

A Simulation-based Study of TCP Dynamics over HFC Networks

Omar Elloumi ^{a,1,2}, Nada Golmie ^{b,3}, Hossam Afifi ^{c,4} and
David Su ^{b,5}

^a*Alcatel Corporate Research Center, Francis Wellesplein 1, 2018 Antwerp,
Belgium*

^b*National Institute of Standards and Technology, Gaithersburg, Maryland 20899,
USA*

^c*Ecole Nationale Supérieure des Télécommunications de Bretagne, BP 78, 35512
Cesson Sévigné, France*

Abstract

New broadband access technologies such as Hybrid Fiber Coaxial (HFC) are likely to provide fast and cost effective support to a variety of applications including Video on Demand (VoD), inter-active computer games, and internet-type applications such as Web browsing, ftp, email, and telephony. Since most of these applications, use TCP as the transport layer protocol, the key to their efficiency largely depends on TCP protocol performance.

We investigate the performance of TCP in terms of effective throughput in an HFC network environment using different load conditions and network buffer sizes. We find that TCP experiences low throughput as a result of the well known problem of ACK compression. An algorithm that controls ACK spacing is introduced to improve TCP performance.

Key words: Congestion avoidance; TCP; ATM; ACK compression; HFC;

¹ This work was partially done while Omar Elloumi was a guest researcher at NIST

² Corresponding author, email: Omar.Elloumi@alcatel.be, Tel: +32 3 240 78 33,
Fax: +32 3 240 99 32

³ email: Nada.Golmie@nist.gov

⁴ email: Hossam.Afifi@enst-bretagne.fr

⁵ email: David.Su@nist.gov

1 Introduction

The emergence of the HFC technology has a significant impact on already deployed Cable TV networks. As a return path from the stations to the head-end becomes available, Cable network operators are able to add more services to television broadcast. A Medium Access Control (MAC) layer protocol is implemented at the root (or headend) and at each of the cable network nodes (or stations) to allow various nodes to share resources. It also controls the upstream (from the stations to the headend) and the downstream link transmissions. MAC protocol specifications are being drafted by the IEEE 802.14 working group to accommodate the needs of current and future network applications.

The IEEE 802.14 Draft document [7] contains various MAC characteristics such as, frame format, station addressing, timing and synchronization procedures, and the ternary-tree mechanism to resolve collisions resulting from two or more stations transmitting at the same time. The MAC draft also provides the necessary “hooks” to support higher layer services such as CBR, VBR and ABR services for ATM. Performance evaluation studies have been conducted on MAC protocol elements such as contention resolution and bandwidth allocation [10]. Also, some preliminary work has been presented on improving the ABR service over HFC in [11]. But so far, little work has been done in studying the details and evaluating the performance of the TCP protocol in an HFC network environment.

The performance of TCP in networks with slow ACKs channel is studied in [4] and [15]. In [15], the authors focus on the effect of asymmetric networks on TCP performance and show, by means of analysis and simulation, the performance degradation of TCP Reno, due to frequent timeouts. However this study does not consider a specific MAC protocol in the reverse path (upstream channel). This model is appropriate for ADSL modems, or configurations that use a telephone line or cellular phone medium in the reverse channel where the only delay is the sum of the queuing delay and the propagation delay. However, in the case of multiple access media, such as HFC, it is important to study the effect of MAC protocol and bandwidth reservation on TCP performance. Finally a study on the effect of random losses on TCP performance in an HFC environment [3] proposes some solutions to improve the performance but again does not take into account the effect of the MAC layer.

We propose an algorithm that improves TCP performance under different offered loads and TCP data buffer sizes.

The rest of the paper is structured as follows. Section 2 presents the MAC model as specified in [7]. Sections 3 and 4 give background information on

TCP and describe the simulation model respectively. Section 5 presents TCP performance results. In section 6 a new algorithm for requesting bandwidth for TCP ACK packets on HFC upstream channel is described along with some performance results. Concluding remarks are presented in Section 7. Additional details on TCP dynamics are given in the Appendix.

2 HFC MAC Protocol Overview

The frame format of the MAC protocol defined in [7] is shown in Figure 1. The upstream channel is divided into discrete basic time slots, called minislots. A variable number of minislots are grouped to form a MAC layer frame as shown in Figure 1. The headend determines the frame format by setting the number of data slots (DS) and contention slots (CS) in each frame and sends this information to the stations on the downstream using a *CS Allocation* message. Several minislots can be grouped together in order to form a DS that carries a MAC Packet Data Unit (MPDU) which is assumed to be an ATM cell plus the MAC layer overhead. In Figure 1 four minislots carry one MPDU. The DS are explicitly allocated to a specific station by the headend using *DS Grant* messages sent on the downstream. CS fit into one minislot and are used by the stations to transmit requests for bandwidth. Since more than one station can transmit a request at the same time, CS are prone to collisions. The headend controls the initial access to the CS slots as well as manages the CRP by assigning a Request Queue (RQ) number to each CS.

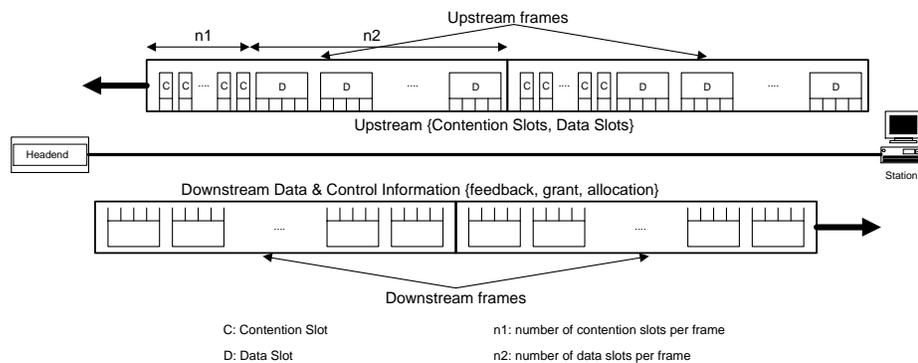


Fig. 1. Frame format of 802.14 MAC protocol

The basic MAC operation is as follows. Upon the arrival of the first data packet, a station generates a Request Minislot Data Unit (RMDU) and waits for a *CS Allocation* message from the headend that reserves a group of CS with $RQ = 0$ for newcomer transmission. The station randomly selects a CS in that group and transmits its RMDU. Since multiple stations may attempt to send their RMDUs in the same upstream CS a collision may occur. A *Feedback* message is sent to the station after a round trip time (which is also equal to

a frame length) informing it of the status of the CS used (note that a station has no means of knowing the status of its request since its transmitter and receiver are tuned on different frequencies).

In case of a successful request transmission (*Feedback=Successful*), the station activates its data transmission state machine and exits the contention process. Subsequently a *Data Grant* message will be sent by the headend. There are two grant scheduling algorithms considered in this paper: First Come First Serve (FCFS) and Round Robin (RR). For FCFS, the HE grants each station the totality of the requested slots before giving any grants to other stations. For RR, the grants are distributed to stations in a round robin fashion. A station sends a cell and waits for all the other stations that have successfully transmitted requests to the HE to send their data. Once a station is assigned a DS to send its data it may use a special field in the MPDU to send other requests in piggybacking thus bypassing the contention process.

In case of a collided CS, the feedback message contains a particular RQ number to be used for collision resolution (*Feedback=RQ*). That is the station needs to retransmit its request in a CS group with that RQ number. The CS groups are usually allocated in the order of decreasing RQ values. For each RQ value, the headend assigns a group of CS. A CS within the group is selected randomly in the range (0..2).

3 TCP Protocol Background Information

Most of today's internet applications use the TCP protocol as defined in [16]. In this section, we describe two basic concepts of TCP related to congestion avoidance and control.

TCP Congestion Avoidance and Control Mechanisms have significantly evolved in the past few years although the protocol packet format and its state machine have remained unchanged. Most versions of TCP control mechanisms aim at improving the estimation of available network bandwidth and preventing timeouts in order to maintain stability and throughput.

