queue backlog in an end-to-end fashion and has been designed starting from the control theoretic analysis developed in [21].

ARC has been tested over many scenarios and compared with the TCP-Friendly Rate Control (TFRC), which is currently considered by the IETF for applications such as video streaming or telephony where a relative smooth sending rate is of importance [5,12,16]. In particular, single- and multi-bottleneck scenarios with and without lossy links and in the presence of homogeneous and heterogeneous traffic sources have been considered. Simulations have shown that (i) ARC exhibits a higher degree of fairness *with respect to* TCP and TFRC, (ii) ARC and Reno TCP are friendly toward each other, (iii) TFRC is not friendly toward Reno, (iv) ARC exhibits a less oscillating rate dynamics with respect to Reno TCP and TFRC in the presence of stationary network load, and (v) ARC remarkably improves the goodput with respect to TFRC and Reno in the presence of lossy links.

The paper is organized as follows. Section 2 provides an overview of related works, Sect. 3 summarizes the control theoretical results that are used as starting points for designing the control law, Sect. 4 describes the proposed algorithm, Sect. 5 shows simulation results, and, finally, the last section draws conclusions.

Correspondence to: S. Mascolo

2 Related work

During the last decade, research on congestion control algorithms has been quite active and has been essentially focused on congestion control for "best effort" reliable data traffic. The current version of TCP congestion control is still largely based on the cornerstone paper [17] and its modifications [2, 11]. TCP congestion control architecture assumes that the network is a "black box" that does not supply any explicit feedback to sources. Therefore, it is designed following the end-to-end principle that is one of the mayor keys to the success of the Internet [10]. In particular, a TCP source follows an *additive increase* mechanism to grab all available bandwidth and a *multiplicative decrease* mechanism to drastically decrease the window when congestion is revealed by a timeout or reception of three duplicate acknowledgements (DUPACKs). A consequence of using this *additive increase/multiplicative decrease* (AIMD) mechanism is that the congestion window size oscillates around its equilibrium point because packet losses are intentionally provoked to probe network capacity [10,23].

TCP Vegas proposes an alternative to the TCP AIMD mechanism in order to get a smoothed window dynamics [7]. In particular, a Vegas source tries to anticipate the reaction to congestion by monitoring the difference between the expected input rate and the actual input rate and adjusts the sender rate in an attempt to keep a small number of packets buffered in the routers along the network. A drawback of TCP Vegas is that it is not able to obtain its own share of bandwidth when competing with Reno sources or in the presence of reverse traffic [9,15,25].

Westwood TCP [23] and its variant [9,14] are recent modifications of Reno that exploit a new *additive increase/adaptive decrease* paradigm. The key idea of TCP Westwood is to exploit the stream of returning acknowledgement packets to estimate the bandwidth that is available for the TCP connection. When a congestion episode happens at the end of the TCP probing phase, the used bandwidth corresponds to the definition of best effort available bandwidth in a connectionless packet network. This bandwidth estimate is then used to adaptively decrease the congestion window and the slow-start threshold after a timeout or three duplicate ACKs.

Classic Reno TCP congestion control has been quite successful in preventing network collapse, but it cannot ensure fair sharing of network resources [14,17]. Other known drawbacks of Reno are (i) the window-based control causes burstiness in data transmission [6,29] and (ii) bursty data sources not only require large playout buffers at the client side to provide a smooth playout but also experience bursty packet losses that makes difficult the recovery via forward error correction techniques [1]. Hence, while TCP congestion control is well suited for applications not sensitive to delay jitter and abrupt changes of the transmission rate, such as FTP data transfer, it is not recommended to deliver video data, whose perceived quality is sensitive to delay jitter, changes in the sending rate, and bursty losses [30,34].

A congestion control algorithm well suited for video delivery should provide smooth rate dynamics in order to reduce playout buffering at the receiver [8]. Moreover, it should be friendly toward Reno sources in order to fairly allocate bandwidth to flows carrying multimedia and bulk data [16]. In order to provide friendliness, many control algorithms with a slower responsive dynamics have been designed by trying to emulate the "long-term" behavior of the Reno algorithm [5]. The TEAR (TCP emulation at receivers) rate control algorithm computes the input rate at the receiver and then feeds it back to the sender [30]. The rate is computed by emulating the average long-term throughput of one hypothetic Reno connection traversing the same path of the rate-based connection. It has been shown that TEAR does not employ the classic self-clocking mechanism [17], is not friendly toward Reno TCP at high loss rate, and does not reduce its sending rate under persistent congestion [35].

To obtain a Reno conformant behavior with a smoothed rate dynamics, linear increase multiplicative decrease algorithms have been proposed. The rate adaptation protocol (RAP) additively increases the transmission rate until a network congestion is detected and, upon a congestion episode, it multiplicatively decreases the transmission rate. RAP does not enforce the self-clocking principle [29].

A linear increase multiplicative decrease algorithm with history (LIMD/H) has been proposed in [19]. The algorithm proposes a linear increasing rate during the probing phase and an adaptive multiplicative rate reduction after congestion that takes into account the loss rate. This algorithm also does not implement the self-clocking mechanism. It has been shown that a simple AIMD rate control algorithm cannot guarantee friendliness toward Reno connections since an AIMD rate mechanism does not match an AIMD window mechanism [5, 6].

A new control algorithm that has been proposed to emulate the long-term behavior of the Reno throughput is the TCP-Friendly Rate Control (TFRC) [12,31]. TFRC aims at obtaining a smooth transmission rate dynamics along with friendliness toward Reno TCP [12]. To provide friendliness, a TFRC sender emulates the long-term behavior of a Reno connection using the equation model of the Reno throughput developed in [27]. In particular, the TFRC sender computes the transmission rate as a function of the average loss rate, which is sent by the receiver to the sender as a feedback report. This approach has the following drawbacks: (i) even though the equation model developed in [27] is a useful tool for carrying out analytical evaluations of Reno TCP throughput, it may exhibit up to 30% of error [27]; (ii) it has been shown that the behavior of equation-based congestion control is influenced by the throughput model employed, the variability of loss events, and the correlation structure of the loss process [33]. Another equation-based rate control algorithm has been proposed in [31] that mainly differs from the TFRC in that the employed throughput equation model is the simpler one proposed in [24].

In [5] the rate-based algorithms proposed in [12,29] and the window-based algorithms proposed in [2,4] have been tested in the presence of dynamic network conditions to show that algorithms that do not employ the self-clocking principle [12,29] may exhibit a huge settling time, that is, they may require many RTTs to adapt the input rate to the bandwidth available in the network. To overcome the disastrous effects due to the violation of the self-clocking principle, an enhanced version of the TFRC algorithm that emulates the self-clocking mechanism has been proposed in [5]. The enhanced TFRC exhibits good dynamic performance.

It is important to point out that the *self-clocking* mechanism implemented by the TFRC is different from the *self-clocking* of the TCP [17]. In fact, the TFRC *self-clocking* acts only after a packet loss by limiting the sending rate at the data rate that has been received during the previous RTT [5]. On the other hand, the TCP *self-clocking* mechanism is always active because the number of outstanding packets is limited. Thus, from a dynamic point of view, TFRC is not Reno conformant. In other words, a new joining TCP flow has an immediate effect on the self-clocking mechanism of existing TCP flows, whereas it affects TFRC flows only after packet loss. This can severely affect the friendliness between TFRC and Reno TCP.

The implementation of the self-clocking principle in a rate-based environment has been theoretically discussed in [21]. In particular, it has been shown that it is possible to design a stable and efficient congestion controller by taking into account the number of outstanding packets when computing the transmission rate. An implementation of this algorithm has been developed in the context of the TCP/IP protocol by proposing the concept of Generalized Advertised Window, which has to be supplied by network routers [13].

3 A control law based on the Smith predictor: background results

This section summarizes some control theoretical results derived in [21] that will be used as starting points in designing the ARC algorithm proposed in this paper. In [21] it was shown that a data connection can be modelled as a time delay system that can be efficiently controlled by following the Smith principle. In particular, to provide bottleneck queue stability and high utilization of the bottleneck link depicted in Fig. 1, the following rate-based control equation has been proposed:

$$r(t) = k[w(t) - q(t - T_{fb}) - \int_{t-RTT_{min}}^{t} r(\tau)d\tau]^{+}, \quad (1)$$

where:

- $[x]^{+} = max\{0, x\}$;
- $r(t)$ is the transmission rate;
- $w(t)$ represents a threshold for the queue length $q(t)$;
- $q(t)$ is the bottleneck queue backlog;
- $RTT_{min} = T_{fw} + T_{fb}$ is the minimum round trip time, where T_{fw} is the forward delay that models the propagation time from the sender to the bottleneck and T_{fb} is the backward delay that models the propagation time from the bottleneck to the destination and then back to the sender;
- $\int_{t-RTT_{min}}^{t} r(\tau)d\tau + q(t - T_{fb})$ represents the in-pipe packets plus the queued packets, that is, they are the *outstanding packets*;
- $b(t)$ is the bandwidth used by the flow;
- k is the proportional gain that relates the transmission rate $r(t)$ to the quantity $[w(t) - q(t - T_{fb}) - \int_{t-RTT_{min}}^{t} r(\tau)d\tau]$.

It is easy to give an intuitive interpretation of Eq. 1: the transmission rate $r(t)$ is proportional, via the constant k, to the difference between the threshold $w(t)$ and the sum of the backlog $q(t - T_{fb})$ with the number of in-pipe packets $\int_{t-RTT_{min}}^{t} r(\tau)d\tau$ [21]. From Eq. 1 it turns out that when the number of outstanding packets $q(t-T_{fb})+\int_{t-RTT_{min}}^{t} r(\tau)d\tau$ is greater than or equal to w, then the computed transmission rate is zero. This implies that the number of outstanding packets can never exceed w. It is also interesting to observe that Eq. 1 can be viewed as the rate-based version of the classic sliding window control. In fact, dividing both sides of Eq. 1 by k, the sliding window control equation

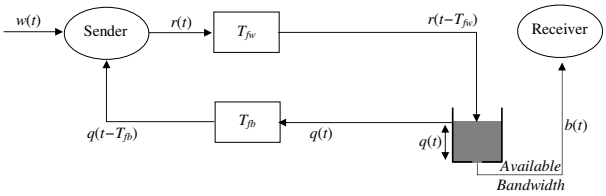


Fig. 1. Schematic of a connection

$$\Delta W = r/k = (w(t) - q(t - T_{fb}) - \int_{t-RTT_{min}}^{t} r(\tau)d\tau)$$

is easily obtained. Notice that if $q(t)$ is the receiver buffer queue length, then $w(t) - q(t - T_{fb})$ is the TCP advertised window and Eq. 1 reduces to the standard TCP flow control [21,22]. If $q(t-T_{fb})+\int_{t-RTT_{min}}^{t} r(\tau)d\tau$ represents the outstanding packets and $w(t)$ the congestion window *cwnd*, then Eq. 1 represents the TCP congestion control. The latter is an important result that will be exploited later when we design a dynamic setting for $w(t)$ that mimics the real-time dynamics of the TCP in order to provide friendliness.

