Experiences with Reducing TCP Performance Problems on ADSL

DIKU - Technical Report 04/07

May 14, 2004

Jesper D. Brouer, Jørgen S. Hansen

Abstract—**This paper presents practical studies of the TCP performance problems caused by the asymmetric nature of ADSL connections. Previously, it has been shown through simulation, wireless and satellite links that TCP throughput may be reduced by asymmetric links due to ack compression and that ack filtering and prioritizing acks can increase TCP throughput. The uplink capacity of ADSL products in general does not result in ack compression unless the uplink is congested. For a single user ADSL installation this is manageable but for larger networks connected to the Internet by ADSL the rise of peer-to-peer file sharing applications may result in a permanently congested uplink.**

We document the TCP throughput problem both through artificial and real world traffic from a network with approximately 200 users. We report on experiences with reducing this problem using the packet scheduler of a Linux based middle-box between the network and the ADSL link. We show how prioritizing acks improves throughput but hurts interactive traffic and that using hierarchical link-sharing and classifying the traffic into interactive, acknowledgments, well-known services and peer-to-peer traffic provides a reasonable solution. However, in order to reliably control latency, it is important to account for the variable ATM/AAL5 overhead when scheduling packets on the uplink. Finally, we show that the ack traffic on the uplink can be significant implying that ack filtering is interesting for ADSL.

We are currently refining the ack filtering, implementing methods for detecting peer-to-peer traffic, and implementing more accurate overhead estimations in the packet schedulers.

Keywords— **Transmission Control Protocol (TCP), asymmetric links, ADSL, link-sharing**

I. Introduction

This paper reports on experiences with TCP performance problems on one category of asymmetric links: ADSL (Asymmetric Digital Subscriber Line) connections. The TCP congestion control [1] based on receiving feedback from the acknowledgments may cause throughput problems when used on asymmetric links [2] as the smaller link may delay or bundle together the TCP acknowledgments (ack compression). This causes the feedback on link conditions supplied by the acknowledgment to be inaccurate and may also result in bursty traffic. Even though ADSL connections have a downstream capacity that is at least a factor two higher than the upstream capacity, the upstream capacity can handle the acknowledgment traffic necessary to ensure maximum utilization of the downstream line. However, in case of heavy upstream traffic TCP throughput on the downstream line may suffer due to large queuing delays and ack compression. The primary target group of ADSL was viewed as home users using the World Wide Web, but the increased use of peer-to-peer file sharing applications has changed the usage pattern dramatically. Now an ADSL upstream line can be permanently congested. A single user can deal with this on the application level but if an ADSL connection is shared by a large group of users other methods are necessary.

The work presented in this paper has been motivated by problems experienced in a real network of about 200 autonomous users connected to the Internet by an ADSL connection. Our goal is to allow peer-to-peer traffic on the upstream line while reducing the negative effects on both interactive traffic and downstream throughput. Due to the nature of the network, it is not a viable solution to modify software on the end hosts in the network nor on the other side of the ADSL connection.

The possibility of throughput problems caused by ack compression on links with two-way traffic was first recognized by [3]. Bottleneck problems for the ack traffic including ack compression have been examined for several varieties of asymmetric links. Kalampoukas et al. [4] provide analytic and simulated evidence that TCP throughput can be increased by transmitting acknowledgments at a higher priority but that bandwidth allocation mechanisms are necessary to ensure fair sharing of the link. Balakrishnan et al. [2] propose to reduce the number of acknowledgments that cross the smallest link either by having the receiver reduce the number of acknowledgments sent or by having a middle-box remove acks when several acks are queued for the same connection. If bursty traffic is to be avoided these schemes should be combined with a sender that emits data packets at a smooth rate. Alternatively, acks can be regenerated at the other side of the low bandwidth link. They also show that prioritizing acks reduces the problem. Samaraweera [5] also employ an ack filtering technique for the low bandwidth terrestrial return link for satellite based Internet access, where the remaining acks are forwarded across the low bandwidth link in an IP tunnel and a suitable number of acks are regenerated at the end of the IP tunnel. Many of the ideas mention above have found their way into an RFC [6] providing guidelines for the use of TCP on asymmetric network paths.

