A Cost-Benefit Flow Control for Reliable Multicast and Unicast in Overlay Networks

Yair Amir, Baruch Awerbuch, Claudiu Danilov, Jonathan Stanton
{yairamir, baruch, claudiu}@cs.jhu.edu, jstanton@gwu.edu

***Abstract*— When many parties share network resources on an overlay network, mechanisms must exist to allocate the resources and protect the network from overload. Compared to large physical networks such as the Internet, in overlay networks the dimensions of the task are smaller, so new and possibly more effective techniques can be used. In this work we take a fresh look at the problem of flow control in multi-sender multi-group reliable multicast and unicast and explore a cost-benefit approach that works in conjunction with Internet standard protocols such as TCP.**

In contrast to existing window-based flow control schemes we avoid end-to-end per sender or per group feedback by looking only at the state of the virtual links between participating nodes. This produces control traffic proportional only to the number of overlay network links and independent of the number of groups, senders or receivers. We show the effectiveness of the resulting protocol through simulations and validate the simulations with live Internet experiments. We demonstrate near optimal utilization of network resources, fair sharing of individual congested links and quick adaptation to network changes.

I. Introduction

THIS paper presents a flow control strategy for multi-group multi-sender reliable multicast and unicast in overlay networks, based on competitive analysis. Our work focuses on maximizing the total throughput achieved by all senders in overlay networks where many participants reliably multicast messages to a large number of groups.

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 of a network resource, 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, and 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 [1] that using a cost function that increases exponentially with the resource's utilization is competitive with the optimal off-line algorithm. Competitive ratio is defined as the maximum, over all possible scenarios, of the ratio between the benefit achieved by the optimal offline algorithm and the benefit achieved by the online algorithm.

Our online algorithm allows the 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 enables us to optimize for various goals. By adjusting the benefit function, performance issues such as throughput and policy issues such as fairness can be taken into account when making flow control decisions. For example, the benefit can be the number of packets sent (sending throughput), the number of packets received by all receivers (receiving throughput), or the average latency given some throughput constraints. In this paper we only use the sending throughput benefit function, seeking to optimize the total sending throughput of all the participants in the network.

Reliability is provided both on each link of the overlay network, and end to end between the multicast members through a membership service. In our approach, each overlay link provides local retransmissions for reliability and uses a standard congestion control protocol that adapts the available bandwidth to the network congestion. This results in a dynamic capacity being available to our flow control framework on every overlay network link. All the traffic generated by our system on a link is seen as one TCP flow on that link, regardless of the number of senders or receivers. This provides a very conservative level of fairness between our multicast traffic and competing TCP flows.

The global flow control problem deals with managing the available bandwidth of the overlay links and the buffers in the overlay nodes. One may also view this problem as congestion control for end-to-end overlay paths. The reason we define it as flow control is that at the physical network level, congestion control is achieved by TCP that runs between overlay nodes, while managing the buffers in the overlay routers is seen as an application level flow control task.

Our framework requires the sender to be able to assign cost to a packet based on the aggregate cost of the links on which it travels. We develop the framework in the context of overlay networks, where the number of network nodes is relatively small compared to the global Internet, while the number of senders, receivers and groups can be very large. For such systems, assigning the aggregate link cost is relatively cheap because dissemination tree information can be available at the sender. Also, as overlay network routers are flexible, it is easy to implement our protocol in the overlay nodes.

Our Cost-Benefit framework is evaluated through simulations and live tests on an emulated testbed and the Internet. The simulations use the ns2 simulator [2] and examine the behavior of several overlay network configurations. To conduct actual network tests we extended the available Spread group communication system [3] to implement our flow control

Y. Amir, B. Awerbuch and C. Danilov are with the Department of Computer Science at the Johns Hopkins University, Baltimore, MD 21218

J. Stanton is with the Department of Computer Science at George Washington University, Washington, DC 20052

This work was partially supported by DARPA and NSA under grants F30602-00-2-0626 and F30602-00-2-0550

protocols, and conducted experiments using this software on both Emulab [4] and the CAIRN network [5].

The contribution of this work is a practical distributed protocol that achieves near optimal global flow control for reliable multicast and unicast in overlay networks. We demonstrate that under varying number of sending and receiving clients, changing link characteristics, external competition from other traffic on the links, and internal competition from clients sending to identical or different groups, the protocol provides a fair sharing of individual congested links between both individual clients in a flow and between different flows. We demonstrate a quick adaptation to changing capacities on the network and to competing traffic. We further demonstrate that senders can each achieve their own near optimal sending rate without being constrained by the ability (or lack thereof) of other senders.

II. Related work

Many different approaches exist in the flow control literature, including TCP-like window based protocols [6], [7], one or two bit feedback schemes [8], [9], [10], and optimization based flow control [11], [12], [13], [14], [15], [16]. The economic framework for flow and congestion control used in many optimization based protocols [12], [14] 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 based on the rationality of the participants, or influence non-cooperative processes to behave, but rather optimizes the throughput under the assumption of minimally cooperative (non-rational or even malicious) senders.

This paper builds on our previous work applying the Cost-Benefit Framework in various resource management problems such as virtual circuit routing [1], job assignment in metacomputes [17], and our earlier work on multicast flow control [18] which forms the foundation for this paper.

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 [19], [20], [21], [22] which continue to provide the traditional strong semantic properties such as reliability, ordering, and membership. These systems predominantly extend a flow control model previously used in local area networks, such as the Totem Ring protocol [21], or adapt a window-based algorithm to a multi-sender group [23], [22]. Our work presents a flow control algorithm designed explicitly for wide-area overlay networks which is motivated more by networking protocols and resource optimization research, than by existing group communication systems.

Work on flow control for multicast sessions has occurred 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 traffic such as defining fairness [24], [25] and determining appropriate metrics for evaluation of multicast traffic [26]. 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 [27], [28], rate based protocols [29], [18], multi-layer based protocols [24], and protocols that use local recovery to optimize congestion control [30]. While IP-Multicast focuses on a single sender, single group approach that scales to many receivers and many intermediate routers, our approach addresses a multi-group multi-sender problem that scales with the number of groups, senders and receivers, but is defined in an overlay network setting rather than on every router in the Internet.

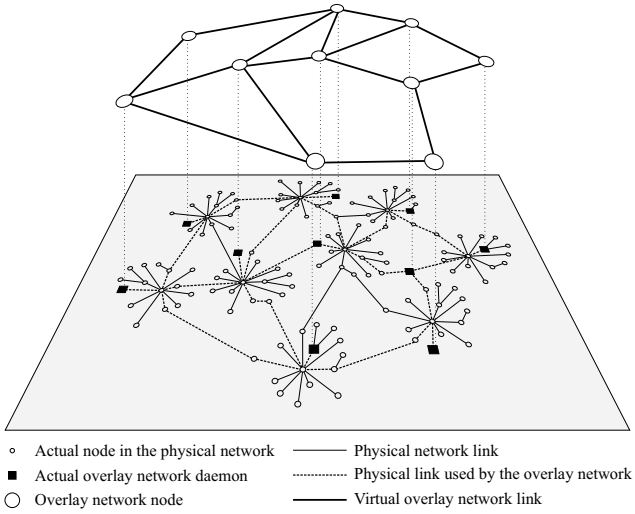

Fig. 1. Overlay Network Architecture

III. Architecture

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 seen in Figure 1. Based on the network topology, each daemon chooses a tree from this graph, in which it will multicast messages. This tree is rooted at the daemon node and may differ from other daemons' trees. In this work, we use the standard TCP protocol on each of the overlay overlay links. The choice of TCP gives us a clean baseline to evaluate the behavior of our Cost-Benefit framework without side effects introduced by a different protocol. However, any other point-to-point reliable protocol could be used instead of TCP.

