ADAPTIVE RATE CONTROL FOR INTERNET VIDEO STREAMING

Luigi A. Grieco, Saverio Mascolo

Dipartimento di Elettrotecnica ed Elettronica,Politecnico di Bari,
Via Orabona, 4 – 70125 BARI, Italy
e-mail {a.grieco, mascolo}@poliba.it

Keywords: Congestion Control, Audio/Video streaming, Internet protocols, Smith's Principle

Abstract

The main contribution of this paper is an end-to-end rate-based congestion control algorithm for unicast quality adaptive video streaming that we call Adaptive Rate Control (ARC). The algorithm is based on end-to-end estimation of both available bandwidth and queue backlog. ARC has been compared with the TCP-Friendly Rate Control (TFRC) algorithm via the *ns*-2 simulator. In particular, both single bottleneck and multi-hop scenarios with and without lossy links have been considered. Simulation results have shown that ARC: (1) is more friendly than TFRC towards Reno; (2) remarkably improves the goodput with respect to TFRC and Reno TCP in the presence of lossy links; (3) provides no rate oscillations in the presence of stationary network load; (4) exhibits a less oscillating rate dynamics with respect to Reno TCP.

1 Introduction

Integration of congestion control with quality adaptation is the key strategy to provide efficient video delivering over time varying capacity networks, such as the Internet, so that when the available bandwidth is scarce a low quality video is transmitted whereas, when an abundant bandwidth is available, video of improved quality is delivered [17]. Classic Reno TCP produces rapidly varying transmission rates due to its window based nature [9, 1]. Thus it is not well suited for video delivering since it requires a large playout buffer at the receiver to provide a smooth video playback [5]. A congestion control algorithm well suited for video delivering should provide a smooth dynamics of the transmission rate and be able to share the network capacity with Reno sources, i.e. it has to be Reno *friendly*.

The concept of *slow responsive algorithm* has been recently introduced to deal with applications such as video streaming where a relative smooth sending rate is of importance [8]. However, recent works indicate that TCP can be still used to transport video contents (see [10]) if the playback buffer is enough wide [5]. In particular, experiments illustrated in [5] show that even if quality fluctuations provided by Reno TCP or TFRC lie in the same range, Reno TCP generates a playback delay 3 times larger than the one provided by TFRC.

Many control algorithms have been proposed that try to emulate the ”long-term” behavior of the Reno algorithm with a more moderate rate dynamics [6, 16, 17, 3, 11]. The most considered among them is the TCP Friendly Rate Control (TFRC) [6], which exploits the equation model of the Reno throughput developed in [15] to compute the transmission rate. In this way, the TFRC sender computes the transmission rate as a nonlinear function of the average loss rate, which is supplied by the receiver as feedback report. From a control theoretic perspective, TFRC employs a *static nonlinear* controller to regulate the input rate of a data connection, which is a time delay system [12]. In [3] the rate based algorithms proposed in [16, 6] and the windows based algorithms proposed in [1, 2] have been tested in the presence of dynamic network conditions to show that algorithms that do not employ the self-clocking principle [9] 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. Moreover, to recover the disastrous effects due to the violation of the self-clocking principle, an enhanced version of the TFRC algorithm has been also proposed in [3]. This paper proposes a new rate-based algorithm to be used in a general video delivering system. 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. It has been designed starting from the control theoretic analysis developed in [12], which shows that it is possible to design a stable and efficient congestion controller by following the Smith principle. ARC has been compared with the TCP Friendly Rate Control (TFRC) over single bottleneck and multi-hop scenarios, using the *ns*-2 simulator [14]. Main results that have been found are that ARC: (1) is more friendly than TFRC towards Reno; (2) remarkably improves the goodput with respect to TFRC and Reno TCP in the presence of lossy links; (3) provides no rate oscillations in the presence of stationary network load; (4) exhibits a less oscillating rate dynamics with respect to Reno TCP.

The paper is organized as follows: Section 2 summarizes the theoretical results that have been used as starting points for designing ARC; Section 3 describes ARC; Section 4 shows simulation results and, finally, the last section draws the conclusions.

