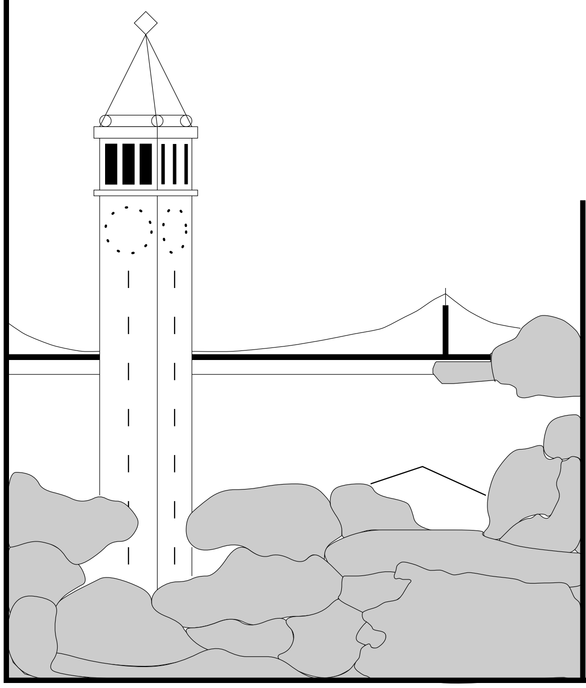
The Breadcrumb Forwarding Service and the Digital Fountain Rainbow: Toward a TCP-Friendly Reliable Multicast

Koichi Yano *Steven McCanne*
University of California, Berkeley
{*yano,mccanne*}@*cs.berkeley.edu*



Report No. UCB/CSD-99-1068
October 1999

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Abstract—**If ubiquitously deployed, IP Multicast promises to provide an efficient datagram service for an arbitrary sending host to reach an arbitrary and dynamic set of destination hosts anywhere in the Internet. Unfortunately, two very difficult problems —interdomain multicast routing and viable end-to-end multicast transport— have yet to be solved and deployed satisfactorily.**

This paper proposes a new multicast service model called the *Breadcrumb Forwarding Service* (BCFS) synthesized from the EXPRESS service model and the network component of the Pragmatic Multicast protocol (PGM). Like EXPRESS, BCFS utilizes explicit-source group join and like PGM, enhances the network forwarding architecture with finer-granularity group control. To demonstrate the flexibility and efficacy of BCFS, we developed a novel reliable multicast transport protocol, *Rainbow*, for bulk data transfer built on top of BCFS. In this approach, each receiver runs an independent, window-based congestion control algorithm modeled after TCP, thereby affording "TCP friendliness" while retaining the efficiency of IP Multicast. To enhance scalability and support asynchronous receiver behavior, we employ a Digital Fountain at the source. In this paper, we detail BCFS service model, describe how Rainbow builds on this service model and on the Digital Fountain abstraction, and evaluate the resulting system. Our simulation results show that Rainbow/BCFS performs well in a variety of configurations and is "TCP friendly".

I. Introduction

The cornerstone of the Internet's resounding success is arguably the end-to-end design principle [1], which says that a given system function should operate at the lowest communication layer in which it can be wholly and correctly realized. When applied to network design, an end-to-end philosophy naturally leads to an architecture where few constraints are placed upon the network itself — e.g., the network can drop, delay, replicate, and corrupt packets — and richer services like reliable, sequenced delivery are defined and implemented at the edge of the network in an "end to end" fashion. In the Internet architecture, the IP network layer offers a *best effort* delivery service and richer transport services like TCP are built on this best-effort IP service.

This split between a deliberately simple network layer and a rich transport layer naturally leads to a robust and scalable system. Because so few presumptions are placed on the network, not only is such a network relatively easy to engineer and deploy at large scale, but end-to-end protocol and application designers must conscientiously account for the indeterminacies of the underlying network. In effect, the best-effort service model calibrates the designer's expectations for an environment like the Internet, where consistent, homogeneous, and high-performance communication is often the exception rather than the rule. Thus, the end-system software that results tends to be *robust*, and as a consequence, the Internet as a whole has continued to scale gracefully despite an onslaught of new and evolving constituent technologies that are decentrally managed, heterogeneous, and imbued with mixed levels of reliability.

Quite naturally, then, when Deering proposed IP Multicast [2] — an enhancement of the traditional Internet architecture for efficient multipoint packet delivery — he very deliberately appealed to the end-to-end design principle. Like unicast, Deering's multicast service model is best effort and richer services like reliability must be implemented in the end-hosts. Unfortunately, whereas the end-to-end approach has enjoyed tremendous success as the bedrock of the unicast Internet, its adaptation to the multicast has created two very difficult design problems that have yet to be satisfactorily solved: First, because multicast routing is so more complex than its unicast counterpart, a viable interdomain multicast routing protocol has yet to be developed; and second, transport services like reliable multicast are confounded by the best-effort network model where packet drops can impact indeterminate subsets of the receiver group. Despite more than a decade's worth of research, a viable interdomain multicast routing has yet to materialize and a reliable multicast transport protocol that offers congestion control and robust and scalable behavior remains a research problem.

The failure of many and varied research efforts to bear truly viable end-to-end multicast transport protocols [3], [4], [5] or truly viable wide-area, interdomain multicast routing protocols [6], [7], [8] brings into question whether the proposed multicast service model is in fact the appropriate core building block. None of the proposals for reliable multicast have satisfied the requirements for "safe" deployment on the public Internet [9], e.g., is scalable, robust, congestion controlled, accommodates heterogeneity, and so forth. Nor have the routing protocols provided the degree of control, stability, flexible, robustness, and scalability required by service providers to deploy them flexibly in the complex peering relationships that will be required for universal deployment.

Reacting to this mixed success, several researchers have proposed alternative multicast service models. The EXPRESS service model, for example, abandons the IP Multicast anonymity of the class D group address [10]. Instead, EXPRESS advocates a model where a multicast tree is rooted at a single source and receivers explicitly indicate that source when subscribing to a multicast channel. Similarly, the so-called *simple multicast* architecture [11] proposes that the group addressing architecture be extended to explicitly embed the IP address of a "rendezvous point" in the multicast group. In either model, the normal unicast routing infrastructure can be used to route multicast data and/or control traffic since unicast addresses are explicit in the revised addressing architecture. The premise is that this approach simplifies the so-called *rendezvous problem*, and because of several other attractive properties, purportedly induces a more viable multicast architecture.

Whereas EXPRESS and simple multicast re-examine the multicast routing architecture from first principles, other work advocates the strategic placement of intelligence within the network infrastructure to solve multicast transport problems. For example, RMTP proposes that *designated receivers* be placed within the network infrastructure to carry out localized retransmissions using subtree multicasts to enhance scalability [12]; LRMP proposes the deployment of logging receivers that provide a similar function [13]; and the Reliable Multicast proXy (RMX) architecture [14] relies on proxy agents within the network to carry out format and protocol conversion to accommodate network heterogeneity and effect congestion control.

Rather than rely on service node deployment within and across the network, other works address the problem of how one might jointly optimize the design of a new multicast transport service with complementary end-to-end transport protocols, thereby retaining many of the merits of the end-to-end approach. The Lightweight Multicast Services (LMS) architecture [15] pioneered this basic approach. In LMS, multicast routers conspire to arrange the receivers into a tree-based hierarchy that is congruent with the underlying network topology. This hierarchy is exposed to the end clients through a service model extension that allows a host to send a packet to its logical parent in the tree. This extension in turn enables an end-to-end multicast transport protocol that implicitly exploits the network topology to optimize its performance. Similarly, the AIM architecture provides a rich addressing structure that from within a single framework offers many different forwarding services, e.g., subtree multicast, standard group multicast, anycast, and so forth [16]. On top of AIM, several multicast transports have been fashioned, including a reliable multicast transport. Finally, the randomcast forwarding service was proposed as an alternative to LMS to enhance robustness by breaking the hierarchy of parent/child relationships with randomized forwarding thus eliminating single points of failure. Reliable multicast protocols can then be layered on top of the randomcast forwarding service, e.g., Search Party and Rumor Mill [17].