The slow start algorithm [12] was proposed by Jacobson as a congestion avoidance and control algorithm for TCP after a congestion collapse of the Internet. This algorithm introduces a congestion window mechanism to control the number of bytes that the sender is able to transmit before waiting for an acknowledgment. For each received acknowledgment, two new segments are sent. When the window size reaches a threshold value, *SSThreshold*, the algorithm operates in Congestion Avoidance mode. The slow start is triggered every retransmission timeout by setting *SSThreshold* to half the congestion win-

down and the congestion window to one segment. In the Congestion Avoidance phase, the congestion window is increased by one segment every Round Trip Time (RTT). Thus when the mechanism anticipates a congestion, it increases the congestion window linearly rather than exponentially. The upper limit for this region is the value of the receiver’s advertised window. If the transmitter receives three duplicate acknowledgments, `SSThreshold` is set to half of the preceding congestion window size while this latter is set to one packet for TCP Tahoe and half of the previous congestion window for TCP Reno. At this point the algorithm assumes that the packet is lost, and retransmits it before the timer expires. This algorithm is known as the fast-retransmit mechanism.

TCP Tahoe experiences low throughput when packets are lost because of the slow-start algorithm. TCP Reno performs better in the case of single packet loss within one window of data because the congestion window is decreased by half rather than set to one. However, in the case of multiple packet losses, TCP Reno also experiences low throughput since it can easily be subject to timeouts leading to long idle periods. This is outlined in [6] and [15]. In [6], the author proposes the so-called “New-Reno” algorithm to avoid this problem.

TCP Self-clocking Principle [12] estimates the bottleneck bandwidth by letting the sender rate exactly match the available bandwidth along the network path. Thus if the time to process packets at the receiver is constant, ACKs are spaced according to the bottleneck in the forward path. If the network is symmetric, the ACK spacing is preserved in the backward direction since the ACK packet size is much smaller than the data packet size and thus less likely to encounter congestion. When the sender transmits a packet for each arriving ACK, the packet sending rate matches the bottleneck service rate and this constitutes a “self-clocking” mechanism, an “idealized state” as mentioned in [14], [19] and [22]. However, this mechanism fails in the case of delay variations in both the forward and backward directions. In particular, in the case of two-way TCP traffic, ACK spacing is not preserved due to the interaction between data packets and ACKs: ACKs accumulate behind large data packets and then leave the bottleneck with a smaller spacing than the spacing corresponding to their data packets (called ACK compression) [14], [22]. Furthermore, ACK compression may lead to a rapid network queue build-up and a high packet loss percentage as shown in [22]. Even with infinite buffers, the network utilization is expected to drop considerably.

4 Simulation model

In this section we provide a description of the simulation environment and parameters. We use the NIST ATM Network Simulator [8] to implement TCP Tahoe and TCP Reno as described in [21]. The simulator also features a MAC

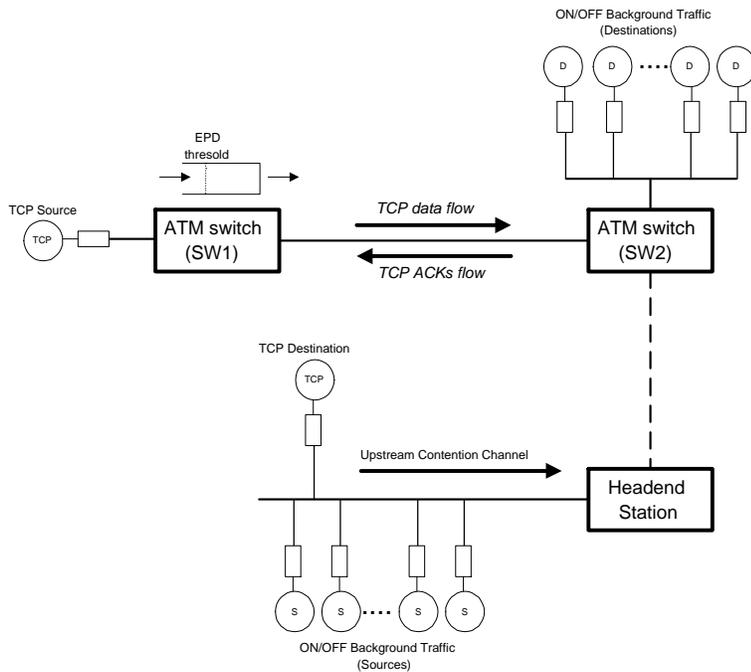


Fig. 2. Simulation model

protocol for HFC networks as described in [7]. The network model considered is illustrated in Figure 2. It consists of a hybrid ATM-HFC configuration where a TCP source in the ATM network sends packets to a TCP destination in the HFC network. We believe that this set-up reflects a typical usage in an HFC network environment since most applications in HFC networks serving residences, such as Web browsing (HTTP) or file downloads (FTP), suggest that the amount of data sent by the stations is much less than the amount of data received.

The selected network topology is chosen with one TCP connection for a better understanding of TCP dynamics. Link capacity between TCP source and *SW1* is set to 155 Mbits/s, while the link capacity between *SW1* and *SW2* is set to 6 Mbits/s to represent the bottleneck link. The propagation delay between *SW1* and *SW2* is set to 1.25 ms. *SW1* and *SW2* implement the Early Packet Discard strategy (EPD) [17].

The TCP source is assumed to have an infinite number of packets to send. In order to stress contention on the upstream channel, we consider 200 stations with on-off sources sending data upstream. This constitutes the background traffic. Sources generate fixed size 48-byte packets (encapsulated in ATM cells) according to a Poisson distribution with a mean arrival rate of $\lambda = \frac{L*UR*48}{53*N}$. L is the percentage of the offered load, UR is the upstream channel rate and N is the number of stations.

Figure 3 illustrates the protocol stack for a TCP connection end system. We use an AAL5 encapsulation of IP packets. After segmentation, cells are queued in a FIFO queue inside the MAC layer. In this paper, unlike what is described in [15], we assume that the FIFO queue inside the MAC layer is large enough to accommodate all incoming ACKs from a single TCP connection. This is a

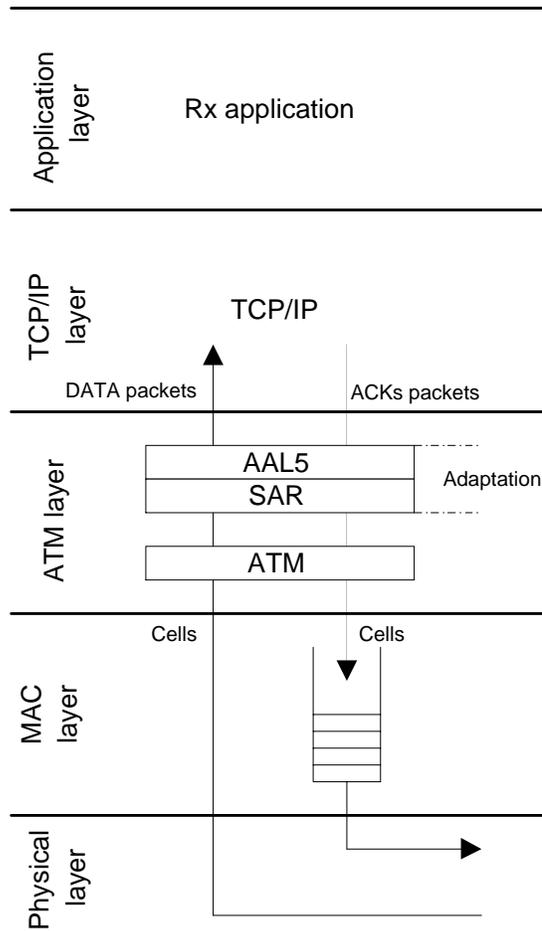


Fig. 3. TCP End Node Model

realistic assumption since for a 1 Kbyte packet size and a 64 Kbyte maximum congestion window, a maximum of 64 ACKs can accumulate in the FIFO queue (we believe that losses should not happen in a transmitter). Also, we assume that driver interface buffers can hold up to 50 IP packets of 1 Kbyte each [20].

Simulation parameters for both the MAC and TCP protocols are given in Table 2 and 3 respectively.

