The Interchangeability Study of ADSL and the Cable Modem

Xiaowei Yang (yxw@mit.edu, MIT LCS)

September 30, 2002

Abstract

Two techniques have been proposed to provide high speed Internet access to homes and small businesses. One is the Asymmetric Digital Subscriber Line (ADSL). The other is the cable modem. This paper describes research into studying the interchangeability of the two techniques. This research consists of two steps. In the first step, a wide-range link speed simulator was developed in the FreeBSD kernel. This simulator is able to simulate links of speed ranging from 10Kb/s to 100Mb/s. Thus, two techniques were simulated without special hardware support. In the second step, a testing environment was set up using this simulator and experiments were conducted to test the two techniques. The results of the experiments show that the two techniques are interchangeable.

1 Introduction

1.1 ADSL and the Cable Modem

As the Internet plays a growing role in our daily life, more and more people want to have Internet access at home. A typical home only has, at most, three lines connected to the outside world, which are the telephone line, the TV cable and the power line. Due to its high voltage and low bandwidth, the power line is not sufficient to transmit data . So the telephone line and the TV cable are used for data network access. Though the telephone line has already been used for network access, the current telephone data modem is slow—even ISDN (Integrated Services Digital Network) at most provides a 128Kb/s data rate. The Asymmetric Digital Subscriber Line (ADSL) [1] and the cable modem [3] are two better solutions. They provide a much higher data rate using the existing telephone line and TV cable, respectively. Multimedia data such as voice and video, which require a high data transmitting rate, can be transmitted with low delay and high quality.

ADSL uses the high spectrum of the copper telephone line to transmit data. As voice signals for telephone conversations only use up bandwidth between 0 and 4Khz, higher bandwidth 4Khz-2.2Mhz could be utilized for data transmission. Because in typical Internet use, end users download more data than they send, more bandwidth is allocated to downstream traffic (towards users) than to upstream traffic (away from users). Thus, the bandwidth

of the old telephone lines is divided into three parts. The bandwidth between 0-3.4Khz still holds for ISDN channels. This bandwidth either provides two 64Kb/s voice channels or one 128Kb/s data channel. The bandwidth between 30Khz-138Khz is allocated for the ADSL upstream channel, which provides a 800Kb/s data transmission rate. The bandwidth between 138Khz-1104Khz is allocated for the ADSL downstream channel, which provides a 9Mb/s data rate. ADSL is a Frequency Division Multiple Access system, which can transmit voice and data simultaneously.

The cable Data modem provides high speed data access over a different medium—the coaxial TV cable instead of the copper phone line. It uses a 6Mhz video channel, which is one of the channels that transmit TV program signals, to provide a 30Mb/s downstream data transmission rate and a 3Mb/s [6] upstream data transmission rate.

The difference between ADSL and the cable modem is the service type: ADSL provides dedicated service and the cable modem provides shared access service. ADSL uses a dedicated circuit connection. A user connects to the host computer via a dedicated telephone line. The cable modem uses a shared TV cable. A single high bandwidth cable is shared among users. The available bandwidth per user depends on how many users are currently active on the same channel. Thus, an ADSL user is guaranteed a 9Mb/s data access rate no matter how many other users use the service simultaneously. The 9Mb/s data rate is the user's maximum available bandwidth. Unlike ADSL, the cable modem user can have a much higher peak data access rate—30Mb/s. As there are other cable modem users sharing the same channel, there is no guarantee what data rate is available to a single user. During a busy period, the available bandwidth may be lower than the data rate ADSL provides.

In this paper, the performance analysis of the two techniques is presented. Although ADSL and the cable modem both have technical advantages and weaknesses, it is not known whether the two techniques perform differently from the users' perspective. Moreover, the Quality of Service (QoS) of the Internet depends on many factors. For example, if the bottleneck of a connection happens in the middle of the Internet, or at the other end of the connection, such as a busy web server, the available data access rate is limited by the bottleneck speed or the busy web server's responding time. Thus, whether the user has a 9Mb/s ADSL connection or a 30Mb/s cable modem connection, it does not improve the QoS. Besides, as a large amount of applications, such as WWW(HTTP), FTP-data and Email(SMTP) are implemented on top of TCP, the performance of these applications also depends on the implementation of TCP [12]. In addition, when evaluating the two techniques, the pattern of human's web surfing behavior has to be considered. When a user requests a web page, he needs some time to process it before he issues another request. Thus, it is unlikely that a single user will fully load a high speed link.