We define a client as an individual connection between a user application and an overlay daemon. A user application

may choose to open multiple connections to an overlay daemon, but then each connection will be treated independently by the daemon. The daemon provides multicast services to clients, and 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 any daemon (preferably the closest one) and that daemon handles the forwarding of their traffic and provides all the required semantics, including reliability and ordering. The connection from a client to a daemon is either a TCP/IP connection or a local IPC mechanism such as Unix Domain Sockets. Each client can reliably 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 entire protocol described in this paper is implemented only at the daemon level and is completely transparent to the multicasting clients. What the clients see is just a TCP/IPC connection to a daemon, and they send their messages via a blocking or non-blocking socket. It is the responsibility of our flow control to regulate the acceptance rate of the client-daemon connection.

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

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

The Spread group communication toolkit: We implemented our global flow control algorithm in the Spread wide area group communication system [3], [31]. 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, total order), and a membership service supporting Extended Virtual Synchrony (EVS) [32] and Virtual Synchrony (VS) [33] models. It is designed to support a small to medium number of members of a group (1-1000's), with a large number of active groups and many senders. As such, it has different design goals than most IP-Multicast systems, which support larger groups but focus on the best-effort, 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 reliability by using a reliable point-to-point protocol for each link on the overlay network [22] and through a group membership service.

IV. Global flow control for wide area overlay networks

The algorithmic foundation for our work can be summarized as follows: We price links based on their "opportunity cost", which increases exponentially with link utilization. We compare different connections based on the total opportunity cost of the links they use, and slow down connections with large costs, by delaying their packets at the entry point.

A. Algorithmic foundation

Whether a message is accepted or not into the system by a daemon is an online decision problem. At the time of acceptance it is not known how much data the sending client (or the other clients) will be sending in the future, nor at what specific times in the future.

The general problem with online allocation of resources is that it is impossible to optimally make irreversible decisions without knowing the future nor the correlations between past and future. Thus, our goal is to design a "competitive" algorithm whose total accrued benefit is comparable to that achieved by the optimal offline algorithm, on *all* scenarios (i.e. input sequences). The maximum possible performance degradation of an online algorithm (as compared with the offline) is called the "competitive ratio". Specifically,

$$\rho = \max_{x} \frac{B_{offline}(x)}{B_{online}(x)} \tag{1}$$

where x is the input sequence, $B_{online}(x)$ is the benefit of the online algorithm, and $B_{offline}(x)$ is the benefit 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.

Theoretical background for the cost-benefi t framework: Our framework is based on the theoretical result in [1]. The framework contains the following components:

- User benefit function is defined, representing how much benefit a given user extracts out of their ability to gain resources, e.g., ability to communicate at a certain rate.
- Resource opportunity cost is defined based on the utilization of the resource. The cost of a completely unused resource is equal to the lowest possible connection benefit, and the cost of a fully used resource is equal to the maximum connection benefit.
- A connection is admitted into the network if the opportunity cost of resources it wishes to consume is lower than its benefit.
- Flow control is accomplished, conceptually, by dividing the traffic stream into packets and applying the above admission control framework for each packet.

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

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.

For the purpose of developing the reader's intuition, it is useful to consider a somewhat restrictive setting where the resources are either assigned forever, or rented out for a 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 $\gamma = \beta/\alpha$. In our framework, the opportunity cost of a unit of resource is:

$$C(u_l) = \gamma^{u_l} \tag{2}$$

As we will describe later, in our approach a unit of resource is a packet slot in the link buffers. Such an exponential cost function leads to a strategy where each $1/\log_2 \gamma$ of the fraction of the utilized resource necessitates doubling the price.

For a path or a multicast tree consisting of multiple links, the opportunity cost of the path is the sum of opportunity costs of all the links which make up the path.

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 a 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 the voice part, additional \$5 for video, and additional \$2 for color.

Notice that concave functions enable one to implement some level of fairness: given 50KB of bandwidth, 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.

An online auction model: Let us focus on the following simple case of auctioning off an arbitrary resource, say link capacity, in an online setting where the bids arrive sequentially and un-predictably.

The *input* to the problem is a sequence of bids with benefits B_1, B_2, B_k that are positive numbers in the range from α to β, generated online at times t_1, t_2, t_k; each bid requests fraction r_i of the total resource.

The *output* is a sequence of decisions D_i made online, i.e. at times t_1, t_2, t_k, so that $D_i = 1$ if the bid is accepted and $D_i = 0$ otherwise. The total benefit of the auction is $\sum B_i \cdot r_i \cdot D_i$ and the inventory restriction is that $\sum D_i \cdot r_i \leq C$ where C is the resource capacity.

The question is what is the optimal online strategy for decision making without knowing the future bids, given that decisions to accept "low" bids cannot be reversed after knowing about future higher bids. On the other hand, it is dangerous to wait for high bids since they may never arrive.

This problem of designing a competitive online algorithm for allocating link bandwidth was shown in [1] to have a lower bound of $\Omega(\log \gamma)$ on the competitive ratio ρ, where γ is the ratio $\gamma = \beta/\alpha$ between maximal and minimal benefit. It is achievable if $1/\log_2 \gamma$ of the fraction of the utilized resource necessitates doubling the price of the resource.

Specifically, at time t_i, the cost of the resource is defined as $C_i = C(u_l) = \gamma^{u_l}$, and the decision to accept $D_i = 1$ takes place if and only if $C_i < B_i$.

Let P be the highest bid accepted by an optimal offline algorithm, but rejected by our algorithm. Since $1/\log_2 \gamma$ of the fraction of the utilized resource necessitates doubling the price, then in order to reject a bid with benefit P, our algorithm should have set the resource cost higher than P, which means that at least $1/\log_2 \gamma$ fraction of the utilized capacity was already sold for at least half of P. So the benefit B_{online} achieved by our online algorithm is at least

$$B_{online} > P/(2 \cdot \log_2 \gamma) \tag{3}$$

The total "lost" benefit, i.e. benefit of all the bids accepted by the offline algorithm and rejected by our algorithm, is at most P, achievable if the entire resource was sold at maximum price P by the offline algorithm. If we define $B_{offline}$ as the total benefit achieved by the offline algorithm, then:

$$B_{offline} - B_{online} < P \tag{4}$$

If we plug P from Equation 4 into Equation 3 we get $B_{online} > (B_{offline} - B_{online})/(2 \cdot \log_2 \gamma)$ which follows to a competitive ratio ρ of:

$$\rho = \frac{B_{offline}}{B_{online}} < 1 + 2 \cdot \log_2 \gamma \tag{5}$$

This shows that our strategy of assigning an exponential cost to the resource leads to a competitive ratio that is within a logarithmic factor of γ.

B. Adapting the model to practice

The above theory section shows how 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 [1].