5 Performance of TCP over HFC

In this section we analyze the dynamics of TCP traffic for the network configuration previously described. We consider the *effective throughput* of a single TCP connection as a function of the EPD⁶ threshold in *SW1* (the bottleneck

⁶ we assume that the switch buffer is large enough to ensure that if the first cell of a packet is accepted (based on the EPD algorithm), all remaining cells from the

Table 1
MAC Model Parameters

Simulation Parameter	Value
Number of active stations	200
Distance from nearest/furthest station to headend	25/200 km
Downstream data transmission rate	30 Mbits/s
Upstream data transmission rate	3 Mbits/s
Propagation Delay	5 μ s/km
Length of Simulation Run	15 s
Length of run prior to gathering statistics	1 s
Guardband and preamble between transmissions	Duration of 5 bytes
Data Slot Size	64 bytes
Contention Slot Size	16 bytes
DS/CS Size Ratio	4:1
Cluster Size	2.27 ms
Maximum Request Size	32
Headend Processing Delay	1 ms

Table 2
TCP parameters

Simulation Parameter	Value
Maximum Transfer Unit (MTU)	1 KB
Timeout granularity	500 ms
Packet processing time	100 μ s
Congestion avoidance algorithm	TCP Reno
Maximum congestion window size	64 KB
Initial SSThreshold	8 KB

switch) for different upstream offered loads and grant scheduling algorithms, namely, FCFS, and RR (see Section 2). We define the effective throughput, as the throughput that is usable by higher layer protocols [17]. It does not account for packets discarded by the EPD algorithm nor successfully retransmitted TCP packets. A detailed analysis of TCP congestion window and bottleneck switch buffer dynamics is given in the Appendix.

same packet are queued.

5.1 Simulation results

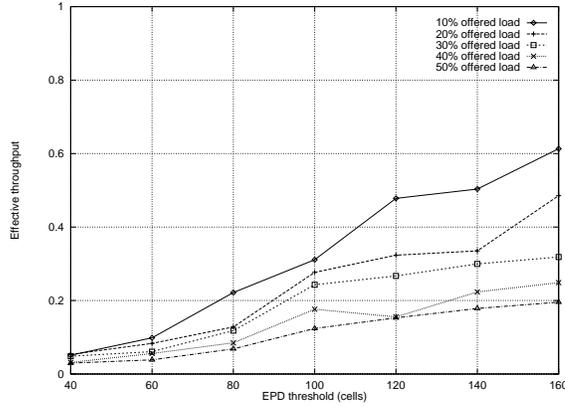


Fig. 4. TCP Effective Throughput vs EPD Threshold, FCFS HE Grant Allocation

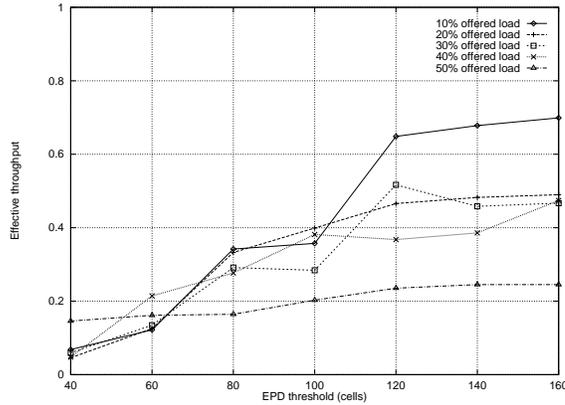


Fig. 5. TCP Effective Throughput vs EPD Threshold, RR HE Grant Allocation

The effective throughput for FCFS and RR scheduling is depicted in Figure 4 and 5 respectively. The y-axis gives the throughput as a percentage of the maximum throughput on the bottleneck link (6Mbits/s), (this includes 5 bytes of ATM header overhead). We note that throughput is low in Figure 4 and 5 meaning that the TCP connection is unable to fill the network pipe. This behavior could be caused by a high bandwidth-RTT product. However the bandwidth-RTT product computed in Table 3 doesn't confirm this assumption. For different upstream offered loads with a maximum congestion window size of 64 Kbytes, we find that the maximum window size is greater than all mean products (with the exception of 50% load).

5.2 Results analysis

The dynamics of the TCP congestion window suggest that the TCP source suffers from frequent timeouts and fast retransmit fast recovery periods. This

Table 3
Bandwidth-RTT Product

Percentage of Offered Load	Mean (Kbytes)	Max (Kbytes)	σ
10%	11.794	26.236	2.715
20%	13.316	28.923	3.253
30%	15.300	34.426	4.081
40%	27.433	103.663	11.262
50%	193.019	462.776	97.942

prevents the congestion window from reaching its optimal value and completely filling the forward link. ACK compression is known to break down the TCP self-clocking algorithm because compressed ACKs clock out data packets at a rate equal to their arrival rate. The ACK compression behavior is usually observed when ACKs encounter non empty queues (in the backward direction). When ACKs leave the bottleneck buffer their spacing is smaller than the original spacing at queue entry [2] [22]. To depict this problem we plot

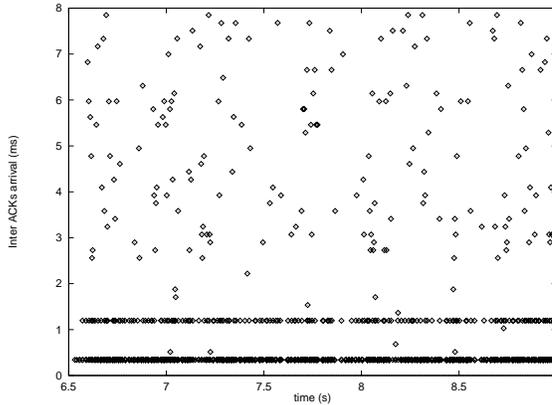


Fig. 6. ACK Interarrival Time with FCFS HE Grant Scheduling, Upstream Offered Load = 30%

the interarrival time of consecutive ACKs for 30% offered load with FCFS and RR in Figure 6 and 7 respectively. In Figure 6, we can identify two groups of ACK interarrival times at $y = 0.34\text{ms}$ and $y = 1.19\text{ms}$. The interval between two data packets sent back to back, is given by $Packet_size/\mu_1 = 1.55$ ms where $Packet_size$ is the size of the data packet and μ_1 is the forward link capacity. This is also equal to the minimum spacing between two TCP data packets. In our simulation we assume that the time to process a TCP packet and send its corresponding ACK is constant. Thus the spacing between ACKs should be equal to the spacing between their corresponding data packets. However, in Figure 6 and 7 for a large number of received ACKs, the spacing between them is less than the minimum spacing between the data packets. This confirms that ACK compression is the main reason for the low effective throughput observed. Further investigations and studies of the

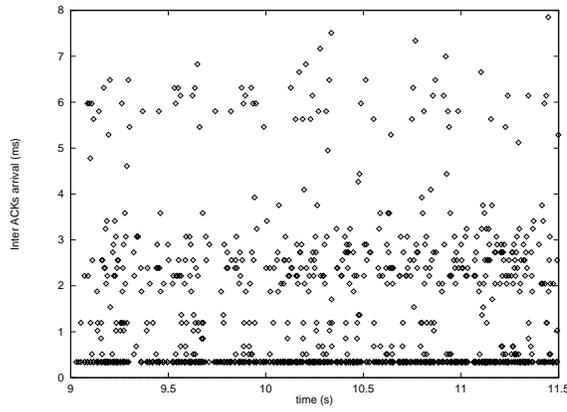


Fig. 7. ACK Interarrival Time with RR HE Grant Scheduling, Upstream Offered Load = 30%

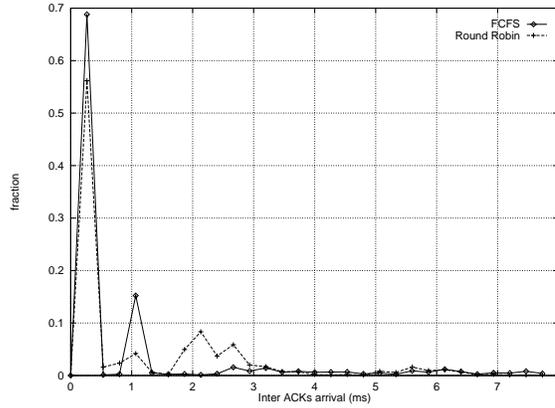


Fig. 8. Distribution of ACK Interarrival Time for FCFS and RR, Upstream Offered Load = 30%