2 Control theoretical background

The ARC algorithm proposed in this paper has been designed starting from the theoretical results derived in [12], which show that a data connection is a time delay system that can be efficiently controlled by following the Smith principle. In particular, to provide stability and high utilization of the bottleneck link depicted in Fig. 1, the following rate-based control equation (1) should be employed:

$$r(t) = k[w(t) - q(t - T_{fb}) - \int_{t-RTT}^{t} r(\tau)d\tau]^{+} \qquad (1)$$

where: $[x]^{+} = max\{0, x\}$; $r(t)$ is the transmission rate; $w(t)$ represents a threshold for the queue length; $\int_{t-RTT}^{t} r(\tau)d\tau$ represents the packets sent by the source and not yet acknowledged by the receiver, i.e. the outstanding packets; T_{fw} is the forward delay that models the propagation from the sender to the the bottleneck; T_{fb} is the backward delay that models the time that the feedback $q(t)$, which is supplied by the bottleneck, needs to reach the destination and then back to the source; $q(t)$ is the bottleneck queue backlog that is received by the sender after the delay T_{fb}; k is the proportional gain that relates the transmission rate $r(t)$ to the quantity $[w(t) - q(t - T_{fb}) - \int_{t-RTT}^{t} r(\tau)d\tau]$. It is easy to give an intuitive interpretation of the 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 bottleneck backlog $q(t-T_{fb})$ with the number of outstanding packets $\int_{t-RTT}^{t} r(\tau)d\tau$. From Eq. (1) it turns out that when the number of outstanding packets plus $q(t - T_{fb})$ is greater 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 the 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 $\Delta W = r/k$ is easily obtained. This result will be exploited later to define a proper dynamic setting of $w(t)$, which mimics the Reno behavior and provides friendliness. Moreover, Eq. (1) can provide a steady state queue length equal to 0 by setting the control window w as follows:

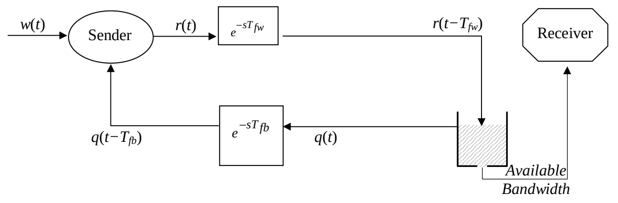


Figure 1: Schematic of a connection.

$$w = B \cdot (mRTT + \frac{1}{k}) \qquad (2)$$

where B is the bottleneck available bandwidth and $mRTT$ is the minimum RTT [12]. An important feature of the setting (2) is that it clears out all the buffers along the connection path, thus improving statistical multiplexing of flows going through FIFO buffers and increasing fairness in bandwidth allocation.

3 The Adaptive Rate Control algorithm

The starting point of the ARC design is the result that Eq. (1) is the rate based form of the classic sliding window control [12], which is employed by Reno TCP and its variants [1, 9]. In particular, we propose to set the control window $w(t)$ in Eq. (1) by following the linear increase behavior of the Reno congestion avoidance phase. Moreover, we propose to adaptively set $w(t)$ after congestion by taking into account an end-to-end estimate of the available bandwidth as dictated by Eq. 2. It follows a description of the main functionalities implemented at the ARC sender and receiver.

3.1 The sender

This section reports details of ARC functionalities implemented at the sender side. The ARC sender computes the transmission rate r using the Eq. (1) in an end-to-end fashion. In order to perform this task, the sender (1) estimates the queue backlog, (2) properly manages the control window $w(t)$ to obtain friendliness towards Reno, (3) implements a timeout mechanism to react also to strong network congestion.

3.1.1 Queue backlog estimation

In order to estimate the queue backlog in end-to-end fashion, we propose to consider the relation between the queuing time T_q, the queue length and the queue depletion rate B, which is:

$$q(t - T_{fb}) = B \cdot T_q.$$

The queuing delay can be computed by monitoring the (RTT) via packet time stamping. In fact, the difference between the actual RTT and the minimum RTT provides an estimate of the queuing time:

$$T_q = RTT - mRTT.$$

3.1.2 Updating the Control Window

The algorithm used to update the control window w is crucial to provide friendliness towards Reno TCP sources. For that purpose, we choose to increase the control window w in Eq. (1) by following a linear pattern that is analogous to the TCP Reno linear increasing behavior during the congestion avoidance phase. In particular, when the queue backlog estimate is less than the $qthresh$ and no losses are reported, the control window w is linearly increased as follows:

$$w(t) = w(t_0) + \frac{t - t_0}{\alpha} \qquad (3)$$

where, t_0 is the time of the last window update and α is a multiplicative constant (a typical value is 0.3s). We choose $\alpha = 0.3s$, which gives a window W that is incremented by 1 packets every $300ms$. Network congestion is discovered when packets are lost or when the estimated backlog exceeds a threshold $qthresh$ (in the sequel we will assume $qthresh$ equal to 10 packets). In particular, when queue backlog estimate exceeds $qthresh$, the control window W is kept constant and equal to the value it has reached, whereas, when losses are detected, the control window W is set using the Eq. (2) to ensure queues depletion. It is important to observe that the proposed additive increase mechanism is equivalent to the additive phase used by Reno during the congestion avoidance phase but *it is not equivalent* to additively increase the input rate as proposed in [11]. In fact, it has been shown that a simple *Additive Increase/Multiplicative Decrease* (AIMD) rate control algorithm, which additively increases the transmission rate to grab the available bandwidth and multiplicatively reduces the sending rate when a congestion happens, cannot guarantee friendliness towards Reno connections since an AIMD rate mechanism does not mach an AIMD window mechanism [4].

3.1.3 Reaction to timeout expiration

A timeout is scheduled to happen if no reports are received within a time interval equal to $2 \cdot SRTT$, where SRTT is the Smoothed RTT that is computed accordingly to the algorithm described in [9]. When a timeout expires the control window w is set accordingly to Eq. (2).

3.2 The Receiver

The role of the receiver is to feed the sender with feedback reports, which indicate the bandwidth used by the flow and potential packet losses. Reports are sent at least every SRTT, or every time a loss is detected, or during the first round trip time when an estimate of the round trip time is not available. A loss is inferred when a hole in the sequence of received packets is detected. When losses are detected the receiver stamps the number of lost packets into the report. Regarding the measurement of the used bandwidth, every SRTT, a sample of used bandwidth is computed as: $B(k) = \frac{D(k)}{T(k)}$, where $D(k)$ is the amount of data received during the last $SRTT = T(k)$. Since network congestion is due to the low frequency components of the used bandwidth [13], we average the $B(k)$ samples using a discrete-time filter obtained by discretizing via bilinear transformation a first order low pass filter with time constant τ [7].

4 Performance evaluation

In this section, the proposed algorithm is investigated via computer simulations using *ns*-2 [14] and a comparison with the enhanced version of TFRC proposed in [3] is also carried out. TFRC parameters have been set as suggested in the *ns*-2 package [14]. The following ARC parameters have been chosen: k=0.5s^{-1}, $\alpha = 0.3s$, $qthresh = 10$ packets. Both single bottleneck and multihop topologies are simulated in the presence of cross and reverse traffic. In all considered scenarios TCP sinks implement the delayed ACK option [1], Packets are 1500 Bytes long, the connections are greedy and bottleneck queues are set equal to the bottleneck bandwidth times the maximum round trip time, which is equal to 250ms.

4.1 Single bottleneck scenario

The considered single bottleneck topology is depicted in Fig. 2. It consists of a single bottleneck link shared by N Reno TCP

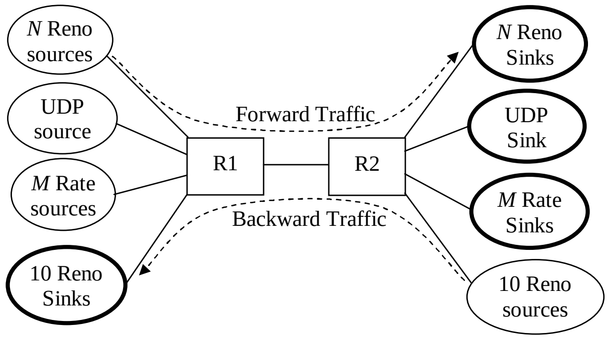
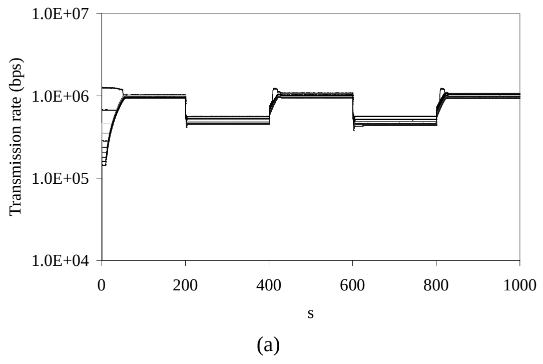