The core theory has several assumptions which do not exactly match the reality in overlay networks. We will examine

these assumptions and adapt the ideas of the Cost-Benefit framework to work in overlay 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, available bandwidth, defined as the link capacity that can be used by our overlay protocols while fairly sharing the total capacity with the other external traffic, is not under the overlay nodes' control. Competing external Internet flows may occupy at any point an arbitrary and time-varying fraction of the actual link capacity. Moreover, it is practically impossible to measure instantaneous available bandwidth without using invasive methods. Therefore, while available 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 that share the network with our overlay. (Recall that our application has to share the link bandwidth "fairly" with other external TCP flows.)

This leads to the following adaptations:

1) Accommodating Packet Switching: Although the model assumed admission control of connections in a circuit switched environment, it can be applied to packet switching in a straight-forward way. The path of each packet can be viewed as a short time circuit that is assigned by the source of the packet. For each packet, we can make a decision to accept or delay that packet individually.

2) Rationed Resource: The underlying model does not specify which resources are to be managed by it. One of the most important issues is deciding what resource is to be controlled (rationed) since not all of the resources used are controllable. Figuratively speaking, available 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 controllable resources.

Thus, we chose buffer space in each overlay node as the scarce resource we want to control. Conceptually, we model our software overlay node 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 overlay node. Note that the number of queues is equal to the number of outgoing links, and does not depend on the number of senders, receivers or groups in the system. If a link is not congested, its corresponding queue will be empty. In this case, once a packet arrives in the buffer, it is immediately sent (maybe with a short delay due to operating system scheduling). If the incoming traffic is more than the capacity of an outgoing link, some packets will accumulate in the corresponding outgoing link buffer.

3) Practical Cost Function: Each overlay node establishes the cost for each of its outgoing links and advertises this cost to all the other nodes. 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 packets 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 a given value C_{max} when no user will be able to buy it.

Equation 2 from Section IV-A gives the basic form of our cost function. The utilization of a link buffer is given by n/M where n is the average number of packets in the buffer and M is the desired capacity of the buffer. The cost of a link l as a function of packets in its buffer is $C_l(n) = \gamma^{n/M}$ which ranges from 1 to γ. We scale the cost of each resource from 0 to C_{max} (the prohibitive cost), so the cost becomes:

$$C_l(n) = C_{max} \cdot \frac{\gamma^{n/M} - 1}{\gamma - 1} \tag{6}$$

The theory does not make any assumptions about the user benefit function, or about the minimum and maximum user benefit that define the base of the exponent γ. If we use a large base of the exponent, then the cost function stays near zero until the buffer utilization is almost 1, and then the cost goes up very quickly. This would be acceptable if we had instantaneous feedback, but as we can only get delayed information from the network our reaction to a cost increase might be too late, allowing the number of used buffers to increase above our desired soft limit before any protocol could react and slow down the sending rate. A practical mechanism will provide incremental feedback on costs before the utilization becomes high, which calls for a small base of the exponent. For simplicity we chose e as the base of the exponent, so finally we get:

$$C_l(n) = C_{max} \cdot \frac{e^{n/M} - 1}{e - 1} \tag{7}$$

Each router will periodically advertise the cost of its links by multicasting a cost update message to all the other daemons through the overlay network. In Section V we show how to minimize the control traffic while maximizing the information it contains.

Each sending daemon can compute the cost of a packet by summing up the cost of all the links that packet will traverse in the multicast tree, plus a constant Δ that will be discussed below in Section IV-B.5 . If we consider MT the set of the links belonging to a multicast tree of a packet p, then the total cost of the packet, C_p, is computed as:

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

$$C_p = \Delta + \sum_{l \in MT} C_l \tag{8}$$

Given the exponential nature of our cost function, it may be possible to approximate the cost of a path with the cost of the single link having the highest utilization, as a higher utilization is likely to yield a dramatically higher cost value.

This approximation can be useful in systems that do not use a link-state propagation mechanism to reduce the control traffic in a way similar to distance vector algorithms. In this work we do not need to use such an approximation as we already have all of the necessary information to compute the cost of a path as the sum of the cost of all the links it uses.

4) Benefit Assignment: The choice of benefit is tightly intertwined with the goal we want to achieve. In this work we chose to maximize the total sending throughput, which means we aim to send globally the maximum number of packets within a unit of time. As we delay packets at the entry point thorough our admission control mechanism, intuitively, the benefit of each packet increases with the time it waits to be sent. A user that has its packets delayed is more "desperate" than a user that has its packets sent immediately. In addition, we would like to encourage users that use cheap, non-congested links to forward their packets, and slow down the ones that use highly congested links.

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 due to the low granularity of the operating system scheduling. This is why we scale both the resource cost and the benefit functions to a value C_{max} higher than γ. We define a "salary" as the amount of benefit units (say dollars) a client is given from time to time. A client is allowed to save up to $S = C_{max}$ dollars of its salary. The mechanism implements a token bucket of dollars with S, the maximum cost of a link, as the bucket size. We define the minimum benefit of a packet to be $\alpha = 1$, and the maximum benefit to be $\beta = e$, achieved when the amount of accumulated tokens is S. This leads to a range from 1 to $\gamma = e$, as our initial link cost function, and we scale it in a linear function from 1 to C_{max}, the prohibitive cost of a link. If the number of tokens available in the bucket is k, with $0 \leq k \leq S$, the benefit of sending a packet is

$$B = 1 + k/S \cdot (C_{max} - 1) \tag{9}$$

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

If the client wants to send a packet that costs more benefit dollars than it currently has in its budget, the daemon blocks the client by not reading from its socket anymore, and keeps the expensive packet until the client affords to buy it. This happens when the client receives more benefit dollars, or when the cost for the sending tree goes down. Since the sender will continue to accrue benefit and the links have a maximum possible cost, the packet will eventually be sent.

5) Packet Processing Costs: A link that is not congested has a cost of zero, and a client that sends through it will see it this way until the next cost update arrives. Therefore, any client would be allowed to send an infinite number of packets on a non congested link between two cost updates, obviously leading to high burstiness and congestion on these links. A solution for this problem is to have a minimum cost per link greater than 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 link cost at zero, but add a constant cost per packet (like a processing fee). This cost is the constant Δ referred to in Equation 8, and in our implementation we define it to be \$1. Therefore we put a cap on the number of packets each client can send between two cost updates, even when the network cost is zero, because of the limited salary.

6) Non-intrusive Bandwidth Estimation: 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. When 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 packet more expensive that it could afford). If the client was not blocked, it means that he was able to buy, and therefore send, all the packets 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 packet that the client bought during the last salary period with a certain threshold H. If the maximum cost is less than the threshold, the time between two salaries is decreased, as the client might have been blocked due to processing fees on a relatively idle network. If the maximum cost is larger then the threshold H, it means that the client tried to buy expensive packets, therefore contributing to congestion in the network. The threshold H can be any arbitrary value larger than the processing fee. However, the larger the threshold is, the more likely each client will get an increase in the salary rate, even in a congested network, resulting in higher buffer occupancy. In our implementation we chose $H = 2$, slightly bigger than the processing fee $\Delta = 1$.

The way we adjust the salary period follows the TCP congestion control algorithm. If the salary period should be decreased then the new salary period is:

$$T_{new} = \frac{T_{old} \cdot T_{update}}{T_{old} + T_{update}} \tag{10}$$

where T_{old} is the previous salary period and T_{update} is the minimum time between two cost updates. In our experiments, the initial salary period is 1 second.