A similar, though less modular, approach has been undertaken in the PraGmatic Multicast (PGM) protocol [18]. Here, routers are enhanced with transport-level knowledge and an end-to-end protocol is built on top of this transport-aware network infrastructure yielding a monolithic solution for reliable multicast loss recovery. In PGM, receivers generate NACKs to repair missing data, and PGM-aware routers coalesce NACKs by maintaining per-packet sequencing state in the routers. This state, in effect, forms a "trail of breadcrumbs" from the receivers missing a given piece of data back to the source of that data. When the source receives notification of new breadcrumb state, it generates a retransmission that in turn follows the breadcrumbs to each requesting receiver and simultaneously tears down the breadcrumb state that represents that retransmission request. This loss recovery scheme is optimal in the sense that retransmissions are sent to only those receivers that are in need of that data.

Unfortunately, in PGM, the network service is not clearly defined as a separable and reusable component network forwarding service. Instead the network component is a PGM-specific router optimization rather than a general extension to the multicast forwarding service. We believe that this codependence between the network and transport layers is unnecessary. In fact, the PGM router-assist component can instead be cast as an explicit-source multicast service that is optimized for fast group establishment and teardown. In this interpretation, PGM NACKs represent explicit-source group joins, as in the EXPRESS service model, and the source-generated PGM retransmissions represent special data packets that simultaneously induce group teardown. Thus, we can recast PGM as a EXPRESS-like *breadcrumb forwarding service* (BCFS) and an end-to-end transport protocol layered on top of BCFS, where PGM is but one of many possible transport protocols. When viewed in this light, each PGM NACK request corresponds to a multicast subscription interaction in a framework in which multiple multicast groups provide multicast loss recovery [19].

In this paper, we present BCFS as a synthesis of the PGM and EXPRESS architectures and further describe a novel reliable multicast transport protocol for bulk transfer built on top of BCFS. Unlike most all other reliable multicasts including PGM, which transmit data using the normal IP multicast delivery service then recover from losses using some other mechanisms, our protocol — which we call *Rainbow* — is built exclusively upon the BCFS service model. Also unlike previous works, Rainbow includes a congestion control algorithm that is modeled after that in TCP and thus provides a viable solution for congestion-controlled reliable multicast. In this approach, each receiver maintains its own congestion window and runs slow-start and congestion avoidance [20] individually by driving the equivalent of the TCP "ack clock" with breadcrumb requests. To enhance Rainbow's scalability and support asynchronous receiver subscriptions, Rainbow utilizes a Digital Fountain [21] at the source to temporally decorrelate what data to send from when it must be sent. This approach allows receivers to exercise asynchronous and autonomous behavior while simultaneously enjoying

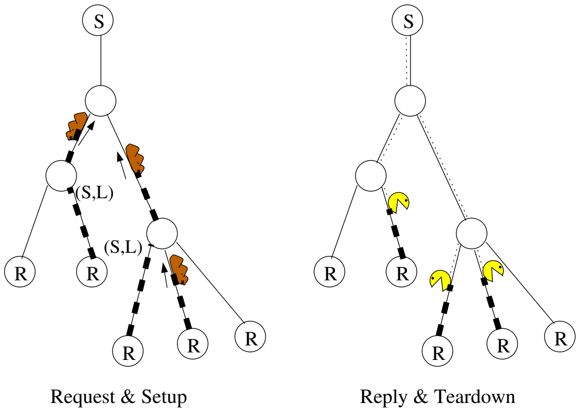


Fig. 1. Basic Service Model by BCFS.

the performance benefit of synchronous multicast communication.

Our core contribution lies not in the particular protocol described herein, which we continue to investigate and refine, but rather in the overall architecture and general direction of the approach — we readily admit that several practical engineering issues and details for how BCFS would be implemented are omitted from this first generation design. In a nutshell, BCFS/Rainbow brings together several novel protocol idioms described elsewhere into a new framework for multicast communication that not only represents a viable method for deployment — as evidence by Cisco's efforts in developing and deploying PGM — but provides a service that can be successfully used across a wide variety of multicast applications and protocols. In addition to supporting Rainbow, BCFS can effectively support a variant of PGM and, as described later, provides a superset of the EXPRESS multicast service model and thus shares many of its attractive properties. Finally, armed with a BCFS network service, the Rainbow transport protocol represents a truly TCP-friendly reliable multicast protocol.

In the remainder of this paper, we develop detail the BCFS service model and describe how Rainbow builds on this service model. In the next section, we describe the BCFS service model and transport API. In section III, the design of Rainbow is detailed and simulation results on exploring its performance is shown. We discuss future work and conclude the paper in section V.

II. Breadcrumb Forwarding Service

A. Service Model

The *Breadcrumb Forwarding Service* (BCFS) unifies the EXPRESS service model and the network component of PGM into a single, flexible multicast service model. To this end, BCFS provides a single-source, request-based multicast service, where groups can be efficiently set up and torn down in tandem with a data exchange, loosely analogous to how T/TCP optimizes the establishment and teardown of a TCP connection in a single response/reply dialogue [22]. BCFS is thus optimized for ephemeral groups that come and go rapidly and is consequently well-suited as a building block for multi-group reliable multicast schemes.

To avoid unnecessary transport-level dependence, BCFS uses an abstract "label" to identify a particular request with respect to some source. The source/label pair (S,L) thus induces an group-oriented address architecture that is precisely analogous to the source/channel (S,E) framework proposed in EXPRESS. BCFS differs from EXPRESS, however, in that messages are sent from the receivers toward the source along the multicast tree and are suppressed if a message with the same label has already been sent up the tree. The group membership protocol is exposed to and run at the application layer, and arbitrary messages can be piggy-backed onto these control messages.

Figure 1 illustrates this breadcrumb forwarding model. Request messages for some piece of data drop breadcrumbs along the path to a source: the breadcrumbs, in turn, guide the reply message from the source back to all requesting receivers. Each breadcrumb is identified by an (S,L) pair to differentiate the forwarding paths for all labels in use.

To enhance the range of transport services that can be built on top of BCFS, the forwarding model includes a level-numbering scheme for selectively tearing down the breadcrumb state. Each breadcrumb carries with it a level number and each request and response includes a level number in the header of the packet. A request packet is propagated up the tree toward the sender only if its level number exceeds the level number in the breadcrumb stored at the router (or if no such breadcrumb exists). Similarly, breadcrumb state is torn down by a response packet only if the level number is equal to or exceeds the breadcrumb level stored in the router.

An application interacts with BCFS through the prototype interface defined in Figure 2. *bcf_bind* allows an application to register its interest in receiving breadcrumb packets sent to a particular label or set of labels using an address/mask pair. If multiple processes match a given label, a copy of the message is delivered to each process.

A request packet is sent via *bcf_request*, which includes the address of the source or tree root (i.e., the destination of the message), a label, a level number, and an optional message body. The message field in a request might include transport protocol payload like sequence numbers of requested packets.

A reply message is sent by *bcf_reply*. The reply contains a label which is copied from the corresponding re-

```
bcf_bind(lab, mask);
bcf_request(src,lab,lev,msg);
bcf_reply(lab,lev,msg );
bcf_recv(src,lab,lev,msg);
```

src:	data source address (destination of request).
lab:	BCF label.
lev:	level number.
msg:	transport message.

Fig. 2. BCFS API for transport protocol

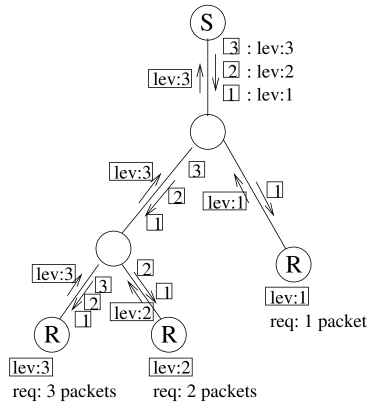


Fig. 3. Request by the same label with different levels.: each receiver receives different number of packets

quest packet. The level field contains the level to be torn down.