Figure 2: Single bottleneck scenario.

sources, one UDP source and M rate-based sources. On the reverse path 10 Reno TCP sources send data. Round trip times of the N Reno connections [M rate-based connections] going along the forward path are uniformly spread from $250/Nms$ to $250ms$ [$250/Mms$ to $250ms$]. Round trip times of the 10 Reno TCP connections feeding the backward path are uniformly spaced in the interval [25ms,250ms]. All the TCP sources start data transmission at the time $t = 0s$ whereas the rate-based sources start data transmission at $t = 10s$. Simulations last 1000s unless otherwise specified.

4.1.1 Ten rate-based connections and one ON-OFF UPD source

This section investigates the behavior of 10 ARC or 10 TFRC rate-based flows in the presence of abrupt changes of the available bandwidth. We consider the single bottleneck scenario in Fig. 2, where the bottleneck capacity is 10Mbps. The ON-OFF UDP source transmits at 5 Mbps during the ON period that lasts 200s and is silent during the OFF period that lasts 200s. The 10 Reno TCP sources feeding reverse traffic are turned off to focus on the interaction with the UDP traffic. Figs. 3 (a) and (b) show the transmission rates of ARC and TFRC respectively. The main result of this investigation is that the rate of the ARC flows are close to each other and track the bandwidth left unused by the UDP source, whereas the rates of the TFRC flows exhibit a much more oscillating behavior with respect to ARC rates. To compare the fairness in bandwidth allocation provided by TFRC and ARC, we have reported the Jain fairness index $F.I. = \frac{(\sum_{i=1}^{10} b_i)^2}{10 \cdot \sum_{i=1}^{10} {b_i}^2}$, where b_i is the throughput of the i^{th} connection. The Jain fairness index belongs to the interval $[0, 1]$. An index equal to one indicates maximum degree of fairness. Fig. 4 shows the Jain fairness index during the simulation. During the first $200s$, TFRC exhibits a significant degree of unfairness. At the end of the simulation the fairness index of TFRC reaches the acceptable value of 0.9 whereas ARC reaches the value 0.9 before $t = 100s$ and the maximum value after $t = 200s$.



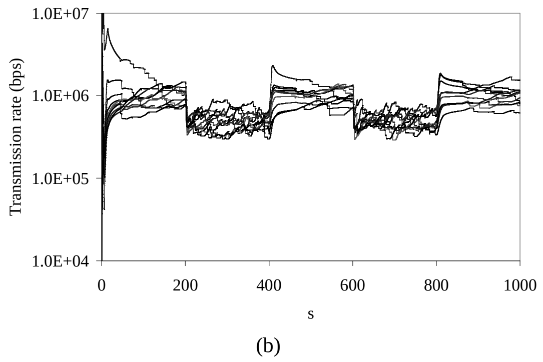


Figure 3: Ten connections sharing a 10Mbps bottleneck with UDP traffic: (a) the 10 connections are ARC; (b) the 10 connections are TFRC.