In [21], it was shown that Eq. 1 ensures both network stability and bounded queue lengths at the routers. Moreover, by considering the control Eq. 1 in steady state condition, i.e., when the sending rate $r(t)$ is constant and matches the available bandwidth $b(t) = B$, and by assuming that the backlog queue length $q(t - T_{fb})$ is zero, so that the round trip time RTT reduces to the minimum round trip delay RTT_{min}, one has that $r = k \cdot (w - B \cdot RTT_{min}) = B$, from which the following relation turns out:

$$w = B \cdot (RTT_{min} + \frac{1}{k}). \quad (2)$$

Equation 2 is important since it provides the window $w(t)$ that ensures there will be no queue backlog when in the presence of the available bandwidth B. It should be noted that, since Eq. 2 clears out all buffers along the connection path, it improves statistical multiplexing of flows going through FIFO buffers and increases fairness in bandwidth allocation.

The constant gain k in Eq. 1 affects the time constant of the system dynamics. In fact, in [21] it was shown that the transfer function from the threshold $w(t)$ to the queue backlog $q(t)$ is

$$\frac{Q(s)}{W(s)} = \frac{k}{s+k} e^{-s \cdot T_{fw}}. \quad (3)$$

This means that the closed-loop dynamics is that of a first-order system, with time constant $\tau = 1/k$, delayed by T_{fw}. To get a further insight into the dynamic behavior of the transfer function Eq. 3, it is worth reporting the response to the step

function $w^0 \cdot 1(t)$,[1] which is

$$q(t) = w^0(1 - e^{-k(t-T_{fw})})1(t - T_{fw}). \quad (4)$$

In principle, the transient mode $e^{-k(t-T_{fw})}$ in Eq. 4 can be made faster and faster by choosing a larger and larger gain k. However, an upper bound must be considered when choosing k. In fact, in packet networks the feedback information is delivered through packets, which implies that the controlled system is a sampled controlled system [3]. By assuming that a feedback report is generated every RTT and that the maximum RTT is 500 ms, which corresponds to the "worst case" of a GEO satellite connection, the system is sampled every 500 ms. The Nyquist-Shannon sampling theorem requires that the system time constant be at least twice as large as the sampling period $T_s = 500\,\text{ms}$. To be conservative, we choose a constant time that is four times the sampling period, that is, we assume $\tau = 1/k = 4T_s = 2\,\text{s}$, i.e., $k = 0.5/\text{s} = 0.5\,\text{s}^{-1}$.

4 The Adaptive Rate Control algorithm

When proposing a new congestion control algorithm two fundamental requirements must be satisfied: (i) the algorithm must provide a degree of fairness in bandwidth allocation *at least equal* to that provided by the Reno algorithm; (ii) the algorithm must exhibit a friendly behavior toward TCP flows, so that coexisting connections controlled by TCP Reno can get their fair bandwidth share.

To provide both fairness and friendliness, we exploit the results summarized in Sect. 3 for which Eq. 1 is the rate-based form of the classic sliding window control employed by Reno TCP for flow and congestion control [2,17]. Thus, similarly to the TCP, the ARC algorithm proposed in this paper is made of two phases: (i) a probing phase, which aims at utilizing the network available bandwidth, and (ii) a shrinking phase, which reduces the input rate in the presence of congestion.

4.1 The probing phase

To be friendly toward Reno, we design the probing phase following the results reported in Sect. 3 for which Eq. 1 is the rate-based form of the TCP flow and congestion control. Therefore, we propose a *quick* probing phase, which corresponds to the TCP slow start, and a *gentle* probing phase, which corresponds to the TCP congestion avoidance phase. The *quick* probing phase is obtained by setting

$$w(t) = w(t_0) \cdot 2^{\frac{t-t_0}{\alpha}}, \quad (5)$$

where t_0 is the time of the last window update and α is a multiplicative constant. The setting (Eq. 5) mimics the exponential increasing of the TCP slow start. The *gentle* probing phase is obtained by linearly increasing the control window $w(t)$ as follows:

$$w(t) = w(t_0) + \frac{t - t_0}{\alpha}, \quad (6)$$

where t_0 is the time of the last window update and α is a multiplicative constant. We choose $\alpha = 0.3\,\text{s}$; this increases w by one packet every 300 ms during the *gentle* probing phase and doubles w every 300 ms during the *quick* probing phase.

It is important to observe that the ARC linear increasing phase mimics the linear phase used by Reno during the congestion avoidance phase. On the other hand, the additive rate increase proposed in [19,29,32] *is not equivalent* to the Reno linear increasing phase (see also [5,6]).

4.2 The shrinking phase

When a congestion episode happens, it is necessary to trigger the shrinking phase in order to reduce the input rate. It is important to realize that end-to-end congestion control algorithms do not explicitly know the congestion status of network nodes, but they must infer it using implicit notifications such as timeouts or duplicated ACKs in TCP.

We consider two events as implicit indications of congestion:

1. A packet is lost and the sequence of received packets contains a hole.
2. The sender does not receive any report from the receiver for a long time so that a timeout expires.

ARC reacts to congestion events by setting $w(t)$ according to Eq. 2, which ensures that all the buffers along the path are cleared out.

To implement Eq. 2, it is necessary to estimate the available bandwidth. For that purpose, it should be noted that the bandwidth used at the time of a congestion episode is, by definition, the end-to-end "best effort" bandwidth available at the end of the probing phase.

To estimate the bandwidth used by a connection, the receiver counts and filters the received packets. In particular, every smoothed round trip time (SRTT), which is computed using the Van Jacobson algorithm [17], a sample of used bandwidth is computed at the receiver as follows:

$$B(k) = \frac{D(k)}{T(k)}, \quad (7)$$

where $D(k)$ is the amount of data received during the last $SRTT = T(k)$ (Fig. 2). Since network congestion is due to the low-frequency components of the used bandwidth [14,20, 23], we average the $B(k)$ samples using the following discrete-time filter:

$$\hat{B}(k) = \frac{2\tau - T(k)}{2\tau + T(k)} \cdot \hat{B}(k-1) + \frac{T(k)}{2\tau + T(k)} \cdot (B(k) + B(k-1)), \quad (8)$$

where τ is the time constant of the filter (we assume $\tau = 0.5\,\text{s}$). This filter is time varying to counteract the fact that the sampling interval $T(k)$ is not constant [23]. It is important to note that the filter in Eq. 7 works well if the Shannon-Nyquist theorem is satisfied, that is, if $T(k) < \tau/2$, in order to avoid aliasing effects [3,23]. To be conservative, we assume $T(k) < \tau/4$ and, when $T(k) \geq \tau/4$, we interpolate and resample using $N = integer(4T(k)/\tau)$ virtual samples $B(k)$ arriving with interarrival time $\tau/4$ and one more virtual sample

[1] The step function is defined as $1(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$; w^0 is a real constant.

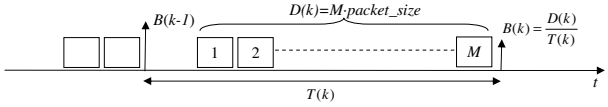


Fig. 2. Bandwidth sample computation

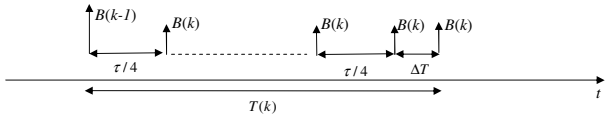


Fig. 3. Resampling algorithm when $T(k) > \frac{\tau}{4}$

$B(k)$ arriving after the interarrival time $\Delta T = T(k) - N\tau/4$. The resampling algorithm is depicted in Fig. 3.

Finally, to implement the control Eq. 1 it is necessary to estimate the backlog $q(t - T_{fb})$ and the in-pipe packets $\int_{t-RTT_{min}}^{t} r(\tau)d\tau$. To estimate the backlog plus the in-pipe packets, ARC uses the outstanding packets.

5 Performance evaluation

The ARC algorithm is here investigated via computer simulation using *ns*-2 [26]. A comparison with the enhanced version of the TFRC proposed in [5] and with the Reno TCP is also carried out. TFRC parameters have been set as suggested in the *ns*-2 package. The ARC parameters have been chosen as follows: $k = 0.5\,\mathrm{s}^{-1}$, $\alpha = 0.3\,\mathrm{s}$. Both single- and multibottleneck topologies have been simulated. In all considered scenarios, TCP sinks implement the delayed ACK option. Packets are 1500 bytes long and connections are greedy. Unless otherwise specified, bottleneck queues are set equal to the bottleneck bandwidth times the maximum round trip time, which is equal to 250 ms. We will focus on throughput, packet losses, intraprotocol long-term and short-term fairness in bandwidth sharing, interprotocol friendliness, and burstiness of the sending rate.

5.1 Single-bottleneck scenario

We consider the single-bottleneck topology depicted in Fig. 4, which consists of a single-bottleneck link shared by N Reno TCP sources, one UDP source, and M rate-based sources. Ten Reno TCP sources send data in the opposite direction, i.e., they generate congestion on the backward path. RTTs of the N Reno connections (M rate-based connections) going along the forward path are uniformly spread in the interval $[(250/N)\,\mathrm{ms}, 250\,\mathrm{ms}]$ ($[(250/M)\,\mathrm{ms}, 250\,\mathrm{ms}]$). RTTs of the ten Reno TCP connections feeding the backward traffic are uniformly spaced in the interval [25 ms, 250 ms]. All TCP sources start data transmission at time $t = 0\,\mathrm{s}$, whereas the rate-based sources start data transmission at $t = 10\,\mathrm{s}$. Simulations last 1000 s unless otherwise specified.