In summary, the sequence of events for effecting the BCFS forwarding service are roughly as follows:
(1) Request: A receiver sends a request packet with a label and level.
(2) Setup: A router that receives a request message maintains state for forwarding links and the level associated with the label.
(3) Suppression: The router forwards the request message toward the source if the label in the request message is new for the that router or the level number is larger than the highest level being maintained. Otherwise, the message is not forwarded.
(4) Reply: A source, in response to the request message, sends the requested data together with the label embedded in the request message and a level number to be torn down.
(5) Forwarding: A router directs a reply message to the links that are associated with the label.
(6) Teardown: The router removes the forwarding state of the link associated with the label, if the reply message includes a level number that is larger than the level maintained by the router.

A.1 Label and Level Use

How applications and protocols generate labels depends on how the protocol designer wishes to differentiate and aggregate higher-layer messages. For example, in PGM, the transport sequence number could be hashed with a port number or some other application specific identifier to produce a label for a retransmission request. Of course, hashing can result in address conflicts so applications must be prepared to deal with superfluous data coming from other, unrelated applications or sessions. But, if the label space is large enough and the label generation functions are chosen well, then the probability of collision will remain low and not adversely impact protocol performance. Moreover, a separate, independent label space exists for each source, so the impact of collisions is quite limited. This contrasts with the existing IP multicast service model where any host in the network can send arbitrary data to an arbitrary group and collisions are prone to happen especially in the absence of a globally consistent multicast address allocation scheme (which remains a difficult research problem).

The level numbering scheme controls request forwarding and tear-down timing. For example, Figure 3 shows how a level number can be used to request a specific number of packets on some label in a receiver-specific fashion. Here, a request message for N packets uses a level number of N, where N can vary among the different receivers, say $N_1 \ldots N_R$. Let $M = \max_{k=1\ldots R} N_k$. The request with the largest level M then reaches the source without suppression, and as a result the source learns M. The source then generates M packets with level numbers $1 \ldots M$, and each receiver receives exactly the number of packets requested.

This level-numbering scheme interacts nicely with previously proposed schemes for FEC-based loss recovery [23]. Here, the above scheme can be extended by having each receiver generate a NACK for a block of packets indicating how many packets were omitted from the block (i.e., corresponding to the N_k's). Then the source would generate M parity packets that recover the lost packets for each receiver, no matter which packets were lost. Details of this mechanism as applied to the traditional multicast service model are described at length in [23].

B. Router Behavior

We call a router that supports BCFS a *BCF Router*. A BCF Router maintains forwarding information associated with labels (*BCF Label*). End hosts exchange *BCF Messages*, which consist of *BCF Request* from receivers and *BCF Reply* from the source.

To effect BCFS, routers maintain "breadcrumb state" tied to a particular label, but they do not store a copy of the message. The breadcrumb also includes level number that corresponds to the highest request level for that label seen so far.

If a router receives a request message for a label that is already stored in the router and the level number is less than or equal to the level number stored in the breadcrumb, then the message dropped, much as DVMRP [2], EXPRESS [10], and PIM [6] join messages are coalesced in a multicast distribution tree. Otherwise, the message is propagated up the tree toward the source S using the normal unicast routing tables (i.e., along the reverse-path multicast route back to S). Thus, as in PGM, routers can fuse requests by suppressing label messages if the state has already been established.

If the message makes it all the way to the source subnet, the router incident to the source delivers it to that source and an application (which has presumably bound itself to the label in question via `bcf_bind()`) receives the message. Because suppression is carried out on a per-label basis, if different receivers send messages on the same label at the same time, only one message will be delivered to the source.

Upon receiving a request, the source may respond with an arbitrary message tied to the label in question. When a source sends a "reply" packet bound to a particular label, the routers forward the packet along all links that have breadcrumb state tied that label. As a side effect of forwarding the packet, the breadcrumb state is deleted, which allows future messages to be propagated back up the tree and frees up router resources.

Unlike PGM NACKs, labeled request messages are not sent in a hop-wise reliable fashion, which means that only breadcrumb state needs to be maintained in the router, not the entire message body. However, this also means that a lost request message that never makes it to the source suppresses further messages sent for that label. To avoid this, label state is refreshed in a soft-state fashion [24], [25]. To this end, when a router suppresses the propagation of a label because of existing breadcrumb state, it verifies that the breadcrumb state has been recently "refreshed", e.g., according to the scalable session messages algorithm in [25]. If the label needs to be refreshed, a null message for that label is sent toward the source. Thus, if the source receives a specially marked null message, it knows the original request message was dropped somewhere in the network and can invoke a higher-layer recovery process if necessary. By using data-driven state updates, the router need not manage timers to otherwise trigger soft-state updates.

To complement the soft-state update process, breadcrumb entries may be deleted by the router if no update is received after a certain time interval. Yet unlike the normal multicast group management machinery, tearing down this state is not critical because updates are data driven from the receivers and generated only if a receiver is explicitly present. Thus, the pool of breadcrumbs could either be timed out by a soft-state aging process or entries could simply be reallocated using LRU replacement.

As defined, BCFS is a superset of and can implement EXPRESS. To do so, each receiver periodically generates a null request message addressed to some channel (S,L) with level number 1, and the source sends data packets as responses to label L with level number 0. Thus, the breadcrumb state is maintained exactly as if it were an EXPRESS channel and packets are sent best-effort to every receiver in the source-specific group identified by the label.

B.1 Setup and Request Suppression

A multicast channel is set up when a request packet arrives at a router by updating state in the router.

If (S,L) in a request message is new for a router, the router appends a new entry for the label and maintains the link identifier for the link that the message came from as a directed link. A level number is maintained coupled with each forwarding link. The request message is then forwarded toward a source.

On the other hand, if the router's state already has a entry for (S,L), the router explores the list of forwarding links. If the link is not in the list, the link identifier is added to the list of directed links tied to the label and a level number is also maintained for the link. If the link is already designated as a directed link but the request has higher level than that in the router's state for the link, the level number is updated. Only in the case where the level number is larger than any level numbers of forwarding links tied to (S,L) is the message forwarded to the up-link. Otherwise, the request message is suppressed.

B.2 Teardown and Forwarding

A forwarding path is torn down by deleting links from the list of directed links in a router's state when a reply packet arrives at the router or a timer expires.

When a BCF reply message arrives at a router, the BCF reply message is forwarded to links listed as directed links tied to (S,L). Furthermore the router compare the level in the message with the level of each directed link. If the level number in the reply is equal to or larger than the level number for a link, the forwarding state of the link is deleted. Otherwise, the link is retained as a directed link.

B.3 PGM-like Multicast over BCFS

To illustrate the power and flexibility of BCFS for structuring the PGM network component as an independent and reusable service, we briefly describe how a PGM-like reliable multicast can be easily adapted to BCFS. In this approach, original data packets are sent to the entire multicast group. When a receiver detects lost packets, a NACK is sent via a BCF request message with a label generated by hashing sequence number of the lost packet into the low-bits of a label, with some well-known upper prefix (perhaps selected as a hash of a session-specific identifier). In response to the NACK, the source retransmits the lost data via a BCF reply message with the same label. In turn, the BCF routers forward the reply packet to precisely those receivers that earlier sent a request.

The label generated by transport sequence numbers can prevent conflicts between different NACKs. Even if the label space is smaller than the sequence number space, the label with sequential order can avoid conflict because a source is expected to send data packets sequentially. In PGM, the size of transmit window is large enough for label space to avoid conflict because only data packets within transmit window are provided for loss recovery.

III. Rainbow on Digital Fountain

Not only can a PGM-like transport be built on top of BCFS, but because BCFS is a generic network service, other transport protocols can exploit it as well. In this section, we describe a reliable multicast transport that differs quite substantially from PGM even though it is built upon precisely the same network service. In particular, our protocol exhibits a viable solution to one of the hardest problems in reliable multicast, namely congestion control.

Multicast congestion control is greatly confounded by heterogeneity amongst receivers in a group: if using only a single multicast group, a single, uniform sending rate cannot satisfy the conflicting requirement of a diverse set of receivers attached to the network at different bit rates. That is, a congestion control strategy must force the sender to transmit data according to the most constrained receiver [26], [27]. This solution is inherently unsatisfying for large-scale deployment in heterogeneous environments. Alternatively, the source can send to multiple multicast groups allowing receivers to individually adjust their reception rate by joining and leaving multicast groups [28], [29], [30]. Unfortunately, the granularity of the layers limits the degree of adaptation and the design of a control law that can manage receiver membership in a scalable and robust fashion is a hard problem that has not been satisfactorily solved.

