Flow Control for Many-to-Many Multicast: A Cost-Benefit Approach

Yair Amir, Baruch Awerbuch, Claudiu Danilov, Jonathan Stanton
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218 USA
{yairamir, baruch, claudiu, jonathan}@cs.jhu.edu

Technical Report
CNDS-2001-1
http://www.cnds.jhu.edu

February 28, 2001

Abstract

Flow control, especially in multicast networks, is a problem of significant theoretical and practical interest. We present a protocol that is analytically grounded, yet also achieves real world goals, such as simplicity, fairness and minimal resource usage. We base our flow control protocol on the Cost-Benefit algorithmic framework for resource management. We base decisions on the "opportunity" costs of network resources, comparing the cost of each individual resource to the benefit it provides. As opposed to existing window-based flow control schemes, we avoid end-to-end feedback by basing decisions on the state of the links between participating nodes. This produces control traffic proportional only to the number of overlay network links and independent of the number of groups.

We apply this algorithm to an existing wide area group communication system, however, it can be applied to other multicast services as well. We show the effectiveness of the resulting protocol through simulations and live Internet experiments.

1 Introduction

In this paper we present the "Cost-Benefit" framework, a new global flow control strategy for many-to-many wide area multicast based on "economic" principles and competitive analysis. Our framework assigns costs to network resources and benefits to achieving user goals such as multicasting a message to a group or receiving a message from a group.

Intuitively, the cost for network resources, such as buffers in routers, should go up as the resource is depleted. When the resource is not utilized at all, its cost should be zero. When the resource is fully utilized, its cost should be prohibitively expensive. Finding the best cost function is an open question, however, it has been shown theoretically that using a cost function that increases exponentially with the resource's utilization is *competitive* with the optimal offline algorithm[AAP93]. This theoretical result was demonstrated to work in practice in the context of cluster computing[AAB$^+$00]. In this paper, we use a similar cost function to price the buffers in the routers.

Our algorithm decides to allow use of resources if the benefit attached to that use is greater than the total cost of allowing the use. The choice of benefit function allows us to optimize for various goals. By adjusting the benefit function, performance issues such as throughput, as well as policy issues such as fairness, can be taken into account when making flow control decisions. In this way our framework allows adjustable trade-offs between conflicting properties such as fairness and throughput. For example, the benefit can be the number of packets sent (sending throughput), the number of packets received by all receivers (receiver throughput), or the average latency given some throughput constraints. In this paper we only use the sender throughput benefit function.

We present a practical realization of our Cost-Benefit framework for global many-to-many flow control in the context of "overlay networks." We define an overlay network as a virtual network constructed such that each link connects two edge nodes in an underlying physical network, such as the Internet. Each virtual link in the overlay network can translate into several hops on the underlying network, and the characteristics, such as latency, bandwidth, and losses of a virtual link are the aggregate of the underlying network links over which it travels.

We use a standard congestion control protocol on each of the overlay links. This results in a dynamic capacity available to our flow control framework on every overlay network link. The global flow control problem is a dynamic optimization problem, optimizing some benefit subject to the dynamic capacity constraints of the overlay links.

Our framework requires the sender to be able to assign a cost to a packet based on the aggregrate cost of the links on which it travels. We developed the framework in the context of a wide area group communication system. In these systems, the goal is to provide high performance multicast service with strong semantics to medium sized groups of receivers and senders who use a large number of distinct groups. For such systems with a membership service, assigning the aggregrate cost is relatively cheap because dissemination tree information can be available at the sender.

The congestion control guarantees that all of the traffic generated by our system on a link will be seen as one TCP flow on that link, regardless of the number of receivers. This provides a very conservative level of fairness between our multicast traffic and competing TCP flows.

Our Cost-Benefit framework is evaluated through both simulations and live tests on the Internet. The simulations use the ns2 simulator[ns2] and examine the behavior of several overlay network configurations. To conduct actual network tests we extended the available Spread group communication system[spr] to implement our flow control protocols, and conducted experiments using this software on the CAIRN network[CAI01]. We studied varying the number of clients sending and receiving packets, changing link characteristics, external competition from other traffic on the links and internal competition from clients sending to the same and different groups.

Both the simulations and the actual CAIRN experiments show promising results. We show a fair sharing of individual congested links between both individual clients in a flow and between different flows. We show a fairly fast and stable response to changing capacities on the network and to competing traffic. We demonstrate that senders can each achieve their own near optimal sending rate without being constrained by the ability (or lack thereof) of other senders.

The rest of this paper is organized as follows. After presenting the related work in Section 2 we describe the network model we use and the overall architecture of our protocol in Section 3. We present our framework, the theoretical model supporting it, and its adaptation to practical settings in Section 4. Section 5 briefly discusses the issues of network and inter-flow fairness. Simulation results and live Internet-based experiments are presented in Section 6 and Section 7. Section 8 concludes the paper.

2 Related work

Many different approaches exist in the flow control literature, including TCP like window based protocols [FJ93, Jac88], one or two bit feedback schemes [RJ88, Flo94, RF99], and optimization based flow control [GG80, GK99, GB98, KMT98, LL98, LL99]. The economic framework for flow and congestion control used in many optimization based protocols[GK99, KMT98] has some similarity with the cost-benefit model used in our work. In both, the links have some cost and packets that are sent must have sufficient benefit to pay the cost of the network resources they require. A significant difference is that our cost-benefit model takes an algorithmic approach using a simple formula to decide when a packet can be sent, and is not based on economic theory. Unlike many economic models our cost-benefit model does not try to reach an equilibrium state or influence non-cooperative processes to behave, but rather optimizes the throughput under the assumption of minimally cooperative senders.

Research on protocols to support group communication across wide area networks such as the Internet has begun to expand. Recently, new group communication protocols designed for such wide area networks have been proposed [KSMD00, KK00, AMMSB98, ADS00] which continue to provide the traditional strong semantic properties such as reliablity, ordering, and membership. These systems predominantly extend a flow control model previously used in local area networks, such as the Totem Ring protocol[AMMSB98], or adapt a window-based algorithm to a multi-sender group[HvR95, ADS00]. Our work presents a flow control algorithm designed explictly for wide-area networks which is motivated more by networking protocols and resource optimization research, than by existing group communication systems.

The research in flow control specificially for group communication systems is sparse. A 1998 work by Mishra and Wu[MW98] surveyed the flow control used in existing systems and compared archetypal optimistic and conservative flow control algorithms. The flow control discussed in that work was used to globally limit the number of outstanding update messages that have not become stable yet. One other paper[HvR95] discusses some general approaches to flow control for group communication and studies extending window-based protocols to groups using both local and global information.