If the salary period should be increased, the new salary period will be:

$$T_{new} = 2 \cdot T_{old} \tag{11}$$

This algorithm resembles the TCP congestion control [7] 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 10 is algebraically equivalent to adding 1 to the congestion window in TCP, while equation 11 is equivalent to reducing the congestion window by half.

7) Cost Update Synchronization: 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 overlay node buffers as every client buys/sends at the same time, then the cost goes up, then everybody waits for the price to go down, etc. To prevent the synchronization, the overlay node may delay a packet even though the client has sufficient funds to send it immediately. This delay will last until either another packet arrives in the sending queue, a

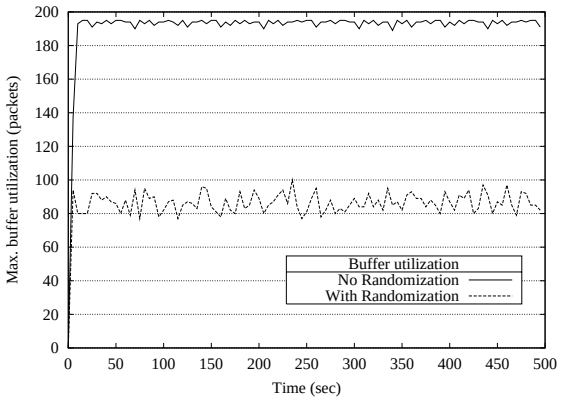

Fig. 2. Randomization effect on buffer utilization

new cost update arrives, or a short timeout elapses. Note that this scheduling delay does not reduce the client's budget, so the main mechanism of admission control remains unchanged. Ideally, this scheduling randomization should depend on the number of senders in the system competing for a link. As we do not have this information we use an approximation in which the client will choose to send a packet with probability $1/C_p$, where C_p is the total cost of the packet, otherwise delay the packet.

The experiment depicted in Figure 2 demonstrates the benefit of randomization. In this experiment, our flow control is deployed on a single 2Mbps link serving 100 streams that compete over the link capacity. The figure shows the maximum buffer utilization with and without randomization.

V. Fairness and scalability

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. We fairly share each link with all the external 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. However, for the purpose of this paper we tried to provide an overlay network flow control that works in any environment and thus made the conservative choice.

The difference between looking at the receiving throughput and at the sending throughput when comparing a multicast protocol with TCP is big, as there can be more than one receiver for one sender. However, we try to be very conservative by taking into account the worst case scenario and analyze 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 packet. That could be because their packets travel over the same multicast tree, or just by coincidence. However, senders who have higher costs, e.g. because they cross more congested links, will be allowed to send at a lower rate. This is depicted in Section VII Scenario 3, where sender A-F who uses all the network links receives much less then its "fair" share of the resources.

A. Router State Scalability

The cost-benefit flow control protocol provides a fine-grained level of control (per-group, per-sender, per-packet flow control) in a complex multi-group multi-sender multicast environment, without keeping any per-flow state in the intermediate routers. The only required router state is one cost record for each outgoing link of the router. Moreover, the amount of control traffic does not depend on the number of groups, senders, or receivers in the system, neither does it carry any information about them. The cost updates carry information only about the state (buffer sizes) of the links - edges in the overlay network graph. Thus, a much larger number of clients and groups, in the order of thousands to tens of thousands can be supported.

B. Frequency of Cost Updates

Each daemon in the overlay network multicasts a cost update at every T_{max} interval as long as its outgoing links are not congested, or their costs did not change significantly. However, if at least one of its links becomes congested - the link cost increases - the daemon will send more frequent cost updates, at T_{min} intervals. This mechanism is based on the observation that, in general, in a multicast tree there are only a few bottleneck links that will limit the speed of the entire tree. Moreover, it is likely that the bottleneck links for different senders or groups will be the same. Therefore, only the daemons that control bottleneck links will send frequent cost updates, while the others will not contribute much to the control traffic. Since the cost updates are very small (64 bytes in our implementation), they are piggy-backed with the data packets whenever possible. Electing the values of the advertising intervals T_{max} and T_{min} is a compromise between the control traffic we allow in the network and the performance degradation due to additionally delayed feedback. They also depend on the diameter of the network, the maximum client link bandwidth, and the size of buffers in the intermediate nodes. In our experiments we show that in practical overlay networks with delays in order of tens of milliseconds and throughput in the order of megabits per second, values in the order of T_{max} = 2.5 seconds, and T_{min} 50 milliseconds, coupled with overlay buffers of about 100 packets, achieve good performance. For an overlay network with the average link bandwidth of 2Mbps this leads to a control traffic of about 0.5 percent per congested link, and 0.01 percent per non-congested link. We believe that for higher throughput networks we may either send cost updates more often or increase the size of the overlay buffers.

VI. The Cost-Benefit protocol

To summarize, we present the Cost-Benefit protocol below:

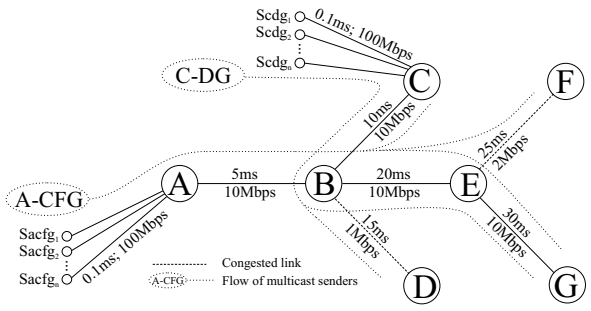

Fig. 3. Scenario 1: Network Configuration

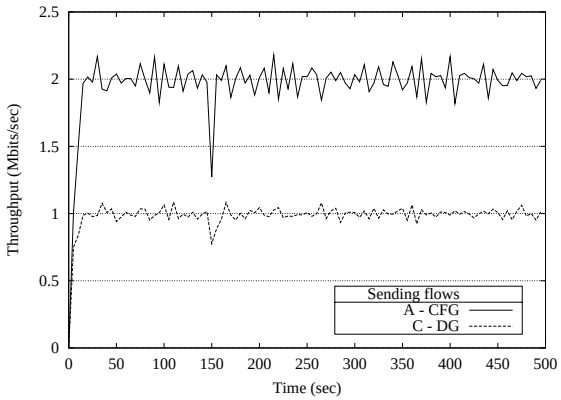

Fig. 4. Scenario 1, Simulation: Throughput

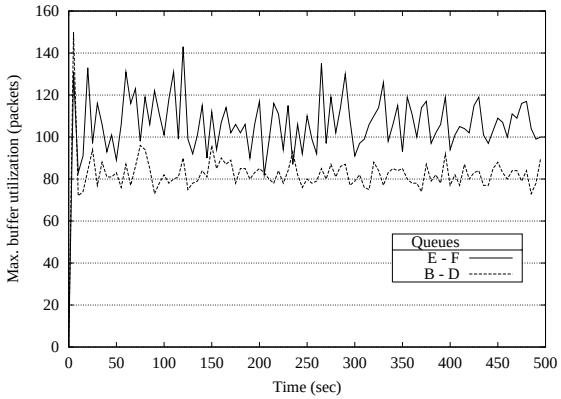

Fig. 5. Scenario 1, Simulation: Buffers

- On each topology change, the overlay routers compute their routing tables, and the set of links in the overlay that will be used by their clients to multicast packets
- When an overlay node needs to forward a packet, if a reliability window of a downstream link is full, the overlay node will buffer the packet. For all its downstream links that have at least a packet in their buffer, the overlay node computes the link cost C_l and multicasts it to the other routers every T_{min} interval. For all other links, the overlay node advertises a zero cost every T_{max} interval.
- Overlay nodes maintain a token-bucket budget and a token rate for each of their clients. The cost of a packet is computed based on the cost of the links that packet will use, and is subtracted from the budget of the client that sent it.
- Clients that cannot afford sending their current packet are blocked until either their budget increases (they get more tokens) or the cost of their current packet decreases.
- The token rate of the clients "salary period" is adjusted only for clients that are blocked: If the most expensive packet they bought over the last salary period was higher than the threshold, their salary period doubles (the rate is reduced by half). Otherwise, the new salary period T_{new} becomes:

$$T_{new} = \frac{T_{old} \cdot T_{update}}{T_{old} + T_{update}} \tag{12}$$

where T_{old} is the previous salary period and T_{update} is the minimum time between two cost updates.