To address these problems, we propose a reliable multicast transport based on BCFS, called *Rainbow*, (ReliAble multicast by INdividual Bandwidth adaptation using winDOW), which includes a congestion control scheme. Rainbow is designed to accomplish the following:

- A receiver receives data at its available rate as if there were a unicast TCP connection between a source and a receiver (i.e., the protocol dynamics are "TCP friendly").
- The bottleneck link or links in a multicast distribution tree are efficiently shared by data aggregation among many receivers.
- The source need not manage state on a per-receiver basis, which would otherwise limit the protocol's scalability.

A. *Digital Fountain*

The Rainbow congestion control scheme utilizes a Digital Fountain [21] on top of BCFS. To establish the context for Rainbow, we first outline the Digital Fountain abstraction.

A Digital Fountain provides a robust mechanism for "implicit" multicast loss recovery as it requires no feedback from the source. Here, a sender simply multicasts a stream of data packets that are generated by the fountain as a function of a fixed input (e.g., a file). A receiver tunes in at any point and gathers up some fixed number of packets. Once this critical number of packets is attained, the receiver leaves the group and decodes the file from the packets.

A key property of the digital fountain is that (almost) any subset of packets may be used to decode thereby alleviating any need for feedback from the receivers to the source. From the perspective of a sender's load, the scheme is extremely efficient because a sender simply transmits packets without involving any sort of loss recovery scheme. Moreover, heterogeneity can potentially be tamed since the fountain can stripe packets across multiple rates and receivers can adjust their reception rate using multiple groups as described above. Though a scheme based on receiver-driven adaptation across multiple multicast groups may eventually be shown to be viable, this approach has not been fully and comprehensively developed and we felt it worthwhile to look for alternatives.

B. *Congestion Control by using BCFS*

In a heterogeneous environment, it is difficult to satisfy all receiver bandwidth requirements with a single multicast channel. To provide different data rates for each receiver without deteriorating network condition, congestion control using BCFS is designed as follows:

- **Individual TCP-like window control:** Each receiver independently executes TCP-like window control [20]. Data transmissions are triggered by the arrival of breadcrumbs at the sender. In turn, packet arrivals at the receiver cause that host to increase its congestion window (either by one for each packet received in slow start or one packet per round-trip in congestion avoidance mode). The invariant we maintain is that the number of breadcrumbs outstanding is less than or equal to the congestion window. Thus, the number of packets in transit from the source to the receiver is bounded by the congestion window. In addition, the congestion window is controlled in response to lost packets according to measured congestion conditions on the path from the source to that receiver. Since the window control behaves as if there were a TCP session between a source and a receiver, each receiver utilizes bandwidth in a TCP-friendly way.

- **Transmission request by BCF messages:** A receiver sends a TRQ (transmission request) as a BCF request using as many labels as its window size. This means that receivers that have the same window size use the same labels for TRQs, and a receiver that has a smaller window uses a subset of the labels that are used by a receiver with a larger window size. If receivers send TRQs with a label after another receiver sends a TRQ with the same label and the TRQs sent later arrives at a BCF router before the reply message of the former TRQ arrives, the TRQs are aggregated and the copies of the identical data packet are sent to all receivers which send the TRQs with the same label. TRQ corredponds to ACK in TCP in the sense that it is sent at packet reception. However it does not need to include sequence numbers of received packets.
- **Simple reply by a Digital Fountain source:** By using a Digital Fountain, the source can merely respond to each TRQ by sending one packet after another as a BCF replie, which includes the same label as the TRQ.

Figure 4 (a) illustrates Rainbow/BCFS data aggregation, where two receivers have a shared bottleneck link, they are probable to have the same window size and it is expected that most TRQs are aggregated at the link. As a result, most of the data packets from a source are directed to both receivers. In case that two receivers have bottlenecks at down-links and one link has half the bandwidth of the other link as shown in Figure 4 (b), the slower receiver receives half of the data directed to the faster receiver, copied at the diverging point. Through using Digital Fountain source, a receiver receives different packets with high probability and reliability is guaranteed by continuing to send TRQs until enough packets arrival to reconstruct the original data,

Of course, there is no guarantee that all packets are delivered efficiently as in the example above, because window control at each receiver is not synchronized in any explicit coordinated way, and a receiver accesses data asynchronously. But when receivers have a shared bottleneck link, it could happen that the same packet loss pattern causes same window control timing, and behave in a nearly synchronous way. Simulation results in section IV demonstrate this aggregation efficiency. As a result, more data aggregation occurs at the shared link, networks can enjoy efficient transmission of multicast, and receivers can receive data at the end-to-end available bandwidth.

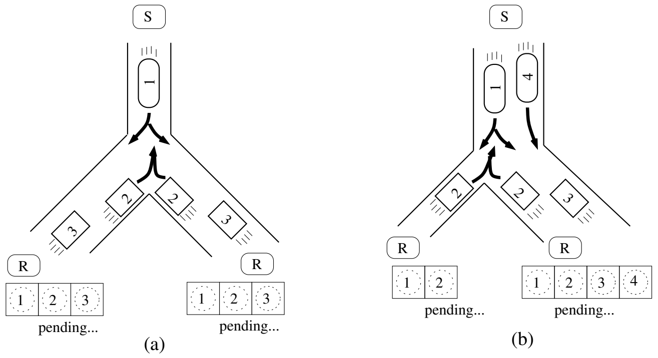


Fig. 4. Multicast Congestion Control using BCFS

C. Window Adaptation at each receiver

A receiver maintains as many labels as its window size (typically labeled from 1 to window size), and sends TRQs with a label and a level number (level 1 except for after loss detection). The forwarding state that is tied to the label at the routers on the path toward the receiver is torn down by the reply packet from the source, because a source is expected to send a reply with the same label and level. In response to the arrival of a data packet, the receiver sends another TRQ with the same label. Thus, the number of outstanding packets conforms to the window size which is also the number of labels in use.

Data transmission is initiated by a receiver's TRQ via a BCF request. At this point the receiver's window size is one, and only one label is maintained. In slow start mode, at every data reception the receiver increases window size by one, which induces two TRQs with different labels, one of which is the label in the reply packet and the other is a new label (equal to the updated window size). When packet loss is detected, window size is decreased in half (labels are 1 to (original window size)/2) and the TRQ remains pending until the number of outstanding packets becomes less than the halved window size. The number of outstanding packets is estimated by $W - L - R$, where W is the original window size, L is the lost packet number, and R is the number of packets received since loss detection. In case for failure of counting outstanding data because of consecutive packet loss, TRQ also resumes when RTT/2 has passed after loss detection. The round trip time between a source and a receiver is measured by time duration since TRQ is sent until a reply data packet arrives, and mean duration is used for the RTT estimation as in TCP. After restarting the TRQ clock, as in congestion avoidance mode, window size is increased by one when the receiver receives reply messages for all labels being maintained. Thus, the window control algorithm mirrors that in TCP.

Packet loss is detected when a timer tied to a TRQ expires. This timer is set by the RTT estimation. We can also detect packet loss when a later TRQ is replied before an earlier TRQ for another label is still waiting for a reply. When packet loss occurs, routers may retain the forwarding state tied to the label of the lost packet without being deleted. The state at the routers causes suppression of a request packet with the same label if it is sent with the original level. To avoid this suppression, when a receiver sends a TRQ with the label formerly tied to the lost packet, a higher level is used for the TRQ (level 2 or higher in case of consecutive packet loss), and the TRQ with the higher level is expected to reach the source. The source replies by sending data with as high level as the TRQ and consequently the reply with the higher level can delete its forwarding state tied to the label by one packet, even if consecutive packet loss leads to state with the large level number. After receiving the higher level reply, the receiver uses the original level (level 1) again.

