Bandwidth Estimation and Rate Control in BitVampire

CPSC 527: Advanced Computer Networks

Mayukh Saubhasik
Department of Computer Science
University of British Columbia
Vancouver, B.C.
mayukh@cs.ubc.ca

Uwe Schmidt
Department of Computer Science
University of British Columbia
Vancouver, B.C.
uschmidt@cs.ubc.ca

ABSTRACT

We consider the problem of bandwidth estimation and rate control in the context of the peer-to-peer video streaming application BitVampire. BitVampire requires each peer to maintain an estimate of its available upstream bandwidth. This is needed by a scheduling algorithm which takes information about peer's bandwidth into account in order to minimize the initial buffering time when watching a video. It is also an interesting problem in general, since P2P applications can generally offer a better quality of service if they are aware of their bandwidth limitations. We looked into the area of bandwidth estimation and evaluated their applicability for the context of P2P applications. We found that it is not easy to use them in this context since most of them require very accurate timing and low level network access which is often not given by high-level programming languages like Java. We went for a pragmatic approach instead, which uses round-trip-time measurements as an indicator for the available bandwidth of a peer. We did a prototype implementation in Java and evaluated it in the LAN environment. Future work has to investigate about the choice of many policy decisions which arose when creating the prototype.

1. MOTIVATION

This project is largely motivated by the need for accurate bandwidth estimation in BitVampire [Liu and Vuong, 2006]. BitVampire is a peer-to-peer (P2P) video streaming software, which is being actively developed here at UBC. BitVampire requires each peer to maintain an estimate of its available upstream bandwidth. This is needed by a scheduling algorithm which takes information about peer's bandwidth into account in order to minimize the initial buffering time when watching a video. Although we are motivated by BitVampire, bandwidth estimation is also an interesting problem in general, since P2P applications can generally offer a better quality of service if they are aware of their bandwidth limitations.

BitVampire is a cost-effective P2P video streaming solution, which aims to collate the individual peer's resources to support large scale on-demand media streaming. The basic idea of BitVampire is to split a published video in various segments and distribute these segments among the peers. When a peer requests to watch a video, it creates a schedule which aggregates the bandwidth from various other peers to stream the video. In order to do that it sends a request to a number of peers, in possession of parts of the movie, demanding a certain bandwidth value. Each of these peers replies whether they can satisfy the request or not, subject to their available bandwidth.

We are giving a brief overview over the rich body of work which has already investigated in this problem. In section 2.4 we will explain why these methods aren't applicable in our context. As a result, we came up with a pragmatic approach which is applicable in the context of any generic P2P application. Although the accuracy of the method is quite low, it makes up for this in terms of speed and ease of implementation.

2. RELATED WORK

The bandwidth of a link can either be measured by passively monitoring the amount of traffic flowing through it, or by actively probing the link to estimate its bandwidth. The passive monitoring mechanism requires access to the intermediate network elements or administrative resources, whereas the active probing techniques can be run just on the end hosts. Thus passive monitoring methods aren't really feasible except in a well controlled network testbed.

Active bandwidth estimation is a well studied field, which has been around for at least 10 years. Despite that, there are quite a few open problems within it. What makes the problem hard is its real-time nature and the need for a balance between the amount of traffic generated, time taken and accuracy of the measurements. Similar to all active measurement techniques, the very act of measuring the bandwidth tends to change it. This makes the problem even more challenging.

2.1 Terminology

A *link* is defined to be the interconnect between any two network elements. For example, this could be between the an end host and its gateway router or between any two intermediate routers.

Each link is modelled to have a constant delay and bandwidth. The *link latency* is a constant delay which is a property of the physical medium being used for transmission, this value is independent of the amount of data being transferred. The bandwidth characterizes the rate at which data can be sent over the link [Curtis and McGregor, 2001].

Therefore for a packet size P, the time taken to transmit it over a link t is given by

$$t = \frac{P}{b} + l \qquad (1)$$

, where b is the link bandwidth and l the link latency. Some of the key terms, used in further discussion, are:

Capacity bandwidth is the maximum bandwidth available on a link. This is a static value which doesn't change over time.

Available bandwidth refers to the *currently* available bandwidth on a link. Thus this value may continuously vary over time.

Bottleneck link of a path refers to the link with the minimum capacity bandwidth among all the links on a path.

Path capacity bandwidth denotes the capacity bandwidth of the bottleneck link.

Path available bandwidth is the available bandwidth of the bottleneck link.