Work on flow control for multicast sessions has occured mainly in the context of the IP-Multicast model. Much of this work has focused on the congestion control problem, avoiding extra packet loss and providing fairness, and has left flow control up to higher level protocols (such as reliability, ordering, or application level services). Research has explored the difficult problems associated with multicast

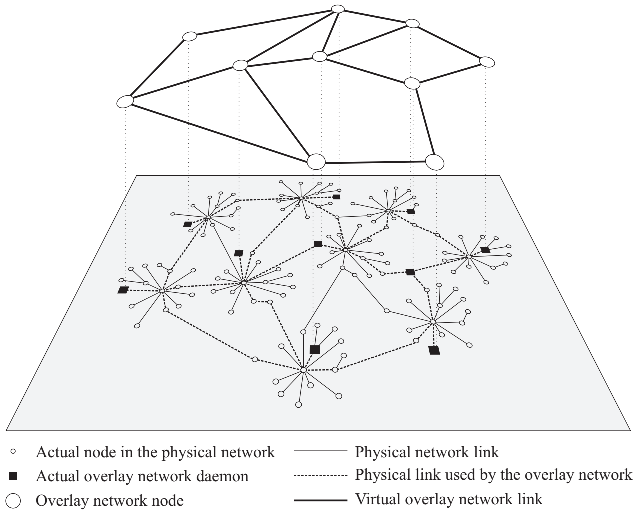

Figure 1: Overlay Network Architecture

traffic such as defining fairness[RKT99, BM00] and determining appropriate metrics for evaluation of multicast traffic[CA00]. A number of congestion control protocols have been developed with the goal of providing some level of fairness with TCP traffic, while taking advantage of the unique characteristics of multicast traffic. These include window based protocols[WS98], rate based protocols[Mon97, SYST98], multi-layer based protocols[RKT99], and protocols that use local recovery to optimize congestion control[CCG00].

The Distributed Core Multicast (DCM)[BB99] routing protocol uses a network architecture similar to ours with tunnels through the backbone network, and focuses on similar scalability goals of supporting many small groups. DCM is complimentary to our work as it focuses on efficient routing for such groups, while we provide flow control algorithms.

3 Architecture

This paper explores an effective way to provide flow control in multi-sender, multi-group group communication or multicast environments. The overlay network model used is a graph with nodes and overlay links. Each node on the graph represents a host running a daemon program. Each overlay link is a unicast link between two nodes which may be a long path traversing multiple routers and physical links in the Internet as is seen in Figure 1. Each node chooses a tree from this graph based on the network topology, in which it will multicast messages. This tree is rooted at the node and will not necessarily be the same as any other node's tree. The daemon provides multicast services to clients. As the daemon also acts as router, we will refer to daemon and router interchangeably. Each daemon can have many clients connected to it. Each client may join an arbitrary number of groups, and may send multicast messages to any number of groups, including ones it has not joined.

Clients connect to whichever node they like, usually the closest one, and that node handles the forwarding of their traffic and providing all required semantics. The connection from a client to a node is either a TCP/IP connection or a local IPC mechanism such as Unix Domain Sockets. Each client can multicast and receive messages at any time.

In this approach each daemon may support many distinct clients who are actually running on many different hosts. The performance of the entire system is improved if each client connects to the daemon closest, network-wise, to them, however clients may connect from anywhere. In practice, the clients connected to a specific daemon may actually use a local group, or local area reliable multicast protocol instead of unicast TCP connections to connect the clients to the daemons. This modification would improve the performance for the group of clients, but does not change the basic model as long as the protocol used allows individual control of each sender's sending rate. The Spread system actually uses such a protocol.

Each message caries some information about its source and destination nodes. When an intermediate node receives a message, it forwards it through its links that have downstream destinations.

In a multiple sender system, each sender may have a different rate at which it can reach the entire receiver group, and it may reach the group over a different multicast tree. So, the bottleneck link for one sender may not be the bottleneck for other senders. The obvious goal is to allow each sender to achieve the highest rate they can to the group, not limit them by what other senders can send to the group. To achieve this, rate regulation must occur on a per-sender basis and not have a single flow control limit for the entire group. The result is a flow control that provides fine granularity of control (per-sender, per-group).

3.1 The Spread group communication toolkit

We implemented our global flow control algorithm in the Spread wide area group communication system[spr, AS98]. The Spread system provides a similar architecture to our model with daemons running on end-hosts acting as routers in an overlay network. Spread provides strong semantics for messages including reliable multicast, message ordering guarantees (unordered, fifo, agreed), and a membership service supporting Extended Virtual Synchrony (EVS)[MAMSA94] and Virtual Synchrony (VS)[VKCD99] models. It is designed to support a small to medium number of members of a group (1-1000's), with a large number of groups active and many senders. As such, it has different design goals then most IP-Multicast systems, which support larger groups but focus on the single-sender model and require state in every network router for every group.

Routing in Spread is based on shortest-path multicast trees rooted at each daemon. The routing is recalculated whenever the set of connected daemons changes (and not when clients join or leave groups, or connect or disconnect from the system).

The Spread system provides end-to-end reliablity by using a reliable point-to-point protocol for each link on the overlay network and requiring that the daemons do not drop any messages between links[ADS00]. Spread provides a choice of several link protocols from which we chose to use TCP.

4 Flow control for many-to-many multicast

The algorithmic foundation for our work can be summarized as follows: allocation of bandwidth in a circuit-switched environment can be performed "near-optimally" using the following approach. We price link bandwidth based on the "opportunity cost", which increases exponentially with link utilization. We compare different connections based on the total opportunity cost of all links, and reject connections with large costs.

Section 4.1 introduces the theoretical foundation of our algorithm. The reader may chose to skip it on first reading and proceed to Section 4.2, which describes how we adjust this purely theoretical framework to the reality of modern communication systems. This adjustment is needed because the assumptions in the model are not met, i.e. link bandwidth is uncontrollable, the current cost of remote links is not known, etc.

This adjustment leads us from algorithms that are "theoretically-optimal" in an unrealistic, but theoretically nice world, to protocols that exhibit good performance in simulations and, ultimately, in real-life experiments.

4.1 Algorithmic foundation

In this section, we proceed as follows. First, we introduce the concept of "competitive analysis". Then, we introduce the concept of "resource opportunity cost" and the concept of "user benefit". We then show how to optimally perform admission control based on bandwidth availability in a circuit-switched environment.

Evaluation of algorithms – Competitive analysis: The general problem with online allocation of resources is that it is impossible to optimally make irreversible decisions without knowing the future or knowing the correlations between past and future. Thus, the goal is to design a "competitive" algorithm whose total accrued benefit is comparable to that achieved by the optimal offline algorithm, on **all** input instances. The maximum possible performance degradation of an online algorithm (as compared with the offline) is called the "competitive ratio". Specifically,

$$\rho = \max_{x} \frac{C(x)}{C^{*}(x)}$$

where x is the input sequence, $C(x)$ is the cost of the online algorithm, and $C^{*}(x)$ is the cost of optimal offline algorithm on sequence x.

Our goal is to design an algorithm with a small competitive ratio ρ; such an algorithm is very robust in the sense that its performance is not based on unjustified assumptions about probability distributions or specific correlation between past and future.

Model of the resource – Cost function: The basic framework revolves around defining, for each resource, the current *opportunity cost*, which is, intuitively, the benefit which may be lost by *high-benefit* connections as a result of consuming the above resource by a *lower-benefit* connection.

Since the goal is to maximize the total benefit, it is "wasteful" to commit resources to applications (connections) that are not "desperate" for that resource, i.e., not enjoying the maximal possible benefit from obtaining this resource. On the other hand, it is equally dangerous to gamble that each resource can be used with maximal benefit gained without knowing the sequence of requests ahead of time.