VII. Simulation results

We used the ns2 network simulator [2] 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 between flows using the same congested links;
- Scalability with number of clients, groups and diameter of the network;

Scenario 1 – achieving the optimal network throughput: 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 $S = C_{max}$ = \$20 in their budget. The processing fee is Δ = \$1/packet.

Two classes of 20 separate clients each initiate multicast messages, S_{acfg} and S_{cdg}. Receiver clients are connected to nodes C, D, F and G. For simplicity we do not show the receiving clients, but only the daemons they are connected to. The S_{acfg} clients multicast to receivers connected to nodes C, F and G, and the S_{cdg} clients multicast to receivers connected to nodes D and G, sharing the links B-E and E-G. S_{acfg} clients are limited by the 2Mb bottleneck link $E - F$, and S_{cdg} clients are limited by the 1Mb link $B - D$. There are no other bottleneck links in the system.

The aggregate sending throughput of the two flows is shown in Figure 4. The two flows achieve maximal network usage, S_{acfg} clients getting on average 1.977 Mbps and S_{cdg} getting 0.992 Mbps.

Rather than looking at the instantaneous buffer occupancy which is very dynamic and depends on the sampling frequency, we chose to analyze the evolution of the upper bound of the buffer utilization. We measure the maximum buffer size over the last sampling period and present it in Figure 5.

The reason for a higher buffer utilization on link $E - F$ is that there is a higher feedback delay from node E to node A (25 milliseconds) than from node B to node C (10

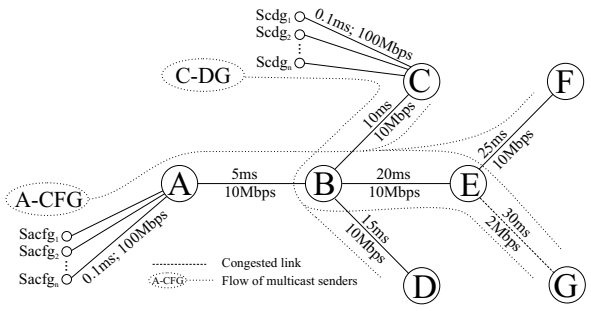

Fig. 6. Scenario 2: Network Configuration

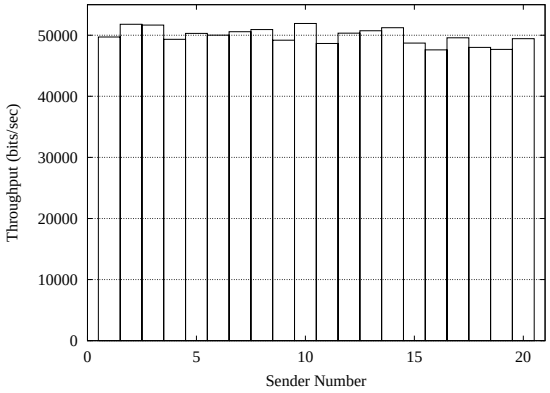

Fig. 7. Scenario 2, Simulation: Fairness

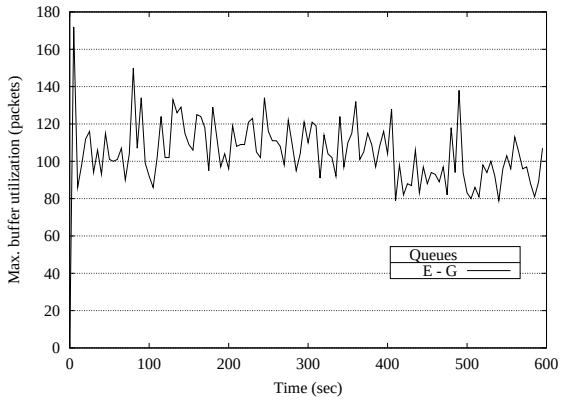

Fig. 8. Scenario 2, Simulation: Buffers with delayed senders

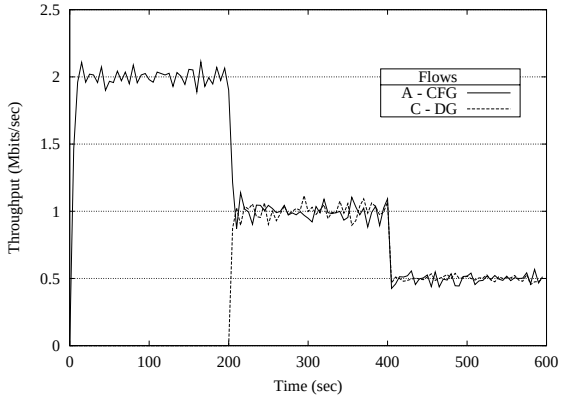

Fig. 9. Scenario 2, Simulation: Throughput with delayed senders

milliseconds), as in Figure 3. Link $E-F$ also experiences higher variability in buffer utilization and throughput. In general, higher latency paths will experience higher variability in throughput and buffer occupancy.

Scenario 2 – fair sharing of network resources: To examine the effect of a congested link, the network shown in Figure 6 is used. Here, link $E-G$ forms the bottleneck for both flows. Each flow represents 20 sending clients.

The two flows share the bottleneck link fairly equal. Flow S_{acfg} gets an average of 997.3 Kbps and flow S_{cdg} gets an average of 997.6 Kbps, while the buffer of the link $G-E$ stays below 150 packets. The various clients who make up each flow also share the bandwidth fairly. In Figure 7, we show the sending throughput achieved by each of the 20 S_{acfg} clients. The variance of the clients throughput was less then 4.6%.

A second experiment uses the same tree configuration as Figure 6 but starts the second group of senders S_{cdgg} only after 200 seconds, and also changes the bandwidth of the link E-G to 1Mbps after 400 seconds. Figure 8 shows the maximum buffer utilization on the links E-G, B-D and E-F. After 200 seconds, as well as after 400 seconds we do not see any major change in the buffer utilization on the bottleneck link. Specifically, there is no large spike in maximum utilization when the second group of clients begins sending all at once, or when the bottleneck link reduces its capacity by half. This is because the link has an existing non-zero cost and so the clients must pay that cost before sending. Figure 9 shows how the throughput of the two groups of clients responds to the new load, or change in the available bandwidth, sharing fairly the congested link. The response time for adjusting the rate of the flow S_{acfg} when the second flow is introduced was under 5 seconds.