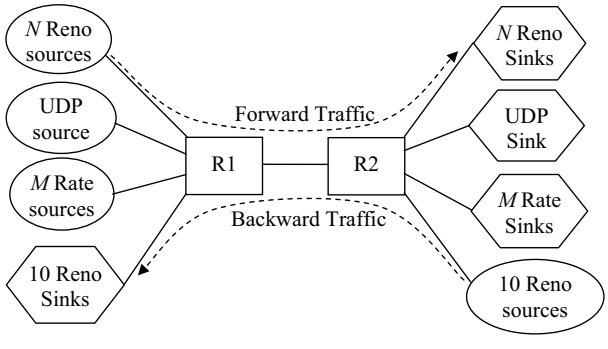
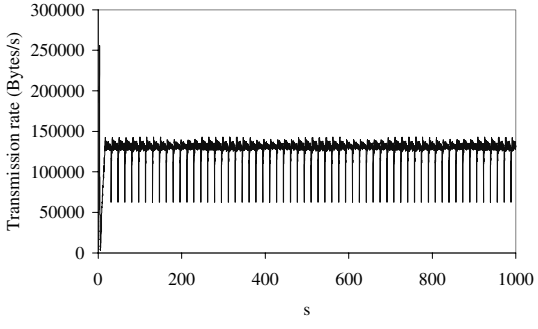


Fig. 4. Single-bottleneck scenario

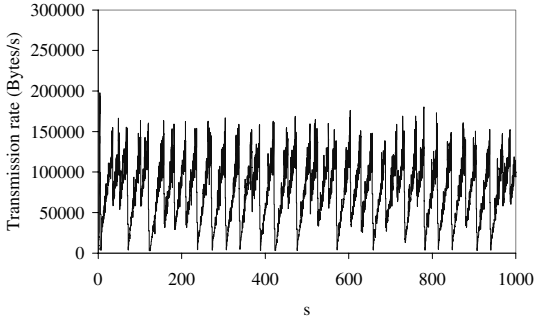
5.1.1 One rate-based or one Reno TCP source in the presence of constant available bandwidth

This section evaluates the behaviors of a single TFRC or a single ARC or a single Reno TCP in the scenario depicted in Fig. 4. The bottleneck link capacity is 1 Mbps, and the RTT of the connection on the forward path is 250 ms. The ten Reno TCP sources, which generate traffic along the backward path, are first turned off and then on to simulate a more realistic scenario where the backward path from the destination to the source is congested. This scenario aims at evaluating the behavior of the considered control algorithms in the presence of a constant available bandwidth. In this condition, we would like to obtain a constant transmission that matches the constant network capacity.

Figure 5 shows the sending rate of a single Reno TCP connection in the above-described scenario. The transmission rate has been computed as $cwnd/RTT$. In particular, it is worth noting that Reno TCP exhibits a highly variable bit rate, which is due to the oscillations of the congestion window that are exacerbated by the phenomenon of ACK compression due to backward traffic (Fig. 5b). It is worth pointing out that while

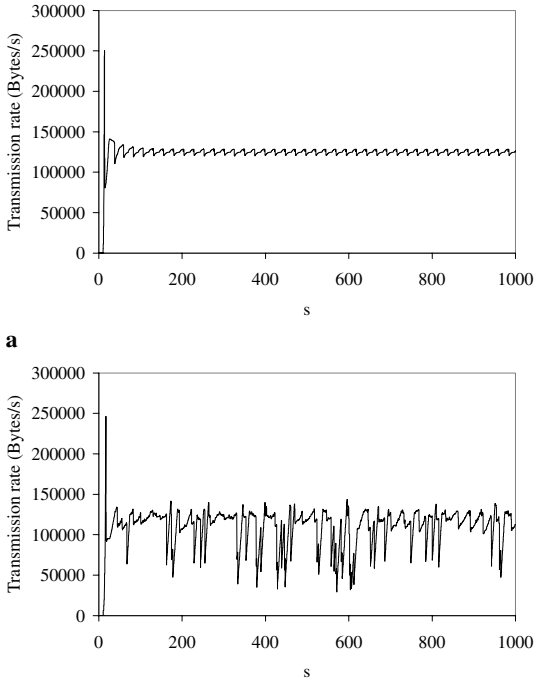


a



b

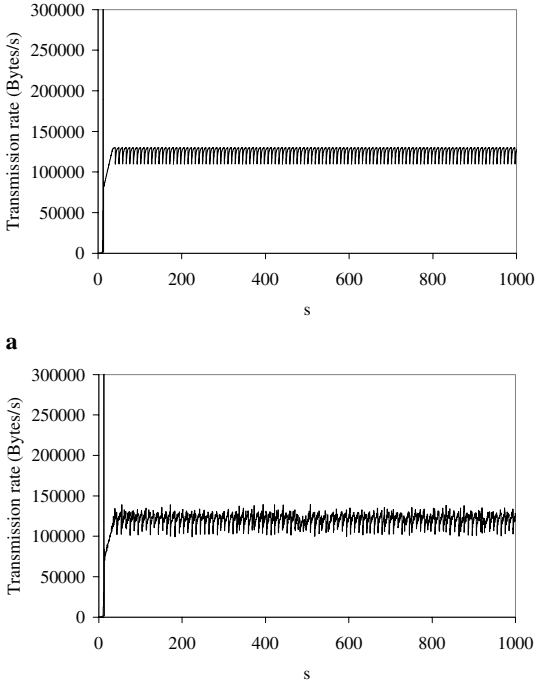
Fig. 5. One Reno TCP source over a 1-Mbps bottleneck. (**a**) Reverse traffic is off. (**b**) Reverse traffic is on



a

b

Fig. 6. One TFRC source over a 1-Mbps bottleneck. (**a**) Reverse traffic is off. (**b**) Reverse traffic is on

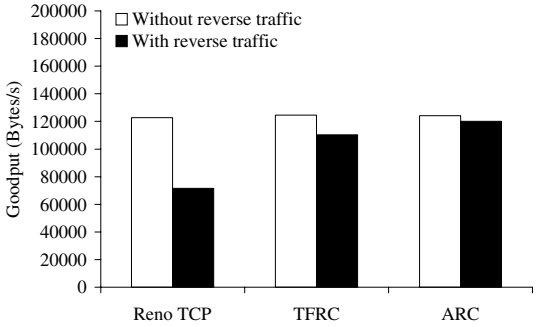


a

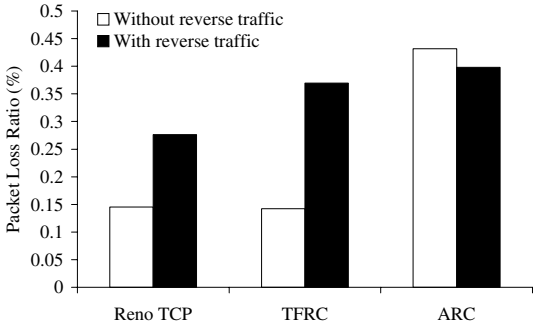
b

Fig. 7. One ARC source over a 1-Mbps bottleneck. (**a**) Reverse traffic is off. (**b**) Reverse traffic is on

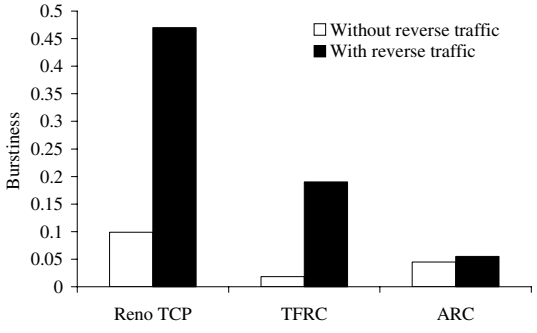
rate fluctuations over short time scales, such as those observed in Fig. 5a, do not affect network utilization, rate oscillations over large time scales can lead to network underutilization. In fact, the bottleneck utilization of Reno TCP, which is defined as



a



b



c

Fig. 8. One Reno TCP or one TFRC or one ARC source over a 1-Mbps bottleneck. (**a**) Goodputs. (**b**) Packet loss ratios. (**c**) Burstiness

$$U = \frac{goodput}{bottleneck\ link\ capacity},$$

diminishes from 98% obtained in Fig. 5a to 57% obtained in Fig. 5b when the reverse traffic is turned on.

Figure 6 shows the sending rate of one TFRC connection in the same scenario. Also, in this case the presence of reverse traffic provokes burstiness in the sending rate. In particular, the bottleneck utilization provided by TFRC drops from 99% to 88% when the reverse traffic is turned on.

Finally, Fig. 7 shows the sending rate of one ARC connection in the same scenario. In this case, the ARC algorithm is not affected by congestion on the backward path and provides a smooth transmission rate, both with and without reverse traffic, reaching a 99% bottleneck utilization.

Figure 8 summarizes simulation results and provides synthetic performance indices for the scenarios considered above. In particular, it reports the goodput, the packet loss ratio, and the burstiness of the sending rate. The burstiness is measured as in [35] using the coefficient of variation, which is

$$Burstiness = \frac{\sigma(r)}{\mathrm{E}[r]},$$

where $\sigma(r)$ is the standard deviation of the transmission rate and $\mathrm{E}[r]$ is the average value. Both $\sigma(r)$ and $\mathrm{E}[r]$ have been evaluated over the last 900 s of simulation to eliminate the initial transient. Figure 8 shows that ARC is not sensitive with respect to reverse traffic. On the other hand, both Reno TCP and TFRC experience increased packet loss ratio and burstiness in the presence of reverse traffic. From now on, sources of reverse traffic will always be turned on to reproduce the conditions of a real packet switching network.

5.1.2 One rate-based or one Reno TCP with one ON-OFF Constant Bit Rate (CBR) source

This section evaluates the behaviors of a single TFRC or a single ARC or a single Reno TCP connection sharing the single-bottleneck scenario in Fig. 4 with one ON-OFF constant bit rate UDP source. The bottleneck link capacity is 1 Mbps. The CBR source transmits at 0.7 Mbps during the 200-s ON period and is silent during the OFF period, which also lasts 200 s. The CBR source provokes 1:3 available bandwidth variations such as in the scenario considered in [5]. The ON-OFF CBR connection provides a piecewise constant best-effort available bandwidth that is useful to test the algorithm reactivity under strenuous conditions. This scenario is particularly interesting since it reproduces the case of bandwidth oscillation that is present in 2.5G/3G wireless systems [18].

Figure 9 shows that the sending rate of one Reno TCP source is bursty and oscillating. Figure 10 shows that TFRC exhibits large fluctuations of the sending rate even though the available bandwidth on the forward path is piecewise constant.