The opportunity cost attempts to "predict" potential future gain of using the resource from a sample of past behavior. Quite counter-intuitively, this attempt to predict the future from the past succeeds, even *without* assuming a known correlation between past and future; e.g. the future may be negatively or adversely related to past, and generally may be controlled by an adversary.

For the purpose of developing the intuition, it is useful to consider a somewhat restrictive setting where the resources are either assigned forever, or rented out for specific time. For a given resource l, (e.g., bandwidth of a given link l), denote by u_l the normalized utilization of the resource, i.e., $u_l = 1$ means the resource is fully utilized and $u_l = 0$ means that the resource is not utilized at all. Also, let α be the minimum benefit value of a unit of a resource used by a connection and β be the maximum value. Let $\gamma = \beta/\alpha$.
The opportunity cost is now defined as follows, let

$$C(u_l) = \gamma^{u_l} \tag{1}$$

be the current opportunity cost $C(u_l)$ of the "unit" of resource. I.e., if r percent of the resource is used, it costs $r/100 \cdot C(u_l)$. Each $1/\log_2 \gamma$ of the fraction of the utilized resource necessitates doubling the price.

Notice that the opportunity cost of an unused resource starts at α per unit, i.e. *any* connection can "afford" it, and the cost of fully utilized resource is β, i.e., no connection can afford it.,

Model of the user – Benefit function: This is part of the users specifications, and is not part of our algorithms. Each user (connection) associates a certain "benefit function" $f(r)$ with its rate r. The simplest function $f(r) = r$ means that we are maximizing network throughput; a linear function means we are maximizing weighted throughput.

More interestingly, a concave function (second derivative negative, e.g. $\sqrt{r}$) means that there is curve of diminishing return associated with rate allocation for this user. For example, imagine that a traffic stream is encoded in a layered manner, and it consists of a number of streams, e.g., first 2KB is voice, next 10KB is black and white video, and last 50KB is color for the video stream. In this case, a concave benefit function may allocate $10 for voice part, $5 for video and $2 for color.

Notice that concave functions enable one to implement some level of fairness: given 50KB of bandwith, it is most "beneficial" to provide voice component for 25 connections, rather than voice + black and white video + color for a single connection since $10 x 25 = $250 is the total benefit in the former case, and $10 + $5 + $2 = $17 is the total benefit in the latter case.

Admission control in a circuit-switched environment: Consider, as a toy example of the cost-benefit framework, the problem of admission control in the case of a permanent virtual circuits environment where an online request for a point-to-point virtual circuit is either *accepted* (with the permanent bandwidth allocation) or *rejected*. The goal is to maximize the bandwidth of all accepted connections.

In this case, the opportunity cost of the path is the sum of opportunity costs of all the links which make up the path.

$$C_p = \sum_{l \in L_p} C(u_l)$$

Also, the minimal benefit of a connection *per link* needs to be adjusted to

$$\alpha' = \alpha / d$$

where d is the number of hops on a path. Thus, the base of exponent becomes

$$\gamma = \frac{\beta}{d\alpha'}$$

The cost-benefit framework suggests the following admission control algorithm [AAP93]: accept a connection if its benefit is larger than the total opportunity cost of the path, and reject a connection otherwise. In [AAP93], the following theorem is proved.

Theorem 1 *The above algorithm is $O(\log \gamma)$ competitive with the optimal prescient strategy.*

4.2 Adapting the model to practice

The above theory section clearly shows that bandwidth can be rationed with a Cost-Benefit framework leading to a near-optimal (competitive) throughput in the case of managing *permanent* connections in circuit-switched environments.

The core theory has several assumptions which do not exactly match the reality in distributed systems and communication networks. We will examine these assumptions and how we adapted the ideas of the Cost-Benefit framework to work in real networks.

- The framework applies to permanent connections in circuit-switched environment, rather than to handling individual packets in packet-switched networks.
- The theoretical model assumes that the senders have instantaneous knowledge of the current costs of all the links at the instant they need to make a decision. It is also assumed that fine-grained clocks are available and essentially zero-delay responses to events are possible.
- The natural application of the framework, as in the case of managing permanent virtual circuits, is to use bandwidth as the basic resource being rationed. However, bandwidth is *not* under our control because competing flows may ocupy at any point an arbitrary and time-varying fraction of the link bandwidth. Therefore, while bandwidth is an essential component for performance, our protocols cannot meaningfully ration (save or waste) it, as its availability is primarily at the mercy of other applications. (Recall that our application has to share the link bandwidth "fairly" with other TCP flows.)

In fact, the underlying model does not specify which resources are to be managed by it. The most important issue is understanding what is the resource that is being controlled (rationed) since not all of the resources used are controllable. Figuratively speaking, bandwidth to flow control is like wind to sailing: it is controlled by adversarial forces rather than by us. We must try to adapt as much as possible to changing circumstances, rationing the controlable resources.

4.2.1 Practical cost function

We chose buffer space in each overlay network router as the scarce resource we want to control. Conceptually, we model our software overlay network router as a router with fixed size output link buffers where packets are placed into the appropriate output link queues as soon as the packet is received by the router. If a link is not congested its corresponding queue will be empty. In this case, once a message arrives in the buffer it is immediately sent, with only a short delay until it is scheduled to be sent. If the incoming traffic is more than the capacity of a link, some packets will accumulate in the corresponding link buffer.

Each router establishes the cost for each of its outgoing links and advertises this cost to all the other nodes using the overlay network. The price for a link is zero if its corresponding buffer is empty. This means that the cost is zero as long as the link is not congested, i.e. the link can accommodate all the incoming traffic. As the link gets congested and messages accumulate in the buffer, the cost of the link increases. The price can theoretically go as high as infinite when the buffer is full. In practice, the cost of the link will increase until nobody is able to buy it.

Equation 1 from Section 4.1 gives the basic form of our cost function. The utilization of a link is given by x/M where x is the average number of messages in the buffer, M is the desired capacity of the buffer. So the cost is:

$$C(x) = \gamma^{x/M}$$

This function ranges from 1 to γ. We want to scale it to range from 0 to S. So the cost becomes:

$$C(x) = S \cdot \frac{\gamma^{x/M} - 1}{\gamma - 1}$$

The theory claims that the maximum benefit given to a client should be sufficient to fully utilize the network. If this maximum benefit is Max_B, and the minimum benefit is 1, then the base of the exponent in the cost function should be $\frac{Max_B}{1}$.

If we use Max_B (say 20) as the base of the exponent, then the cost function stays near zero until the average buffers reach around 20. Then, the cost goes up very quickly. This allows actual congestion to occur (average used buffers > 0) without increasing the cost almost at all. In the real world, the cost of a link cannot be known instantly. Therefore a practical mechanism will provide incremental feedback on costs before the utilization becomes high. This calls for a small base of the exponent.

Using a small base for the exponent has real-world advantages and still fits the model. First, it provides an increase in costs almost immediately as the link first becomes congested. Second, it is still exponential, which is required by the theoretical model. We chose e as the base of the exponent.

So finally we get:

$$C_l(x) = S \cdot \frac{e^{x/M} - 1}{e - 1}$$

It is interesting to see whether other functions which also react quickly could work. We tried functions such as $f(x) = x$ and $f(x) = x^2$, however, they were unstable and unable to provide consistent throughput.