Cross traffic refers to all traffic going over a link, apart from the traffic generated by the method itself.

We are mostly interested in the path capacity bandwidth and the path available bandwidth. Note that a path in our case is the sequence of links from one peer to another.

2.2 Single Packet Techniques

Single packet techniques work by sending probe packets, which are then echoed back by the destination. The measured round-trip-time (RTT) values are used to form an estimate for the link bandwidth [Curtis and McGregor, 2001].

For example there are nodes A and B and a single link between them. Node A sends a probe packet over the link to node B. On getting the probe packet, node B simply echos it back. On receipt of the response packet, node A calculates the RTT value for it. This process is repeated for different packet sizes to give a graph similar to figure 1 where packet sizes are plotted against the RTTs. If the points do not align on a perfect line, a linear fit is used to arrange the measured data onto a line. As is evident from equation 1 the slope in such a plot gives the bandwidth value and the constant intercept on the time axis is the fixed link latency.

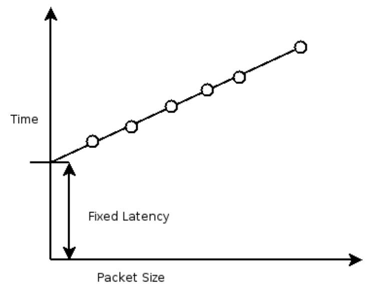

Figure 1: Plot of packet size vs transmission time

The problem with this approach is that cross-traffic is delaying the transmission of the probe packets. Thus to calculate the link capacity bandwidth, the measurements for each packet size are repeated multiple times to discount for cross traffic. The assumption is that at least one probe for each packet size won't get delayed by cross-traffic. For each packet size, only the minimum recorded RTT is taken into account while calculating the linear fit as illustrated in figure 2.

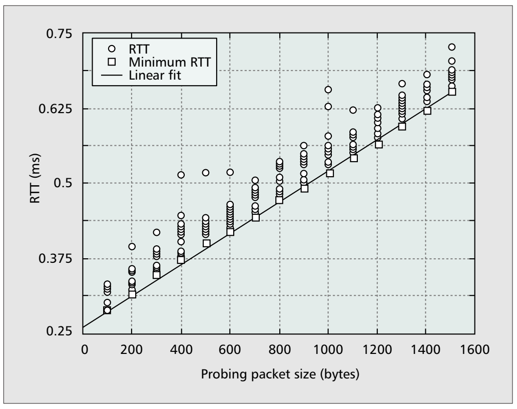

Figure 2: Multiple measurements. Source: [Prasad et al., 2003]

In order to get the path capacity bandwidth using this method, the link capacity bandwidth for each link along the path has to be calculated. The RTT value for a link N is given by equation 2, wherein we take into account the RTT values for all preceding links $1, \ldots, N$. Therefore if one knows the RTT values for all the preceding links, one can easily calculate the link bandwidth for the current link. Starting from the source, each successive router along the path to the destination is forced to reply to the UDP ping, by setting the appropriate time to live (TTL) field in the IP packet header. Each router decrements the value of this field, and the router which decrements it to zero, replies back to the source with a special ICMP packet. Thus by increasing the range of the UDP ping packet one hop at a time, the bandwidth for each individual link can be calculated.

$$t_N = \frac{P}{b_n} + l_N + \sum_{i=1}^{N-1} t_i \qquad (2)$$

2.3 Packet Pair Techniques

The packet pair techniques are ideal for measuring the path capacity / available bandwidth. They work by sending two equal sized packets one after another from the source node to the destination node. The destination node measures the arrival delay between the two packets. The delay between the packets is determined by the queuing delay on the bottleneck link along the way [Lai and Baker, 2001]. A simple proof for this is as follows:

Let us consider a scenario as shown in figure 4. The packets arrive at some intermediate router 0 with a inter-packet delay of x. Further for the sake of simplicity let us assume $packet_1$ arrives at time zero, therefore $packet_2$ arrives at time x.

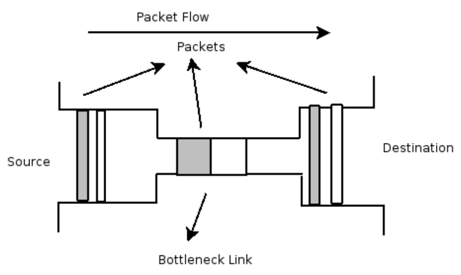

Figure 3: The delay between the packets, caused by queuing at the bottleneck link, remains constant throughout the rest of the path.