As we only consider solutions that do not depend on changes to the end hosts, we have chosen to introduce a middle-box between the network and the ADSL connection that uses a combination of prioritizing acknowledgments and link-sharing techniques. Our experiences with a 8 Mbit/512 Kbit ADSL line connecting a 200 user network to the Internet show that:

- giving TCP acknowledgments highest priority results in high

Jesper is a research assistant / graduate student and Jørgen is assistant professor both working at the distributed systems research group at Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen. Email: {hawk|cyller}@diku.dk.

This extended abstract have been submitted May 14, 2004, to ACM Internet Measurement Conference 2004, but have not yet been approved.

downstream utilization but may significantly increase the latency experienced by interactive users.

- link-sharing based on traffic classes for interactive, acknowledgments, well-known services, peer-to-peer applications and default traffic provides both good downstream utilization and low latency for interactive use.
- a simple ack packet dropping scheme on the upstream traffic can reduce bandwidth consumption of TCP acknowledgments with 40% but only reduces downstream line utilization by 17%.
- to reliably control traffic accurate estimates of link layer overhead must be accounted for in the middle-box.

The rest of this paper is organized as follows. Section II describes the TCP throughput problem in more detail. In section III we describe our experiences with reducing the problems in practice, and finally we conclude in section IV.

II. TCP Throughput Problems

The TCP throughput problems experienced on asymmetric links have their origin in the TCP congestion avoidance and control algorithm [1], where the sender uses acknowledgments to clock the transmission of new data segments. To ensure congestion free transmission of data segments, the sender should receive the acknowledgments with a temporal spacing close to that of the receipt of the data segments by the receiver, as this spacing reflects the capacity of the network path from sender to receiver. This is based on the assumption that the return path has little effect on the average spacing of the acknowledgments. However, it has been shown [3] that in the case of a network path with two-way traffic, router packet queues may cause several acknowledgments from the same connection to be grouped together (known as *ack compression*) thereby canceling the spacing of the acks and resulting in bursty traffic patterns with a higher risk of packet loss. Furthermore, throughput may drop due to increased queuing delays. This can be handled by increasing the TCP window size, but doing so can negatively affect the throughput in the opposite direction as a result of even longer bursts and thereby packet queue lengths. An additional impact of ack congestion is that the slow start period of a TCP connection is unnecessarily prolonged.

A variation of this problem is experienced on asymmetric network paths. If the capacity of the network path is insufficient for carrying the acknowledgments for a TCP bulk transfer, TCP throughput will be limited in the opposite direction. Balakrishnan et al. [2] define the *normalized bandwidth ratio k*, as the ratio of the raw bandwidth of the two directions divided by the ratio of the packet sizes used in the two directions. For example, if k = 4 the smallest link will be congested if it has to cope with more than one acknowledgment for every 4 data packets in the other direction. As described in the introduction, it is possible to perform ack filtering on the low bandwidth link to increase throughput in the other direction. For ADSL connections, k is less than one meaning that the link can support maximum throughput on the downstream line. However, due to the asymmetry an ADSL connection is still vulnerable to ack compression and large queuing delays in the case of upstream traffic.

Kalampoukas et al. [4] analyze this situation and conclude that prioritizing acks higher on the smallest link does indeed result in maximum utilization of the larger link (if $k < 1$). However, the ack traffic on the smaller link can cause starvation of other traffic and they instead propose a bandwidth allocation scheme for the smallest link, where the bandwidth is divided between acks and other traffic. Kalampoukas et al. consider these algorithms for implementation in end hosts but they may as well be implemented as router-based solutions.

Our focus is on networks with multiple autonomous users connected to the Internet by an ADSL connection. In order to be easily deployable, a solution should be based on techniques that can be implemented in a router or middle-box placed between the network and the ADSL connection. Such a solution provides us with control of the upstream traffic. In the following, we will therefore concentrate on the different forms of bandwidth allocation schemes for TCP acknowledgment traffic, as they can be applied to our middle-box solution. Additionally, we will examine a limited use of ack filtering.

III. Experiences with packet scheduling

In this section, we describe our experiences with improving downstream line utilization and interactive latency on a real network with approximately 200 users. We begin by describing the physical setup. Then we document the TCP throughput problems of the network and thereafter we describe our approach to reducing these problems. Finally, we describe our problems with precisely mimicking the transmit rate of the physical link.