From the cost of the links we can compute the cost for a packet. That cost is the sum of the cost on all the links that the packet will traverse plus a constant ρ that will be discussed later.

$$MT = \{l | l \in \text{Multicast tree of p}\}$$

$$C_p = \rho + \sum_{l \in MT} C_l \tag{2}$$

4.2.2 Benefit assignment

The choice of benefit is tightly intertwined with the choice of the goal we want to achieve. If our goal is to maximize the sending throughput of the network, then the benefit function should reward each packet sent onto the network with a fixed benefit. If our goal is to maximize sending throughput while still allowing every sender to send at least some traffic (i.e. no one starves), then the benefit function could be a concave function (like $1/x$) of the number of packets already sent by this connection. The benefit function could also be based on the number of received packets, instead of sent packets. That would advantage senders who send to larger groups. We chose to maximize sending throughput so our benefit is one unit per packet sent.

Although one would like to handle the benefit as a pure rate, in practice giving several units of benefit to a client at a time is more efficient. We define a "salary" as the amount of dollars a client is given from time to time. A client is allowed to save up to S dollars of its salary. Every time a client gets a salary, they lose whatever they did not spend previously and which is more than S dollars. This acts like a leaky bucket of dollars.

The sending rate of the clients is actually regulated by the daemon a client is connected to. The daemon acts as the client's "agent", purchasing packets whenever possible for the client.

If the client wants to send a message that costs more dollars than it currently has, the daemon blocks the client by not reading from its socket anymore, and keeps the expensive message until the client can afford to send it. The client can afford to send it, and will be unblocked, when it receives more money, or the price of the sending tree goes down.

4.2.3 Other considerations

If the minimum cost of a link is zero and it stays zero until the clients receive a cost update, any client would be allowed to send an infinite number of messages between two cost updates. One solution is to have a minimum cost per link greater then zero, however, this will not scale with the size of the network (long paths could have a very large cost even if idle). Our solution is to keep the minimum cost at zero, but add a constant processing fee per message of \$1. This fee is the constant ρ referred to in Equation 2. This fee puts a cap on the number of messages each client can send, even when the network cost is zero, because of the limited salary.

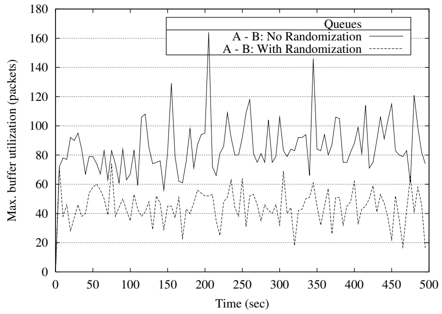

Figure 2: Randomization effect on buffer utilization

Since we do not know the network capacity (assumed known by the theory), we approximate such knowledge by adaptively probing for the current network bandwidth. We chose to do this in a way very similar to TCP. We decrease the time between salaries in a linear manner as the network appears uncongested. When the network appears congested, we multiplicatively increase the time between salaries to slow down the clients. At the time a client receives a salary the daemon checks whether the client was blocked during the last salary period (due to an intention to buy a message more expensive that it can afford). If the client was not blocked, it means that he was able to buy all the messages he initiated, so the next salary period will be exactly the same as the previous one. However, if the client was blocked, the daemon compares the cost of the most expensive message that the client bought during the last salary period with a threshold. If this cost is less than the threshold, the time between two salaries is decreased, because the client might have been blocked due to processing fees on a relatively idle network. The new salary period is:

$$T_{new} = \frac{T_{old} \cdot T_{update}}{T_{old} + T_{update}} \tag{3}$$

where T_{old} is the previous salary period and T_{update} is the time between two cost updates. If this cost is larger then the threshold, this client has assumed the network has more capacity then it actually does, so the client slows down. The new salary period will be:

$$T_{new} = 2 \cdot T_{old} \tag{4}$$

This algorithm resembles the TCP congestion control where T_{new} and T_{old} would be the average time between two packets sent, and T_{update} would be the round trip time. Equation 3 is algebraically equivalent to adding 1 to the congestion window in TCP.

Finally, the coarse granularity of cost updates causes a high degree of synchronization between clients at the time an update is received. This synchronization phenomenon could cause oscillations in the router buffers as every client buys at the same time, then the cost goes up, then everybody waits for the price to go down, etc. To prevent the synchronization, each client may *not* send a message even though it has sufficient funds to do so. It will choose to not send with a probability $1 - 1/C_p$, where C_p is the total cost of the message. This bad behavior and the benefit of randomization are shown in Figure 2.

5 Fairness

What definition of fairness is best in a multicast environment is an area of active research. For this work we chose a conservative approach of considering each link on our overlay network as one TCP flow. So, on each link we will fairly share that link with all competing traffic. Some might argue that this is too conservative, as many people may be using our multicast service at once, and each one would receive their own TCP flow if they were using a separate unicast service, but here they will share only one TCP flow. This is true.

The difference between looking at the receiving throughput and looking at the sending throughput when comparing a multicast protocol with TCP is big. However, we try to be very conservative in our approach by taking into account the worst case scenario and analyzing only the sending throughput.

Giving a "fair" amount of traffic to all the senders, regardless of their intended use of network resources, is at odds with maximizing throughput of the network as a whole. We choose, by default, to provide a fair share of our overlay network resources to all senders who have the same cost per messsage. That could be because their messages travel over the same multicast tree, or just by random coincidence. However, senders who have higher costs, say because they cross more congested links, will be allowed to send at a lower rate. This is depicted in Section 6 Scenario 4, where sender A-F who uses all the network links receives much less then its "fair" share of the resources.

6 Simulation results

We used the ns network simulator[ns2] to evaluate the performance and behavior of our flow control protocol. The main issues we focused on are:

- optimal network resource utilization;
- automatic adjustment for dynamic link capacities;
- optimal sharing of network resources to achieve maximum throughput;
- fairness with TCP;
- fairness between flows using the same congested links;

Scenario 1 – one bottleneck link per group: We used the multicast tree shown in Figure 3, with the link capacities and latencies as shown in the figure. All the intermediate buffers in the network have a soft limit of 100 packets. Clients receive a \$10 salary, and they can save up to \$20 in their budget. The processing fee is \$1/packet. Cost updates are sent from each intermediate node every 50ms.

Two classes of clients initiate multicast messages, G_{ace} and G_{aef}. The G_{ace} clients multicast to nodes C and E, and and the G_{aef} clients multicast to nodes E and F, sharing the links A-B, B-D and D-E. In this scenario the network has enough resources to provide 2Mbps to G_{ace} clients and 1Mbps to G_{aef} clients.

Rather than looking at the buffer occupancy which is very dynamic and depends on the sampling frequency, we chose to analyze the evolution of the maximum buffer utilization in Figure 4.