IV. Evaluation

In this section, we present simulation results of several simple network topologies to study how Rainbow adapts to heterogeneous receivers and network conditions. In section IV-A, we explore the basic behavior of Rainbow based on a small scale session with simple topologies. In section IV-B, we execute simulations for larger scale sessions and compare results with RLC [29].

We implemented the BCFS network service and Rainbow/Digital Fountain on the network simulator ns-2 [31]. For forwarding of BCF request to the source, reverse path toward a source is necessary. In ns-2, because shortest-path routes are computed for the input topology, a reverse path is available from each node via the route. Our BCFS transport API can be deployed for other types of transport protocols, and in fact we have implemented PGM/BCFS as well.

A. Exploring the behavior of Rainbow

A.1 Evaluation Metrics

We investigate how Rainbow realizes each receiver's satisfaction under a heterogeneous environment, efficient utilization of network resources. Thus, following metrics are used to evaluate the congestion control scheme.

- **Bottleneck bandwidth utilization:** The data receiving rate of a receiver against its end-to-end available bandwidth shows how much it can satisfy heterogeneous receivers. A hundred percent utilization of available bandwidth is not expected, because of the TCP-like window control, which oscillates by its inherent nature.
- **Data packets aggregation:** A total packet number sent to a cluster of receivers that share the same link characterizes network resource efficient utilization. In ideal case, all receivers under the same bottleneck link should share all data packets with each other and the total packet number is equal to the number required for reconstruction.
- **Intra fairness:** Clusters of receivers under the same link share the link fairly depending on down-link capacities. Even if the shared link is not an end-to-end bottleneck, the link might not have enough capacity to accommodate all the down link bandwidth. In that case, the shared link should be utilized efficiently by BCFS data aggregation mechanism and data packets should be directed to each down-link according to its bandwidth.

A.2 Topologies and Settings

We simulated two scenarios, shown in Figure 5 to investigate the points described above. The topology of Scenario A-(i) consists of one shared up-link and five different down-links. One down-link is narrower than the shared up-link, but others have broader link capacity than the shared bottleneck and each link has different transmission delay. Through this topology, adaptation to heterogeneous receivers and sharing bandwidth at a bottleneck link is investigated. Four faster receivers should receive the same service at up-link capacity and the slowest receiver should receive a portion of the data directed to faster ones at its down-link capacity.

Scenario A-(ii) has two clusters of five receivers with the shared backbone link (L1) by all receivers and the same capacity down-link for each cluster. For the shared backbone link, we use three different bandwidth. Receivers in both clusters should receive data at the same rate in all situations in terms of intra-fairness, and the degree of data aggregation should change depending on the backbone capacity. We expect that as the backbone capacity becomes narrower, the more packets are aggregated at the link and all the receivers come to receive the same data packets if the backbone becomes an end-to-end bottleneck.

In all of the experiments, the data packet size is 512 Bytes, and the simulation run comprises 2000 packets, which means a receiver stops sending TRQs after receiving 2000 packets. To "randomize" each run, each receiver initiates its session at a uniformly random start time in [0...5] seconds. All routers are RED gateways with a queue size of 10 packets. In each scenario, 100 simulations are executed with randomized different start time of receivers and average results are shown in the following sections.

A.3 Scenario A-(i)

The results for scenario A-(i) are illustrated in Figure 6. The slowest receiver receives data at around 200 Kbps against its 256 Kbps bottleneck capacity and the average receiving rate throughout the duration of the simulation is 214.4 Kbps, with some variance at the slow-start phase. This average is also calculated from all 100 simulation

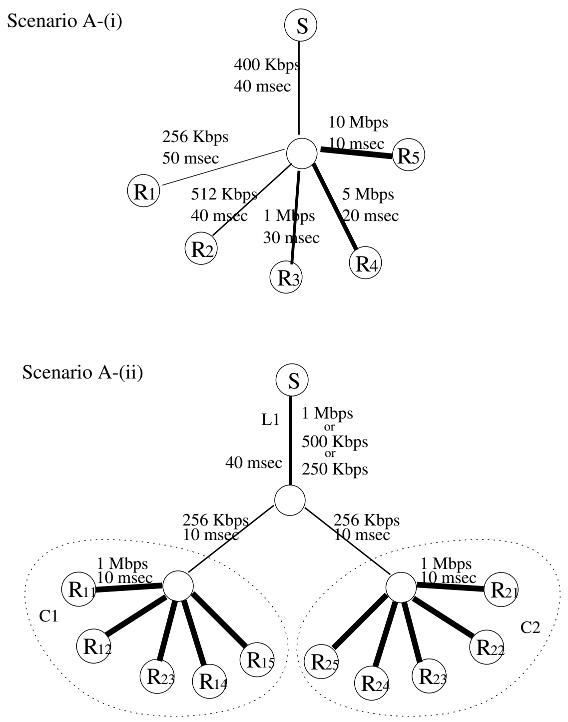


Fig. 5. Simulation Settings.

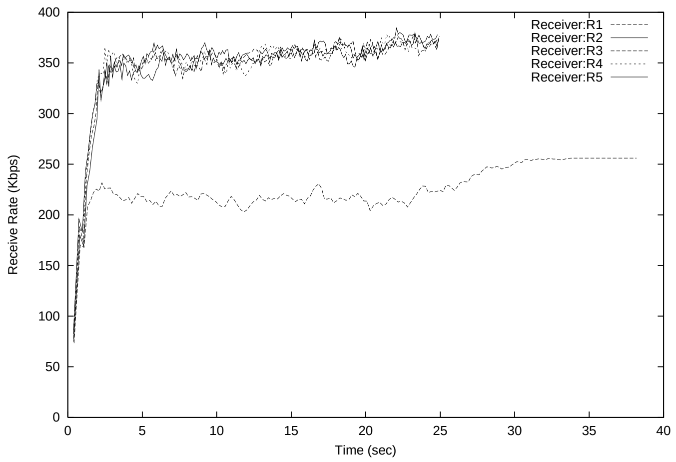


Fig. 6. Simulation Result (scenario A-(i)).

runs, and we use the average rate for later explanation of simulation results. The receiving rate of all four faster receivers reaches around 350 Kbps against 400 Kbps up-link bottleneck bandwidth after different increase rate in slow start phase because of bandwidth and delay difference and the average rate of four receivers is 329.7 Kbps.

In this scenario, four faster receivers should be dealt with as if on a single multicast channel because they share the identical bottleneck link. The overhead against the single multicast session is 15.2 % ,which is calculated by four times of the packet number sent from the source over the total packet number received by the four faster receivers during the stable condition, between 10 and 20 seconds in the simulation. Furthermore, 1264.4 packets for the slowest receiver (R1) out of 1279.9 packets are shared with other faster receivers, while at least one other receiver is receiving data.

From this simulation, we have seen all receivers can receive at their appropriate rates according to each available bandwidth through data aggregation at bottleneck link. Some overhead exists even for receivers sharing a bottleneck link, but this overhead is not expected to grow as the number of receivers increases, because the more receivers exist under the same bottleneck link, the higher the probability TRQs are aggregated resulting in the same data packet is sent to more receivers at the same time. The simulation results for larger session size is shown in section IV-B.3.

A.4 Scenario A-(ii)

In Table 1, the average receiving rate for each cluster of five receivers and the average total packet number sent from the source are shown for different bandwidth backbone (L1) cases. Figure 7 shows receiving rates of receivers in a cluster (C1) for three different backbone capacity cases. According to the average receiving rate in Table 1, receivers on both subtrees (C1,C2) attain approximately equal throughputs for all bandwidth cases.

When the shared backbone link has adequate bandwidth (1 Mbps) compared with the down-links, all receivers receive data at the down-link capacity rate. The behavior of packet aggregation can be similar to the situation in which two different multicast channels exist for each subtree because the bandwidth of the backbone link is broad enough to accommodate two different 256 Kbps multicast sessions. However, the total number of sent packets is 3402.7, which is less than the double of the necessary packet number for reconstruction from the source as shown in Table 1 because some packets are eventually aggregated for both of clusters.