1.2 Testing Methods

Network protocols or techniques are tested in two ways: simulation experiments and experiments on real functional networks or hardware. In [10], Henrik and others tested network performance effects of HTTP on three different environments: LAN(Local Area Network), WAN(Wide Area Network) and PPP(Point-to-Point Protocol). According to their experience, data collected from WAN were unstable due to the fluctuation of the Internet traffic. Only 3-5 repeated experiments were listed in their paper. Our tests faced the same prob-

lem. Operational networks and hardware are less controllable than a simulated environment. Therefore, it is difficult to reproduce experiments under the same environment. In addition, different vendors of ADSL modems and cable modems have products of different qualify. Our results should not be based on specific hardware. Thus, we decided to perform our tests on a software simulation environment.

The most widely used simulator in academia is ns [5]. The old version of ns does not emulate real traffic. A new software module—the ns emulator—is still under development. Luigi Rizzo has built a simulator called "*dummynet*" [11], which is capable of simulating four features of a link: the maximum queue length, the propagation delay, the bandwidth and the loss rate. However, the implementation of *dummynet* uses the unix timer interrupt to account for the time elapse. Usually, the granularity of unix timers, by default, is 10ms, which implies *dummynet* cannot accurately simulate links with speed faster than 1 packet per 10ms. Increasing the timer's granularity causes other performance problems. Another wide area network emulation tool, Delayline[7], is built into user space. Applications have to be recompiled in order to use the simulation tool.

As no satisfying simulation tools were found, we developed our own one as the first step of our project. This simulation tool will also facilitate other researchers' work. The simulator was developed in FreeBSD kernel. It is capable of simulating a link of speed ranging from 28.8Kb/s to 100Mb/s. Using this simulator, we set up a testing bed for comparing ADSL and the cable modem. We conducted several experiments on this testing bed. The experimental results show that the two techniques are interchangeable.

2 Design and Implementation of the Link Simulator

2.1 The Simulating Algorithm

A link is characterized by three parameters: the bandwidth, the propagation delay and the maximum queue length. We want to simulate links of arbitrary combinations of the three parameters using a fixed-speed physical link. Our simulator was implemented in FreeBSD kernel.

The design goal of the simulator is to avoid using timer interrupts to account for the time elapse. There are three reasons. First, timer interrupts happen at a fixed interval and a high priority. The default interval of a FreeBSD system is 10ms, which means the resolution of the timer is 10ms. A link that takes 11-19ms to transfer a certain size of packet may be approximated by a link that takes 20ms to transfer the packet. For example, in the implementation of *dummynet*, packets are pulled out from the delay pipe at the simulated rate per tick, e.g., every 10ms. Packets that take 11-19ms to transfer will wait for 2 ticks to be transferred. Thus, the effect of simulation is not accurate. Second, increasing the timer granularity increases the system overhead. If it takes t time to run a timer interrupt and the interrupt happens at the interval T, the system overhead for timer interrupts is t/T. If t is greater than T, the timer will miss the next interrupt, causing the system to lose time [9]. As timer interrupts run at a high priority, most of the other system activities are blocked during timer interrupts. This blocking might cause I/O controllers such as the network controller and the disk controller to lose incoming data. The third reason to avoid

using timer interrupts is that packets cannot be evenly spaced out according to the simulated bandwidth. Suppose the simulated bandwidth is $simbw$. Every T time, $simbw * T$ data will be scheduled to leave the system. Since this amount of data is transfered at the real link speed bw, the spacing between each packet is $pktlen/bw$, instead of $pktlen/simbw$.

Instead of using timer interrupts, we utilize a link's own clocking ability—its bandwidth—for timing. Figure 1 illustrates the underlying concept of our algorithm. We insert dummy packets[1] in front of a real packet to simulate the phenomenon that the packet is transfered by a link of speed $simbw$ and delay $simdelay$. This algorithm can simulate a wide range of link speeds accurately without heavy loading the system. Also, inserting dummybytes evenly spaces out each packet.