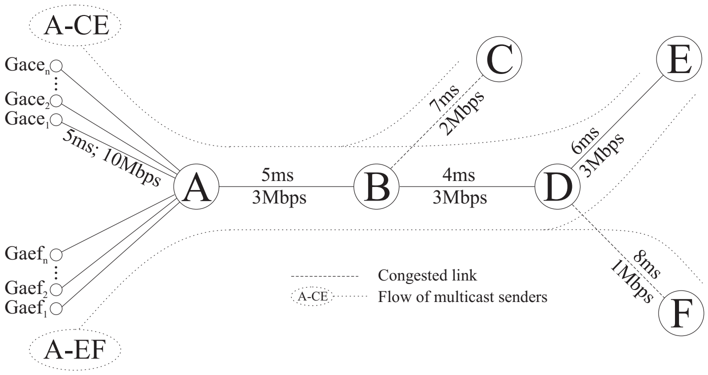

Figure 3: Scenario 1: Network Configuration

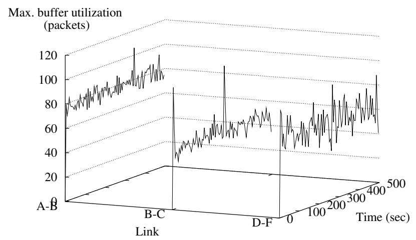

Figure 4: S1: Buffers–20 senders/flow

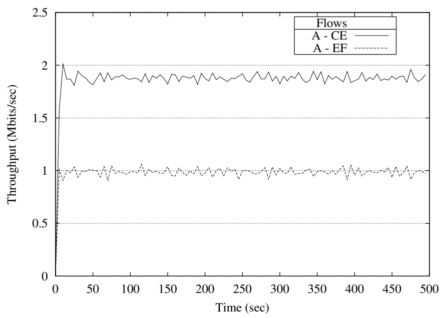

Figure 5: S1: Throughput–20 senders/flow

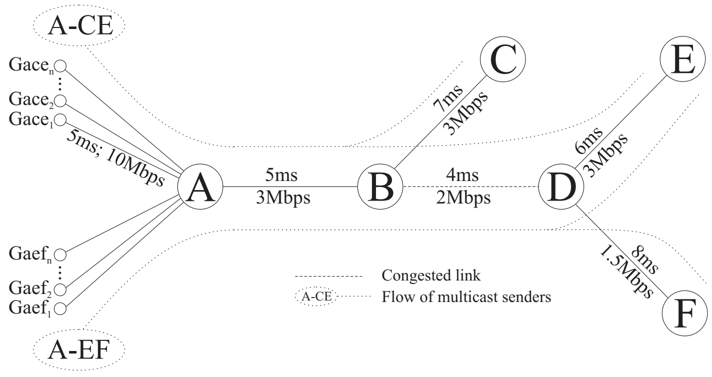

Figure 6: Scenario 2: Network Configuration

Since the network can accommodate all the traffic that link A-B allows, we see that buffers stay usually below 90 packets, going up and down mainly because of changes in the available bandwidth on the link. As we described before, the bandwidth considered available to us is that provided by a TCP connection between the two nodes. So as TCP varies its window size to adapt to congestion or changes in available network bandwidth, our available bandwidth also changes. Notice that link A-B has a higher maximum buffer usage. This is because link A-B has the highest input-output throughput ratio (for a scenario with 20 G_{ace} and 20 G_{aef} clients, the ratio is more than 100:1), and thus small bursts on the input links cause large bursts on the output link.

As can be seen in Figure 5, the throughput is optimally divided between senders for maximal network usage, G_{ace} getting on average 1.840 Mbps and G_{aef} getting 0.9679 Mbps.

Scenario 2 – shared bottleneck link: To examine the effect of a congested link, the network shown in Figure 6 is used. Here, link B-D forms the bottleneck for both flows, while link DF would be the bottleneck for flow G_{aef} if it was alone. Also, even though links BC and DF have different capacities, they should both show a minimal cost since they will not be congested when they each receive a fair share of link B-D.

The maximum buffer utilization for the congested link (B-D), shown as the top line in Figure 7, stays around (and below) 100 packets, and is the largest component in the cost of the messages. We can see that the buffer of the first link is still used because of the large disparity between client connection bandwidth (10 Mbps per client) and output bandwidth (3Mbps total).

The two flows share the bottleneck link equally. Flow G_{ace} gets an average of 0.9685 Mbps and flow G_{aef} gets an average of 0.9698 Mbps. The various clients who make up each flow also share the bandwidth fairly. In Figure 9, we show the percentage difference of each of the 20 clients throughput from the average throughput achieved by the group as a whole. This shows that the variance of the clients throughput is less then 3.5%.

We examined the effect of increasing the number of clients in Figure 10. This figure shows how the maximum buffer utilization grows at a decreasing rate as the number of clients increases. The link buffer represented in this graph is the B-D link which is the bottleneck link. Also, we looked at the aggregrate

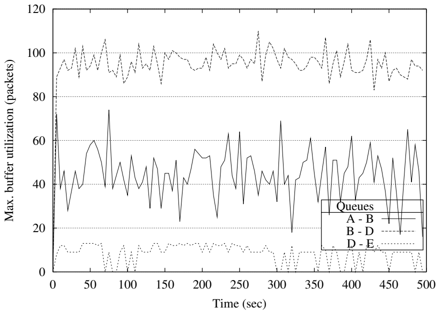

Figure 7: S2: Buffers–20 senders/flow

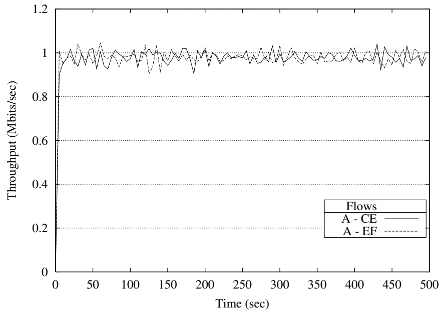

Figure 8: S2: Throughput–20 senders/flow

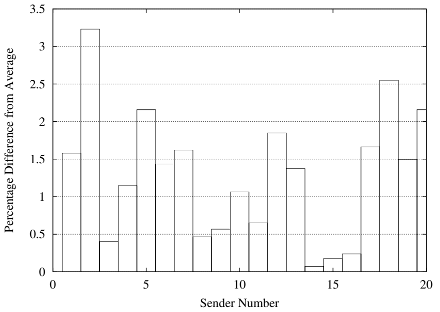

Figure 9: S2: Fairness

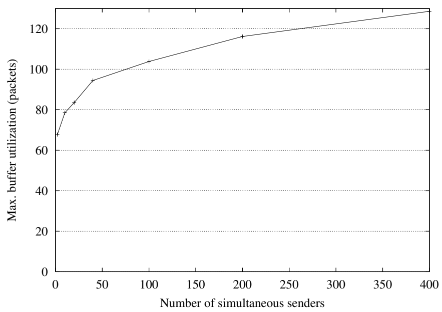

Figure 10: S2: Clients vs. Buffers

rate of all senders. This rate stays between 1.928 and 1.944 Mbps while the number of senders varies from 2 to 400.