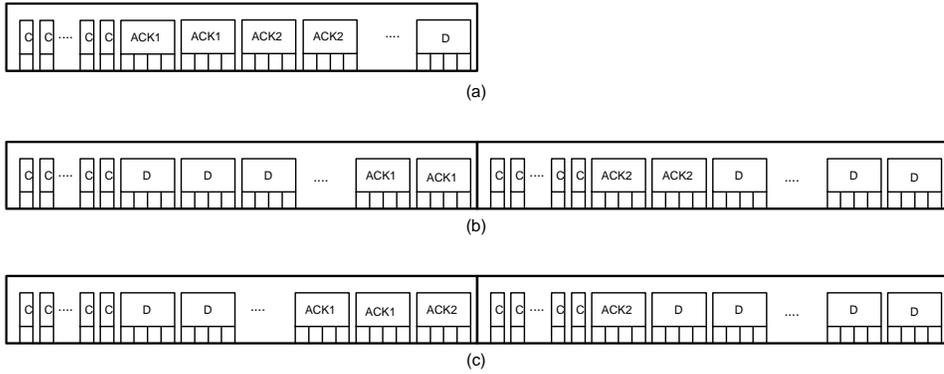


Fig. 9. ACK Compression Scenarios

MAC layer protocol help us determine that the ACK-compression identified in this case is the result of the grant allocation mechanism that is used to send ACK packets at the MAC layer. The interval between two ACKs is equal to $ACK_size/Rate_{upstream}$, where $ACK_size = N_{DataSlot} * Size_{DataSlot}$. Values of inter ACK spacing equal to $2 * 64 * 8/3000 = 0.34ms$ (Figure 6 and 7)

correspond to the case where two ACKs are sent back to back in the same frame (Figure 9-(a)). The second group of points at $y = 1.19\text{ms}$ corresponds to ACKs being sent back to back in two different frames. In this case, the spacing between two ACKs is equal to two DSs (corresponding to an ACK packet) and $n1$ CS (Figure 1). Since each DS corresponds to four CSs, the spacing is given by $7 * 64 * 8 / 3000 = 1.19\text{ms}$. Note that each ACK requires two DSs and there are 20 CS (or 5 DSs) in each cluster for a total of 7 DSs. This is illustrated in Figure 9-(b) and 9-(c). With RR scheduling, we note that the minimum ACK spacing is also equal 0.34ms when the request queue at the HE contains only one request from the TCP receiver station. Figure 8 shows the distribution of ACK spacing for both FCFS and RR. It is clear that although the minimum spacing between packets is 1.55ms (for packets sent back to back), more than 85 % of the ACK spacing is lower than the theoretical minimum spacing between packets for FCFS. However, for RR scheduling we can see that the percentage of ACK spacing below 1.55ms is relatively low since the RR algorithm introduces spacing during scheduling. This explains why better throughput is obtained in Figure 5 with RR than with FCFS in Figure 4. Figure 10 illustrates the HFC grant compression of TCP ACKs. TCP data packets arrive to the TCP destination at intervals equal to $Packet_size/\mu_1$. For each TCP packet, an ACK is generated and queued in the MAC layer FIFO queue. The MAC layer sends requests with a request size equal to its queue size. Since the time between the first request transmission and the reception of grants can be relatively large compared to $Packet_size/\mu_1$ (due to collisions and contention resolution), the FIFO queue builds up. When a successful request is received at the HE, depending on the scheduling algorithm used, grants for more than one ACK are sent to the station. If we consider the FCFS grant allocation at the HE, (Figure 10), ACKs are sent with a spacing equal to ACK_size/μ_2 where μ_2 is the upstream rate. For FCFS scheduling, the following condition, $Packet_size/\mu_1 > ACK_size/\mu_2$ guarantees ACK compression. This condition is necessary but not sufficient for RR scheduling since in case of multiple requests queued at the HE, the ACK spacing can be greater than ACK_size/μ_2 .

If we denote by $Reserv_time(t)$ the time between the first ACK arrival (at the MAC transmitting queue) and the receipt of grants (to send waiting ACKs) the necessary and sufficient condition to have an idle period is: $W(t)/\mu_1 > Reserv_time(t) + D$, where $W(t)$ is the congestion window size at the time the packet corresponding to the first ACK is sent and D is the propagation delay between the TCP receiver and sender. This idle period is due to the fact that TCP must wait for incoming ACKs before sending new data and is equal to $Reserv_time(t) + D - W(t)/\mu_1$. For FCFS HE grants scheduling, the queue occupancy $q(t)$ in switch1 after an idle period is characterized by:

$$\begin{cases} q'(t) = \frac{2*\mu_2*Packet_size}{ACK_size} - \mu_1 & \text{if TCP is operating in Slow Start phase} \\ q'(t) = \frac{(1+1/W(t))*\mu_2*Packet_size}{ACK_size} - \mu_1 & \text{if TCP is operating in Congestion Avoidance phase} \end{cases}$$

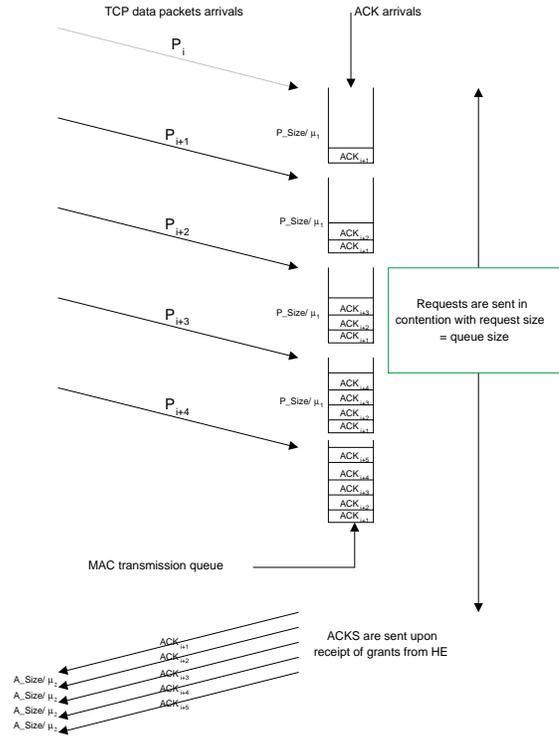


Fig. 10. HFC Grant Compression of TCP ACKs

For RR HE grants scheduling, the queue occupancy $q(t)$ in switch1 after an idle period is characterized by:

$$\begin{cases} q'(t) = \frac{2*\alpha(t)*\mu_2*Packet_size}{ACK_size} - \mu_1 & \text{if TCP is operating in Slow Start phase} \\ q'(t) = \frac{(1+1/W(t))*\alpha(t)*\mu_2*Packet_size}{ACK_size} - \mu_1 & \text{if TCP is operating in Congestion Avoidance phase} \end{cases}$$

where $\alpha(t)$ is a reduction factor inversely proportional to the number of requests $R(t)$ being served by the HE scheduler. $\alpha(t)$ is given by $\alpha(t) = \frac{1}{R(t)}$. In all cases the buffer occupancy in switch1 starts building up when $q'(t) > 0$. Thus the necessary condition to have packet losses is given by: $\frac{q'(t)}{B} > \frac{\mu_2*Packet_size}{ACK_size*W(t)}$ where B is the buffer size in switch1.

6 Solutions to the ACK compression behavior

In this section we discuss possible improvements of TCP performance in an HFC network environment. As explained in Section 5, TCP low throughput is mainly attributed to ACK compression. For a better TCP throughput, ACKs spacing in the upstream channel should be equal to the TCP data spacing in the downstream path. Solutions to this problem vary in complexity and implementation. In [14], authors propose the use of separate queues for ACKs

and data respectively to improve TCP performance over ATM in the case of two-way traffic. This ensures that ACKs are not subject to delay and delay variation caused by TCP data packets. Similarly, a possible improvement of the TCP effective throughput is to prevent ACKs from being subject to delay and delay variation inside the MAC layer. For example, the HE can predict the number of data slots needed by a station to send its ACK packets based on the number of TCP packets it forwards in the downstream direction. However such scheme would require the processing of each ATM cell header in order to ensure that the ATM cell is part of a data packet. It also requires a good estimator for the RTT (between the HE and the station) and the processing delay at the HE. We believe that other mechanisms for ACK spacing can be implemented in the station MAC layer with less complexity.

We first investigate the performance of TCP over HFC using piggybacked requests for sending ACK packets. Then we propose an algorithm for ACK spacing that significantly increases TCP throughput.