A. Environment

The real world network in this paper is a student residence[1] connected to the Internet with an 8 Mbit/512 Kbit ADSL line, using a fully qualified C-class IP-range.

The Internet Service Provider (ISP, TeleDanmark) delivers a Cisco 1401 router which has a 10Mbit/s Ethernet connection and an ATM (ATM25) connection. The ATM connection is connected to the ADSL modem. The router encapsulates the traffic over ATM according to RFC1483[7] / RFC2684[8][2] in routed mode. Our middle-box is acting as a normal router and is placed between the LAN and the Internet ADSL router.

All performance measurements performed below are done between a local host in the network and a couple of well connected external hosts. All hosts used the standard TCP/IP implementation of Linux 2.4.

B. Documenting the problem

We document the effects of upstream traffic on the downstream line using both artificial and real traffic. The artificial traffic experiment was conducted so that no other traffic was present on the ADSL connection. It consists of one continuous TCP bulk transfer of 200 MB on the downstream line and two non-overlapping TCP bulk transfers of 4 MB each on the upstream line. Figure 1 shows the impact on downstream throughput from using the upstream line. It is very clear that the upstream traffic has a negative impact on the performance of the downstream throughput, illustrated by the dips in the downstream throughput around time 60 to 144, and time 204

[1] Kollegiegården, with 307 permanent residents.

[2] The manual and Cisco IOS refers to rfc1483 mode, even though rfc2684 obsoletes rfc1483

to 289, where the two upstream TCP connections are transferring data. The downstream throughput is reduced from approximately 7 Mbit/s to approximately 250 Kbit/s (jumping between 150 Kbit/s and 300 Kbit/s). The window size from our downstream TCP transfer (Figure 1) is 34752 bytes. On an 8 Mbit line with a window size of 34752 bytes, the ack latency has to be under 34.75 msec to utilize the full capacity. The average round trip time on the ADSL connection increases to around 300 msec during the upstream transfers, corresponding well with a downstream throughput of 250 Kbit/s given the window size.

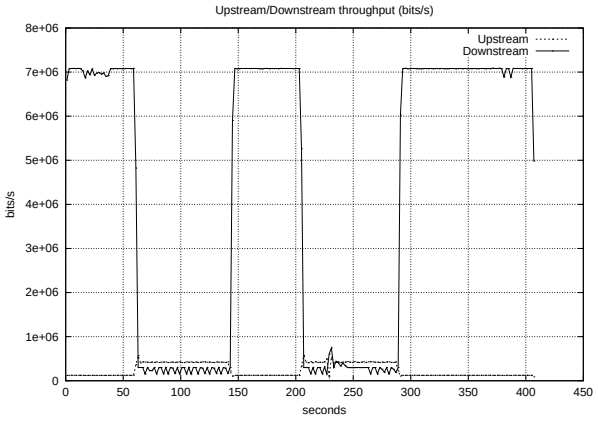

Fig. 1. Illustrating the problem on a clean 8Mbit/512Kbit ADSL line. One continuous downstream bulk TCP transfer (200Mb), and two upstream bulk TCP transfers (4MB each).

To estimate the effects of real upstream traffic on downstream transfers, we performed the 200 MB downstream TCP transfer during the afternoon concurrently with real traffic from the network. As figure 2 shows, the downstream transfer still experiences throughput problems during the bulk transfer started at 15:20, as the link utilization during the TCP bulk transfer is only 1.98 Mbit/s instead of 7 Mbit/s. The TCP bulk transfer itself only achieves an average of 1.1 Mbit/s. The average round trip time of the ADSL connection was approximately 380 msec. This shows that the real upstream traffic consisting of several TCP connections to a number of different hosts has lower overall link utilization than our artificial upstream transfer, but that the effects are still significant. As our experiments show an increase in RTT, we increased the maximum window size of the downstream TCP bulk transfer to 2 MB (Linux adjusted the window size to 1497 Kbytes) as this should be sufficient for fully utilizing the downstream bandwidth. Figure 3 shows our expectations fulfilled. However, the large window size allows considerable bursts of packets, which triggers packet losses as illustrated by the regular drops in throughput.