A second experiment used the same tree configuration as Figure 6 but started the second group of senders G_{aef} after 200 seconds. Figure 11 shows the maximum buffer utilization on all the links. At the 200 second mark, the maximum buffers used on the B-C link drops to zero as the link goes from 2/3 used to 1/3 used. The bottleneck link from B-D increases only slightly in maximum utilization as the number of clients doubles. Specifically, there is no large spike in maximum utilization when the second group of clients begins sending all at once. This is because the link has an existing non-zero cost and so the new clients must pay that cost before sending. Figure 12 shows how the throughput of the two groups of

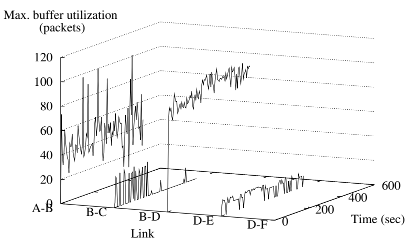

Figure 11: S2: Delayed senders, buffers–20 senders/flow

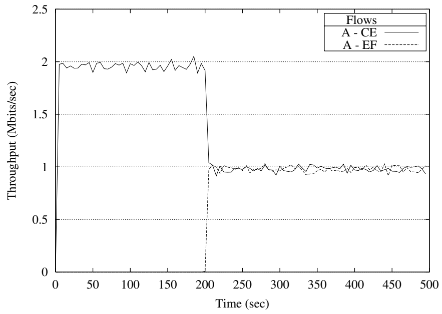

Figure 12: S2: Delayed senders, throughput–20 senders/flow

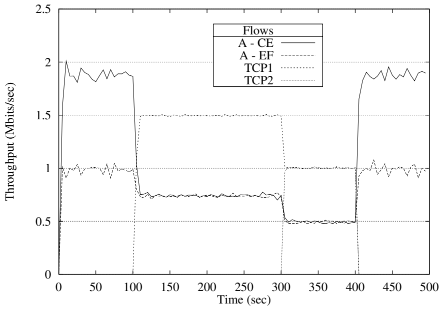

Figure 13: S3: Throughput–20 senders/flow

clients responds very quickly to the new load and they both get essentially the same bandwidth.

Scenario 3 – competing with TCP: We next examine how our flow control algorithm responds to substantial changes in the available bandwidth on some of the links. We use the same tree network as Scenario 1, shown in Figure 3, with one modification. To simulate changes in resources as well as demonstrate our fairness with TCP, we added two TCP flows over link D-E which start at different times during the simulation. The first flow starts at 100 seconds, the second flow starts at 300 seconds, and they both end at 400 seconds.

Figure 13 shows the throughput for 20 clients per flow (40 clients total). During both the initial 100 seconds and the last 100 seconds, the behavior is very similar to the initial experiment on this network shown in Figure 5. After the first TCP flow starts, the client flow G_{ace} drops from 1.9 Mbps to 742 Kbps and the client flow G_{aef} drops from 0.98 Mbps to 739 Kbps, and the TCP flow receives 1.49 Mbps (one half of the DE link bandwidth). This change in available bandwidth on the DE link also changes the bottleneck link for all the clients from link BC and DF, respectively, to link DE. Since they now have the same bottleneck link and none of the other links have noticeable costs, both flows receive equal throughput. When the second TCP flow starts, each flow receives 495 Kbps and each of the two TCP flows receives 1.0 Mbps. Note that we have a total of 40 senders in our flows, that compete as a single connection with TCP. Similar results were obtained using 1, 5, 10 and 50 clients per flow.

Scenario 4 – chain network with our flow control: Our flow control tries to maximize throughput by allowing low cost messages to pass, and reducing high cost traffic. The very simple way to show this is to set up a chain network in which some clients try to send their messages across the entire network, while other clients use only one link in the chain. Figure 14 shows such a network with 5 links connected in series. One flow of clients sends from node A to node F, and 5 other flows send only over one link, i.e. from B-C or E-F.

Figure 15 shows the throughput on the chain network as short connections start up every 150 seconds. The A-F flow starts using the entire capacity of the network. When the flow A-B starts, they share the congested link, AB, about equally. When the third flow, B-C, starts at time 300, the long flow A-F slows down letting short flows use the available bandwidth. As we add more congested links by starting more short connections, the A-F flow goes almost to zero, thus almost maximizing the global throughput of

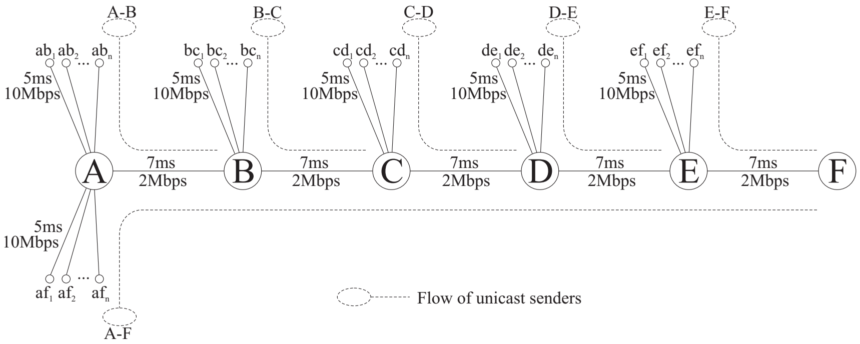

Figure 14: Scenario 4: Network Configuration

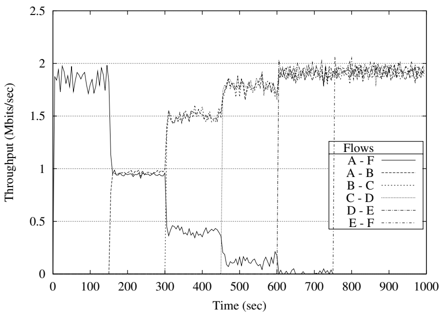

Figure 15: S4: Throughput–20 senders/flow

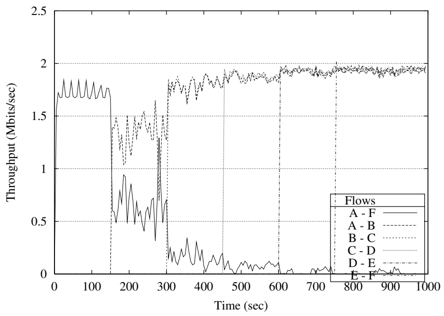

Figure 16: S5: Throughput of TCP

the system. If the flow control had been fair, the aggregrate throughput would be 6 Mbps, 1 Mbps for each flow. We achieved an aggregate throughput after all flows have started of 9.61053 Mbps, while the theoretical maximum is 10 Mbps.

Scenario 5 – chain network with end to end TCP: The results of the previous simulation show a definite bias towards short flows and show how such a bias can increase network throughput. Actually, this is the same behavior as end to end connections using TCP would show given the same situation. Figure 16 shows the throughput on the same chain network, only instead of hop by hop connections regulated by our flow control, we run end to end TCP connections. With TCP, the long A-F connection is biased against in the same way as our flow control. Moreover, when competing with only one other TCP flow A-B, the longer A-F flow receives less bandwidth. We believe this is because TCP is biased against both long RTT connections as well as having to cross multiple congested links. So even when only one link is congested, the longer RTT of the A-F flow causes it to receive lower average bandwidth then the short RTT A-B flow.