If the time required to send the packet over link 1 is d, then $packet_1$ arrives at router 1 at time d. Since link 1 is the bottleneck link $x < d$, therefore $packet_2$ has to wait till $packet_1$ is completely transferred before it begins transmission. Therefore $packet_2$ arrives at time x, and waits up to time d to begin transmission, and consequently arrives at router 1 at time $d + d = 2d$. Therefore the new inter-packet delay is $2d - (d) = d$.

Now, let us consider the next hop, assume the time to transfer over this link is d'. $d > d'$ as link 1 is the bottleneck. Now $packet_1$ arrives at router 2 at time $d + d'$ and $packet_2$ arrives at time $2d + d'$. $packet_1$ finishes transmitting from router 1 at time $d + d'$, and $packet_2$ arrives at time $2d$. Since $d > d' \implies 2d > d + d'$, thus there is no queuing at router 1.

The new inter-packet delay $= 2d + d' - (d + d') = d$, thus the inter-packet delay stays constant at d.

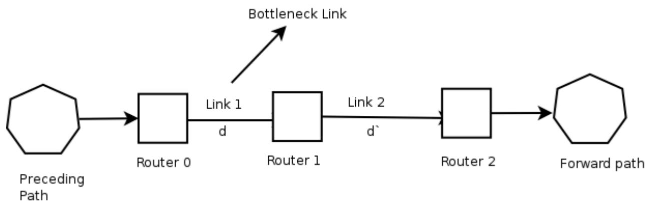

Figure 4: Example scenario, for a simple proof

The inter-packet delay is equal to the time the router at the end of the limiting link spent receiving the second packet, after the first packet was received. Since the second packet was already queued up in the sending router for a while before it got transmitted, it can utilize this time to do all the required look-ups needed to forward the packet. Thus the second packet does not incur the overhead of the fixed delay while being sent. Equation 3 is used to calculate the bandwidth value.

$$d = \frac{P}{\tilde{b}} \tag{3}$$

where d is the inter-packet delay, P is the packet size, $\tilde{b}$ is the is the bottleneck link bandwidth. Similar to the single packet technique, multiple measurements are needed to discount for cross traffic and thus to accurately calculate the path capacity bandwidth.

2.4 Shortcomings

Both of the above methods for bandwidth estimation have their limitations. Additionally there are problems to use them in our context of BitVampire.

2.4.1 Single Packet Techniques

- It assumes that the link bandwidths are symmetric in nature, i.e. the bandwidth for the forward and reverse path is the same. This assumption is mostly not true.
- To calculate the bandwidth for a certain link on the path, the method relies on the RTT measurements for all the preceding links. The method fails if any of the routers along the path does not respond to the probe packets.
- It assumes that the packet transfer time is directly proportional to the packet size. This may not be true. For example, some router may copy a 128 byte data packet disproportionately faster than a 129 byte packet.

2.4.2 Packet Pair Techniques

- The measurement software needs to be running at both ends.
- The method assumes that the routers perform *First in First Out (FIFO)* queuing, this may not always be true. Some routers perform a weighted fair queuing.
- Similar to the single packet technique it assumes that the time to transfer is directly proportional to the packet size.
- It requires the ability to measure time at a fine grained level, since the inter-packet delay tends to be quite small as compared to a RTT measurement.

2.4.3 Problems in the context of BitVampire

Both methods are hard to implement and use in the context of BitVampire. The BitVampire code base is entirely written in Java. Java networking does not have any support for raw sockets. Therefore it is not possible to set the TTL value for a packet or even send out an ICMP packet using Java's networking library. Java only supports time measurement at the level of milliseconds, this is not enough for the packet pair techniques to work in all scenarios. In case of high-speed links, the inter-packet delay can be in the range of a few microseconds.

To calculate the path capacity bandwidth, both methods need to send a lot of probe packets and thus a considerable amount of time to discount for the cross traffic. This is not really a feasible solution in our context, since we need to update the estimated bandwidth value frequently.

3. PRAGMATIC APPROACH

This section deals with the pragmatic approach that we took because of the aforementioned problems.

3.1 Assumptions

All traffic, UDP and TCP, has to go through the same IP send buffer on a single peer. Uploading at a very fast rate causes the IP send buffer to be congested and thus slows down other uploads on the same machine. But not only that, TCP downloads are also affected because TCP requires to send acknowledgment packets (ACKs) to provide reliable data transfer. In summary, indiscriminate uploading will slow down all downloads which is a serious problem for a P2P application.