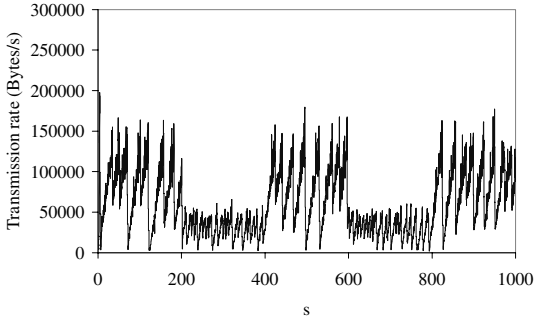


Fig. 9. One Reno TCP source sharing a 1-Mbps bottleneck with one ON-OFF CBR source

Figure 11 reports analogous results obtained using the ARC algorithm. In this case, ARC provides a piecewise constant sending rate that nicely matches the available bandwidth left unused by the ON-OFF CBR source.

It is interesting to investigate how fast the input rate matches the network available bandwidth. For that purpose we zoom in on Figs. 9, 10, and 11 during the transients, i.e., after the CBR is turned on and off. In particular, Fig. 12a shows the dynamic behavior of the sending rates when the CBR source is turned off and new bandwidth becomes available. ARC and TFRC are smooth and exhibit similar rise times, which are equal to 200 RTTs, whereas Reno TCP produces an oscillating transmission rate. Figure 12b shows the behavior of the sending rates in response to the sudden reduction of available bandwidth that happens when the CBR source is turned on.

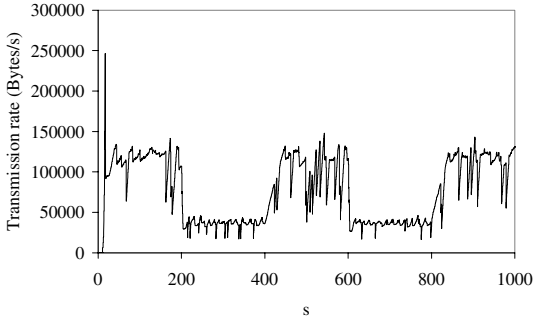


Fig. 10. One TFRC source sharing a 1-Mbps bottleneck with one ON-OFF CBR source

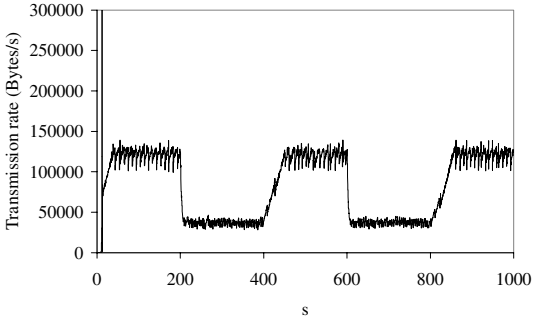
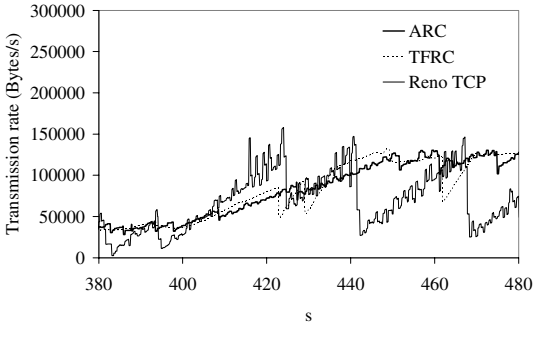
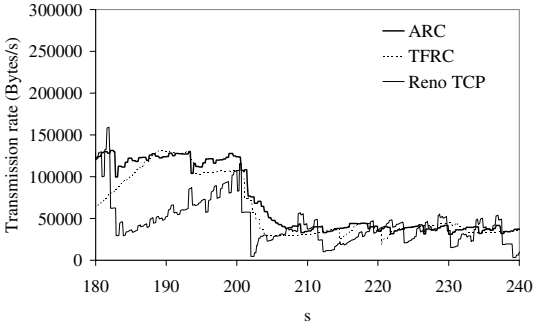


Fig. 11. One ARC source sharing a 1-Mbps bottleneck with one ON-OFF CBR source



a



b

Fig. 12. Transient behaviors of one Reno TCP, one TFRC, and one ARC sending rates in the presence of (**a**) sudden increase of the available bandwidth when the CBR source is turned off and (**b**) sudden decrease of the available bandwidth when the CBR source is turned on

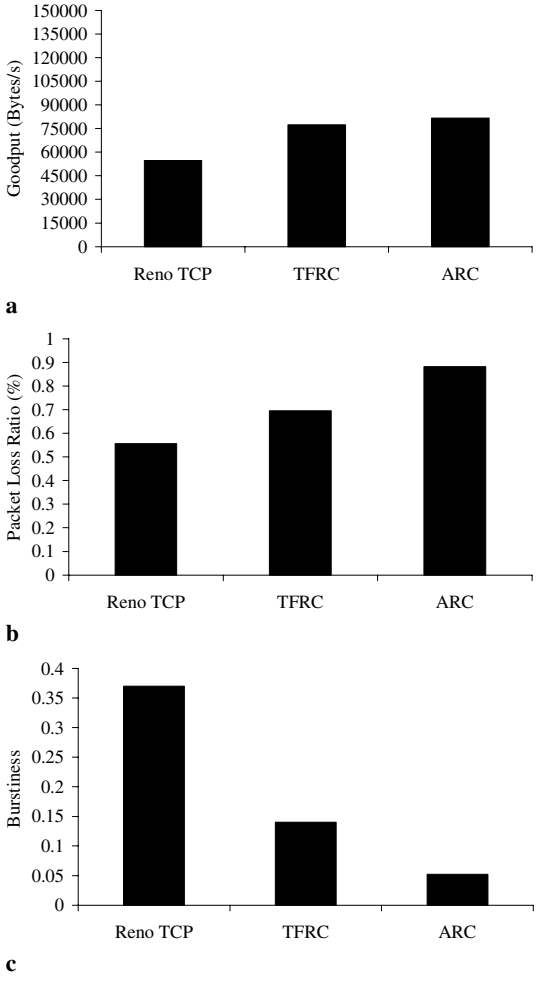


Fig. 13. One Reno TCP or one TFRC or one ARC source sharing a 1-Mbps bottleneck with an ON-OFF CBR source. (**a**) Goodputs. (**b**) Packet loss ratios. (**c**) Burstiness

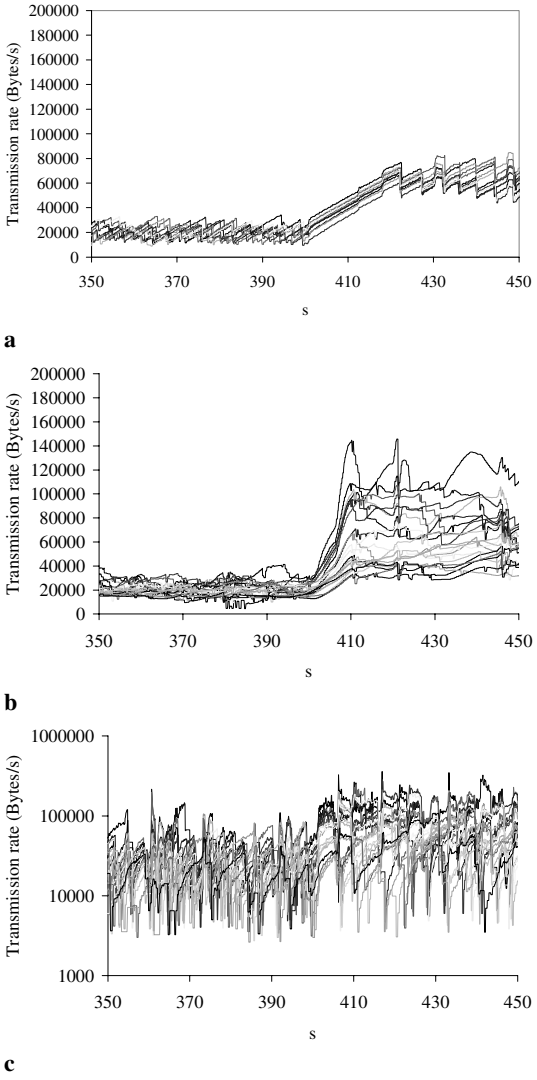


Fig. 14. Transmission rates of 20 connections when the CBR source is suddenly turned off. (**a**) The 20 connections are ARC. (**b**) The 20 connections are TFRC. (**c**) The 20 connections are Reno TCP

Also in this case, the transmission rates of ARC and TFRC take the same time (40 RTTs) to match the available bandwidth, whereas Reno TCP exhibits a larger oscillating transmission rate.

Figure 13 reports the goodput, the packet loss ratio, and the burstiness of the sending rate. For this scenario the burstiness has been evaluated over the last 100 s of simulation during which the available bandwidth is constant in order to discard the rate variability due to the interacting ON-OFF CBR source. ARC and TFRC basically achieve similar goodput and packet loss ratios, which are slightly larger than those obtained by Reno TCP. Moreover, ARC provides the smallest burstiness index.

5.1.3 Twenty rate-based or twenty TCP connections and one ON-OFF CBR source

This section investigates the behavior of 20 Reno TCP or 20 ARC or 20 TFRC flows sharing a 10-Mbps bottleneck (Fig. 4) in the presence of abrupt changes of the available bandwidth caused by an ON-OFF CBR source. The ON-OFF CBR source transmits at 7 Mbps during the ON period, which lasts 200 s, and is silent during the OFF period, which also lasts 200 s. Also, in this case the UDP source provokes 1:3 available bandwidth variations such as in the scenario considered in [5].

Figures 14a–c show the transmission rates of ARC, TFRC, and Reno TCP during the time interval [350 s,450 s], when there is a sudden increase of the available bandwidth left unused by the CBR source. They show that both ARC and TFRC track the available bandwidth within similar rise times, whereas Reno TCP exhibits a bursty transmission rate. The main difference between ARC and TFRC is that TFRC rates are much more spread out than those provided by ARC, i.e., ARC is fairer than TFRC in bandwidth sharing.

Figures 15a–c show the transmission rates of ARC, TFRC, and Reno TCP during the time interval [150 s,250 s], after a sudden decrease of the available bandwidth. These figures confirm that Reno TCP exhibits a bursty behavior, whereas ARC and TFRC gracefully adapt the transmission rate to the available bandwidth.