When the bandwidth of shared backbone link is 500 Kbps, its capacity is a little less than aggregation of downlinks bandwidth. Even in this case, the same receiving rate is realized as in 1 Mbps case, as shown in both Table 1 and Figure 7. The reason is that more identical packets are aggregated at the the backbone link and directed to more receivers at the same time, which is also evident by less total sent packets than 1 Mbps case in Table 1.

When the shared backbone link is 250 Kbps, the link becomes the bottleneck and all receivers should be dealt with as if on one multicast channel. In Table 1, a decrease in overall packets shows that more data are aggregated at the node of the backbone link.

As the results show, depending on the placements of bottleneck link, Rainbow aggregates data packets in dif-

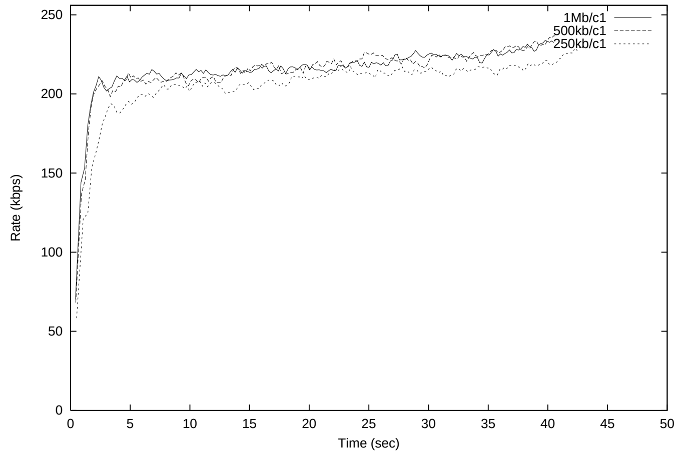


Fig. 7. Simulation Result (scenario A-(ii)).

ferent ways and as a result, traffic behaves as if a subtrees of bottleneck link is formed as the same multicast channel without explicit coordinated mechanism for synchronization.

Table 1
SIMULATION RESULTS (SCENARIO 2).

L1 Bandwidth	Average rate (Kbps)		Packet number
1 Mbps	C1	202.4	3402.7
	C2	202.6	
500 Kbps	C1	201.6	3320.5
	C2	200.7	
250 Kbps	C1	190.8	2849.1
	C2	191.9	

B. Comparison with RLC

In this section, we present simulation results that explore scalability to large session size, heterogeneity and TCP fairness. We also compare results of Rainbow with those of RLC [29], which is a TCP-friendly congestion control scheme for layered multicast.

B.1 Evaluation Metrics

We use the following metrics to compare Rainbow with RLC in terms of large session size, heterogeneous environments and TCP fairness.

• **Overhead for large scale session:** The total number of packets sent to a cluster of receivers through the shared bottleneck link characterizes efficient network resource utilization. In ideal case, if there are N homogeneous receivers under the shared bottleneck link, the total number of packets received by the N receivers is N times of the number of packets transmitted through the shared bottleneck link:

$$overhead = \frac{N * ps}{\sum_{i=1}^{N} pr_i} - 1 \tag{1}$$

is used as a metric of overhead, where ps is the number of packets transmitted through the shared link and pr_i is the number of packets received by the ith receiver.

• **Bottleneck bandwidth utilization for heterogeneous receivers:** The data transmitted through a link against its bandwidth shows utilizatoin of the link:

$$utilization = \frac{average\, transmisson\, rate}{link\, bandwidth} \tag{2}$$

the utilization of heterogeneous links is used for comparison between Rainbow and RLC.

• **Fair bandwidth share with TCP:** We use the following fairness metric proposed in [32] to investigate fair link share of concurrent Rainbow and TCP flows.

$$fairness = \frac{(\sum x_i)^2}{n \sum {x_i}^2} \tag{3}$$

where, x_i is normalized throughput of measured throughput T_i over fair throughput O_i,

$$x_i = T_i / O_i \tag{4}$$

Fair throughput is a fraction of a shared link bandwidth over a total number of Rainbow (or RLC) and TCP flows We assume that the link should be shared evenly among Rainbow (or RLC) and TCP flows and .

B.2 Topology and Settings

Figure 8 shows the topologies used in the simulations. The topology in scenario B-(i) consits of one sender and N receivers. We examine how overhead changes as the number of receivers N increases. The N receivers are homogeneous: they have the same bandwidth and delay, and the shared up-link is narrower than a down-link for each receiver. Ideally all data packets should be aggregated at the shared link and provided to all receivers, which results in 0% overhead. However, unsyncronized behavior of each receiver's congestion control and packet loss at the shared link can introduce overhead. The number of receivers N varies from 4 to 125.

Scenario B-(ii) has one sender and 13 heterogeneous receivers with different bandwidths. The nth receiver R_n's link bandwidth is $r_n = 2^{\alpha n} r_0$ $(n : -6...6)$. Thus, α indicates the degree of heterogeneity, which increases as α increases. We investigate how Rainbow and RLC adapt to such a heterogeneous environment and whether they can realize high utilization for each link. In this scenario We set r_0 to 512 Kbps, and α varies from 0.0 to 0.5. When α is 0.0, receivers are homogeneous with 512 Kbps links. In the most heterogeneous case, or $\alpha = 0.5$, the bandwidth of the narrowest link is $r_{-6} = 64$ (Kbps) and the bandwidth of the broadest link $r_6 = 4$ (Mbps).

Scenario B-(iii) and B-(iv) investigate TCP friendliness. In Scenario B-(iii), there are only two receivers for TCP and Rainbow (or RLC) respectively. Using this simple topology we examine how Rainbow (or RLC) shares a link with a TCP flow. For the shared link bandwidth, we use from 128 to 512 Kbps.

On the other hand, scenario B-(iv) has multiple sender-receiver pairs both for Rainbow (or RLC) and TCP under the same shared link. Because the bandwidth of the shared link is fixed, proper bandwidth share for each receiver is determined by the number of sessions. Rainbow (or RLC) and TCP have the same number of sessions, N. We study how Rainbow and TCP share a link under this environment of many concurrent flows with the fairness metrics as the number of sessions $2N$ varies from 4 to 16.

Data packets are 512 Bytes for all simulation runs. TCP NewReno implemantation in ns-2 [31] is used. For RLC, the base layer's sending rate is 32 Kbps and synchronization points for base layer come every second. For all scenarios, 50 simulation runs are executed and each session starts at uniformly randomized time in [0...5] seconds.

B.3 Scenario B-(i)

Figure 9 (a) illustrates the results of overhead for Rainbow and RLC as the number of receivers incereases. Overhead of RLC does not depend on the number of receivers, because receivers conduct synchronized congestion control of joining or leaving a layer. The overhead of RLC comes from lost packets, as shown in Figure 9 (b).

On the other hand, Rainbow's overhead increases with the number of receivers because of unsychoronized window control at each receiver. But this overhead does not grow linearly with number of receivers. and converges at around 25 %.

Intuition about this result is shown in Figure 10. If there two receivers, executing unsynchronized window contrrol, each receiver increases window size up to maximum value W additively acccording to their bottleneck line and reduce window in half to $W/2$ in reaction to packet loss. In this case, overhead are areas of diamonds. If there are many receivers, the overhead exptends to sum of triangle areas, which makes 25 % overhead traffic compared with a single receiver case. Actual behavior is more complicated. Some receivers are probable to suffer from shared packet loss before maximum window, which leads to more inefficiency. However this shared packet loss could result in more synchronized behavior among receivers' window control and thereby reduce overhead.

B.4 Scenario B-(ii)

Figure 11 (a) shows the results of link utilization for Rainbow and RLC as the heterogeneous index α increases. Average utilization of all receivers are calculated and displayed. We also plot the maximum and minimum utilization among receivers.

The worst resuls mostly came from the highest bandwidth link (l_6) and the result of the worst utilizatoin degrades as heteroneneity increases, where the bandwidth of the highest link becomes broader. For Rainbow, the reason of this poor utilization is considered that heterogeneous link bandwidth causes many reordered packets. Reordering occurs because some TRQs are aggregated at an intervening node and reply data packets for other receivers are copied and sent to the link. but ohther TRQs have to go up all through the tree to the source and then reply packets are sent. This reordering makes receivers misunderstand that packet loss occurs and fail to extend window size large enough for high bandwidth links.