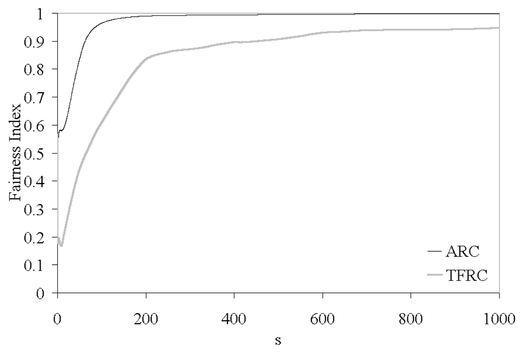
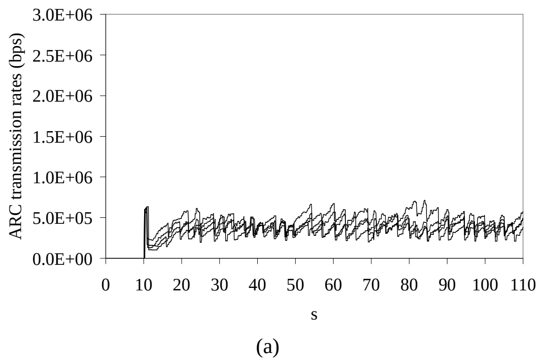
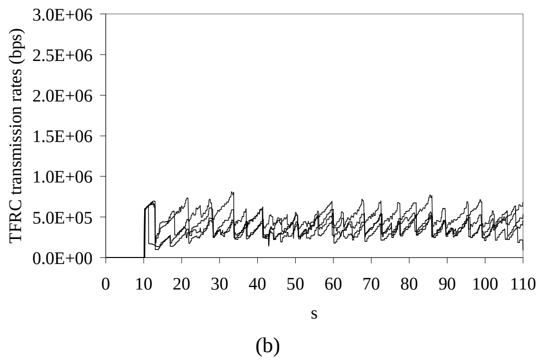


Figure 4: Fairness Index of 10 ARC or TFRC connections sharing a 10 Mbps bottleneck with an UDP source.

4.1.2 Four rate-based and four Reno connections

In this section, the single bottleneck scenario in Fig. 2 is considered to evaluate the burstiness obtained using ARC, TFRC or Reno TCP. The Reno reverse traffic is turned ON and the bottleneck capacity is set equal to 2Mbps. Two scenarios are considered: in the first, four ARC sources share the bottleneck with four Reno sources; in the second four TFRC sources share the bottleneck with four Reno sources. Fig. 5 shows the transmission rates of four ARC or TFRC or Reno TCP sources. Figs. 5 shows that ARC and TFRC provide a smoother data delivering with respect to Reno TCP.





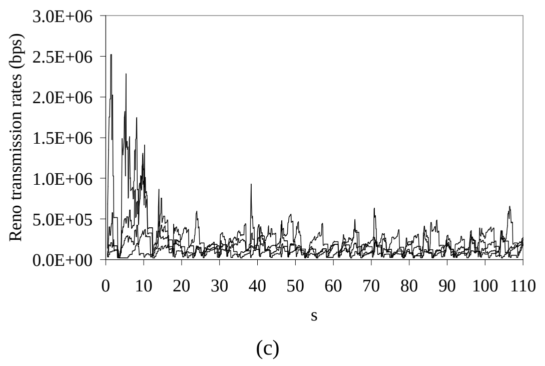


Figure 5: Sending rates obtained when 4 rate-based sources share a 2 Mbps bottleneck with 4 Reno sources: (a) ARC; (b) TFRC; (c) Reno TCP.

4.2 Multihop scenario

To investigate the ARC algorithm in a scenario that better models the real Internet, we consider the multihop topology depicted in Fig. 6. It is characterized by: (1) N hops; (2) one persistent connection C_1 going through all the N hops; (3) $2N$ persistent sources of cross traffic $C_2 - C_{2N+1}$ transmitting data over every single hop. The simulation lasts 110s during which the cross traffic sources always send data. The connection C_1 starts data transmission at time $t = 10s$. The round trip propagation time of the long C_1 connection is 250ms whereas the round trip propagation times of the connections C_2-C_{2N+1} are equal to 50ms. The capacity of the entry/exit links is 100Mbps, the capacity of the links between the routers is 1Mbps. Notice that this is a worst case scenario for the source C_1 since: (1) it starts data transmission when all the network bandwidth has been grabbed by the cross traffic sources; (2) it has the longest RTT and experiences drops at all the hops it goes through on both the forward and backward path.

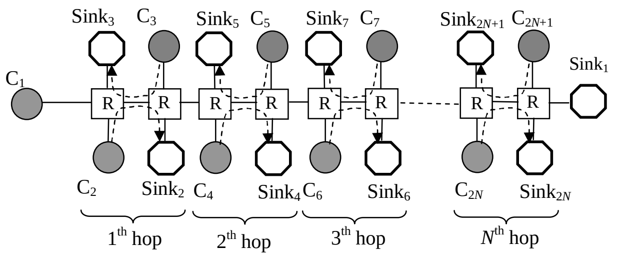

Sink3 C3 Sink5 C5 Sink7 C7 Sink2N+1 C2N+1
Sink1
C1
R
C2 Sink2 C4 Sink4 C6 Sink6 C2N Sink2N
1th hop
2th hop
3th hop
Nth hop

Figure 6: Multihop scenario.