C. Prioritizing Acknowledgments and Interactive Traffic

Our first approach to increasing the throughput on the downstream line was to prioritize TCP acknowledgments higher than all other traffic. This ensures, that upstream transfers will impose a maximum delay of one packet transmission on an acknowledgment. This scheme resulted in maximum utilization of the downstream line, but had the negative side effect that interactive traffic experienced large delays as all upstream traffic

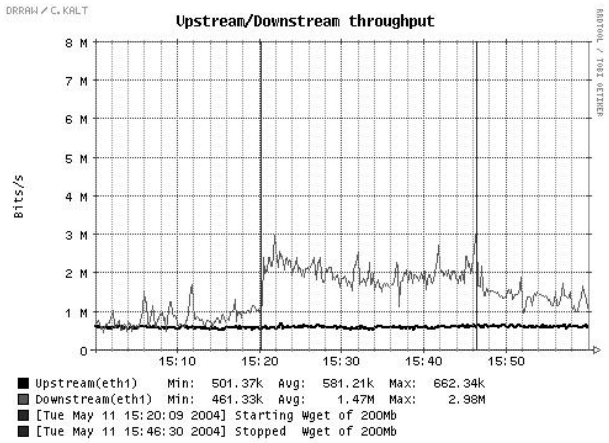

Fig. 2. Real world traffic and a 200 MB downstream transfer on a 8Mbit/512Kbit ADSL connection without prioritizing.

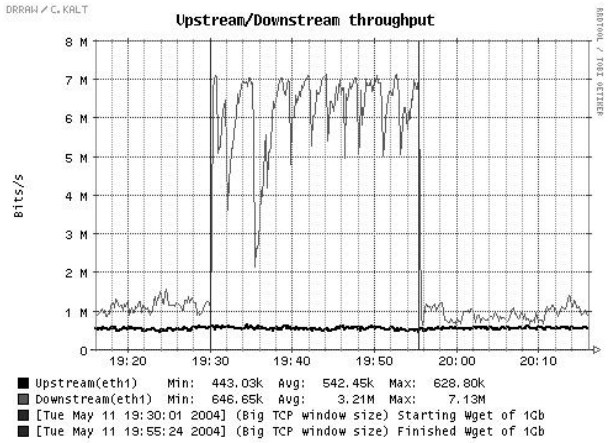

Fig. 3. Using a very large TCP window size (1497 Kbytes), makes it possible to utilize the downstream bandwidth without any prioritizing.

was squeezed into the remaining capacity (approx. 30%).

To improve the latency of interactive traffic, we further refined the traffic categories into the following five classes:

Interactive consisting of secure shell traffic and various chat protocols.

Acknowledgments consisting of pure TCP acknowledgments from all traffic classes. The ack packets influence the downstream utilization, thus every ack packet is given high priority as we are trying to maximize the downstream throughput.

Good traffic consisting of as many well-known services as possible including protocols such as http, ftp, dns, and tcp-handshake.

Bad traffic consisting of peer-to-peer traffic that can be identified from know port numbers. This includes protocols such as eDonkey, Kazaa/Fasttrack and BitTorrent.

Default consisting of traffic not matched by the other categories.

The *bad* class is traffic that has little real value to local users. The obvious example are external peer-to-peer clients receiving data, which add load to the upstream line. Experiences with real world traffic show that our current classification of *bad* traffic based on default port numbers of known peer-to-peer services

only captures part of the real peer-to-peer traffic. This is because current peer-to-peer protocols include a number of evasive techniques for avoiding classification and firewalling. These evasive techniques result in peer-to-peer traffic being classified as either *good* traffic when they use standard port numbers or *default* traffic when random port numbers are used. In practice, it has been necessary to manually figure out local users peer-to-peer port numbers based on the traffic patterns of a local host. More specifically, if the bandwidth consumption of a local host increases above a certain threshold, all connections between that host and external hosts are examined. If any of the external hosts use known peer-to-peer ports, all traffic from the local node is assumed to be peer-to-peer traffic. We are currently implementing a way to categorize the bad traffic and the interactive traffic [9], [10] based on its network behavior.