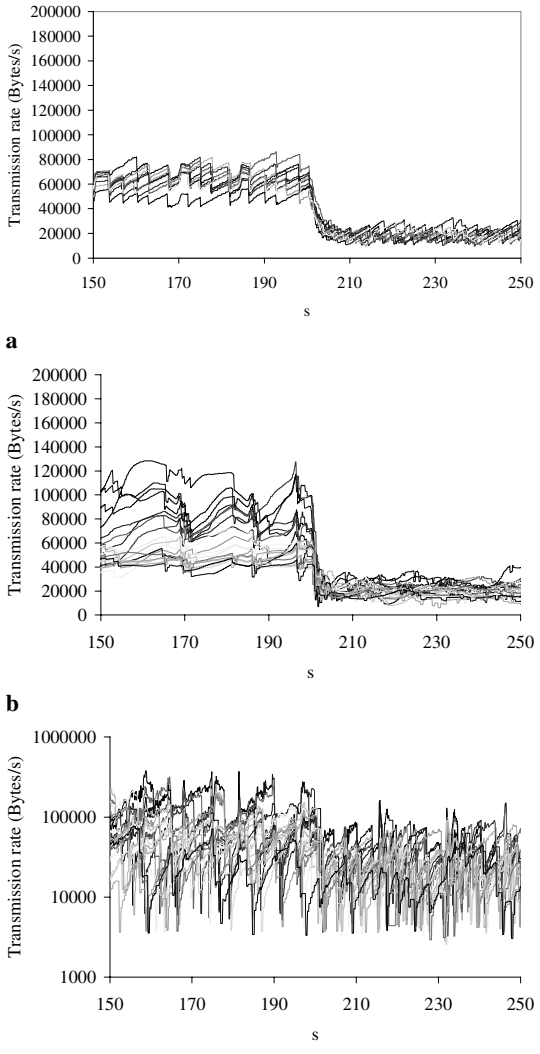


Fig. 15. Transmission rates of 20 connections when the CBR source is suddenly turned on. (**a**) The 20 connections are ARC. (**b**) The 20 connections are TFRC. (**c**) The 20 connections are Reno TCP

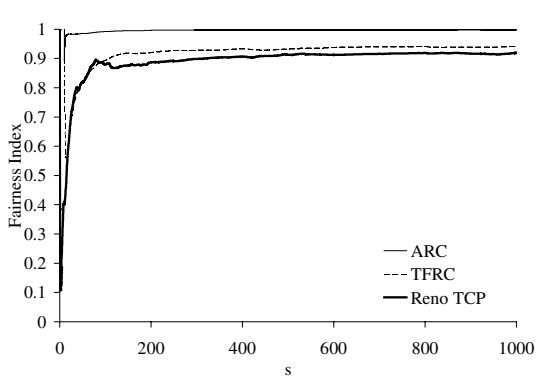


Fig. 16. Fairness index of 20 Reno TCP or ARC or TFRC connections sharing a 10-Mbps bottleneck with an ON-OFF CBR source

To provide further insight into the fairness in bandwidth allocation provided by Reno TCP, TFRC, and ARC, we evaluate the Jain fairness index $J_{FI}(t) = \frac{(\sum_{i=1}^{20} b_i(t))^2}{20 \cdot \sum_{i=1}^{20} b_i^2(t)}$, where $b_i(t)$ is the goodput achieved by the i-th connection during the time interval $[0, t]$. Moreover, we define the instantaneous Jain fairness index $IJ_{FI}(t) = \frac{(\sum_{i=1}^{20} r_i(t))^2}{20 \cdot \sum_{i=1}^{20} r_i^2(t)}$, where $r_i(t)$ is the transmission rate of the i-th connection at time t. The $J_{FI}(t)$ index evaluates the long-term fairness, whereas the $IJ_{FI}(t)$ index evaluates the short-term fairness. Both indices belong to the interval $[0, 1]$. An index equal to 1 indicates maximum degree of fairness. Figure 16 shows the Jain fairness index vs. time. During the first 20 s, Reno TCP and TFRC exhibit a significant degree of unfairness. At the end of the simulation their fairness indices reach the acceptable value of 0.9. On the other hand, ARC reaches the maximum value of the Jain index after 10 s.

Figure 17 shows that the instantaneous fairness indices provided by ARC vary in the range [0.9,1] whereas those of TFRC and Reno TCP oscillate in the range [0.7,0.95]. To summarize these investigations, ARC reaches the steady state fair bandwidth allocation faster than TFRC and Reno TCP (Fig. 16) and provides a higher degree of fairness in bandwidth sharing on shorter time scales (Fig. 17).

5.1.4 Many rate-based connections sharing a bottleneck

This section investigates the behavior of M ARC or M TFRC or M Reno TCP connections sharing a 10-Mbps bottleneck, with M ranging from 20 to 200. We focus on (a) bottleneck utilization, which is the total goodput of the connections along the forward path over the bottleneck link capacity; (b) intraprotocol fairness in bandwidth sharing; and (c) packet loss ratio for various degrees of statistical multiplexing. Moreover, the impact of the bottleneck buffer size on these indices is also investigated.

Figure 18 shows that ARC improves fairness in bandwidth sharing with respect to Reno TCP and TFRC. This is mainly due to the probing phase of ARC, which does not depend on the connection RTT such as in the case of Reno TCP and TFRC. Note that the fairness index of TFRC decreases when the number of connections sharing the bottleneck is larger than 70. This means that the TFRC fairness degrades in the presence of high loss ratios. This behavior has also been reported in [35] and will be confirmed later when we investigate the impact of the bottleneck buffer size.

Figure 19 reports the total goodput as a function of the number of connections sharing the 10-Mbps bottleneck. Again, Fig. 19 shows that both ARC and TFRC slightly improve the goodput with respect to Reno TCP.

Figure 20 shows that ARC, TFRC, and Reno TCP achieve similar loss ratios that grow with the number of connections sharing the bottleneck. The reason for this behavior is that many connections are more aggressive than few when probing for available bandwidth.

Finally, we have investigated the sensitivity of ARC, TFRC, and Reno TCP with respect to the bottleneck buffer size. This investigation is important since TFRC computes the transmission rate using the packet loss ratio, which is affected by the buffer size. For that purpose, we have considered 40

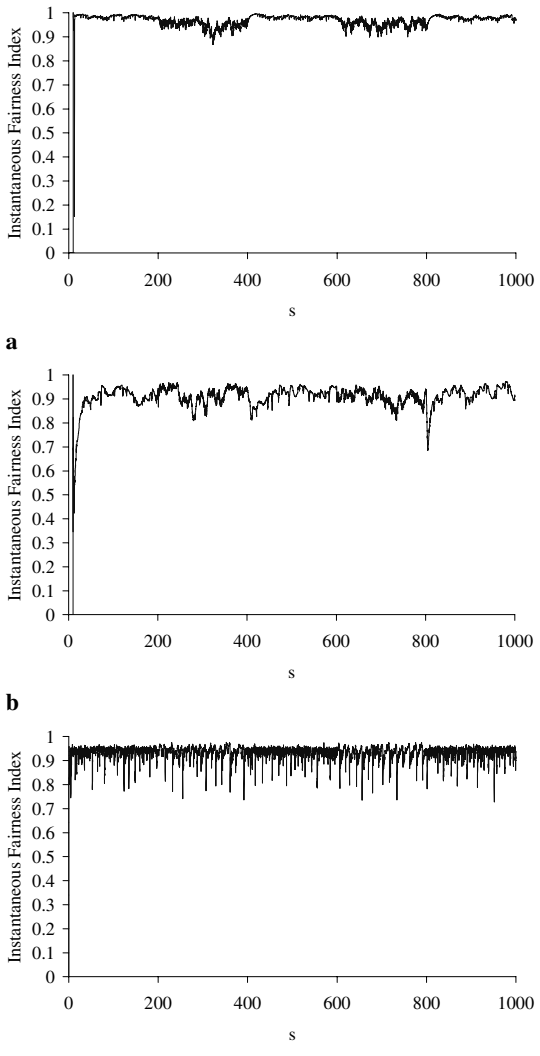


Fig. 17. Instantaneous fairness index of 20 connections sharing a 10-Mbps bottleneck with an ON-OFF CBR source. (**a**) The sources are ARC. (**b**) The sources are TFRC. (**c**) The sources are Reno TCP

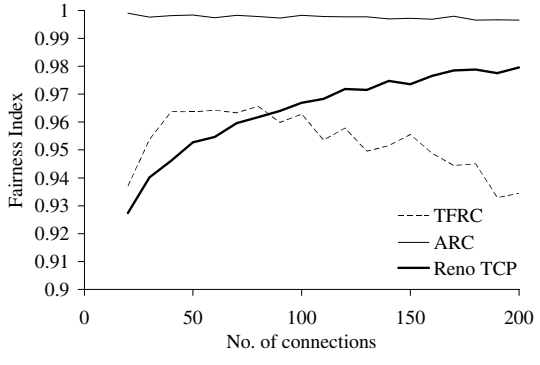


Fig. 18. Fairness indices as a function of the number of coexisting connections

long-lived connections sending data over a 10-Mbps bottleneck. The buffer size has been varied from 0.1 to 1.5 times the bandwidth delay product, which is equal to 200 packets.

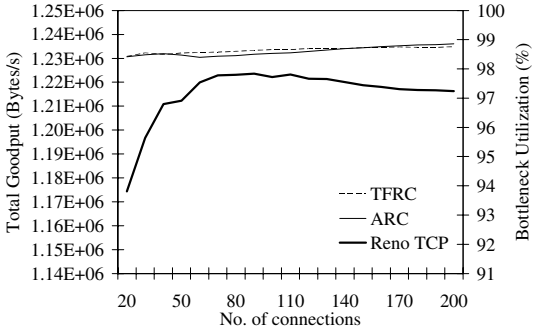


Fig. 19. Total goodput as a function of the number of coexisting connections

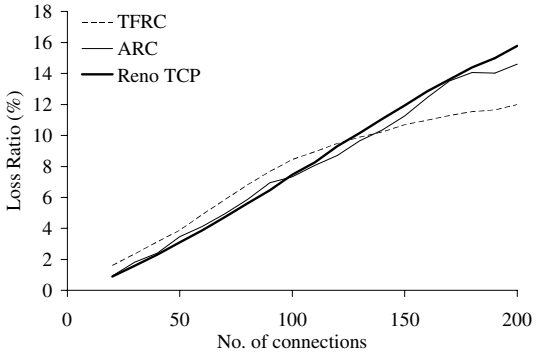


Fig. 20. Packet loss ratio as a function of the number of coexisting connections

Figure 21 shows the total goodput achieved by the 40 connections as a function of the bottleneck buffer size. Both TFRC and ARC provide similar goodputs, which do not depend on the buffer size. On the other hand, Reno TCP achieves the smallest goodput when the buffer size is smaller that 0.3 times the bandwidth delay product. The reason is that a small buffer is not able to absorb packet bursts generated by the TCP Reno connections.