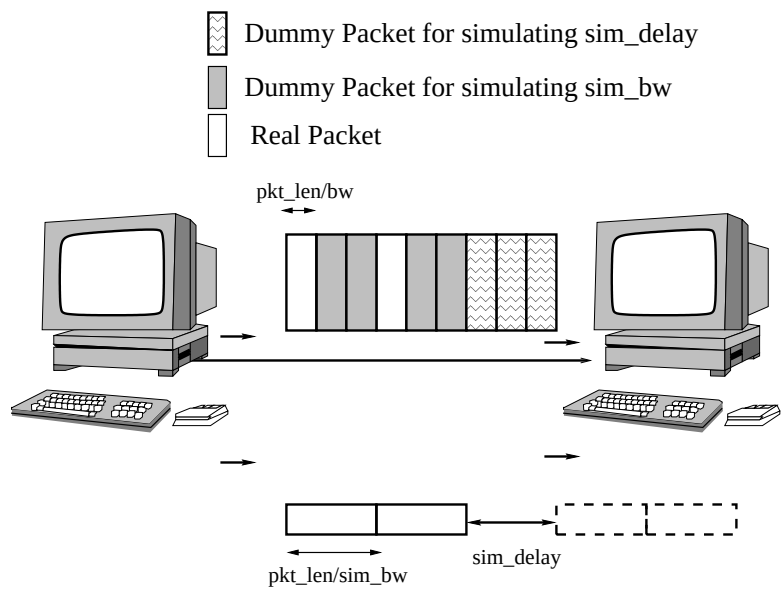

Figure 1: Inserting dummy packets to simulate $simbw$ and $simdelay$

Suppose the simulated link has bandwidth $simbw$, propagation delay $simdelay$, and link layer overhead $simoh$. The real link in our simulation environment has bandwidth bw and link layer overhead oh. The number of dummy bytes to insert can be computed based on $simbw$ and $simdelay$. Suppose the packet i of length L_i arrives at time a_i. Imagine that packet i arrives at the simulated link instead of the real link. The departure time d_i^s of i at the simulated link depends on its arrival time a_i and the departure time of packet $i-1$. If a_i (the arrival time of packet i) is later than d_{i-1}^s (the departure time of last packet $i-1$), packet i sees an empty queue at the simulated link and it is transmitted immediately. The total bytes transimitted, including the simulated link layer overhead, is $L_i^s = L_i + simoh$. So $d_i^s = a_i + L_i^s/simbw$. If a_i is earlier than d_{i-1}^s, packet i will not be transmitted until packet $i-1$ departures. So $d_i^s = d_{i-1}^s + L_i^s/simbw$. In a summary, the departure time of packet i at the simulated link can be computed as

$$d_i^s = max(d_{i-1}^s, a_i) + L_i^s/simbw \tag{1}$$

[1] All dummy packets have an illegal Ethernet address and will be discarded at the other end.

After packet i departures from one end of the simulated link, it takes *simdelay* (the propagation delay of the simulated link) for packet i to arrive at the other end of the link. The arrival time A_i of i at the other end of the simulated link is

$$A_i = d_i^s + simdelay \tag{2}$$

In our simulator, we insert a certain number of *dummybytes* in front of packet i to actually hold it up until time A_i. Thus, we make the packet behave as if it passes through a link of simulated bandwidth *simbw* and delay *simdelay*. Similarly, the departure time d_i of the packet i at the real link depends on its arrival time a_i and d_{i-1} (the departure time of last packet $i-1$ at the real link). The total bytes transmitted for packet i is $L_i^r = L_i + oh$. The inserted *dummybytes* can be computed as

$$d_i = max(d_{i-1}, a_i) + (dummybytes + L_i^r)/bw \tag{3}$$

The effect of simulation is to make the departure time d_i at the real link equal to the simulated arrival time at the other end of the simulated link A_i. That is

$$d_i = A_i \tag{4}$$

When the first packet arrivals at the system, it sees an empty queue. Thus, we know the initial conditions

$$\begin{aligned} d_0^s &= a_0 + L_0^s/simbw && (5)\\ d_0 &= a_0 + (dummybytes + L_0^r)/bw && (6)\\ A_0 &= d_0^s + simdelay && (7)\\ d_0 &= A_0 && (8) \end{aligned}$$

Starting from the inial conditions and combining Equations 1, 2, 3 and 4, we can recursively compute each packet's departure time d and the amount of inserted *dummybytes*.

When solving the equations, three scenarios may happen:

1. When a packet i arrives, both the output queue at the real link and the output queue at the simulated link are empty. So,

$$dummybytes = (L_i^s/simbw + simdelay) * bw - L_i^r \tag{9}$$

2. When a packet i arrives, the output queue at the simulated link is empty. However, since the output queue at the real link also simulates the propagation delay, it is not empty. In this case,

$$dummybytes = (a_i + L_i^s/simbw + simdelay - d_{i-1}) * bw - L_i^r \tag{10}$$

3. When a packet i arrives, both queues are not empty. The number of inserted dummybytes is

$$dummybytes = bw/simbw * L_i^s - L_i^r \tag{11}$$

2.2 The Implementation

We coded our algorithm into the FreeBSD kernel ether_output routine. The major challenge of the implementation lies in the management of memory buffers. The real link speed in our simulation environment is 100Mb/s. To simulate a 10Kb/s link without considering the propagation delay, we need to insert 9999 dummy packets for every real packet, which means one single packet will consume about 20M bytes[2] of memory. To queue 10 real packets, 200M bytes memory is needed. Obviously, the cost is expensive. As dummy packets do not contain any real data, there is no need to allocate extra memory for them. In our implementation, we allocate a master mcluster for all dummy packets. This mcluster is never freed. Every dummy packet has a pointer pointed to this mcluster. Thus, only one extra mbuf, which is of size 128 bytes in our system, is allocated for every dummy packet.