BitTorrent had the same problem, and as a effect most BitTorrent clients introduced the possibility for the user to manually set an upload rate limit to prevent the slowdown of incoming traffic.

3.2 Problems to be solved

We are concerned with maintaining a single value Bw for the available upstream bandwidth which can be used by the BitVampire application. Bw varies over time because of other applications using network resources. We additionally need the value Bw_{avail} which is needed to decide whether to admit a new upload request (with a certain rate) by another peer. Let $U_1, \dots, U_N$ be the currently running uploads, then Bw_{avail} can be simply calculated as

$$Bw_{avail} = Bw - \sum_{i=1}^{N} rate(U_i) \tag{4}$$

, where $rate(U_i)$ is the maximum allowed upload rate for this upload. This value is explicitly requested by the recipient in the beginning and it is guaranteed that the upload rate will stay below that value at all times. To give this guarantee, we have to throttle the upload to each peer individually.

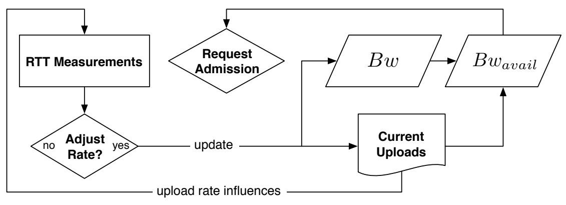

Figure 5: Our approach

The most important and challenging part is how to make the decision whether the global upload rate has to be adjusted. Even when the decision has been made, it still has to be decided by how much to adjust the rate and how to distribute the change to multiple running uploads. The goal of *decreasing* the upload rate is to avoid slowing down other connections on the same machine. Using a very slow upload rate, far below the available upstream bandwidth, wastes precious network resources on the other hand. Thus we also have to consider *increasing* the upload rate.

Our proposed solution about these problems is discussed in the following.

3.3 Available Bandwidth Detection

We were initially concerned with the automatic detection of the user's maximum upstream bandwidth. We wanted to avoid letting the user set it manually because she may often not know what her maximum upload rate, assigned by her Internet Service Provider (ISP), is. Even if the user is aware of that, she could change locations and connect to a different network and thus would have to reconfigure this value. In addition to that, what we really want is the currently available bandwidth to the BitVampire application (Bw). This value potentially varies a lot over time and is therefore not static like the maximum upstream bandwidth.

Our approach is to periodically send probe packets (like *ping*) to other peers and measure the observed RTT. The RTT is an indicator for the queue length of the *IP send buffer*, since a longer queue causes a longer delay to send the data (figure 6).

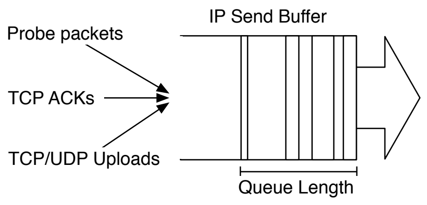

Figure 6: IP Send Buffer

The delay caused by the recipient of a probe packet is included in the RTT value, so that an increased RTT not necessarily gives us information about our own queue length. But if we send probes to multiple peers at the same time and collect all RTT measurements, we can assume that the delays caused by each peer are uncorrelated. Thus if we aggregate all RTT measurements to multiple peers at the same time, we can find out about the level of congestion of our IP send buffer. This level of congestion gives information about the utilization of the network link. For instance high RTTs indicate a long IP send buffer queue which indicates low available bandwidth.

Additional to the RTT, we also measure the inter-packet delay, using a packet pair technique like in section 2.3. We send two packets back-to-back to the other side. When the first packet is received, the recipient records the arrival-time for that packet. Upon the arrival of the second packet, the delay between the packet arrivals is calculated and included in the response to the sender. We included this form of measurements because it is easy to implement and gives us additional information which may help to develop better decision methods.

3.4 Closed-loop Control Systems

Our model of RTTs and upload rate can be considered a *closed-loop control system* [Barr, 2003], since it utilizes *feedback* in form of RTT measurements to control the *input signal* (the upload rates) of a *plant* (the currently running uploads). A diagram of this can be seen in figure 5.

A simple form of control in these systems is *proportional-integral-derivative (PID)* control. This method makes use of the proportional, integral and derivative part of the measured feedback value. To use an example from [Barr, 2003], the goal could be to control the temperature of a room with a heater. The measured input signal is the temperature given by a thermometer. Then the state of the heater is controlled until the desired temperature is achieved. One thing must be noticed here: A desired value for the temperature must be given, otherwise it is not clear how to control the heater.