6.1 TCP performance using Piggybacking

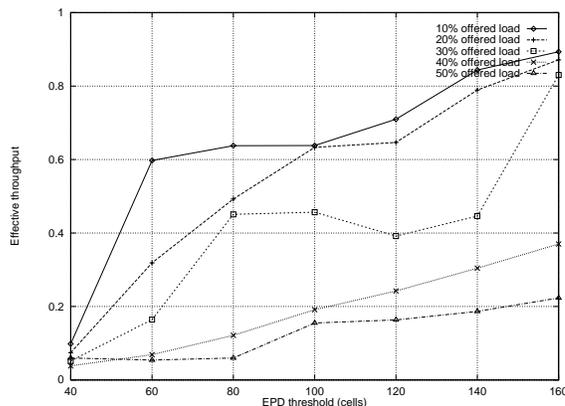


Fig. 11. TCP Effective Throughput vs EPD Threshold, FCFS HE Grant Allocation and Piggybacked Requests

Piggybacking in the MAC layer consists in sending requests for additional data transmission along with a data packet without having to go through contention. After the first request for data transmission (sent in contention), subsequent requests are transmitted in the Extended Bandwidth Request (EBR) field of the MAC data PDU. Since the request is piggybacked in the DS, it is not subject to contention. We plot in Figure 11 and 12 the effective throughput of a TCP connection, using piggybacking for sending ACK packets, as a function of the EPD threshold for both FCFS and RR HE scheduling. Note that in order to keep the offered load comparable to the simulations performed in Section 5, we only use piggybacking for the transmission of TCP ACK packets (i.e. not for background traffic). As a direct result of piggybacking the effective

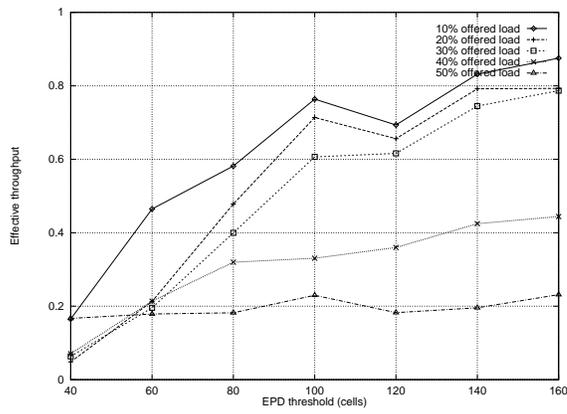


Fig. 12. TCP Effective Throughput vs EPD Threshold, RR HE Grant Allocation and Piggybacked Requests

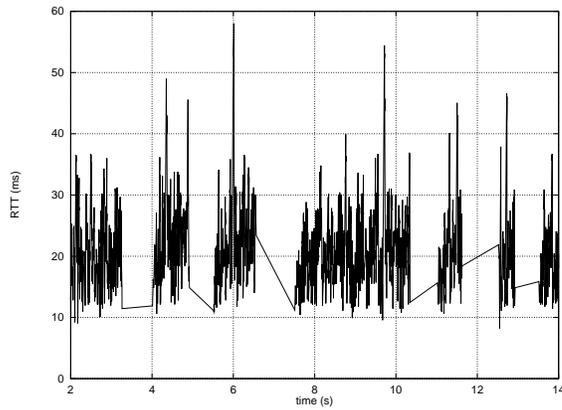


Fig. 13. RTT with RR, EPD threshold = 160 cells, Upstream Offered Load 30%

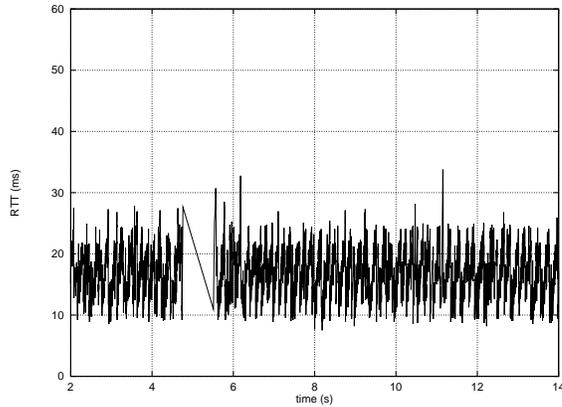


Fig. 14. RTT with RR + Piggybacking, EPD threshold = 160 cells, Upstream Offered Load 30%

throughput increases for both FCFS and RR scheduling (especially for EPD threshold values larger than 100 cells).

To illustrate this improvement we plot the RTT measured at the TCP sender for 30% offered load with and without piggybacking in Figure 13 and Figure

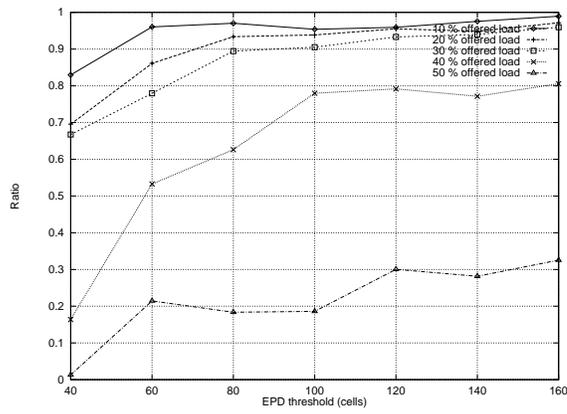


Fig. 15. Ratio of successful Piggybacked Requests over the total number of Requests, FCFS HE Grant allocation

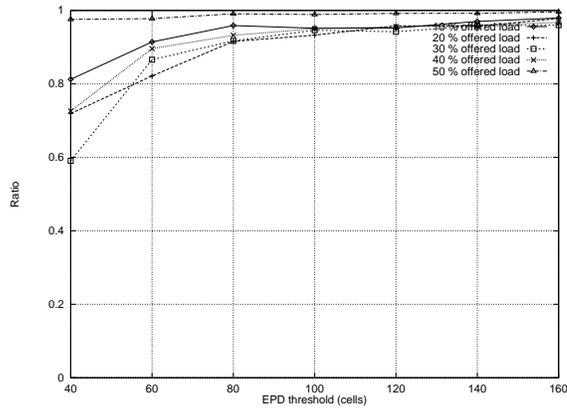


Fig. 16. Ratio of successful Piggybacked Requests over the total number of Requests, RR HE Grant allocation

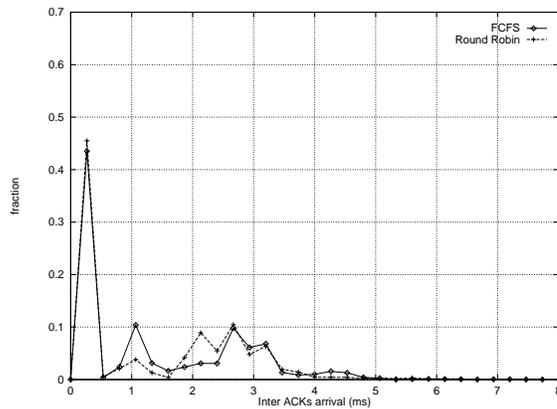


Fig. 17. Distribution of ACK Interarrival Time for FCFS and RR, Upstream offered load = 30%

14 respectively (with EPD threshold set to 160 cells). We first observe that the RTTs obtained are considerably reduced. We also compute the standard delay deviation with and without piggybacking to be equal to 1.38ms and 4.17ms respectively. We note that RR scheduling gives better performance results

than FCFS scheduling. We attribute this to the spacing introduced by the RR scheduling that increases the probability to send requests for additional ACK transmissions in every DS granted. In Figure 15 and Figure 16 the ratio of successful piggybacked requests over the total number of requests is shown for FCFS and RR scheduling respectively. As the EPD threshold in the forward direction is increased, the ratio of successful requests for both scheduling algorithms increases as well. Thus a larger bottleneck buffer enables larger TCP congestion window values and leads to a *continuous* TCP data flow. However, with FCFS scheduling we observe that the ratio of successful requests decreases as the upstream offered load increases. The continuity of data flow is perturbed with FCFS and the station cannot take full advantage of piggybacking. A contrario for the case of RR scheduling where TCP ACKs are sent with a spacing proportional to the number of stations having data to send. This spacing allows the station to send requests for subsequent ACKs along with the ACK packets using piggybacking which improves TCP performance. Note that the highest ratio of successful piggybacked requests happens at 50% offered load due to the "large" spacing introduced on TCP ACKs. From the results obtained, we can conclude that piggybacking reduces the contention percentage in the case of multiple TCP sources and gives a shorter upstream delay. However, the throughput for 50% offered load remains low due to the relatively large delay incurred. In Figure 17, we plot the interarrival of TCP ACKs. Compared to Figure 8, we note a reduction in the ratio of ACKs sent back to back: with a spacing equal to 0.34ms. Moreover a relatively large number of ACKs arrives with a spacing greater than 1.55ms (which is the minimum spacing between data packets) and hence further alleviates the ACK compression phenomenon reported.