Another implementation problem is related to simulating the maximum queue size. Packets are transfered from memory to network interface before they leave a computer system. Usually we only know how many packets are inside the memory and we do not know how many packets are inside the interface. Thus, we do not know how many packets are queued inside a computer system. In order to keep track of the real output queue, we implemented a simulated queue. This queue keeps track of each packet's departure time. When a new packet arrives, based on its arrival time, we check how many packets have left the system and how many packets are in the simulated output queue. Therefore, we obtain the current queue size. The current arrival packet will be either enqueued, if the current queue size is less than the simulated maximum queue size or dropped otherwise.

3 Testing Results

We set up a test bed using this simulator. The test bed consists of three computers. As shown in Figure 2, one computer is set up as a web client that issues HTTP requests. The second computer acts as a router and runs the link speed simulator at both interfaces. The third computer is a web server. When the client sends a packet to the web server, the router intercepts this packet. At the output interface connecting the router to the web server, the router transmits the computed dummy bytes before it transmits the real packet. Similarly, in the other direction, when the web server sends the client a packet, the router also intercepts the packet and insert the computed number of dummy bytes before the real packet.

3.1 Ping Tests

To test how accurate our simulator works, we configured the simulator to simulate different link speeds, ranging from 10Kbps-100Mbps. The router sent 1000 ping packets of length 1066 (data + header) to the client. The client ran *tcpdump* to capture the ping packets and timestamped them. On the other hand, based on the simulated bandwidth, we computed the inter-arrival time of each packet, which is $pktlen/simbw$. Figure 3 shows the computed

[2] One packet usually consumes one mcluster, which is of size 2K in our system. Ten thousand dummy packets need 20M bytes memory.

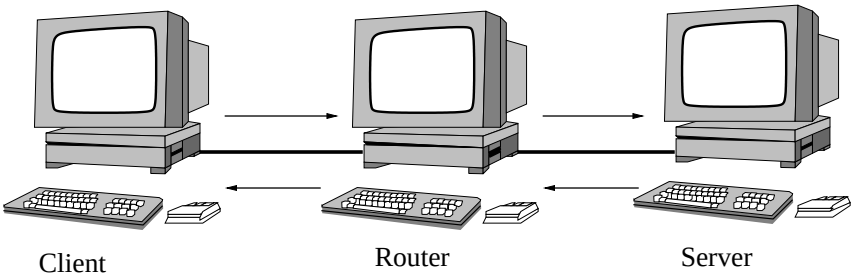

Figure 2: The Testing Bed

arrival time and the real arrival time of the last ping packet at different link speed. As can be seen in this figure, the simulator simulates a wide range of link speed accurately.

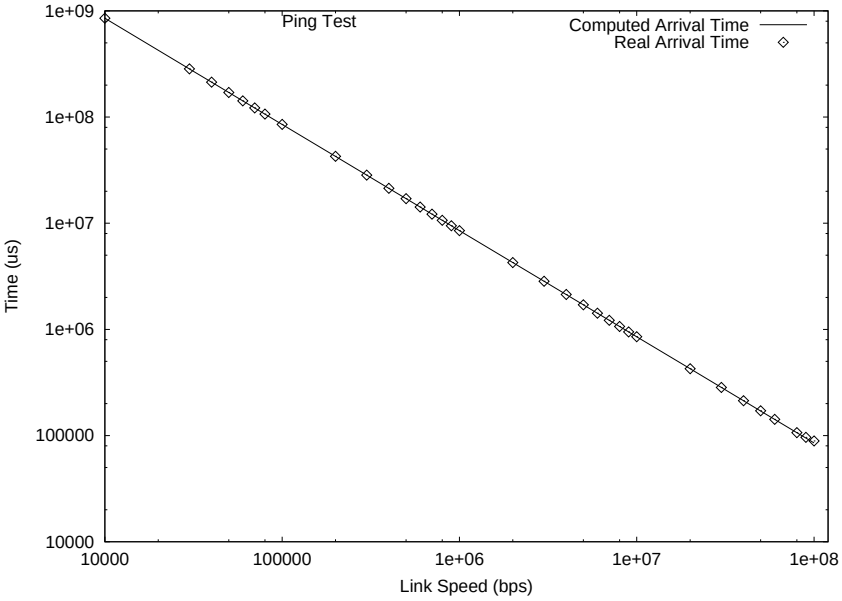

Figure 3: Ping Testing Results

Figure 4 and 5 present tests of bandwidth 10Kb/s and 80Mb/s. The computed arrival time and the real arrival time for each ping packet are shown in the figures. As they are too close, we cannot really see two lines. Figures 6 and 7 are one enlarged piece of each figure, respectively. The simulated results of both bandwidths match closely to their theoretical values.