Scenario 3 – unicast behavior and comparison with TCP:

Our flow control tries to maximize throughput by allowing low cost packets to pass, and reducing high cost traffic. A simple way to demonstrate this is to set up a chain network in which some clients try to send their packets across the entire network, while other clients use only one link in the chain. Figure 10 shows such a network with 5 links connected in a chain. One client sends from node A to node F, and 5 other clients send only over one link, i.e. from B to C or from E to F.

Figure 11 shows the throughput on the chain network as short path connections start up every 150 seconds. The client A-F starts trying to use the entire capacity of the network. When the client A-B starts, they share the congested link, AB, about equally. When the third client, 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 throughput of the flow A-F goes almost to zero, thus almost maximizing the global throughput of the system. If the flow control had been fair, the aggregate throughput would be 6 Mbps, 1 Mbps for each

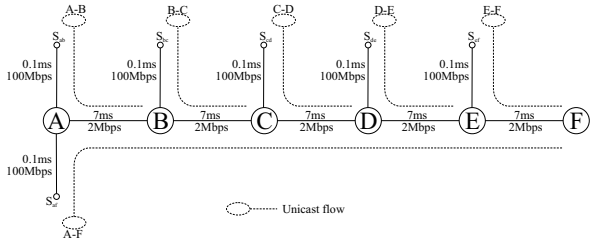

Fig. 10. Scenario 3: Network Configuration

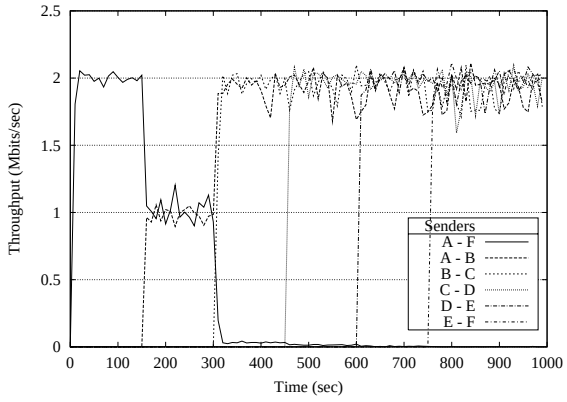

Fig. 11. Scenario 3, Simulation: Throughput

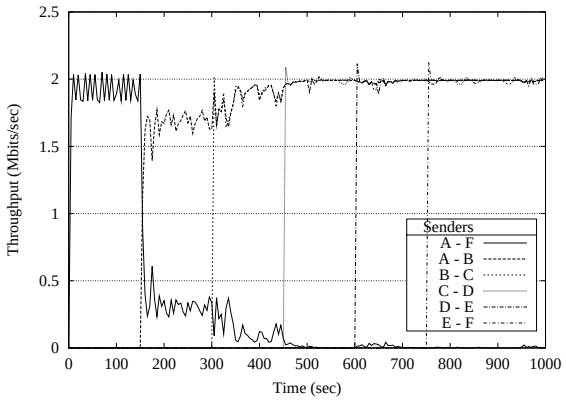

Fig. 12. Scenario 3, Simulation: TCP throughput

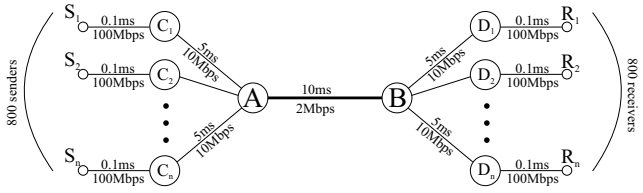

Fig. 13. Scenario 4: Network Configuration

client. We achieved an aggregate throughput after all clients have started of 9.677 Mbps, while the theoretical maximum is 10 Mbps.

The results of the previous simulation present a definite bias toward short flows and show how such a bias can increase network throughput. One can view reliable unicast connections as a special case of reliable multicast, and in this experiment we show that our cost-benefit flow control achieves similar behavior to that of a set of end-to-end connections using TCP on the same network.

Figure 12 presents 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 end-to-end TCPs, 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 flow A-F 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.

Scenario 4 – scalability with number of nodes and groups: In order to see how a large number of clients multicasting to many different groups share the network resources, we set up the network presented in Figure 13. The overlay network consists of 1602 nodes, and there are 1600 clients, each of them connected to a separate daemon, joining 800 different groups. We could not run a bigger scenario due to memory limitation using the ns simulator on our machines.

Each of the clients S_1 to S_{800} multicasts to a different group composed of three different receivers. S_1 sends to R_1, R_2 and R_3, S_2 sends to R_2, R_3 and R_4, and so on, until S_{800} that sends to R_{800}, R_1 and R_2. All the senders share the same bottleneck link, A-B.

We ran the simulation with different number of senders, from 5 to 800. As shown in Figure 14 the maximum buffer utilization on the bottleneck link A-B stays about the same until the number of senders reaches to the buffer soft limit (in our case, 100), and then it starts increasing. However, the Cost-Benefit framework kept the buffer size under controllable limits (under 170 packets for 800 senders). The aggregate throughput was not affected by the number of senders, getting an average of 1.979Mbps for the aggregate sending rate of 800 senders.

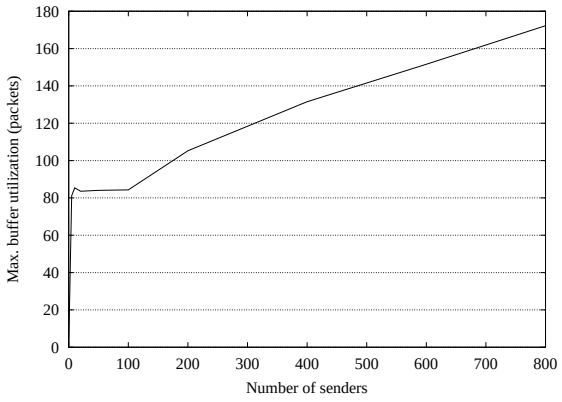

Fig. 14. Scenario 4, Simulation: Buffers

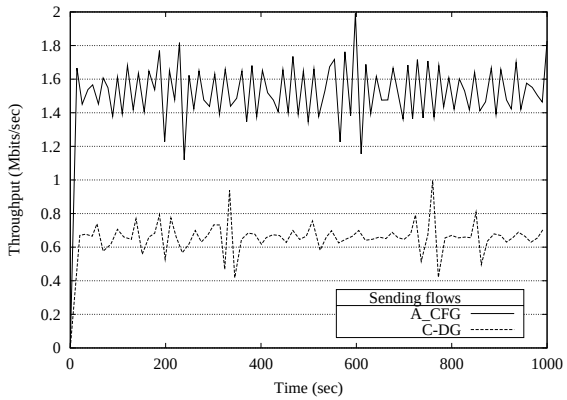

Fig. 15. Scenario 1, Emulab: Sending throughput

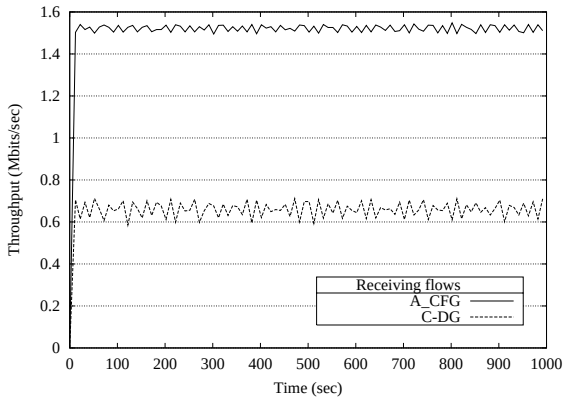

Fig. 16. Scenario 1, Emulab: Receiving throughput

VIII. Simulation validation on an emulated wide area testbed

In order to validate our simulation experiments we extended the Spread toolkit [3] to use our Cost-Benefit framework for global flow control. We then run Spread on Emulab [4] where we created the network setups of **Scenario 1** and **Scenario 2** presented in Section VII.