3.4.1 Azureus Auto Speed

The *Auto Speed* - plugin [Chalouhi, 2006] for the BitTorrent-Client *Azureus*[1] is presumably based on the same assumptions as explained in section 3.1. It adjusts the global upload-rate of Azureus based on the periodically measured RTTs (pings) to a single fixed host (e.g. `www.google.com`). At the time of writing, it is one of the most popular plugins for Azureus, being downloaded over 33500 times.

It can be considered a simple proportional controller to adjust the global upload rate given the RTT measurements. Given a desired target value *desired* for the RTT, it controls the upload rate using the average of the last 5 RTT measurements *average*. It checks whether

$$2/3 \cdot desired \leq average \leq 3/2 \cdot desired \tag{5}$$

, and only if *average* lies outside these bounds, it adjusts the upload rate as follows:

- Decrease upload rate *proportionally* by $1 + \frac{4}{9}\frac{average}{desired}$, if $average > 3/2 \cdot desired$.
- Increase upload rate *proportionally* by $1 + \frac{4}{9}\frac{desired}{average}$, if $average < 2/3 \cdot desired$.

Please note that this simple controller needs a user set target value *desired*, otherwise it is not possible to control the RTT measurements.

3.4.2 Problems

The problem with the above approach is that we ping multiple peers at the same time in our case, which can be located anywhere in the world. Thus the observed RTTs for each peer will vary a lot and it is not clear what the desired value for each peer is.

We could take the same approach as the Auto Speed plugin and just ping a single host (ping server), in which case we may assume a known target value, at least for peers in a certain geographic area. This approach also has its problems. If all peers are pinging the same host, it could become overloaded given a large number of peers pinging at the same time. The ping server would also become a single source of failure. The whole bandwidth estimation and rate control would not work anymore if it is down. But even variations in buffer congestion on the ping server side will affect the RTTs for all peers in the network. Thus all peers will change their upload rate, which would lead to a sudden change for the whole P2P network.

These are all reasons why we took the approach to ping a set of peers. We have the problem that we don't have a target value for the RTTs. This makes it impossible to control the upload rates in a way which results in stable RTT measurements. Our current decision method only takes the change in RTTs into account and controls the upload rate based on that. But this will never result in a stable system.

[1]`http://azureus.sourceforge.net`

4. IMPLEMENTATION

This section deals with the implementation of our proposed approach and gives details on the various parts of it. We didn't modify the current source code of BitVampire since it is heavily modified at the moment. Instead we wrote a separate server and client application to test our approach.

The client is started on demand and and connects to the server. It simply requests a download with a random bit-rate. During this start-up, the client also starts a `UdpPingServer`, which runs in a separate thread on the client side. Its purpose is to wait in the background until the server sends a UDP packet probe. It replies in an appropriate way (see section 3.3) on receipt of a packet.

The server, once started, waits for incoming upload requests by clients. An incoming request with rate r is only accepted if $r \leq Bw_{avail}$. Bw_{avail} is calculated as in equation 4. If accepted, a new throttled upload with the requested rate as limit is created. The details on upload throttling are explained in the next section. Additionally, the new client is added to the list of peers which are constantly "pinged" to measure the RTTs. This is done by the class `RoundTripProber` which periodically sends probe packets to a set of peers. Two components are fed the RTTs after each "round of probe packets": the GUI to visualize the data and the `Brain` which is currently recording the history of RTTs for each peer. The Brain is also responsible for making the decision whether to adjust the upload rate. If the decision is yes, it also has to decide by how much to change the upload rate.

4.1 Bandwidth Throttling

To throttle the bandwidth of a stream, we sub-divide each second into smaller windows depending on the granularity of required control. Given the desired bandwidth in terms of bytes per second, and given the window size, one can calculate the bytes per window. Then keep track of the number of bytes written per window, and disallow any more writes once the per window quota is reached. The write call is blocked till the end of this current window. Once a new window begins the quota is replenished. The accuracy of control depends on the window size. A smaller window size gives better accuracy at the cost of higher and more frequent computation. A window size of 10 milliseconds seems to work well in practice.