The actual packet scheduling uses the Hierarchical Token Buckets (HTB) implementation under Linux, which is a class-based queuing implementation inspired by [11]. Traffic is marked and classified into the different classes with the use of Linux's Netfilter. The class hierarchy is illustrated in Figure 4. The classes good, bad and default are grouped together to minimize their effect on the delay of the two other classes [12], [13].

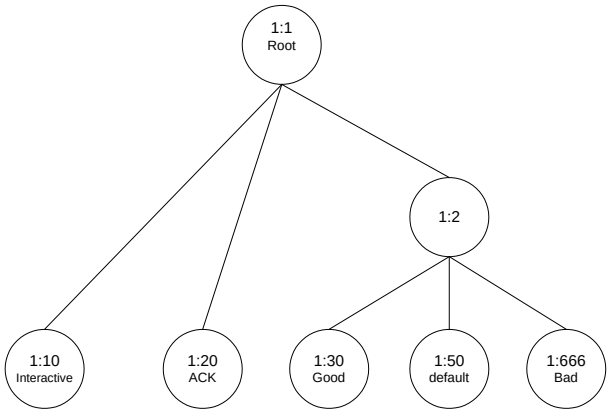

Fig. 4. Hierarchical Token Bucket class hierarchy

The actual upstream line capacity is 500 Kbit/s (and not 512 Kbit/s) and subtracting the fixed ATM overhead (see section III-D) results in 432 Kbit/s, which is used as the HTB maximum rate. The *ack class* is assigned a rate of 380 Kbit/s and given high priority (prio 0), to ensure low delay by avoiding the class ever being backlogged (Figure 8 shows an ack usage of 313K bit/s including overhead). The remaining bandwidth is assigned by ratios among the other classes. The ceil rate is limited for some of the classes to avoid starvation of low priority classes. The *interactive class* is assigned 30% (15.6 Kbit/s) rate, but is limited to a ceil rate of 20% (86 Kbit/s) due to its high priority (prio 0, highest). The *good class* (prio 3) is assigned a rate of 35% (18.2 Kbit/s), but is limited to a ceil rate of 80% (346 Kbit/s) to avoid starvation of the default class. The *default class* is assigned a rate of 25% (13 Kbit/s) and allowed to use 100% of the ceil rate (prio 5). The *bad class* is assigned a rate of 10% (5.2 Kbit/s) and allowed to use 100% of the ceil rate, but is assigned the lowest priority (prio 7).

Repeating the experiment from the previous section with a 1 GB downstream TCP bulk transfer concurrently with real traffic shows that the HTB solution gives high priority to the ack packets resulting in a high downstream utilization of 5.77 Mbit/s (see Figure 5). The TCP throughput of the downstream transfer is still below the maximum capacity as the TCP window size is 63712 bytes and the round trip time occasionally exceeds the maximum value (approximately 73 msec) for fully utilizing the downstream line.

The round trip time measurements were obtained by having our middle-box classify ICMP packets to a set of different hosts into our different traffic classes, and then measure the average round trip time for a series of 10 ping packets every 10 seconds. The obtained round trip times for the *interactive, acknowledgment* and *good* classes on the link are shown in Figure 6, where it can be seen that the acknowledgment traffic experiences an average delay of 110 msec resulting in an average throughput of 4.63 Mbit/s for the bulk transfer. This corresponds to the average utilization of 5.77 Mbit as the bulk transfer coexists with real world downstream traffic. The increased average round trip time for acknowledgments and to a certain extent interactive traffic has two causes: (1) the increase in ack traffic causes longer queues when acks have to wait for the transmission of a lower priority packet to complete and (2) increases in packet queue lengths on the downstream line due to higher utilization. The downstream line may experience bursty traffic caused by ack compression on the upstream line as the bandwidth ratio is 1/16 meaning that up to 8 acks (one for every two data segments) may get queued behind a low priority data packet. Figure 6 also shows the impact of the ack traffic on the *good* traffic, that experiences an average delay of 628 msec with a maximum of 4.48 seconds. This was also the condition for interactive traffic when only ack traffic was assigned a high priority which is clearly unacceptable.