Although piggybacking introduces spacing between consecutive acknowledgements, it hardly adapts to variable load (see next section).

6.2 Acknowledgments spacing

Piggybacking does not adapt itself to a situation where the offered load periodicity (rate) is different from the arriving packet rate (it does not conserve the optimal spacing between acknowledgments). In this case requests that are piggybacked arrive at the HE with a rate relative to the offered load and do not match the packet rate. We logically loose some bandwidth. We present now a new algorithm that, in addition of reducing the delay using piggybacking, dynamically controls the spacing between TCP ACKs. It is reactive so that the interval value is adjusted according to the variation between TCP data packet arrivals. This interval is chosen as to increase the bandwidth and to prevent packet losses. We first describe the algorithm and give a pseudo code description of the necessary calculations. We then comment the simulation results, compare the two solutions and discuss feasibility issues.

Dynamic rate tracking algorithm The algorithm tracks the bottleneck rate capacity μ_1 . It then calculates the adequate ACK rate μ_p for the measured μ_1 .

$$\mu_p = \mu_1 / \text{Packet_size}$$

Different methods may be used to estimate the bottleneck rate. Some have been already proposed in rate based flow control algorithms [13], [18]. We use a very simple method well adapted to TCP behavior in SlowStart and Congestion-Avoidance working regions. It measures the minimal interarrival time τ between two back to back packets to decide of the ACK sending rate (spacing). Since each ACK corresponds to two ATM cells (for Classical IP over ATM), the requested spacing is then set to $\tau/2$. This is adapted to TCP, since it usually sends packet pairs in both Slow-Start and Congestion-Avoidance regions. We can measure and always use the shortest interarrival time between two consecutive packets. The pseudo code for the proposed algorithm is as follows:

```

When a new ACK is received
  if MAC_queue_is_not_empty()
    if  $\tau == 0$ 
       $\tau = \text{clock}() - \text{last\_ack\_arrival}$ 
    else
       $\tau = \text{filter}(\text{clock}() - \text{last\_ack\_arrival}, \tau)$ 
      /* filter can be Min, Max or Mean */

  last_ack_arrival = clock()

Each time a request for new data transmission is sent:
  request_queue_size() grants with a transmission rate  $\tau/2$ 
   $\tau = 0$ 

```

Fig. 18. The ACK spacing algorithm

Simulation Results Figure 19 gives the effective throughput of TCP over HFC using the ACK spacing algorithm given. We note that it improves TCP performance for different offered loads. It prevents premature and frequent packet losses due to ACK compression. As expected, no improvement can be achieved beyond a load of 50%. This is due to the large bandwidth delay product incurred at this load. In order to validate the choice of the minimal interarrival time between two consecutive ACKs, we did simulations using the

average and the maximum interarrival values. Results are given in Figure 20 and 21 respectively. Although TCP performance is improved with a mean value based estimation, it degrades for the maximum value based estimation. Minimum based estimation proves to be optimal in our case. In Figure 22, we plot the distributions of ACK interarrival times for the minimum, mean and maximum based estimations. With the minimum and mean estimation more than 80% of the ACK interarrival times fall close to 1.55ms, which is also the minimum spacing between two consecutive TCP data packets. There is relatively few ACKs compressed (spacing less than 1.55ms). However for the maximum based estimation, while the ACK compression behavior is corrected, a large number of ACKs arrive with an interarrival time greater than 1.55ms. This results from large delays in grant reception increasing ACK spacing and triggering periods of TCP inactivity.

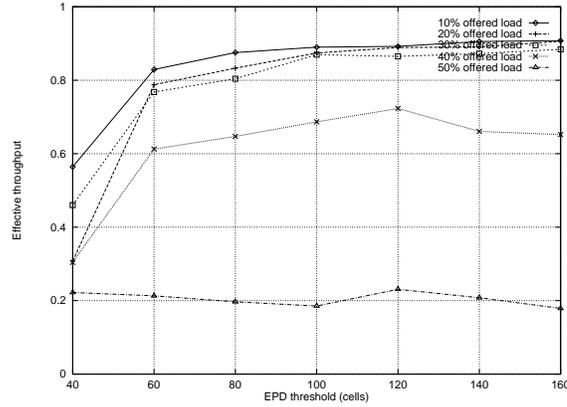


Fig. 19. TCP Effective Throughput vs EPD Threshold, RR HE Grant Allocation, min spacing

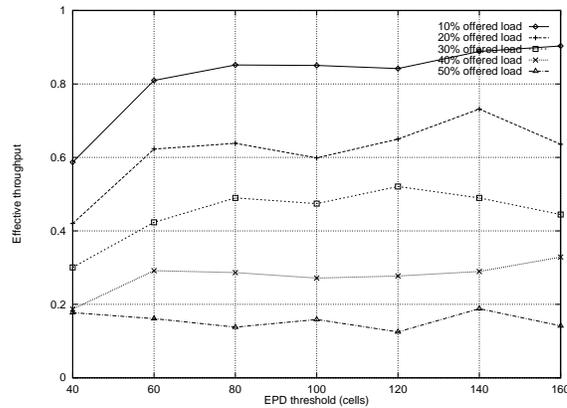


Fig. 20. TCP Effective Throughput vs EPD Threshold, RR HE Grant Allocation, max spacing

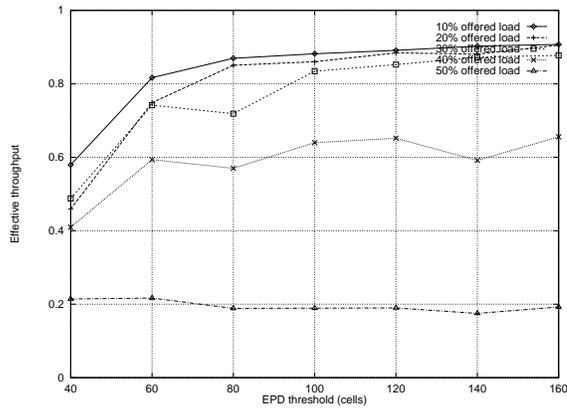


Fig. 21. TCP Effective Throughput vs EPD Threshold, RR HE Grant Allocation, mean spacing

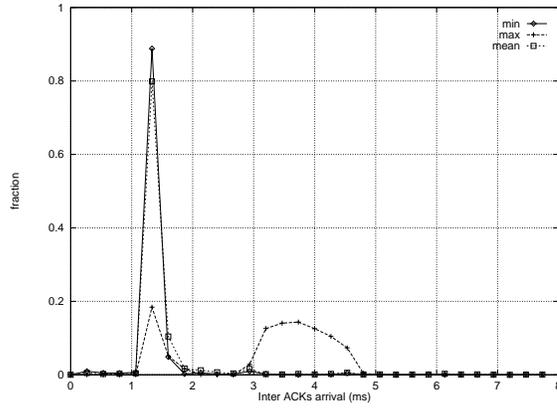


Fig. 22. Distribution of ACK Interarrival Time for ACK spacing algorithm

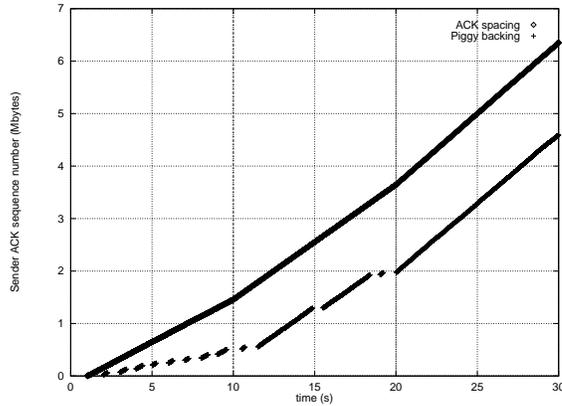


Fig. 23. Comparison of Piggy backing and ACK spacing with different bottleneck rates: 1.5, 2, 2.5 Mbits/s for 20% offered load, EPD threshold = 100 cells

Piggybacking vs. ACK spacing As we have mentioned in the beginning of this paragraph, although piggybacking reduces the ACK compression behavior, by introducing some kind of spacing on successive ACKs, it does not adapt to the offered load and it may not match the original spacing between ACKs. In this situation ACKs spacing can result in ACK compression and

bad performance. To show this shortcoming, we simulated different bottleneck rates (link capacity between SW1 and SW2): 1.5 Mbits/s from 0 to 10 s, 2 Mbits/s from 10 to 20 s and 2.5 Mbits/s from 20 to 30 s of the simulation time (see 23). Our goal is to compare the stability of piggybacking and ACK spacing. We found that ACK spacing algorithm prevents from TCP timeouts for the three bottleneck rates. For piggybacking, while the performance is similar to ACK spacing when the bottleneck rate is under 2.5 Mbits/s, a lot of timeouts are observed when the bottleneck rate is set to 2 Mbits/s and 1.5 Mbits/s. For 1.5 Mbits/s the throughput degradation is very visible (large periods of inactivity). The spacing introduced by piggybacking does not always match the bottleneck rate, and ACKs can be subject to compression.