4.2 RTT Probing

The collected RTTs usually show a lot of variation. They have a lot of outliers in form of high RTTs even if most values are very low (figure 7). For this reason, we calculated the gradients (rate of change) for the last 20 measurements and took the average value of them as an indicator whether the RTTs in general are going up or not. We currently only consider the RTT values to a single peer. The average gradient is very useful to "filter out" outliers in form of occasional spikes.

4.3 Policy Decisions

In our implementation we took a number of policy decisions. These techniques and parametric values are subject to improvement for better performance and accuracy. Presented

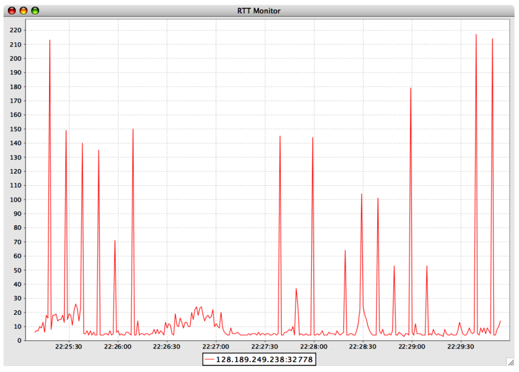

Figure 7: RTTs for transfer over Wireless LAN

below are the most important decisions we made, along with an explanation of the trade-offs.

Ping frequency: How often to ping potentially determines the rate at which we make adjustments to our estimated bandwidth value. If this frequency is too low, our estimate would be out of sync with the actual current value. A too high frequency would increase the computational costs and also doesn't leave enough time to ping all the peers and to process the data in that short period of time.

Ping packet size: The size of the packet size determines how long it takes to send it. The packet size should be such that it doesn't cause a disproportionate jump in the transfer time for the intermediate routers.

Hosts to ping: We must decide on the set of hosts to ping. This can either be a set of fixed hosts or we can ping a subset of the peers to whom we're currently uploading. The size of the set is also an important decision, it must balance the time taken to ping them all versus the increased accuracy gained from pinging multiple peers. The advantages and disadvantages are discussed in section 3.4.2.

Collating the data: We have to decide on the methods used to collate the RTT values from each host we ping. One possible approach would be to consider the trend in the majority of them. That is calculating whether the RTT values are increasing or decreasing for the majority of them, and take this as the overall trend.

Threshold for taking an action: We have to decide when to take any action, when to increase / decrease our estimated bandwidth depending on the RTT data collected.

Rate of increment / decrement: Decide on the amount by which the estimated bandwidth will be incremented / decremented. If the value is too low then it will take quite a long time to converge to the actual bandwidth value. If the value is too high, it will cause oscillations around the desired bandwidth value, wherein the desired value is constantly under- or over-estimated.

Decrease in rate: Decide on the policy by which the global decrease in bandwidth is distributed among the various currently running uploads of a peer.

5. FUTURE WORK

Future work has mainly to investigate how to control the upload rates to get stable RTT measurements. And this has to be done without the need of the user to set desired RTTs, which is also not feasible for our scenario of pinging multiple peers.

In order to do that, we need to deploy the application on several peers connected over the Internet. Only then can we run realistic tests. For these tests the maximum upload and download rate for each peers should be known (the up- and down-load rate from the ISP). Additional software on the peer's machines can measure the total up- and down-load rate at any given time, an observation which is normally not available for a P2P application. Experiments with different policies (as described in the previous section) can then be done. All data like the RTTs and inter-packet delays should be collected for later analysis. The analysis could show which policies prove to be successful in which scenarios.

It would also be interesting to apply machine learning techniques to this problem. The trace data of the aforementioned experiments could be used as input for machine learning techniques to learn good polices, given certain situations.

Acknowledgment

We would like to thank Dr. Son Vuong and Stanley Chiu for the helpful discussions and ideas.

References

M. Barr. Closed-Loop Control. *Embedded Systems Programming*, pages 55–56, August 2003. `http://www.netrino.com/Publications/Glossary/PID.php`.

O. Chalouhi. Azureus Auto Speed Plugin, 2006. `http://azureus.sourceforge.net/plugin_details.php?plugin=autospeed`.

J. Curtis and T. McGregor. Review of bandwidth estimation techniques. *New Zealand Computer Science Research Students Conference*, 8(3), 2001.

K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 134, 2001.

X. Liu and S.T. Vuong. A Cost-Effective Peer-to-Peer Architecture for Large-Scale On-Demand Media Streaming. *JOURNAL OF MULTIMEDIA*, 1(2), 2006.

R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6):27–35, 2003.