3.2 TTCP Tests

We ran the program ttcp under different link speeds to test how TCP performs. During each run of the experiment, the program ttcp sends 100 buffers, which is of size 8192 bytes each, using TCP. The results are shown in Figure 8 and 9. The maximum queue length at the router is set to the maximum between 10 and the delay and bandwidth product of each connection.

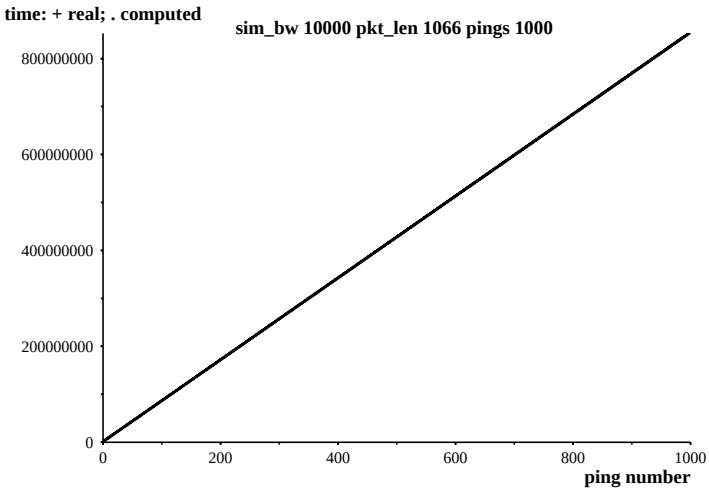

Figure 4: Ping Testing Results: 10Kb/s

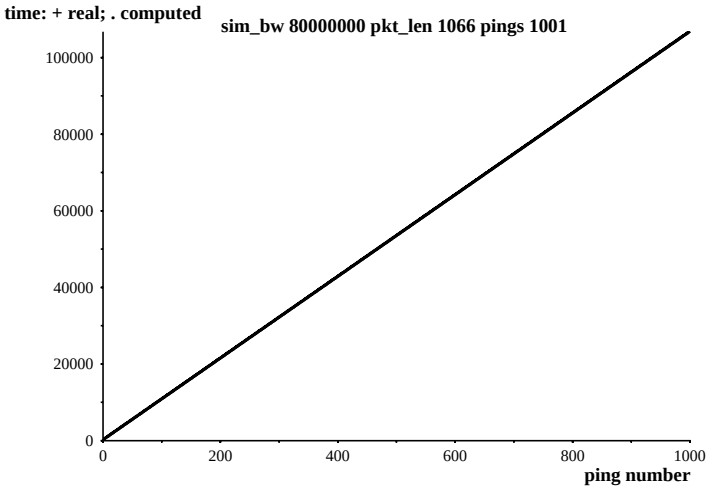

Figure 5: Ping Testing Results: 80Mb/s

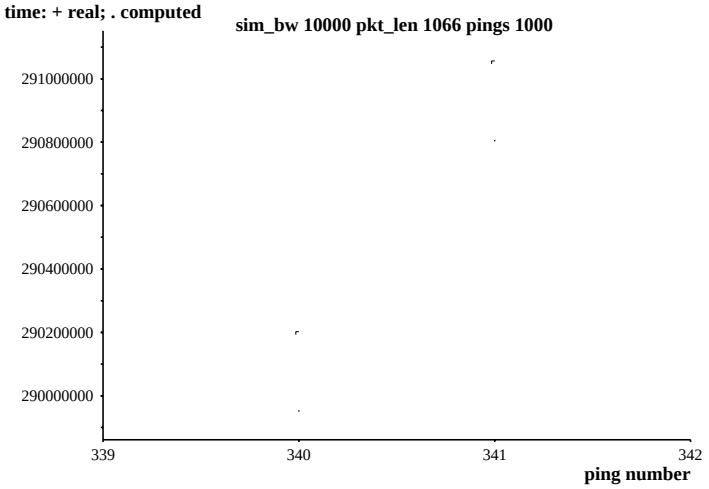

Figure 6: Ping Testing Results: 10Kb/s

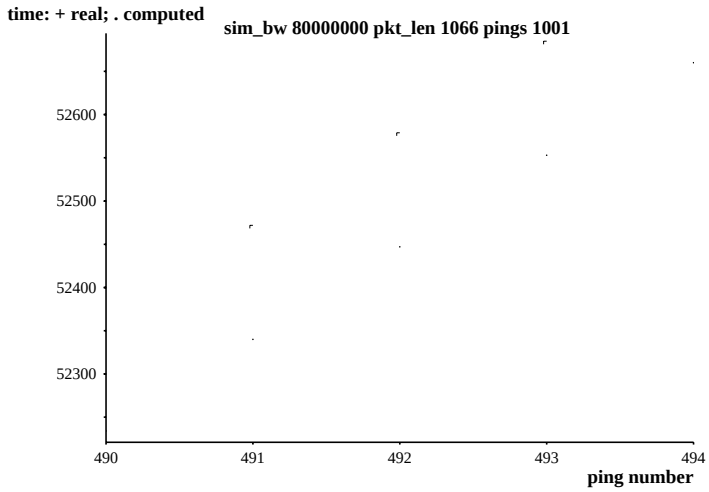

Figure 7: Ping Testing Results: 80Mb/s

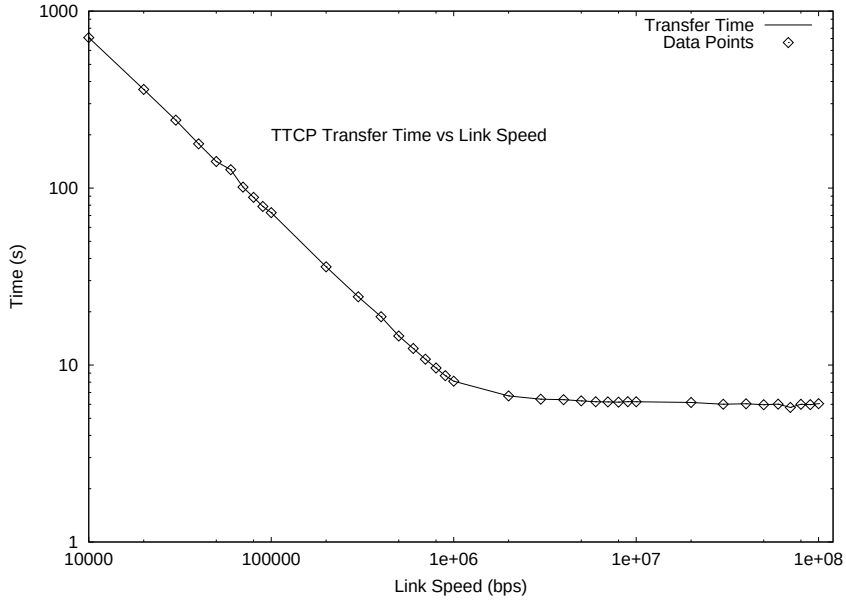

Figure 8: TTCP Testing Results: Transfer Time vs Link Speed

As can be seen from the figures, the throughputs under different link speeds were leveled out when the simulated speed reaches 1Mb/s. The link utilization drops dramatically after 1Mb/s. A further investigation of data traces shows that TCP at the server side has a maximum window size of 16384 bytes. As is known, TCP can send at most a window size of packets every round trip time. Thus, the throughput is limited by 16384*8/((50+50)/1000) = 1.3M. We show the data traces of 10Kb/s and 80Mb/s in Figure 10 and 11. In Figure 10, as the delay and bandwidth product is small, the maximum queue length is set to be 10. The window size of the connection is limited by the congestion window size. In Figure 11, no packet drop is present. The window size is limited by the sender's sending buffer size. So, even given infinite link speed, with the current implementation of TCP, a connection is unable to make full utilization of the link's bandwidth.