We consider the following four scenarios:

Scenario 1. The cross C_2-C_{2N+1} sources 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. Fig. 7 (a) depicts the goodput of the C_1 connection when it is controlled by TFRC, ARC or Reno algorithm and goes through multiple congested gateways in the presence of Reno cross traffic. It shows that goodputs decrease when the number of traversed hops N increases and that ARC exhibits the largest goodput for $N > 3$. This means that ARC is able to grad a reasonable share of available bandwidth when competing with Reno TCP flows.

Scenario 2. The sources of cross traffic C_2-C_{2N+1} are controlled by TFRC, ARC or Reno, whereas the C_1 connection is Reno. This scenario is useful to investigate the friendliness of ARC or TFRC towards Reno. Fig. 7 (b) reports the goodputs of the C_1 Reno connection when going through multiple congested gateways in the presence of Reno or TFRC or ARC cross traffic. It turns out that Reno TCP achieves a very poor goodput in the presence of TFRC cross traffic, that is, TFRC does not seem to be friendly towards Reno in this scenario. Moreover, the C_1 Reno connection achieves the largest goodput when the cross traffic is provided by ARC sources. This investigation along with the previous one shows that a connection controlled by the ARC algorithm is not only able to grab its bandwidth share when competing with many Reno sources (see Fig. 7 (a)) but also that cross traffic controlled by ARC is friendlier than TFRC and Reno itself towards Reno (see Fig. 7 (b)).

Scenario 3. All traffic sources are controlled by the same control algorithm. This is a homogeneous scenario, which is useful to evaluate TFRC, ARC and Reno in absolute terms. Fig. 7 (c) 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 when the number of considered hops is larger than 3. This means that the TFRC cross traffic is too much aggressive and does not allow the C_1 connection to achieve its bandwidth share, which means that TFRC is less fair than Reno TCP and ARC.

Scenario 4. We consider the multihop scenario depicted in Fig. 6 where the last hop connecting the $Sink_1$ is a lossy wireless link, $N = 10$ and the C_2-C_{2N+1} cross traffic is Reno. Wireless losses affect the link in both directions. We assume an uniform independent packet loss distribution. We vary the packet loss probability of the wireless link from 0 to 10%. The C_1 connection is controlled by TFRC, ARC or Reno. Fig. 7 (d) shows the goodputs provided by Reno, ARC and TFRC. Both Reno and TFRC provide a very low goodput, whereas ARC provides an improvement of the goodput up to 1300%.

5 Conclusion

In this paper we have proposed an adaptive rate-based algorithm for Internet congestion control which is particularly suited for video delivering. The algorithm has been designed by following control theoretical results developed in [12] and proposes to estimate the connection available bandwidth and its path backlog in an end-to-end fashion. Simulation results have shown that ARC: (1) is more friendly than TFRC towards Reno; (2) remarkably improves the goodput with respect to TFRC and Reno TCP in the presence of lossy links; (3) provides no rate oscillations in the presence of stationary network load; (4) exhibits a less oscillating rate dynamics with respect to Reno TCP.

6 Acknowledgements

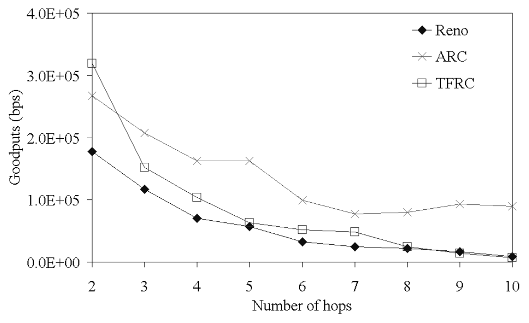
This work was partially supported by the FIRB Research Project ”Traffic models and Algorithms for Next Generation IP networks Optimization (TANGO)”.