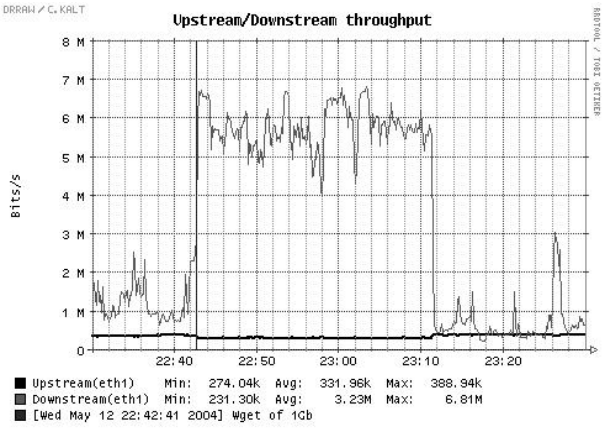

Fig. 5. Link utilization of real world traffic and 1 GB downstream transfer on a 8Mbit/512Kbit ADSL connection with HTB packet scheduling.

The good downstream throughput is achieved at the expense of the lower priority traffic classes on the upstream line, as the ack traffic is allowed to use a maximum of 380 Kbit/s (76% of the total bandwidth). We therefore conducted an experiment where we reduced the allocated bandwidth for ack traffic by 50% to 190 Kbit/s. When performing the downstream TCP bulk transfer, the ack traffic uses 82% of its traffic rate. The band-

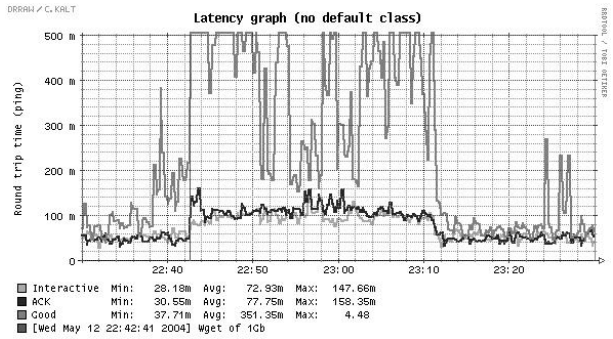

Fig. 6. Round trip time for interactive, acknowledgments and good traffic classes using real world traffic and 1 GB downstream transfer on a 8Mbit/512Kbit ADSL connection with HTB packet scheduling.

width reduction will therefore cause a reduction in the number of acks by close to 40% with a proportional drop in the downstream utilization. An alternative to only delaying the acks is to perform ack filtering. We can emulate ack filtering by combining the reduced ack traffic rate with dropping ack packets. We therefore tried to limit the length of the ack FIFO to 5. In theory, this should provide better throughput as long as not all acks for a given TCP connection are dropped. Figure 7 shows the result of these experiments. It can be seen that reducing the ack rate by 50% results in a drop in downstream line utilization by 50%. After a while, we reduce the queue length to five packets, resulting in a packet drop rate of 100 acks per second (the 40% reduction). This improves throughput significantly without reaching the level of the higher ack rate. This lower utilization is to be expected, as the FIFO drop scheme results in dropping acks of higher sequence numbers. We are currently working on implementing a better ack drop policy, that can drop acks of the lowest sequence numbers on a per connection basis. Dropping acks did not result in any TCP retransmissions, so the increased burstiness did not result in a congested link. This was as expected, as the window size used by the TCP connections was only 63712 bytes.

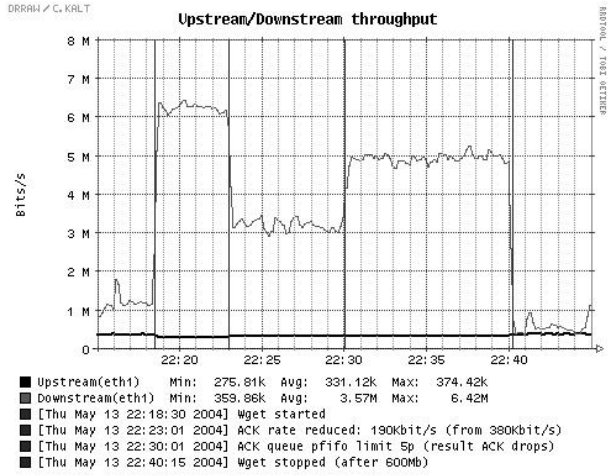

Fig. 7. Effects of limiting ack rate on link utilization of real world traffic and 1 GB downstream transfer on a 8Mbit/512Kbit ADSL connection with HTB packet scheduling.