Feasibility and implementation As far as feasibility and implementation are concerned, we believe that three conditions are necessary to ensure proper operation of the proposed algorithm. First the HE grant allocation algorithm must be able to schedule grants for ACK transmissions at a constant rate. This can be easily achieved since HE scheduling algorithms are expected to provide Constant Bit Rate (CBR) service. The second concern is the value of the clock granularity. As pointed out in [19], clock granularity can bias the estimation of the interarrival of consecutive packets or ACKs. This problem is likely to happen in a high speed network environment. However, in this case, the MAC layer has a very fine timer granularity: depending on the upstream bit rate, minislot intervals could be in the order of few μs . Finally, the algorithm presented can be implemented using an additional option in the HFC MAC layer that permits to specify an interval for transmitting data. This is achieved at the cost of a slight increase in the MAC grant request PDU size, namely an 8-bit field containing the transmission rate.

7 Concluding Remarks

This paper gives performance results and improvements of the TCP protocol over HFC networks. First we have shown by means of simulations that poor TCP performance is observed due ACK compression. We compared the performance of FCFS and RR HE scheduling algorithms and found that RR scheduling results in better performance due to the "natural" spacing introduced on successive TCP ACKs. This reduces the ACK compression behavior. Second, we investigated the effect of piggybacking on TCP performance and found that it reduces the delay and the delay variation of TCP ACKs. As a result TCP efficiency is improved in all cases. However piggybacking can still result in bad performance when the data path capacity is small. Finally, an algorithm for ACK spacing is proposed and is shown to give optimal performance for different offered loads and network buffer sizes. This

algorithm is very simple and aims at conserving the ACKs interarrival time, to respect the TCP self-clocking mechanism. This algorithm can be generalized for other applications, since even if packets are subject to delay (due to grant requests), it is desirable to conserve, at the MAC layer, the spacing introduced by the applications. However since the MAC layer can handle packets from different flows, conserving the delay between packets on a per-flow basis is a complex problem, and is in our opinion a good research area in the case of HFC networks. In practice, in the case of residential access, it is expected that only one network application is used in each terminal.

In this paper we studied TCP efficiency, however, we believe that more studies need to be led in order to investigate other issues such as TCP fairness and delay over HFC. Although, we used TCP-RENO in our simulations to reflect the large majority of TCP/IP stacks, it may be interesting to examine the performance of different TCP algorithms, such as SACK-TCP and New-Reno. These algorithms are known to give better performance in the case of frequent packet losses. Our simulations were made using a single TCP connection since our goal was to focus on the effect of the MAC protocol on TCP performance rather than to study the interaction between different TCP connections. Our future work will also include the interaction between multiple TCP connections.

References

- [1] C. Bisdikian. "msStart: A Random Access Algorithm for the IEEE 802.14 HFC Network," *Technical Report, RC 20466*, IBM Research Division, T.J. Watson Research Center, June 1996.
- [2] J.C. Bolot, "Charaterizing End-to-End Packet Delay and Loss in the Internet" , *Journal of High-Speed Networks*, vol. 2, no. 3, pp. 305-323, December 1993.
- [3] R. Cohen, S. Ramanathan "TCP for High Performance in Hybrid Fiber Coaxial Broad-Band Access Networks", *IEEE/ACM Transactions on Networking*, vol. 6, no. 1, Feb. 1998, pp. 15-29
- [4] O. Elloumi, H. Affi, M. Hamdi. "Improving Congestion Avoidance Algorithms in Asymmetric Networks". *Proc. IEEE ICC '97*. Montreal. June 1997.
- [5] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", *Computer Communications Review*, V. 26 N. 3, July 1996, pp. 5-21
- [6] J. C. Hoe, "Improving the Startup Behavior of a Congestion Control Scheme for TCP", *Proc. ACM SIGCOMM'96*, August 1996, pp 270-280, Stanford, CA.
- [7] IEEE 802.14 Working Group, Media Access Control, IEEE Draft Std. 802.14, Draft 2 R2, October 1997.

- [8] N. Golmie, A. Koenig, and D. Su, "The NIST ATM Network Simulator, Operation and Programming," Version 1.0, *NISTIR 5703*, March 1995.
- [9] N. Golmie, S. Masson, G. Pieris, and D. Su: "Performance Evaluation of MAC Protocol Components for HFC Networks," Proceedings of the International Society for Optical Engineering, Photonics East'96 Symposium, 18-22 November 1996, Boston, Massachusetts. Also appeared in *Computer Communication*, June 1997.
- [10] N. Golmie, S. Masson, G. Pieris and D. Su: "Performance Evaluation of Contention Resolution Algorithms: Ternary-tree vs p-Persistence," *IEEE 802.14 Standard Group*, IEEE 802.14/96-241, October 1996.
- [11] N. Golmie, M. Corner, J. Liebherr, D. Su, "Improving the Effectiveness of ATM Traffic Control over Hybrid Fiber-Coax Networks", Proc. of IEEE Globecom 1997, Phoenix, Arizona.
- [12] V. Jacobson, "Congestion Avoidance and Control", Proc. ACM SIGCOMM'88, pages 314-329, August 1988.
- [13] R. Jain, "Rate Based Flow Control", Proc. M SOMM'96, Aust b.c. 22, pp 270-280, Leonides, Gr.
- [14] L. Kalampoukas, A. Varma, K. K. Ramakrishnan, "Two-Way TCP Traffic over ATM: Effects and Analysis", Proc. Infocom'97, April 1997, Kobe, Japan.
- [15] T. V. Lakshman, U. Madhow, B. Suter, "Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance", Proc. Infocom'97, April 1997, Kobe, Japan.
- [16] J. Postel, "Transmission control protocol", Request for comment 793, DDN Network Information Center, SRI International, September 1981.
- [17] A. Romanow, S. Floyd, "Dynamics of TCP Traffic over ATM Networks", IEEE JSAC, V. 13 N. 4, May 1995, p. 633-641.
- [18] S. Keshav, "Packet Pair Flow Control", to appear in IEEE/ACM Transactions on Networking, available from <http://www.cs.cornell.edu/skeshav/papers.html>
- [19] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics", Ph.D. Thesis, LBNL-40319, UCB//CSD-97-945, University of California, Berkley.
- [20] W. R. Stevens, "TCP/IP Illustrated, volume 1", Addison-Wesley Publishing Compagny, 1994.
- [21] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", Request for Comments 2001, January 1997.
- [22] L. Zhang, S. Shenker, D. D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of 2-way traffic", Proc. ACM SIGCOMM'91, pp 133-147, Sept 1991, Zurich, Switzerland.