References

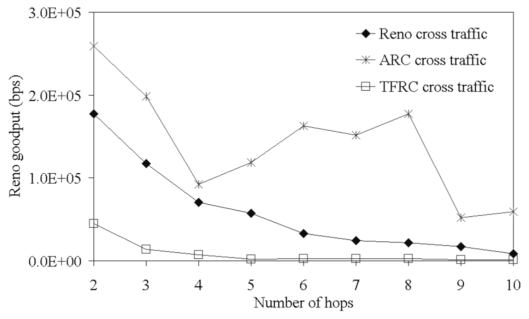
[1] M. Allman, V. Paxson, and W. R. Stevens. TCP congestion control. RFC 2581, April 1999.

[2] D. Bansal and H. Balakrishnan. Binomial congestion control algorithm. In *IEEE Infocom 2001*, pages 631–640, Anchorage, AK, April 2001.

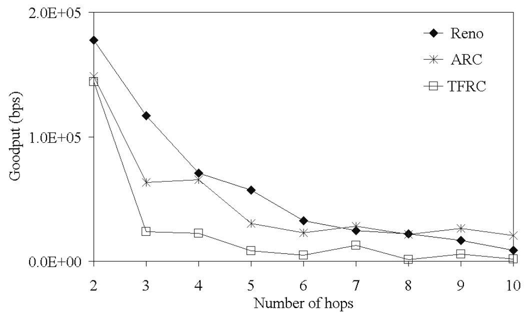
[3] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *ACM SIGCOMM 2001*, pages 263–273, UC San Diego, CA, USA, August 2001.



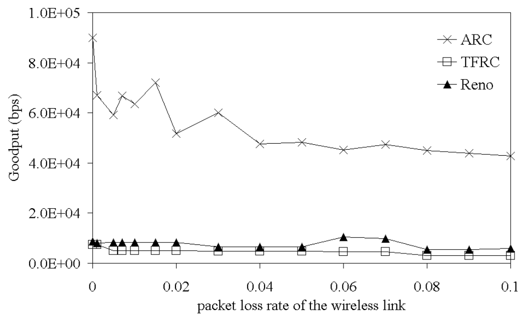
(a)



(b)



(c)



(d)

Figure 7: Goodputs of the C_1 connection over the multihop topology: (a) Scenario 1; (b) Scenario 2; (c) Scenario 3; (d) Scenario 4.

[4] J. C. Bolot and T. Turletti. Experience with control mechanisms for packet video in the internet. *ACM SIGCOMM Computer Communication Review*, 28:4–15, January 1998.

[5] P. de Cuetos and K. W. Ross. Adaptive rate control for streaming stored fine-grained scalable video. In *ACM NOSSDAV '02*, pages 3–12, Miami, Florida, USA, May 2002.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast application. In *ACM SIGCOMM 2000*, pages 43–56, Stockholm, Sweden, August 2000.

[7] L. A. Grieco and S. Mascolo. End-to-end bandwidth estimation for congestion control in packet networks. In *Second International Workshop, QoS-IP 2003*, pages 645–658, Milano, Italy, February 2003.

[8] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. RFC 3448, January 2003.

[9] V. Jacobson. Congestion avoidance and control. In *ACM Sigcomm '88*, pages 314–329, Stanford, CA, USA, August 1988.

[10] T. Kim and M. H. Ammar. Optimal quality adaptation for MPEG-4 fine-grained scalable video. In *Infocom*, San Francisco, CA, USA, April 2003.

[11] T. Kim and V. Bharghavan. Improving congestion control performance through loss differentiation. In *International Conference Computer and Communications Networks*, Boston, MA, October 1999.

[12] S. Mascolo. Congestion control in high-speed communication networks using the Smith principle. *Automatica, Special Issue on Control methods for communication networks*, 35:1921–1935, December 1999.

[13] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP Westwood: End-to-end bandwidth estimation for efficient transport over wired and wireless networks. In *ACM Mobicom 2001*, Rome, Italy, July 2001.

[14] Ns-2. network simulator, 2002.

[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM Sigcomm '98*, pages 303–314, Vancouver BC, Canada, 1998.

[16] R. Rejaie, M. Handley, and D. Estrin. Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *IEEE Infocom 1999*, pages 1337–1345, New York, March.

[17] W. Tian and A. Zakhor. Real-time internet video using error resilient scalable compression and TCP-friendly transport protocol. *IEEE Transaction on Multimedia*, 1:172–186, June 1999.