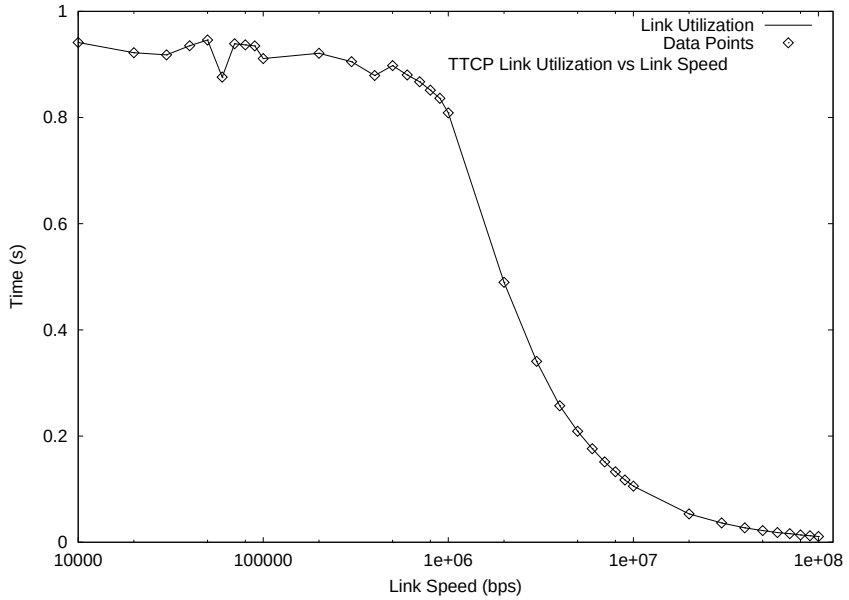

Figure 9: TTCP Testing Results: Link Utilization vs Link Speed

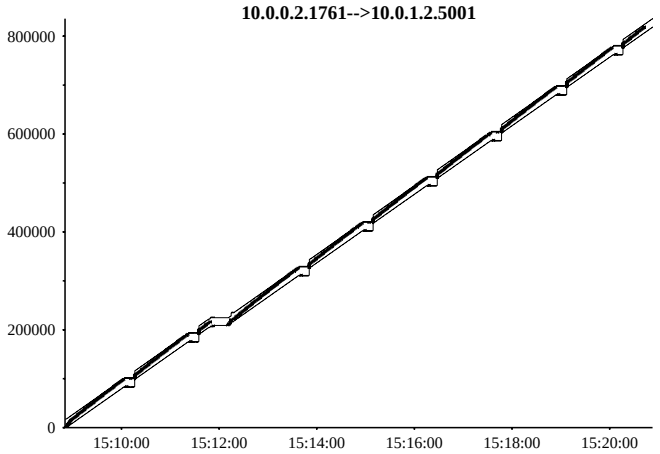

Figure 10: TTCP Testing Results: 10Kb/s

3.3 Web Transfer Tests

A significant portion of Internet traffic is WWW (HTTP) [8]. So we chose the web as a typical application to test ADSL and the cable modem. Webbot 5.1m from libwww [4] is used as the client program. Apache/1.3b5 [2] is used as the web server. We used the same testing page as used in [10]. The page consists of 42 inlined images and one main document. The webbot sequentially requests 43 documents via one TCP connection. This mimics the human's web surfing pattern, i.e., issuing a new request after processing a received page. The total size of the documents is 167KB. The propagation delay of each way is 50ms.

The results are shown in Figure 12 and 13. As can been seen, both the transfer time and the link utilization flatten out after the link speed reaches about 1Mb/s.

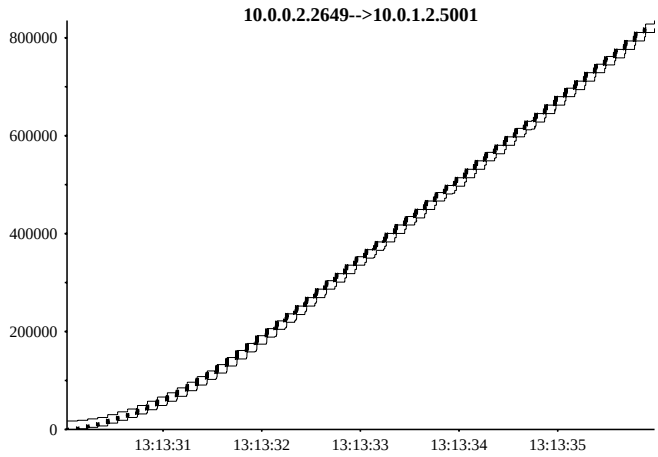

Figure 11: TTCP Testing Results: 80Mb/s

As we chose HTTP1.1 without pipelining, the request for the next file is sent after the reception of the current file. At least 43 round trips are needed to fetch all documents. So the throughput is limited by $167KB/(43 * 100ms) = 311Kb/s$. Though HTTP1.1 without pipelining is not an efficient implementation, it simulates human's web surfing behavior. In our test, the link utilization of the simulated ADSL link is about 2.9%. And the link utilization of the simulated cable modem link is about 0.87%. The results show that the performance is not limited by the link speed. About $1/0.87\% = 115$ simultaneous active users will fully load a cable modem channel. Suppose the average number of homes on a single cable channel is 2000 [3], 65% homes take cable. The average number of cable subscribers is 1300. 33% of cable subscribers take data service and during peak usage, 30% of them are active. Then the average number of simultaneous active users is about $1300 * 33\% * 30\% = 129$. This number is just slightly above the number of users that fully load the system. Thus, we conclude in most cases, ADSL and the cable modem will provide the same quality of service.

4 Conclusions

In this paper, we described the design and implementation of a wide range link speed simulator. Testing results show that this simulator is capable of simulating links of various speeds accurately. Using this simulator, we tested the performance of HTTP1.1 under different link speed. The results show that when link speed exceeds a certain threshold, the performance of the application levels out. Thus, we conclude ADSL and the cable modem will basically provide the same service. Users are free to choose either technique.

5 Acknowledgments

This paper was inspired by my advisor David Clark. I also like to thank my officemate Tim Shepard for his help and patience.