For buffer size larger than 0.7 times the bandwidth delay product, the total goodput of Reno TCP gracefully diminishes due to the larger queueing delay that increases the RTT and slows down both congestion avoidance and slow-start phases [17].

Figure 22 shows the packet loss ratio as a function of the buffer size. TFRC provokes a larger fraction of lost packets with respect to both Reno and ARC when the buffer size is

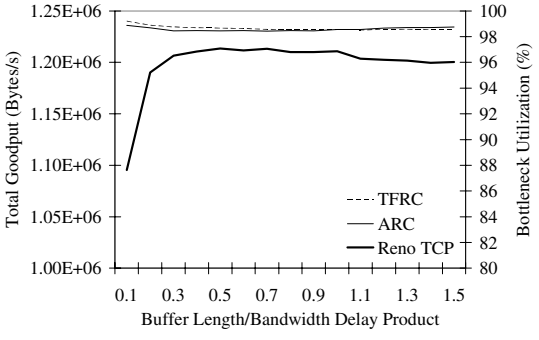


Fig. 21. Total goodput as a function of the buffer size when 40 connections share a 10-Mbps bottleneck

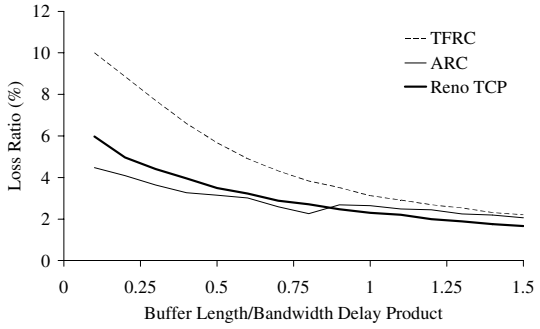


Fig. 22. Packet loss ratio as a function of the buffer size when 40 connections share a 10-Mbps bottleneck

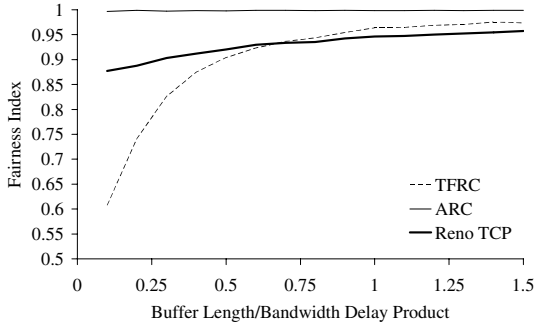


Fig. 23. Fairness index as a function of the buffer size when 40 connections share a 10-Mbps bottleneck

smaller than 0.7 times the bandwidth delay product. Figure 23 shows that TFRC becomes unfair when the buffer capacity is smaller than 0.7 times the bandwidth delay product. Moreover, ARC provides the largest fairness index for any buffer size.

To summarize, Figs. 20–22 show that the performance of the ARC algorithm is less sensitive than Reno TCP and TFRC with respect to buffer size. In particular, in the presence of small buffers TFRC exhibits a significative loss ratio and a small fairness index.

5.1.5 One rate-based and one Reno TCP connection with one ON-OFF CBR source

So far we have investigated intraprotocol behavior of ARC, TFRC, and Reno TCP over a single-bottleneck scenario for various degrees of statistical multiplexing. Now we investigate also interprotocol friendliness of ARC and TFRC toward Reno TCP, i.e., the interaction between one ARC or one TFRC source with one Reno TCP source. For that purpose, the single-bottleneck topology shown in Fig. 4 is considered where the bottleneck is 1 Mbps. We consider an ON-OFF CBR source that transmits at 0.7 Mbps during the 200-s ON period and is silent during the 200-s OFF period.

Figure 24 shows the transmission rates of TFRC and ARC and the fair share, which is 0.5 Mbps when the UDP is silent and 0.15 Mbps when the UDP is on. TFRC exhibits many dips in the transmission rate, whereas the oscillation range of the ARC rate is smaller.

Figure 25 reports the goodput, the packet loss ratio, and the burstiness measured in the scenarios considered above. They show that (i) Reno TCP achieves the same goodput when it

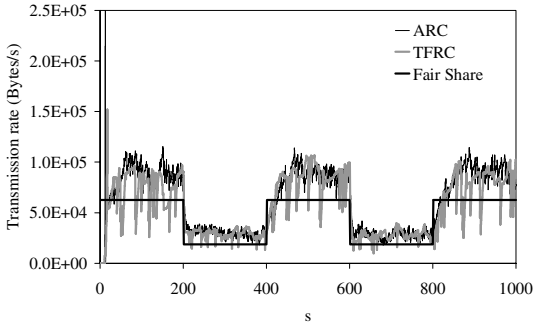


Fig. 24. One ARC or one TFRC sharing a 1-Mbps bottleneck with one Reno source and one ON-OFF UDP source

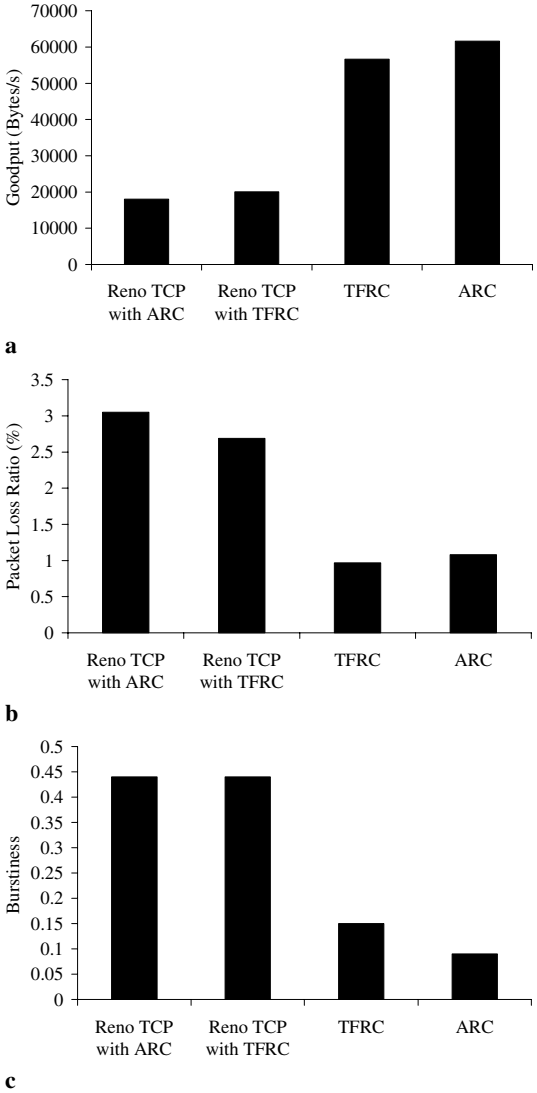


Fig. 25. One Reno TCP source share a 1-Mbps bottleneck with one ON-OFF CBR source and one TFRC or one ARC source. (**a**) Goodputs. (**b**) Packet loss ratios. (**c**) Burstiness

shares the bottleneck with ARC or with TFRC, (ii) the total goodput, which is the sum of the Reno with the ARC or the TFRC goodput, is roughly the same for both scenarios, and (iii) ARC exhibits the lowest burstiness index.

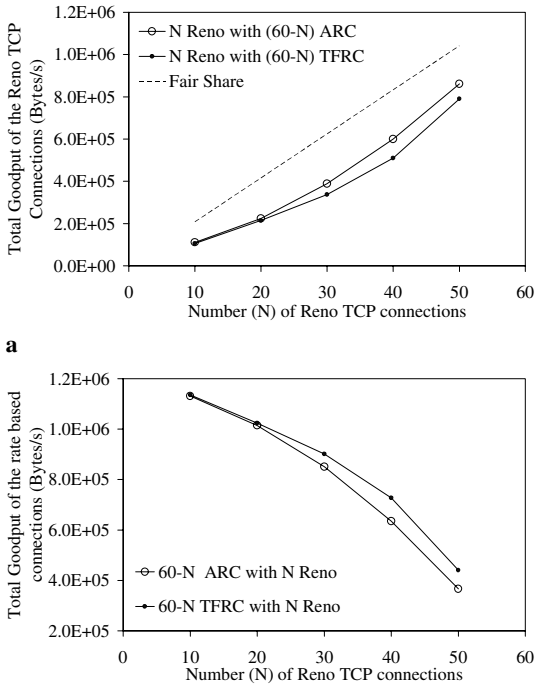


Fig. 26. N Reno TCP and $(60-N)$ ARC or $(60-N)$ TFRC long-lived connections sharing a 10-Mbps bottleneck. (**a**) Total goodput of Reno connections. (**b**) Total goodput of rate-based connections

5.1.6 Many rate-based connections mixed with many Reno TCP connections (interprotocol friendliness)

To evaluate interprotocol friendliness between ARC and Reno TCP and between TFRC and Reno TCP, we consider M rate-based connections and N Reno TCP connections sharing a 10-Mbps bottleneck, with $N+M=60$ and M ranging from 10 to 50. For all mixes, we have measured a bottleneck utilization larger than 99%. Figure 26a plots the total goodput of the N Reno TCP connections, which is obtained by summing the goodputs of the N Reno TCP connections. Also, the curve of the fair share rate, which is equal to $(Bottleneck\_Capacity \cdot N)/60$, is depicted. Figure 26a shows that when N Reno TCP connections share the 10-Mbps bottleneck with $(60-N)$ ARC connections, they get more bandwidth share than in the case of sharing the bottleneck with $(60-N)$ TFRC connections. This effect is more evident in the presence of a large number of Reno TCP connections. The reason why ARC *is friendlier* than TFRC toward Reno TCP is that ARC strictly mimics the real-time dynamics of Reno TCP by using Eq. 1 and settings given by Eqs. 5 and 6, whereas TFRC mimics only the *long-term* behavior of the TCP. Figure 26b reports the goodput achieved by the $(60-N)$ rate-based connections.

5.2 Multihop scenario

To investigate the ARC algorithm in a more complex scenario, we consider the multihop topology depicted in Fig. 27, which is a more realistic model of the real Internet. It is characterized by (i) N hops, (ii) one persistent connection C_1 going through all the N hops, and (iii) $2N$ persistent sources

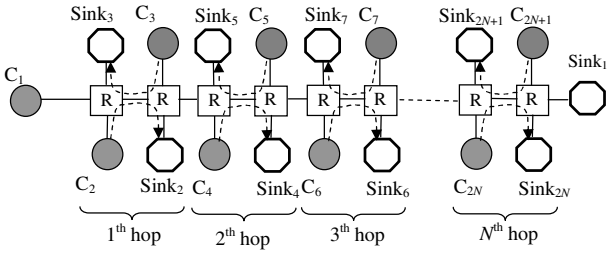


Fig. 27. Multihop scenario

$C_2, C_3, C_4 \ldots C_{2N+1}$ of cross traffic transmitting data over every single hop.