Emulab allows real instantiation in a hardware network (composed of actual computers and switches) of a simulation topology, simply by using the ns script in the configuration setup. Link latencies and bandwidths are emulated with additional nodes that hold packets for a while, or drop them when the traffic increases above the bandwidth requirement. The emulated link latencies measured with ping were accurate up to a precision of $\pm$ 3ms, while the throughput measured by TCP flooding was 1.91Mbps for the 2Mbps bottleneck link and 0.94Mbps for the 1Mbp link.

Spread has its own overhead of about 15% of data sent due to headers required for routing, group communication specific ordering and safety guarantees, as well as to provide user-friendly group names of up to 32 characters. In addition, any node that is not part of receiver set of a multicast message does not receive the actual message, but must receive a 96 byte ordering header required for group communication guarantees to be maintained, no matter how big the message is. Therefore, receiver D in **Scenario 1** recives a message header for each message sent in the S_{ACFG} flow. Note that Spread allows messages to be as large as 100 KB.

What we measured in our results is *actual user data sent and received* by clients connected to Spread, sending 1200 byte messages. 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 buffer limit of 100 packets.

Figure 15 shows the sending throughput of the two flows in **Scenario 1**, while Figure 16 shows the receiving throughput at the nodes behind bottleneck links.

The S_{acfg} flow achieved a sending rate of 1.53Mbps while the S_{cdg} flow achieved 664Kbps. Taking into account the Spread overhead and meta-headers this leads to a total throughput of 1.9Mbps for the S_{acfg} flow and 904Kbps for the S_{cdg} flow. Comparing these numbers with the available bandwidth offered by the Emulab setup, we obtain a difference of about 4% between what we get and the available network resources.

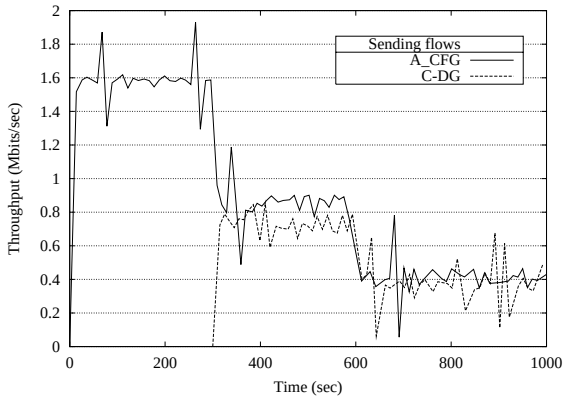

Fig. 17. Scenario 2, Emulab: Sending throughput

Figure 17 and Figure 18 show the sending and receiving throughput achieved by Spread clients in **Scenario 2**. As we start the S_{cdg} flow at time 300 we see the two flows fairly share the bottleneck link. Similarly, when the available bandwidth on the bottleneck link drops to 1Mbps at time 600, both flows adapt to the network conditions by reducing their rate to half.

The above experiments show that the system implementation of our cost-benefit flow control achieves good performance on a controlled emulated testbed with real computers and networks. We achieve similar results to the simulated experiments, showing the feasibility of our adaptation of the theoretical model to practical networked environments.

IX. Real-life Internet experiments

To further validate our results and demonstrate real-life behavior we conduct experiments over a portion of the CAIRN network [5]. 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 CAIRN routers are Intel machines that run FreeBSD. Figure 19 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

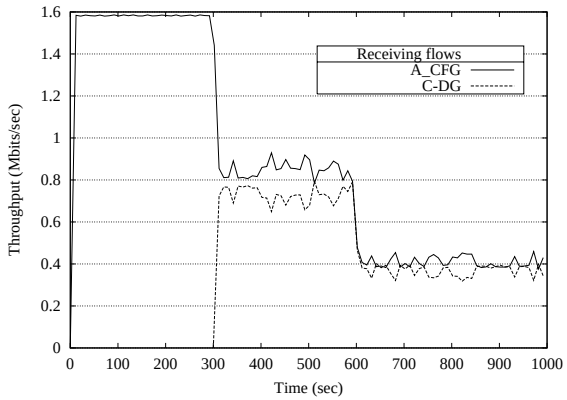

Fig. 18. Scenario 2, Emulab: Receiving throughput

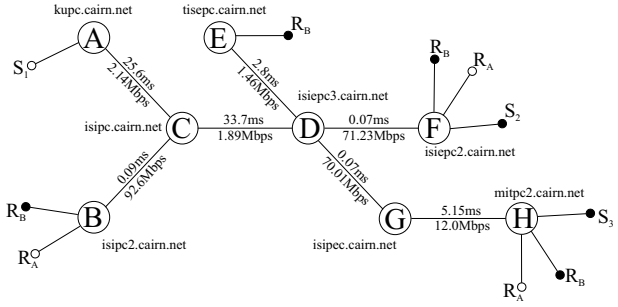

Fig. 19. CAIRN: Network Configuration

bandwidth with point to point TCP connections for each 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.

Sender S_1 multicasts messages to a group A joined by the receivers R_A, while senders S_2 and S_3 multicast to a group B joined by the receivers R_B. All the clients run directly on the overlay network machines, connected to the daemons through Unix Domain Sockets. Obviously, S_1 was limited by the bottleneck link C-D, while S_2 and S_3 had to share the bottleneck link D-E. Taking into account the data overhead in Spread, we can see in Figure 20 that the sending clients use the network resources optimally and share them fairly between senders S_2 and S_3, S_1 getting 1.417 Mbps, while S_2 and S_3 got 0.618 and 0.640 Mbps respectively. Comparing these numbers with the available bandwidth offered by CAIRN we achieve a difference between 1% and 14%.

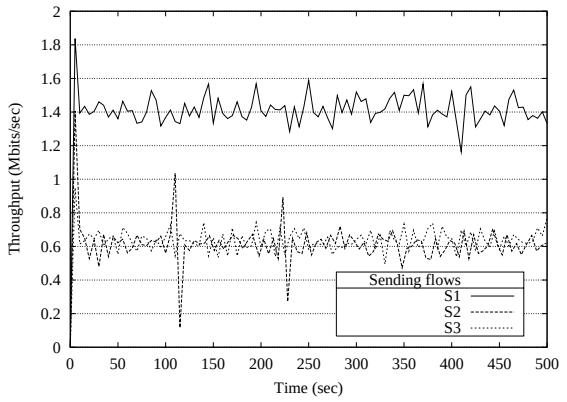

Fig. 20. CAIRN: Sending throughput

The uncontrollability of the Internet network conditions did not affect the performance of our protocol. The real-life Internet experiments show that different senders located at different sites and multicasting messages to the same or different groups achieve near optimal bandwidth utilization and fairly share the network resources.