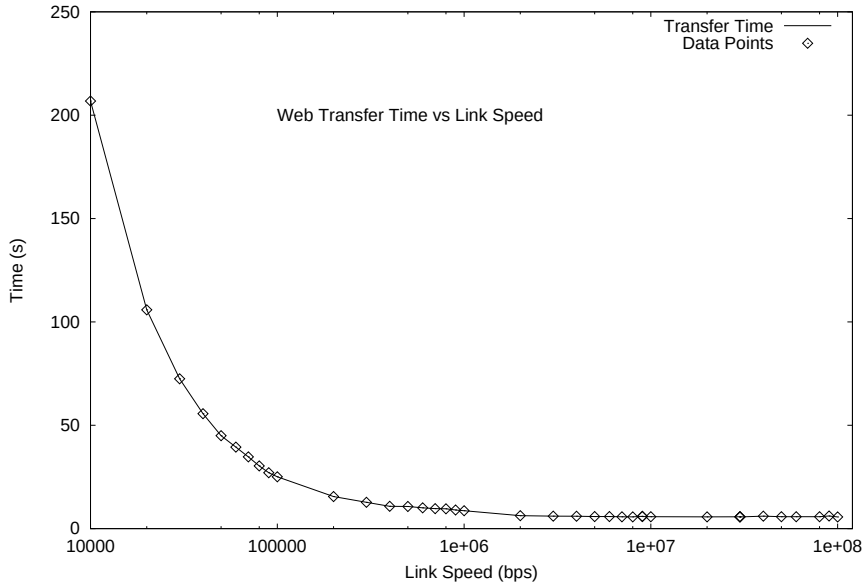

Figure 12: Web Testing Results

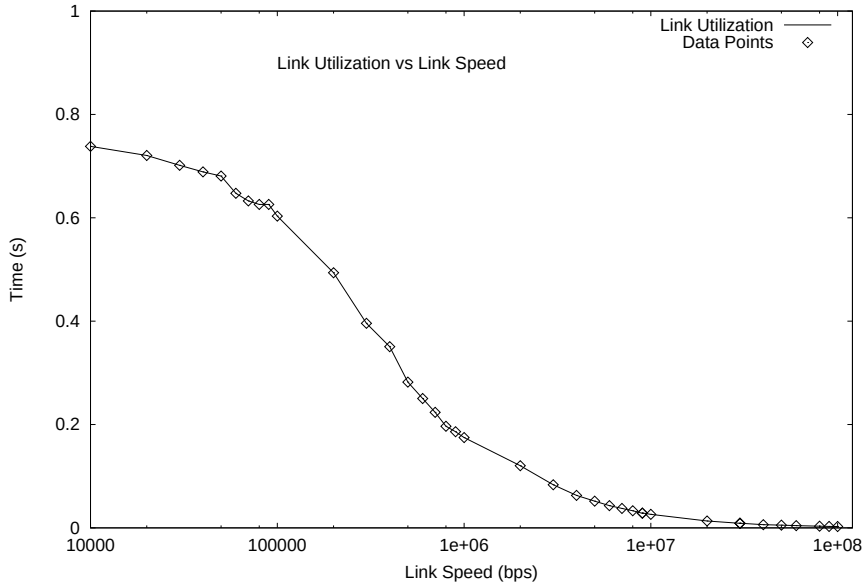

Figure 13: Link Utilization

References

[1] Adsl tutorial. http://www.adsl.com.

[2] apache. http://www.apache.org/.

[3] Cable data modems. http://www.cablelabs.com/Publications.html.

[4] libwww. http://www.w3.org/.

[5] ns. http://www-mash.cs.berkeley.edu/ns/ns.html.

[6] What is a cable modem. http://www.godset.dk/cablemodem/.

[7] D.B. Ingham and G.D. Parrington. Delayline: A wide-area network emulation tool. In *USENIX Computing Systems*, volume 7, pages 313–332, 1994.

[8] K.Thompson, G.Miller, and M.Wilder. Wide-area Internet Traffic Patterns and Characteristics. *IEEE Network*, 11(6), November-December 1997.

[9] Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, and John S. Quarterman. *The Design and Implemenation of the 4.4BSD Operating System*. Addison-Wesley Pulishing Company, 1996.

[10] Henrik Frystyk Nielsen, Jim Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Hakon Lie, and Chris Lilley. Network performance effects of http/1.1, css1, and png. In *ACM SIGCOMM'97*, September 1997.

[11] Luigi Rizzo. Dummynet. In *ACM Computer Communication Review*, volume 27, pages 31–41, January 1997.

[12] Jeffrey Semke, Jamshid Mahdavi, and Mattew Mathis. Automatic TCP Buffer Tuning. In *SIGCOMM'98*, volume 28, October 1998.