From	To	Throughput (Mbps)	RTT (ms)
isiepc3	isipc	2.0285	61.529
isipc	isiepc3	2.0156	
isiepc3	tisepc	1.4522	5.742
tisepc	isiepc3	1.4565	
isiepc3	isiepc	27.5601	0.158
isiepc	isiepc3	28.1297	
isiepc2	isiepc	28.4132	0.164
isiepc	isiepc2	28.1811	
isiepc	mitpc2	10.9173	9.461
mitpc2	isiepc	11.2697	
isiepc2	saicpc	1.4707	2.311
saicpc	isiepc2	1.4729	
isiepc2	udelpc	1.39821	7.214
udelpc	isiepc2	1.4497	

Figure 17: CAIRN Resources

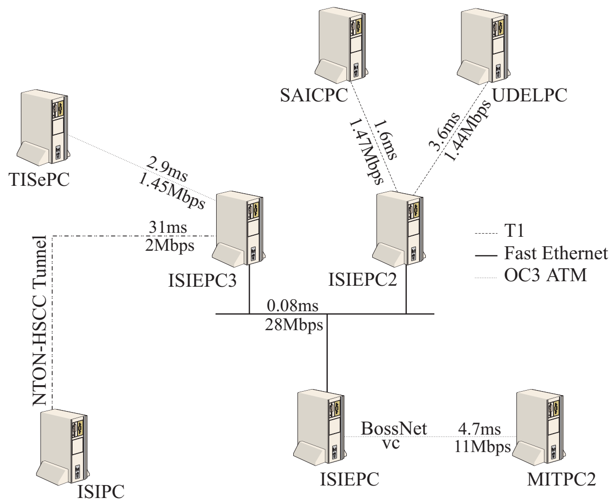

Figure 18: CAIRN Network Configuration

7 Real-life experiments

We ran our experiments over a portion of the CAIRN network[CAI01]. This is a wide-area network that crosses the entire United States, and consists of links that range from 1.5Mbps to 100 Mbps. The routers are Intel machines that run FreeBSD 3.4. Figure 18 shows the portion of the CAIRN network that we used for our experiments.

We measured individual link latencies using ping under zero traffic, and the available bandwidth with simple point to point TCP connections between each two ends of a link. Note that our flow control uses the available bandwidth given by the underlying TCP link protocol, and not the physical bandwidth of the network. Table 17 shows the performance of each individual link of the overlay network.

We extended the Spread toolkit[spr] to use our Cost-Benefit framework for global flow control. Spread has its own overhead of about 15% of data sent due to adding headers required for routing, ordering, and safety guarantees as well as to provide user-friendly group names. The control traffic overhead of our flow control was less then 0.16%. What we measured in our results is **actual user data sent and received** by clients connected to Spread. For these experiments we gave each client a \$10 salary, and allowed up to \$20 of savings. The processing fee was \$1. All the overlay network links had a soft limit of 100 packets buffered for sending. The cost updates were sent every 100 ms.

Scenario 1 – one bottleneck link per group: Having different congested links for each group limits the senders to their own bottleneck, similarly to scenario 1 of the simulation. In this scenario, mitpc2 and udelpc join a group G_1, isipc and isiepc2 join a group G_2, and we send from isiepc3 to G_1 and from mitpc2 to G_2. The sender to G_1 is limited by the 1.4Mbps isiepc2-udelpc link, while the sender to G_2 is limited by the 2Mbps isiepc3-isipc link. As expected, Figure19 shows that the sender to the group G_2 gets a higher throughput (1.502 Mbps) than the sender to the group G_1 (1.110 Mbps).

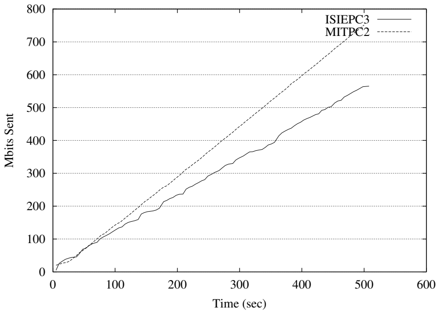

Figure 19: CAIRN: Scenario 1

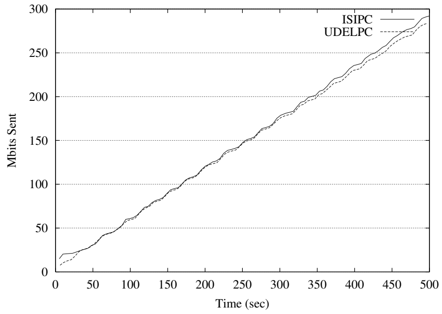

Figure 20: CAIRN: Scenario 2

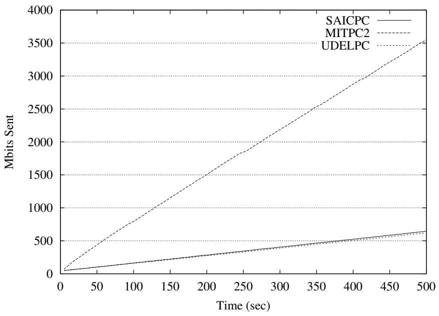

Figure 21: CAIRN: Scenario 3–bits

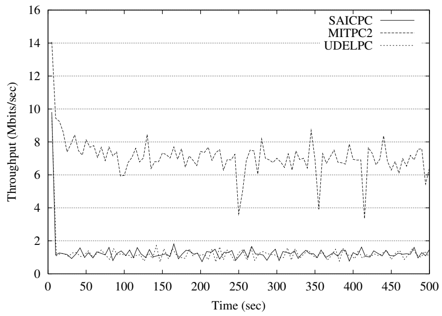

Figure 22: CAIRN: Scenario 3–throughput

Scenario 2 – one shared bottleneck: To examine the effect of a congested link we run an experiment with a shared bottleneck link similar to scenario 2 of the simulation. In this scenario tisepc and mitpc2 join group G_1, and tisepc and saicpc join group G_2. The link isiepc3-tisepc is the common bottleneck for both groups. We send from isipc to the group G_1 and from udelpc to group G_2. Figure 20 shows that both senders send at about the same rate where isipc sends at 0.584 Mbps and udelpc sends at 0.568 Mbps.

Scenario 3 – three high speed receivers: This scenario uses one group with three receivers, isiepc, isiepc2 and isiepc3. All the receivers are located in a high speed network, fairly close to each other.

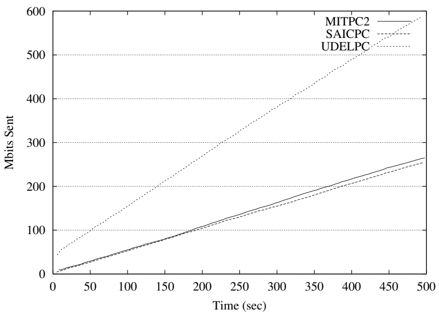

Figure 23: CAIRN: Scenario 4–bits

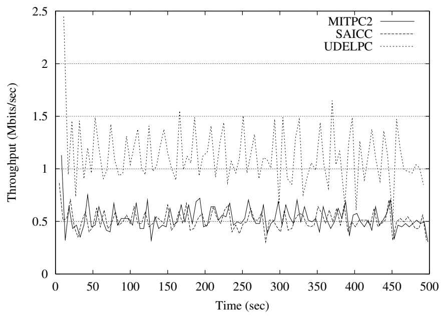

Figure 24: CAIRN: Scenario 4–throughput

The senders are mitpc2, saicpc and udelpc. Each of them has a non-overlapping route to the receiver group. Each sender should be limited only by their link to the receiver group, and not by any other links. Figure 21 and Figure 22 show the senders data bits sent, and their sending rate over time. In the figures one can see that udelpc (1.271 Mbps) and saicpc (1.289 Mbps) send slightly below the capacity of their T1 links, while mitpc sends at a significantly higher rate (7.099 Mbps).