In case of RLC, it does not have slow start phase (exponential increase) and synchronized points when join experiments are executed for higher layer become infrequent. This causes slow adaptation for higher bandwidth links. For example, appropriate layer for 4 Mbps link is 7th or 8th layer and join experiments for 7th layer are executed every 32 seconds based on parameters that are used for this simulation.

We expected the utilization of RLC would become worse as the heterogeniety increases, because prepared layers limit the adaptation granularity. But in terms of average link utilization for all links, RLC seems realizing more settled link utilization than Rainbow. However, it is because RLC sends packets aggressively to low bit rate link at the cost of packet loss. Figure 11 (b) shows the loss rate of the link which suffers from packet loss most. The lowest bandwidth link tends to suffer from packet loss most and also record the best utilization. According to the graph, as α increases and the lowest bandwidth link (l_{-6}) becomes narrower, the best link utilization is going up but the loss rate increases. This high loss rate is unacceptable considering coexistence with other traffic. According to this result, in order to lower loss rate, synchronized points for RLC should be extended, but it causes slower adaptation to link bandwidth and degrade link utilization especally for high bandwidth case.

B.5 Scenario B-(iii)

According to the results above, we expect RLC's behavior is not TCP-friendly especially for low bit rate link. We study how a link is shared between TCP and Rainbow (or RLC) flows for different bandwidths. Figure 12 (a) shows the fairness metric (3). Figure 12 (b) shows the ratio of average receiving rate of Rainbow (or RLC) and TCP. A value of more than 1.0 means Rainbow's (or RLC's) average rate is over TCP's average rate. Packet loss rate at the shared link is shown in Figure 12 (c).

According to the fairness metric Figure 12 (a), Rainbow behaves fairly against TCP traffic, but RLC gets unfair bandwidth share except of the case of 256 Kbps. Figure 12 (b) shows that this unfairness comes from the fact that RLC get too much bandwidth share in case of 128 Kbps, but less bandwidth share for 512 Kbps. As Figure 12 (c) shows, the loss rate of RLC for low bit rate link also indicates RLC's aggressive behavior when sharing a single queue of the shared link with a TCP flow. On the other hand, RLC fails to get enough bandwidth share in case of 512 Kbps because of RLC's infrequent adaptation interval for higher layers.

B.6 Scenario B-(iv)

In scenario B-(iv), we investigate link sharing among multiple Rainbow (or RLC) and TCP flows. The bandwidth of the shared link is fixed but the number of sessions varies. As a result, appropriate bandwidth share for each sesion is similar to the former scenario B-(iii). Figure 13 (a),(b),(c) shows fairness metric, ratio of average receiving rate, and packet loss rate respectively. Note that to compare with Figure 12 x-axis is reverse in terms of shared bandwidth.

The tendency of transition of fairness is similar to scenario B-iii. Rainbow shows more settled fairness change for different number of sessions. When the number of session is four and bandwidth share is broader, RLC cannot get enogh bandwidth share and fairness metric is worse. As the number of sessions increases and bandwidth share decreases, RLC gets more than its fair share of the bandwidth. However degradation of fairness for RLC is less than scenario B-(iii) as appropriate bandwidth share decreases. The reason is considered that when number of sessions increase, the packet loss pattern seems random from each session and behave equivalently to the model described in [29].

V. Conclusion

In this paper, we have presented an alternative multicast service model, *BCFS*, and congestion control for reliable multicast, *Rainbow*, built on top of the BCFS service model.

As future work, we plan to tackle practical implementation issues for BCFS in terms of a router's required memory and processing load, and describe more detailed protocol specification. Other efficient application examples on BCFS will enforce the significance of the service model. As for Rainbow, its scalability will be clarified by large-scale simulation and mathematical analysis about data aggregation mechanism.

We believe BCFS provides a new direction for multicast forwarding service. By factoring PGM into a reusable network component that is modeled after EXPRESS, we have created a network service that is not only a good building block for PGM, but also provides a foundation for new transport protocols like Rainbow.

References

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, Nov. 1984.

[2] S. Deering, *Multicast Routing in a Datagram Internetwork*, Ph.D. thesis, Stanford University, Dec. 1991.

[3] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *IEEE/ACM Transactions on Networking*, 1995.

[4] R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," in *Proceedings of ACM Multimedia '95*, San Francisco, CA, Nov. 1995, ACM.

[5] B. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves, "The Case for Concurrent Reliable Multicasting Using Shared Ack Trees," in *Proceedings of ACM Multimedia*, Boston, MA, Nov. 1996, ACM.

[6] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*, Jun 1998, RFC-2362.

[7] T. Ballardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing," in *Proceedings of Sigcomm '93*, San Francisco, CA, Sept. 1993, ACM, pp. 85–95.

[8] K. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP Architecture for Inter-Domain Multicast Routing," in *Proceedings of Sigcomm '98*, Vancouver, Canada, September 1998.

[9] A. Mankin, A. Romanow, S. Bradner, and V. Paxson, *IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols*, Jun 1998, RFC-2357.

[10] H. Holbrook and D. Cheriton, "IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications," in *Proceedings of Sigcomm '99*, Cambridge, MA, September 1999.

[11] R. Perlman, C-Y Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green, "Simple Multicast: A Design for Simple, Low-Overhead Multicast," Feb. 1999, Internet Draft (Work in Progress).

[12] J. Lin and S. Paul, "RMTP: A Reliable Multicast Transport Protocol," in *Proceedings IEEE Infocom '96*, San Francisco, CA, Mar. 1996, pp. 1414–1424.

[13] H. Holbrook, S. Singhal, and D. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," in *Proceedings of Sigcomm '95*, Boston, MA, Sept. 1995, ACM.

[14] Y. Chawathe, S. Fink, S. McCanne, and E. Brewer, "A Proxy Architecture for Reliable Multicast in Heterogeneous Environments," in *Proceedings of ACM Multimedia*, Bristol, England, September 1998.

[15] C. Papadopoulos, G. Parulkar, and G. Varghese, "An Error Control Scheme for Large-Scale Multicast Applications," in *Proceedings IEEE Infocom '98*, San Francisco, CA, March 1998.

[16] B. Levine, , and J.J. Garcia-Luna-Aceves, “Improving Internet Multicast with Routing Labels,” in *Proceedings of IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[17] A. Costello and S. McCanne, “Search Party: Using Randomcast for Reliable Multicast with Local Recovery,” in *Proceedings of IEEE Infocom ’99*, New York, NY, March 1999.

[18] T. Speakman, D. Farinacci, S. Lin, and A. Tweedly, “PGM Reliable Transport Protocol Specification,” Aug. 1998, Internet Draft (Work in Progress).

[19] S. Kasera, J. Kurose, and D. Towsley, “Scalable Reliable Multicast Using Multiple Multicast Groups,” in *Proceedings of ACM Sigmetrics Conference*. ACM, June 1997.

[20] V. Jacobson, “Congestion avoidance and control,” in *Proceedings of Sigcomm ’88*, Stanford, CA, Aug. 1988.

[21] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A Digital Fountain Approach to Reliable Distribution of Bulk Data,” in *Proceedings of Sigcomm ’98*, Vancouver, Canada, September 1998.

[22] R. Braden, *T/TCP – TCP Extensions for Transactions Functional Specification*, Jul 1994, RFC-1644.

[23] J. Nonnenmacher, E. Biersack, and D. Towsley, “Parity-Based Loss Recovery for Reliable Multicast Transmission,” in *Proceedings of Sigcomm ’97*, Cannes, France, Sep 1997, ACM.

[24] S. Raman and S. McCanne, “A Model, Analysis, and Protocol Framework for Soft State-based Communication,” in *Proceedings of Sigcomm ’99*, Cambridge, MA, September 1999.

[25] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson, “Scalable timers for soft state protocols,” in *Proceedings IEEE Infocom ’97*, Kobe, Japan, Apr. 1997.

[26] S. Bhattacharyya, D. Towsley, and J. Kurose, “The Loss Path Multiplicity Problem for Multicast Congestion Control,” in *Proceedings of IEEE Infocom ’99*, New York, NY, March 1999.