D. Implications of ATM transport

Our packet scheduling for the upstream is performed in front of the equipment provided by the service provider and as such has no immediate feedback concerning packet queue lengths internally in that equipment. In order to be able to control both latency and throughput of the ADSL upstream line we need to ensure that we are not emitting packets at a higher rate than the ADSL equipment supports but at the same time we wish to be as close to the maximum capacity as possible to fully utilize the bandwidth. Therefore we need to have a good estimate of the link size of a given packet. Unfortunately, the Linux queuing system only accounts for IP overhead, so it is not feasible to use the hardware bandwidth rate directly, when setting up the HTB queuing discipline. To specify the correct rate to HTB, the line overhead needs to be calculated. But unfortunately the overhead of an ADSL line encapsulating the traffic over ATM using ATM Adaption Layer type 5 (AAL5) [8], [14] depends on the number and size of packets.

The ATM overhead can be categorized into three categories, 1) Fixed overhead, 2) Overhead per packet (AAL5) and 3) ATM cell padding. The *fixed* overhead is the ATM cell header overhead, which is 5 bytes out of every 53 bytes ATM cell (9.43%). Another fixed overhead, at the physical layer one out of every 27 cells is an ATM-OAM (Operations And Maintenance) cell (3.7%). The *AAL5* overhead is 16 bytes (8 bytes header + 8 bytes tail), which is applied to every IP packet. A patch to HTB makes is possible to specify an overhead per packet. The *ATM cell padding* overhead is the most difficult to calculate, as it is a function of the packet size (plus the AAL5 overhead and possible TCP options, modulus 48). There is currently no support to account for this kind of overhead in the Linux queuing system, but we are in progress of implementing this[3]. Until then, we have to estimate the average ATM padding overhead except for the ack traffic, where the packet size is fixed (minimum two ATM cells) . Figure 8 shows the overhead of the ATM encapsulation for both the ack traffic and the other kinds of traffic for the experiment depicted in Figure 5. The bandwidth has been adjusted for the fixed part of the ATM encapsulation overhead. Although not surprising when doing the math, it is dispiriting to see that (at maximum) 313 Kbit/s (63%) of the bandwidth is consumed by the ack packets, where 160 Kbit/s (32%) is due to pure overhead. This shows that even for ADSL connections it might be beneficial to perform some kind of ack filtering.

E. Alternative Packet Scheduling Algorithms

The choice of HTB as our link-sharing mechanism is mainly based on the overhead calculation feature. We would prefer to use Hierarchical Fair Service Curve (HFSC)[4] [15] because of the ability to decouple delay and bandwidth allocation. Our practical experiences with HFSC have shown that we can achieve a better delay bound in our interactive class. However, we cannot use HFSC currently due to the inability to account for the variable ATM overhead, without sacrificing to much up-

[3]The Linux implementation (of cbq, HTB, police and tfb) uses a rate table, where a packet size is mapped to a delay via a precomputed table, which makes it difficult to implement.

[4]That has been ported from AltQ to Linux, and recently have been accepted into the main Linux kernel (from v.2.4.25)

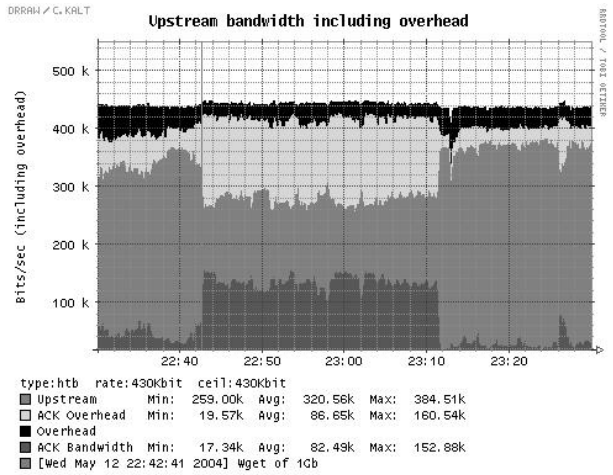

Fig. 8. IP traffic size and per IP packet ATM encapsulation overhead for acknowledgments and other traffic using real world traffic and 1 GB downstream transfer on a 8Mbit/512Kbit ADSL connection with HTB packet scheduling.

stream bandwidth. We are in the progress of implementing this type of overhead accounting, as it seems easier to implement into HFSC than HTB where packet sizes are mapped to delays via a table with only 256 entries.