X. Conclusions

This paper presented a global flow control approach for multicast and unicast in overlay networks that is scalable with the number of groups and participants and is 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 live experiments on Emulab and CAIRN network to validate the simulations and show the real-world performance of the framework.

References

[1] B. Awerbuch, Y. Azar, and S. Plotkin, "Throughput-competitive on-line routing," in *Proceedings of 34th IEEE Symposium on Foundations of Computer Science*, vol. 30, 1993, pp. 32–40.

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

[3] "Spread group communication system," http://www.spread.org/.

[4] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *OSDI02*. Boston, MA: USENIXASSOC, Dec. 2002, pp. 255–270.

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

[6] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[7] V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, vol. 18, 4, pp. 314–329, 1988.

[8] K. K. Ramakrishnan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks with a connectionless network layer," *Proceedings of the 1988 SIGCOMM Symposium on Communications Architectures and Protocols; ACM; Stanford, CA*, pp. 303–313, 1988.

[9] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, October 1994.

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

[11] R. G. Gallager and S. J. Golestaani, "Flow control and routing algorithms for data networks," in *Proceedings of 5th International Conference on Computers and Communication*, 1980, pp. 779–784.

[12] R. J. Gibbens and F. P. Kelly, "Resource pricing and the evolution of congestion control," *Automatica*, vol. 35, December 1999.

[13] S. J. Golestani and S. Bhatacharyya, "End-to-end congestion control for the Internet: A global optimization framework," in *Proceedings of International Conference on Network Protocols*, October 1998, pp. 137–150.

[14] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, March 1998.

[15] D. Lapsley and S. Low, "An IP implemention of optimization flow control," in *Proceedings of IEEE Globecom*, 1998, pp. 3023–3028.

[16] D. E. Lapsley and S. Low, "Random early marking for Internet congestion control," in *Proceedings of IEEE Globecom*, vol. 3, 1999, pp. 1747–1752.

[17] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, and A. Keren, "An opportunity cost approach for job assignment and reassignment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 760–768, July 2000.

[18] Y. Amir, B. Awerbuch, C. Danilov, and J. Stanton, "Global flow control for wide area overlay networks: A cost-benefit approach," in *Proceedings of IEEE Openarch*, June 2002.

[19] I. Keidar, J. Sussman, K. Marzullo, and D. 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*. Taipei, Taiwan: IEEE Computer Society Press, Los Alamitos, CA, April 2000, pp. 356–365.

[20] I. Keidar and R. 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*. Taipei, Taiwan: IEEE Computer Society Press, Los Alamitos, CA, April 2000, pp. 344–355.

[21] D. 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*, vol. 16, no. 2, pp. 93–132, May 1998.

[22] Y. Amir, C. Danilov, and J. Stanton, "A low latency, loss tolerant architecture and protocol for wide area group communication," in *Proceeding of International Conference on Dependable Systems and Networks*. IEEE Computer Society Press, Los Alamitos, CA, June 2000, pp. 327–336, fTCS 30.

[23] T. M. Hickey and R. van Renesse, "Incorporating system resource information into flow control," Department of Computer Science, Cornell University, Ithaca, NY, Tech. Rep. TR 95-1489, 1995.

[24] D. Rubenstein, J. Kurose, and D. Towsley, "The impact of multicast layering on network fairness," in *Proceedings of ACM SIGCOMM*, ser. Computer Communication Review, vol. 29, October 1999, pp. 27–38.

[25] T. Bonald and L. Massoulie, "Impact of fairness on Internet performance," in *SIGMETRICS/Performance*, 2001, pp. 82–91.

[26] R. C. Chalmers and K. C. Almeroth, "Developing a multicast metric," in *Proceedings of GLOBECOM 2000*, vol. 1, 2000, pp. 382–386.

[27] H. A. Wang and M. Schwartz, "Achieving bounded fairness for multicast and TCP traffic in the Internet," in *Proceedings of ACM SIGCOMM*, 1998.

[28] L. Rizzo, "pgmcc: a TCP-friendly single-rate multicast congestion control scheme," in *ACM Computer Communications Review: Proceedings of SIGCOMM 2000*, vol. 30, October 2000, pp. 17–28.

[29] T. Montgomery, "A loss tolerant rate controller for reliable multicast," West Virginia University, Tech. Rep. NASA-IVV-97-011, August 1997.

[30] S. Chang, H. J. Chao, and X. Guo, "TCP-friendly window congestion control with dynamic grouping for reliable multicast," in *Proceedings of GLOBECOM 2000*, vol. 1, 2000, pp. 538–547.

[31] Y. Amir and J. Stanton, "The Spread wide area group communication system," Johns Hopkins University Department of Computer Science, Tech. Rep. 98-4, 1998.

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

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

Yair Amir is a Professor in the Department of Computer Science, Johns Hopkins University where he served as Assistant Professor since 1995, Associate Professor since 2000, and Professor since 2004. He holds a BS (1985) and MS (1990) degrees from the Technion, Israel Institute of Technology, and a PhD degree (1995) from the Hebrew University of Jerusalem. Prior to his PhD, he gained extensive experience building C3I systems. He is a creator of the Spread and Secure Spread messaging toolkits, the Backhand and Wackamole clustering projects, and the Spines overlay network platform. He has been a member of the program committees of the IEEE International Conference on Distributed Computing Systems (1999, 2002, 2005), the ACM Conference on Principles of Distributed Computing in 2001, and the IEEE International Conference on Dependable Systems and Networks (2001, 2003, 2005). He is a member of the ACM and the IEEE Computer Society.

Baruch Awerbuch is currently a (full) professor at the Computer Science Dept. at Johns Hopkins University (www.cs.jhu.edu/ baruch). His current Research interests include: Security, Online Algorithms, Distributed and Peer-to-Peer Systems, Recommendation Systems, and Wireless Networks.

Baruch Awerbuch has published more than 100 papers in journals and refereed conferences in the general area of design and analysis of online algorithms, combinatorial and network optimization, distributed algorithms, learning, fault tolerance, network architecture, and others.

Baruch Awerbuch is a co-director of the JHU Center for Networks and distributed systems http://www.cnds.jhu.edu.

Dr. Awerbuch served as a member of the Editorial Board for Journal of Algorithms, Wireless Networks and Interconnection Networks. He was a program chair of the 1995 ACM Conference on Wireless Computing & Communication and a member of the program committees of the 2004 ACM Mobihoc, as well as PC member ACM PODC Principles of Distributed Computing (PODC) Conference in 1989 and of the Annual ACM STOC (Symposium on Theory of Computing) Conference in 1990 and 1991.

Claudiu Danilov is an Assistant Research Scientist in the Department of Computer Science, Johns Hopkins University. He received the BS degree in Computer Science in 1995 from Politehnica University of Bucharest, and the MSE and PhD degrees in Computer Science from The Johns Hopkins University in 2000 and 2004. His research interests include distributed systems, survivable messaging systems and network protocols. He is a creator of the Spines overlay network platform.

Jonathan Stanton received the BA degree in Mathematics in 1995 from Cornell University, and the MSE and PhD degrees in Computer Science from The Johns Hopkins University in 1998 and 2002. He is currently an Assistant Professor in the Computer Science department of the George Washington University. He also holds an appointment as an adjunct assistant professor in the Computer Science department of The Johns Hopkins University. His research interests include distributed systems, secure distributed messaging, network protocols, and middleware support for clustered systems. He is a member of the ACM and the IEEE Computer Society.