The simulation lasts 1000 s during which the cross traffic sources always send data. The connection C_1 starts data transmission at time $t = 10$ s when all the network bandwidth has been grabbed by the cross traffic sources starting at $t = 0$ s. The capacity of the entry/exit links is 100 Mbps, that of the links between the routers is 1 Mbps, and link propagation delays are equal to 10 ms. Queue sizes have been set equal to 12 packets, which correspond to the bandwidth delay product of a typical RTT of 150 ms. Notice that the described scenario is a "worst case" scenario for the source C_1 since (i) C_1 starts data transmission when all the network bandwidth has been grabbed by the cross traffic sources and (ii) C_1 has the longest RTT and experiences drops at each router it goes through.

We consider the following three scenarios:

Scenario 1. All traffic sources are controlled by the same control algorithm. This is a homogeneous scenario aiming at evaluating TFRC, ARC, and Reno in absolute terms. Figure 28a shows that in the presence of homogeneous cross traffic, the connection C_1 achieves the worst goodput when it is controlled by TFRC, whereas ARC and Reno achieve similar goodputs. Reno TCP and ARC exhibit similar behaviors because they are the rate-based and window-based versions of the same sliding window algorithm (Sects. 3 and 4). On the other hand, the connection C_1 is not able to grab an acceptable share of the network capacity when it is controlled by TFRC. For instance, for $N \geq 5$, TFRC provides goodputs that are two orders of magnitude smaller than those provided by TCP or ARC. Figure 28b reports the average goodputs of the $C_2, C_4 \ldots C_{2N}$ connections. It shows that the sources of TFRC cross traffic get almost all the network bandwidth so that the C_1 TFRC source is basically unable to send data when the number of hops it goes through is larger than 5. This result suggests that *TFRC could lead to starvation of connections traversing many hops*. Figure 28c shows the total goodput computed as goodput of C_1 connection+average goodput of $C_2, C_4 \ldots C_{2N}$ connections. Again, both ARC and TFRC slightly improve the goodput with respect to Reno TCP; however, TFRC is not fair.

Scenario 2. The $C_2, C_3, C_4 \ldots C_{2N+1}$ sources of cross traffic are controlled by Reno TCP, whereas the C_1 connection is controlled by TFRC, ARC, or Reno, respectively. This scenario aims at comparing TFRC, ARC, and Reno behaviors when going through an Internet dominated by Reno traffic. In other words, this scenario allows us to investigate the ability of TFRC and ARC to grab the network bandwidth when competing with Reno cross traffic, i.e., the friendliness of Reno

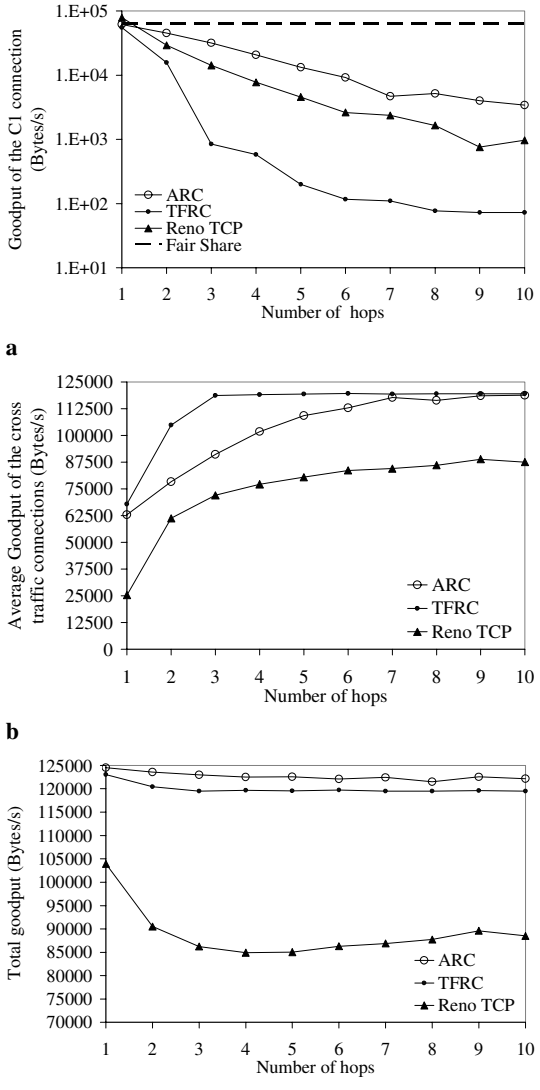


Fig. 28. Homogeneous N-hop scenario with N varying from 1 to 10. (**a**) Goodput of C_1 connection. (**b**) Average goodput of $C_2, C_4 \ldots C_{2N}$ connections. (**c**) Total goodput

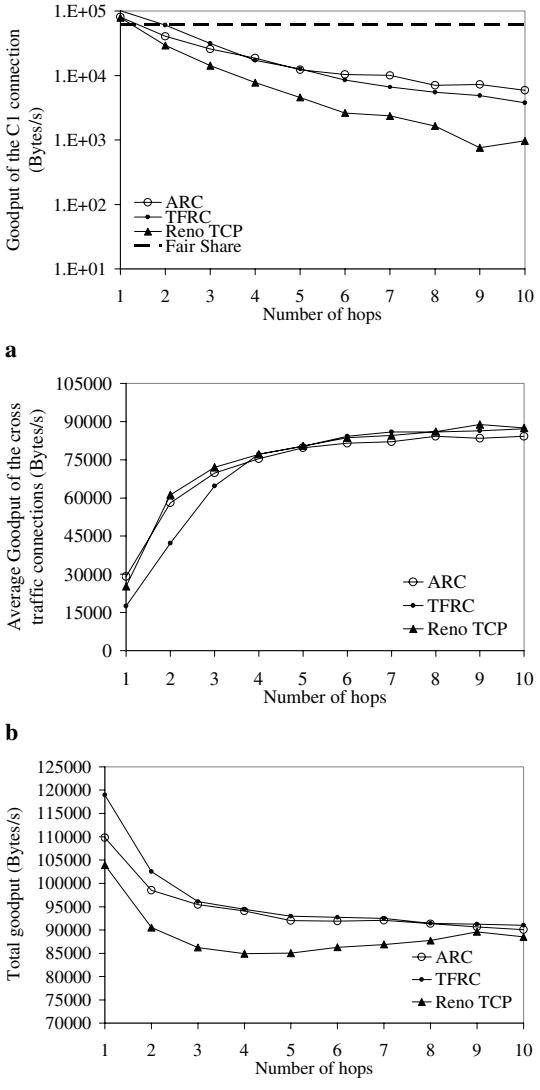


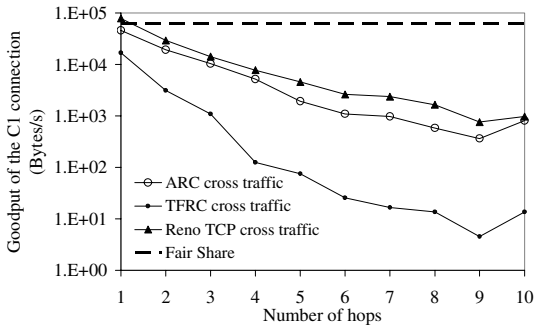
Fig. 29. The C_1 connection is controlled by Reno TCP or ARC or TFRC and goes through multiple congested gateways in the presence of Reno cross traffic. (**a**) Goodput of C_1 connection. (**b**) Average goodput of $C_2, C_4 \ldots C_{2N}$ connections. (**c**) Total goodput

TCP toward ARC and TFRC. Figure 29a depicts the goodput of the C_1 connection when it is controlled by a TFRC, ARC, or Reno algorithm. It shows that all the considered control algorithms exhibit similar behaviors, which means that the Reno TCP cross traffic sources allow the joining C_1 connection to get an acceptable share of the network capacity. When the number of traversed hops is larger than 5, ARC improves the goodput *with respect to* Reno TCP and TFRC. This result can be explained by noting that the RTT of the C_1 connection increases with N. Therefore, since the goodputs of TFRC or Reno TCP are inversely proportional to RTT [27], the goodput of C_1 diminishes when TFRC or Reno TCP is employed.

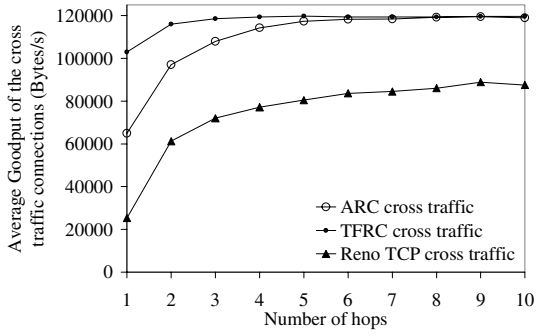
Figures 29b and c show that the average goodput of the $C_2, C_4 \ldots C_{2N}$ connections and the total goodput are similar using the three algorithms. This result is due to the fact that the simulated scenarios differ only in the control algorithm of C_1, which has a negligible impact on the overall network performance.

Scenario 3. The sources of cross traffic $C_2, C_3, C_4 \ldots C_{2N+1}$ are controlled by TFRC, ARC, or Reno, whereas the C_1 connection is Reno. This scenario aims at evaluating the behaviors of Reno when going through an Internet dominated by TFRC, ARC, or Reno traffic, that is, it aims at investigating the friendliness toward Reno of TFRC and ARC cross traffic. In fact, in the case of a wide deployment of TFRC or ARC or in the case of a path where TFRC or ARC is heavily used, a joining Reno TCP flow must get an acceptable share of network capacity.