IV. Conclusion and Future Work

Achieving low latency for interactive traffic and high link utilization on shared ADSL connections can be problematic, unless some form of resource control is used on the upstream line. In this paper, we have shown that TCP throughput problems are present on ADSL connections in real world networks due to a high load on the upstream line caused mainly by peer-to-peer traffic. Our experiences with link-sharing implemented in a middle-box between the network and the Internet show that using a class-based scheduler provides both good downstream utilization and low latency for interactive use. The scheduling is based on the following five traffic classes: interactive, acknowledgments, well-known services, peer-to-peer applications and default traffic. However, our experiences tell us that it is difficult to capture all peer-to-peer traffic as the protocols specifically try to avoid such classification. Finally, we have described how accurately estimating the link layer overhead is necessary to avoid queuing delays in network hardware beyond our control, and that limited ack filtering may be a way to reduce ack bandwidth consumption.

Future work includes conducting experiments with other link-sharing algorithms, especially hierarchical fair service curve. Furthermore, we are in the progress of implementing more precise link layer overhead calculations. Finally, we are currently developing methods for automatically detecting a larger class of peer-to-peer traffic and interactive traffic, which will allow us to better enforce link-sharing policies.

V. Acknowledgments

Walter Karshat has provided the HTB overhead patch and has in general been a great help concerning the Linux HTB implementation. Patrick McHardy and Oleg Cherevko have provided valuable advice on configuring HFSC. The use of drraw by Christophe Kalt has been invaluable in analyzing the data.

References

[1] Van Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM '88*, Stanford, CA, Aug. 1988, pp. 314–329.

[2] Hari Balakrishnan, Venkata N. Padmanabhan, and Randy H. Katz, "The effects of asymmetry on TCP performance," in *Mobile Computing and Networking*, 1997, pp. 77–89.

[3] Lixia Zhang, Scott Shenker, and David D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic," in *Proceedings of the conference on Communications architecture & protocols*. 1991, pp. 133–147, ACM Press.

[4] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan, "Improving TCP throughput over two-way asymmetric links: analysis and solutions," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. 1998, pp. 78–89, ACM Press.

[5] N. K. G. Samaraweera, "Return link optimization for internet service provision using dvb-s networks," *ACM Computer Communications Review (CCR)*, vol. 29, no. 3, pp. 4–19, 1999.

[6] H. Balakrishnan, V. N. Padmanabhan, G. Fairhurst, and M. Sooriyabandara, "RFC3449 – TCP performance implications of network path asymmetry," Tech. Rep., RFC, Dec. 2002.

[7] Juha Heinanen, "RFC1483 – Multiprotocol encapsulation over atm adaptation layer 5," Tech. Rep., RFC, July 1993.

[8] D. Grossman and J. Heinanen, "RFC2684 – Multiprotocol encapsulation over atm adaptation layer 5," Tech. Rep., RFC, Sept. 1999.

[9] Yin Zhang and Vern Paxson, "Detecting stepping stones," in *In Proc. of 9th USENIX Security Symposium*, 2000.

[10] Yin Zhang and Vern Paxson, "Detecting backdoors," in *In Proc. of 9th USENIX Security Symposium*, 2000.

[11] Sally Floyd and Van Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365–386, 1995.

[12] Martin Devera, *Hierachical token bucket theory*, May 2002.

[13] Sally Floyd, *Notes of Class-Based Queueing: Setting Parameters*, Feb. 1996.

[14] "I.363.5, B-ISDN ATM adaptation layer specification:, Type 5 AAL," Standard, ITU-T, Aug. 1996.

[15] Ion Stoica, Hui Zhang, and T. S. Eugene Ng, "A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 185–199, 2000.