Appendix: Analysis of TCP behavior

In this Appendix we examine TCP congestion window and SW1 buffer (corresponding to the bottleneck link) dynamics for 30% of the upstream offered load using different MAC layer algorithms: 1) “pure RR”, 2) RR + piggybacking and, 3)RR + ACK spacing. In Figure 24 we identify two pathological

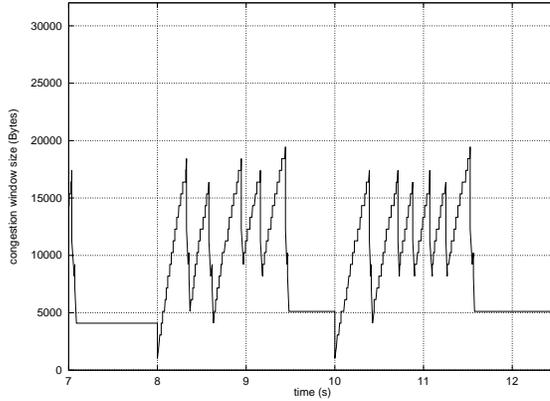


Fig. 24. TCP Congestion Window for 30% Upstream Offered Load

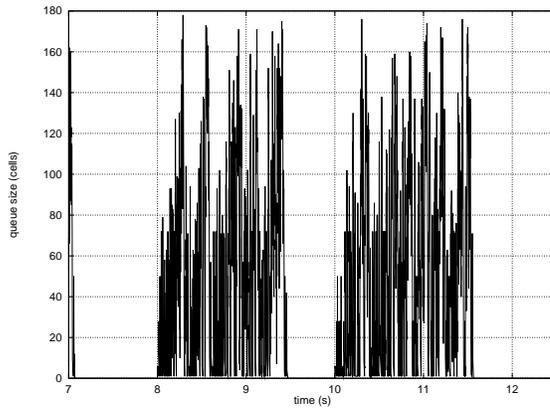


Fig. 25. SW1 Buffer Dynamics for 30% Upstream Offered Load

problems with the TCP congestion window. We note that TCP suffers from frequent timeouts leading to large periods of inactivity. As pointed out by Hoe in [6], if more than one packet belonging to a window of data is lost, TCP Reno retransmits only one packet using the fast retransmit algorithm. The remaining lost packets are retransmitted after a timeout. This is depicted in Figure 24 at $t = 8\text{s}$ and $t = 10\text{s}$. Multiple packet losses are the result of ACK compression as explained in Section 5. The second problem leading to TCP low throughput can be observed from Figure 24 at $t = 8.35\text{s}$ and $t = 8.63\text{s}$. Immediately after a fast retransmit of a lost packet, multiple ACK packets are received in a burst due to grant compression at the MAC layer leading to additional packet losses. This results in setting the congestion window and the slow start threshold to a fourth of its initial value (when the first packet

loss is detected). As shown in [5], this slows down the TCP connection since TCP operates in the congestion avoidance phase with a very small congestion window.

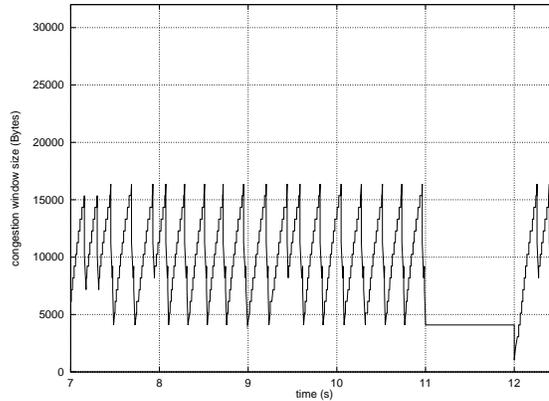


Fig. 26. TCP Congestion Window for 30% Upstream Offered Load, Piggybacking

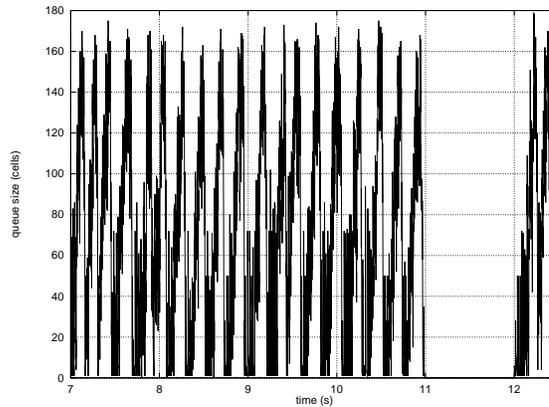


Fig. 27. SW1 Buffer Dynamics for 30% Upstream Offered Load, Piggybacking

In Figure 25, we note that the frequency of the buffer occupancy cycles is high. The queue size drops frequently to zero, due to the size of the window that is frequently reduced to half or fourth of its size and never reaching its optimal value. In Figure 26, we plot the dynamics of the congestion window using piggybacking. Timeouts are less frequent than those observed in Figure 24. This explains why piggybacking improves TCP performance by reducing the delay and the delay variation in the backward direction. However the congestion window is still at half of its optimal size. The study of the buffer behavior, in Figure 27, reveals less burstiness than with “pure RR”. However the queue size still drops down to zero frequently due to the premature reduction of the congestion window.

Figure 28, gives the dynamics of the congestion window when using the ACK spacing algorithm specified in Section 6. The congestion window reaches higher value than with “pure RR” and “RR+ piggybacking” (Figure 24 26). Furthermore, since there is no ACK compression, timeouts are avoided. The corresponding buffer dynamics in Figure 29 exhibit a rather stable periodic behavior

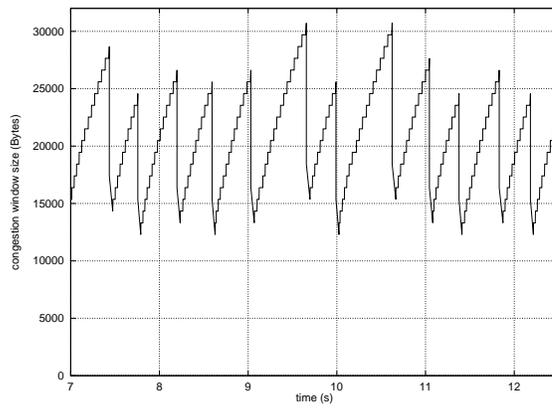


Fig. 28. TCP Congestion Window for 30% Upstream Offered Load, ACK Spacing

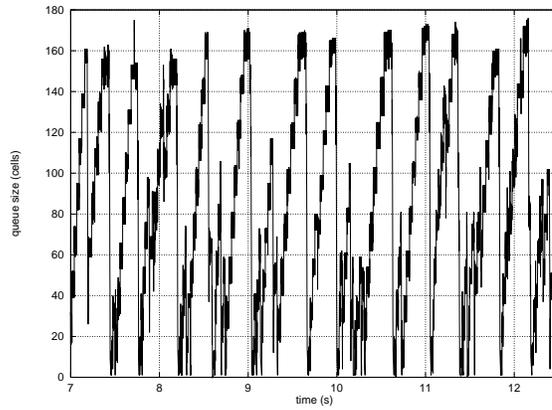


Fig. 29. SW1 Buffer Dynamics for 30% Upstream Offered Load, ACK spacing where the buffer size is rarely equal to zero.

Omar Elloumi has obtained the engineering degree in computer science from the Ecole Nationale des Sciences de l'Informatique, Tunisia and the Ph.D. from the University of Rennes I, France in 1995 and 1999, respectively. During his Ph.D. studies he was with the Networks and Multimedia department of the Ecole Nationale Supérieure des Télécommunications de Bretagne working on architectural and performance aspects of TCP/IP and ATM integration. Since April 1999, he has been a member of the Traffic and Routing Technologies project, Network Architecture Department, Alcatel Corporate Research Center. His research interests are in the area of traffic analysis, QoS in the Internet and flow admission control.

Nada Golmie received the M.S. degree in Electrical and Computer Engineering from Syracuse University, New York, in 1993 and the B.S. in Computer Engineering from the University of Toledo, Ohio, in 1992. Since 1993, she is a research engineer in the high speed networks technologies group at the National Institute of Standards and Technology (NIST). Her research interests include modeling and performance evaluation of network protocols, media access control, and quality of service for ATM, IP, HFC, WDM and wireless network technologies.

Hossam Afifi has graduated from Cairo University. He obtained the DEA and Ph.D. from University of NICE - France in the INRIA laboratories. After a Post.Doc in Washington University St. Louis , he joined the ENST Bretagne, Rennes-France as assistant professor. He obtained his tenure in September 1999 and he is now involved in Internet telephony protocols and performance evaluation for fixed and mobile infrastructures.

David Su is the manager of the High Speed Network Technologies group of the Information Technology Laboratory at the National Institute of Standards and Technology. His main research interests are in modeling, testing, and performance measurement of communications protocols. He has been involved in modeling and evaluation of protocols as they are being developed by standardization organizations. These include protocols for the Asynchronous Transfer Mode (ATM) networks, Hybrid Fiber-Coaxial networks, optical networks, and pico-cell wireless networks. He has also participated in the development of standard conformance test suites for testing of X.25, Integrated Services Digital Network (ISDN), Fiber Distributed Data Interface (FDDI), and ATM network protocols.

Before joining NIST in 1988, Dr. Su was with GE Information Service Company as the manager of internetworking software for support of GE's world wide data network. From 1973-1976, he was an Assistant Professor in Computer Science at the Florida International University in Miami, Florida. Dr. Su received his Ph.D. degree in Computer Science from the Ohio State University in 1974.