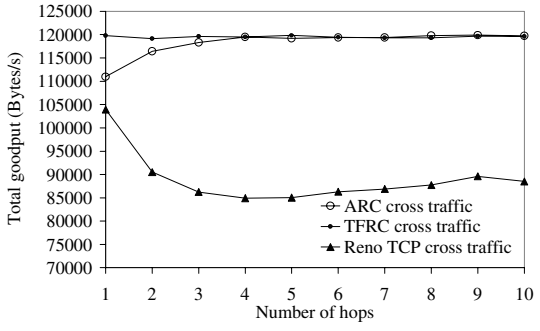
Figure 30a reports the goodputs of the C_1 Reno connection in this scenario. Reno TCP achieves a very poor goodput in the presence of TFRC cross traffic, that is, TFRC *reveals itself not to be friendly* toward Reno in a multihop scenario. On the other hand, Reno achieves similar goodput when the cross traffic is of the ARC or Reno type. To summarize, the latter and the former cases show not only that a connection controlled by the ARC algorithm is able to grab its bandwidth share when



a



b



c

Fig. 30. C_1 is Reno TCP, whereas the cross traffic is ARC, TFRC, or Reno TCP. (**a**) Goodput of C_1 connection. (**b**) Average goodput of $C_2, C_4 \ldots C_{2N}$ connections. (**c**) Total goodput

competing with many Reno sources (Fig. 29a) but also that ARC-type cross traffic is friendly toward Reno TCP (Fig. 30a). The average goodputs of the cross traffic sources (Fig. 30b) and the total goodputs (Fig. 30c) are similar to those observed in the case of a homogeneous scenario (Fig. 28).

Scenario 4. We consider the N-hop scenario depicted in Fig. 27, where $N = 10$, the last hop connecting the $Sink_1$ is a lossy wireless link, and the $C_2, C_3, C_4 \ldots C_{2N+1}$ cross traffic is Reno. This scenario represents a mixed wired/wireless topology where a flow goes through many congested gateways in the presence of concurrent Reno traffic and finally goes through an unreliable wireless last hop. We assume that independent uniformly distributed packet losses affect the wireless link in both directions. We vary the packet loss probability of the wireless link from 0.1% to 10%. Figure 31 shows the C_1 goodputs obtained using Reno, ARC, and TFRC. In this case, ARC provides a goodput improvement ranging from 300% up to 2500% with respect to Reno and from 50% to 1000% with

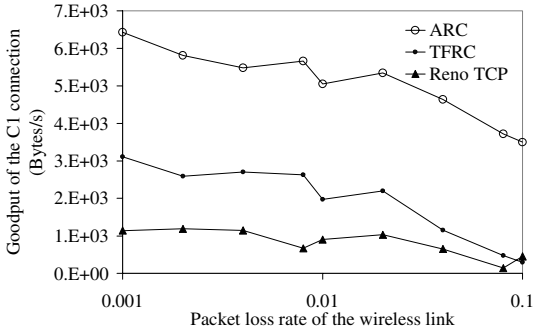


Fig. 31. Goodputs of the C_1 connection when controlled by ARC, TFRC, or Reno and going through ten multiple congested gateways in the presence of Reno cross traffic. The last hop is a lossy wireless link

respect to TFRC. The reason for such an improvement is that ARC adaptively reduces the window w by taking into account an estimate of the available bandwidth (Eq. 2). This mitigates the impact of random losses not due to congestion. On the other hand, Reno TCP and TFRC interpret packet losses as a symptom of congestion and reduce the transmission rate by following the multiplicative decrease mechanism and the throughput equation model of Reno, respectively.

6 Conclusions

This paper has proposed an adaptive rate-based congestion control algorithm for streaming flows over the Internet. The algorithm has been designed by following a control theoretical approach. Computer simulations using *ns*-2 have been developed to compare ARC with Reno TCP and with TCP-friendly rate control (TFRC) algorithm. Single- and multibottleneck scenarios with and without lossy links and in the presence of homogeneous and heterogeneous traffic sources have been considered. Simulations have shown that ARC improves fairness and is friendly toward Reno. On the other hand, TFRC *revealed itself not to be friendly toward Reno*. The reason why ARC is friendlier than TFRC toward Reno TCP is that ARC strictly mimics the real-time dynamics of Reno TCP, whereas TFRC mimics only the *long-term* behavior of the TCP. Finally, simulations have shown that ARC remarkably improves the goodput with respect to TFRC and Reno in the presence of wireless lossy links.

Acknowledgements. This work was supported by the MIUR-FIRB project no. RBNE01BNL5 "Traffic Models and Algorithms for Next Generation IP Network Optimization (TANGO)". We thank Cormac J. Sreenan, Klara Nahrstedt, and anonymous reviewers for their suggestions, which allowed us to greatly improve the quality of the paper. We also thank Prof. M. Ajmone Marsan for encouraging and supporting this research.

References

1. Aagarwal A, Savage S, Anderson T (2000) Understanding the performance of TCP Pacing. In: Proceedings of IEEE INFOCOM '2000, Tel-Aviv, Israel, March 2000, pp 1157–1165

2. Allman M, Paxson V, Stevens WR (1999) TCP congestion control. RFC 2581, April 1999
3. Astrom KJ, Wittenmark B (1995) Computer controlled systems: theory and design, 3rd edn. Prentice-Hall Information and System Sciences series. Prentice-Hall, Englewood Cliffs, NJ
4. Bansal D, Balakrishnan H (2001) Binomial congestion control algorithm. In: Proceedings of IEEE INFOCOM 2001, Anchorage, AK, April 2001, pp 631–640
5. Bansal D, Balakrishnan H, Floyd S, Shenker S (2001) Dynamic behavior of slowly-responsive congestion control algorithms. In: Proceedings of ACM SIGCOMM 2001, San Diego, August 2001, pp 263–273
6. Bolot JC, Turletti T (1998) Experience with control mechanisms for packet video in the Internet. ACM SIGCOMM Comput Commun Rev 28:4–15
7. Brakmo LS, Peterson L (1995) TCP Vegas: end-to-end congestion avoidance on a global Internet. IEEE J Select Areas Commun 13(8):1465–1480
8. De Cuetos P, Ross KW (2002) Adaptive rate control for streaming stored fine-grained scalable video. In: Proceedings of ACM NOSSDAV'02, Miami, FL, May 2002, pp 3–12
9. Grieco LA, Mascolo S (2004) Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control. ACM Comput Commun Rev (to appear)
10. Floyd S, Fall K (1999) Promoting the use of end-to-end congestion control in the Internet. IEEE/ACM Trans Netw 7(4):458–472
11. Floyd S, Henderson T (1999) NewReno modification to TCP's fast recovery. RFC 2582, April 1999
12. Floyd S, Handley M, Padhye J, Widmer J (2000) Equation-based congestion control for unicast application. In: Proceedings of ACM SIGCOMM 2000, Stockholm, Sweden, August 2000, pp 43–56
13. Gerla M, Locigno R, Mascolo S, Weng R (2002) Generalized window advertising for TCP congestion control. Eur Trans Telecommun (6):1–14
14. Grieco LA, Mascolo S (2002) Westwood TCP and easy RED to improve fairness in high speed networks. In: Proceedings of the IFIP/IEEE 7th international workshop on protocols for high-speed networks, Berlin, April 2002, pp 130–146
15. Grieco LA, Mascolo S (2003) Performance comparison of Reno, Vegas and Westwood+ TCP congestion control. Technical Report 07/03/S, Politecnico di Bari
16. Handley M, Floyd S, Padhye J, Widmer J (2003) TCP friendly rate control (TFRC): protocol specification. RFC 3448, January 2003
17. Jacobson V (1988) Congestion avoidance and control. In: Proceedings of ACM SIGCOMM '88, Stanford, CA, August 1988, pp 314–329
18. Khafizov F, Yavuz M (2002) Running TCP over IS-2000. In: Proceedings of the IEEE international conference on communications (ICC 2002), New York, April 2002, pp 3444–3448
19. Kim T, Bharghavan V (1999) Improving congestion control performance through loss differentiation. In: Proceedings of the international conference on computer and communications networks, Boston, October 1999, pp 412–418
20. Li SQ, Hwang C (1995) Link capacity allocation and network control by filtered input rate in high speed networks. IEEE/ACM Trans Netw 3(1):10–25
21. Mascolo S (1999) Congestion control in high-speed communication networks using the Smith principle. Automatica 35:1921–1935
22. Mascolo S (2003) Modeling the Internet congestion control as a time delay system: a robust stability analysis. In: Proceedings of the IFAC workshop on time-delay systems, Inria, Rocquencourt, September 2003
23. Mascolo S, Casetti C, Gerla M, Sanadidi M, Wang R (2001) TCP Westwood: end-to-end bandwidth estimation for efficient transport over wired and wireless networks. In: Proceedings of ACM MOBICOM 2001, Rome, Italy, July 2001, pp 287–297
24. Mathias M, Semke J, Mahdavi J, Ott T (1997) The macroscopic behavior of the TCP congestion avoidance algorithm. ACM SIGCOMM Comput Commun Rev 27:67–82
25. Mo J, La RJ, Anantharam V, Walrand J (1999) Analysis and comparison of TCP Reno and Vegas. In: Proceedings of IEEE INFOCOM 1999, New York, March 1999, pp 1556–1536
26. Ns-2 Network simulator http://www.isi.edu/nsnam/ns/
27. Padhye J, Firoiu V, Towsley D, Kurose J (1998) Modeling TCP throughput: a simple model and its empirical validation. In: Proceedings of ACM SIGCOMM '98, Vancouver, BC, Canada, September 1998, pp 303–314
28. Rejaie R, Reibman A (2001) Design issues for layered quality-adaptive internet video playback. In: Proceedings of the international workshop on digital communications, Taormina, Italy, September 2001
29. Rejaie R, Handley M, Estrin D (1999) RAP: an end-to-end rate-based congestion control mechanism for realtime streams in the Internet. In: Proceedings of IEEE INFOCOM 1999, New York, March 1999, pp 1337–1345
30. Rhee I, Ozdemir V, Yi Y (2000) TEAR: TCP emulation at receivers: flow control for multimedia streaming. Technical report, NCSU, April 2000
31. Tian W, Zakhor A (1999) Real-time Internet video using error resilient scalable compression and TCP-friendly transport protocol. IEEE Trans Multimedia 1:172–186
32. Turletti T, Huitema C (1996) Videoconferencing on the internet. IEEE/ACM Trans Netw 4(3):340–351
33. Vojnović M, Boudec JL (2002) On the long-run behavior of equation-based rate control. In: Proceedings of ACM SIGCOMM 2002, Pittsburgh, August 2002, pp 103–116
34. Wang Y, Claypool M, Zuo Z (2001) An empirical study of real video performance across the Internet. In: Proceedings of the ACM SIGCOMM workshop on Internet measurement, San Francisco, November 2001, pp 295–309
35. Yang YR, Kim MS, Lam S (2001) Transient behaviors of TCP-friendly congestion control protocols. In: Proceedings of IEEE INFOCOM 2001, Anchorage, AK, April 2001, pp 22–26