Scenario 4 – adding a slow receiver: This scenario has one of the slower senders (udelpc) from scenario 3 join the group as a receiver. This scenario is interesting because the link from udelpc to isiepc2 can be fully utilized by udelpc's generated traffic to the group. In contrast the incoming link into udelpc has to be shared between the other two senders (mitpc2 and saicpc). Figure 23 and Figure 24 show that udelpc does achieve a higher sending rate (1.189 Mbps) than the others. The other two receivers approximately share the link getting 0.508 Mbps for saicpc and 0.529 Mbps for mitpc2.

Note that the numbers reported in all the scenarios are fairly close to the network capacity once the Spread overhead is taken into account, except for the link (MITPC2-ISIEPC). We believe this difference is due to implementation details in Spread, which currently limit the utilization of high speed links.

8 Conclusions

This paper presented a new global flow control approach for many-to-many multicast based on sound theoretical foundations. Our Cost-Benefit framework provides a simple and flexible way to optimize flow control to achieve several desirable properties such as near optimal network throughput and automatic adjustment to dynamic link capacities. The resulting algorithm provides fairness between equal cost internal flows and is fair with outside traffic, such as TCP. We implemented the framework in the ns2 simulator and showed results similar to those predicted by theory. We then implemented the framework in the Spread group communication system and conducted experiment on the CAIRN network to validate the simulations and show the real-world performance of the framework.

References

[AAB+00] Yair Amir, Baruch Awerbuch, Amnon Barak, Ryan Borgstrom, and Arie Keren. An opportunity cost approach for job assignment and reassignment. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768, July 2000.

[AAP93] Baruch Awerbuch, Yossi Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of 34th IEEE Symposium on Foundations of Computer Science*, 1993.

[ADS00] Yair Amir, Claudiu Danilov, and Jonathan Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proceeding of International Conference on Dependable Systems and Networks*, pages 327–336. IEEE Computer Society Press, Los Alamitos, CA, June 2000. FTCS 30.

[AMMSB98] D.A. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia. The totem multiple-ring ordering and topology maintenance protocol. *ACM Transactions on Computer Systems*, 16(2):93–132, May 1998.

[AS98] Yair Amir and Jonathan Stanton. The spread wide area group communication system. Technical Report 98-4, Johns Hopkins University Department of Computer Science, 1998.

[BB99] Ljubica Blazevic and Jean-Yves Le Boudec. Distributed core multicast (DCM): a multicast routing protocol for many groups with few receivers. *Computer Communications Review*, 29(5):6–21, October 1999.

[BM00] Thomas Bonald and Laurent Massoulie. Impact of fairness on Internet performance. Submitted, December 2000.

[CA00] Robert C. Chalmers and Kevin C. Almeroth. Developing a multicast metric. In *Proceedings of GLOBECOM 2000*, volume 1, pages 382–386, 2000.

[CAI01] Cairn network. Information available at http://www.cairn.net/, 2001.

[CCG00] Shuming Chang, H. Jonathan Chao, and Xiaolei Guo. Tcp-friendly window congestion control with dynamic grouping for reliable multicast. In *Proceedings of GLOBECOM 2000*, volume 1, pages 538–547, 2000.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

[Flo94] Sally Floyd. TCP and explicit congestion notification. *ACM Computer Communication Review*, 24(5), October 1994.

[GB98] S. Jamaloddin Golestani and S. Bhatacharyya. End-to-end congestion control for the Internet: A global optimization framework. In *Proceedings of International Conference on Network Protocols*, pages 137–150, October 1998.

[GG80] R. G. Gallager and S.Jamaloddin. Golestani. Flow control and routing algorithms for data networks. In *Proceedings of 5th International Conference on Computers and Communication*, pages 779–784, 1980.

[GK99] R. J. Gibbens and Frank P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, 35, December 1999.

[HvR95] Takako M. Hickey and Robbert van Renesse. Incorporating system resource information into flow control. Technical Report TR 95-1489, Department of Computer Science, Cornell University, Ithaca, NY, 1995.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, 1988.

[KK00] Idit Keidar and Roger Khazan. A client-server approach to virtually synchronous group multicast: Specifications and algorithms. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 344–355, Taipei, Taiwan, April 2000. IEEE Computer Society Press, Los Alamitos, CA.

[KMT98] Frank P. Kelly, A. K. Maulloo, and David. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[KSMD00] Idit Keidar, Jeremy Sussman, Keith Marzullo, and Danny Dolev. A client-server oriented algorithm for virtually synchronous group membership in wans. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, pages 356–365, Taipei, Taiwan, April 2000. IEEE Computer Society Press, Los Alamitos, CA.

[LL98] David Lapsley and Steven Low. An IP implemention of optimization flow control. In *Proceedings of IEEE Globecom*, pages 3023–3028, 1998.

[LL99] David E. Lapsley and Steven Low. Random early marking for Internet congestion control. In *Proceedings of IEEE Globecom*, volume 3, pages 1747–1752, 1999.

[MAMSA94] L. E. Moser, Yair Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pages 56–65. IEEE Computer Society Press, Los Alamitos, CA, June 1994.

[Mon97] Todd Montgomery. A loss tolerant rate controller for reliable multicast. Technical Report NASA-IVV-97-011, West Virginia University, August 1997.

[MW98] Shivakant Mishra and Lei Wu. An evaluation of flow control in group communication. *IEEE/ACM Transactions on Networking*, 6(5):571–587, October 1998.

[ns2] ns2 network simulator. Available at http://www.isi.edu/nsnam/ns/.

[RF99] K. K. Ramakrishnan and Sally Floyd. A proposal to add explicit congestion notification (ECN) to IP. RFC 2481, January 1999.

[RJ88] K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer. In *Proceedings of ACM SIGCOMM*, 1988.

[RKT99] Dan Rubenstein, Jim Kurose, and Don Towsley. The impact of multicast layering on network fairness. In *Proceedings of ACM SIGCOMM*, volume 29 of *Computer Communication Review*, pages 27–38, October 1999.

[spr] Spread group communication system. http://www.spread.org/.

[SYST98] Tetsuo Sano, Nagatsugu Yamanouchi, Teruji Shiroshita, and Osamu Takahashi. Flow and congestion control for bulk reliable multicast protocols – toward coexistance with TCP. In *Proceedings of INFOCOM*, March 1998.

[VKCD99] R. Vitenberg, I. Keidar, G. V. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. Technical Report CS99-31, Institute of Computer Science, The Hebrew University of Jerusalem, 1999.

[WS98] Huayan Amy Wang and Mischa Schwartz. Achieving bounded fairness for multicast and TCP traffic in the Internet. In *Proceedings of ACM SIGCOMM*, 1998.