[27] I. Rhee, N. Ballaguru, and G. Rouskas, “MTCP: Scalable TCP-like Congestion Control for Reliable Multicast,” in *Proceedings of IEEE Infocom ’99*, New York, NY, March 1999.

[28] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven layered multicast,” in *Proceedings of Sigcomm ’96*, Stanford, CA, Aug. 1996, ACM.

[29] L. Vicisano, L. Rizzo, and J. Crowcroft, “TCP-like congestion control for layered multicast data transfer,” in *Proceedings of Infocom ’98*, San Francisco, CA, March 1998.

[30] D. Rubenstein, J. Kurose, and D. Towsley, “The Impact of Multicast Layering on Network Fairness,” in *Proceedings of Sigcomm ’99*, Cambridge, MA, September 1999.

[31] “UCB/LBNL/VINT Network Simulator - ns (version 2),” http://www-mash.cs.berkeley.edu/ns/.

[32] R. Jain, A. Durresi, and G. Babic, “Throughput Fairness Index: An Explanation,” Feb. 1999, ATM Forum/99-0045.

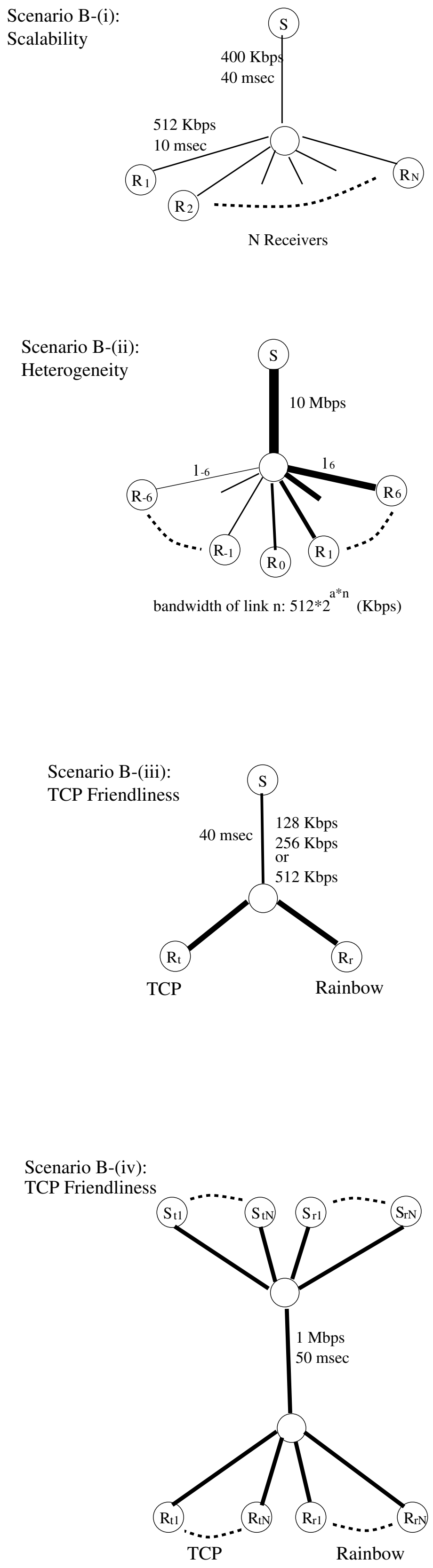
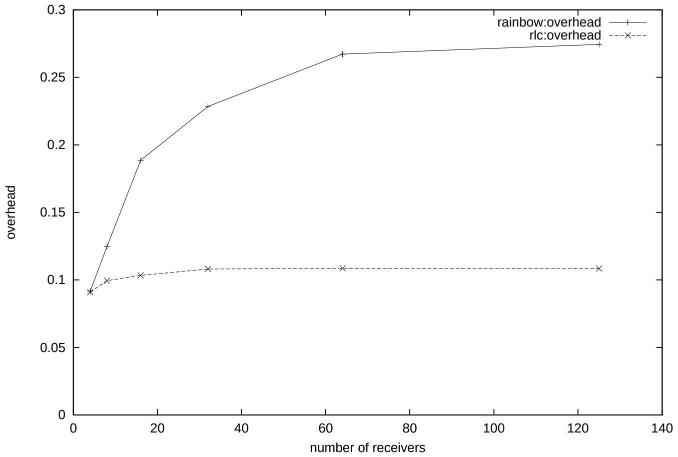
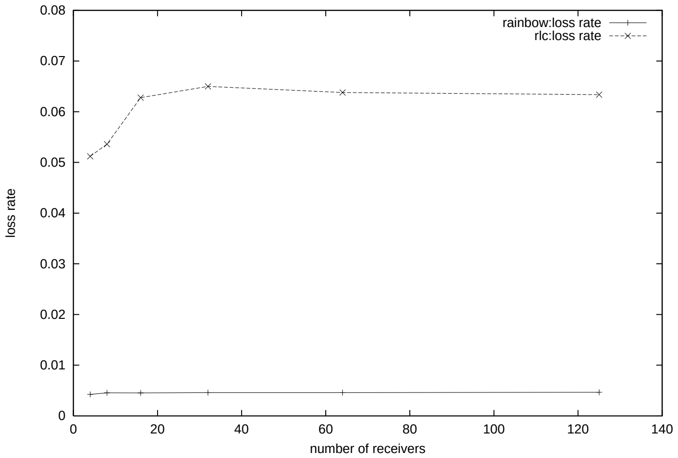


Fig. 8. Simulation Settings.



(a)



(b)

Fig. 9. Simulation Result (scenario B-(i)).

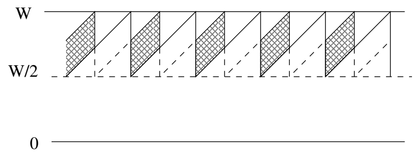
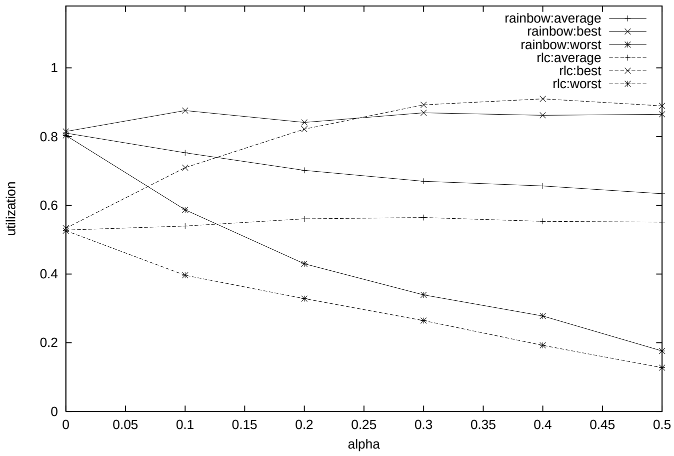
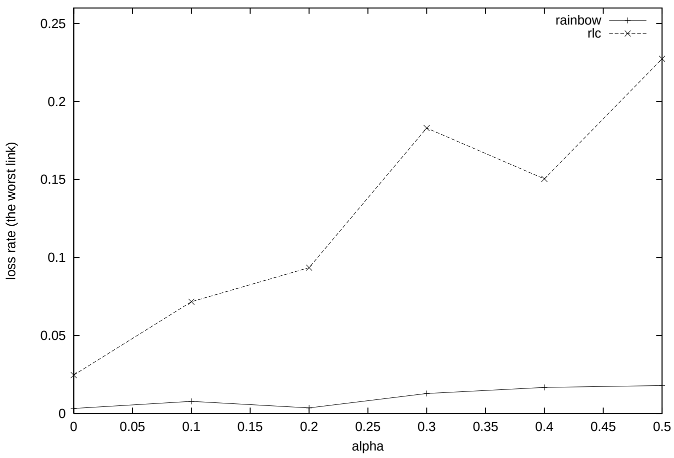


Fig. 10. Overhead by two individual window controls.



(a)



(b)

Fig. 11. Simulation Result (scenario B-(ii)).

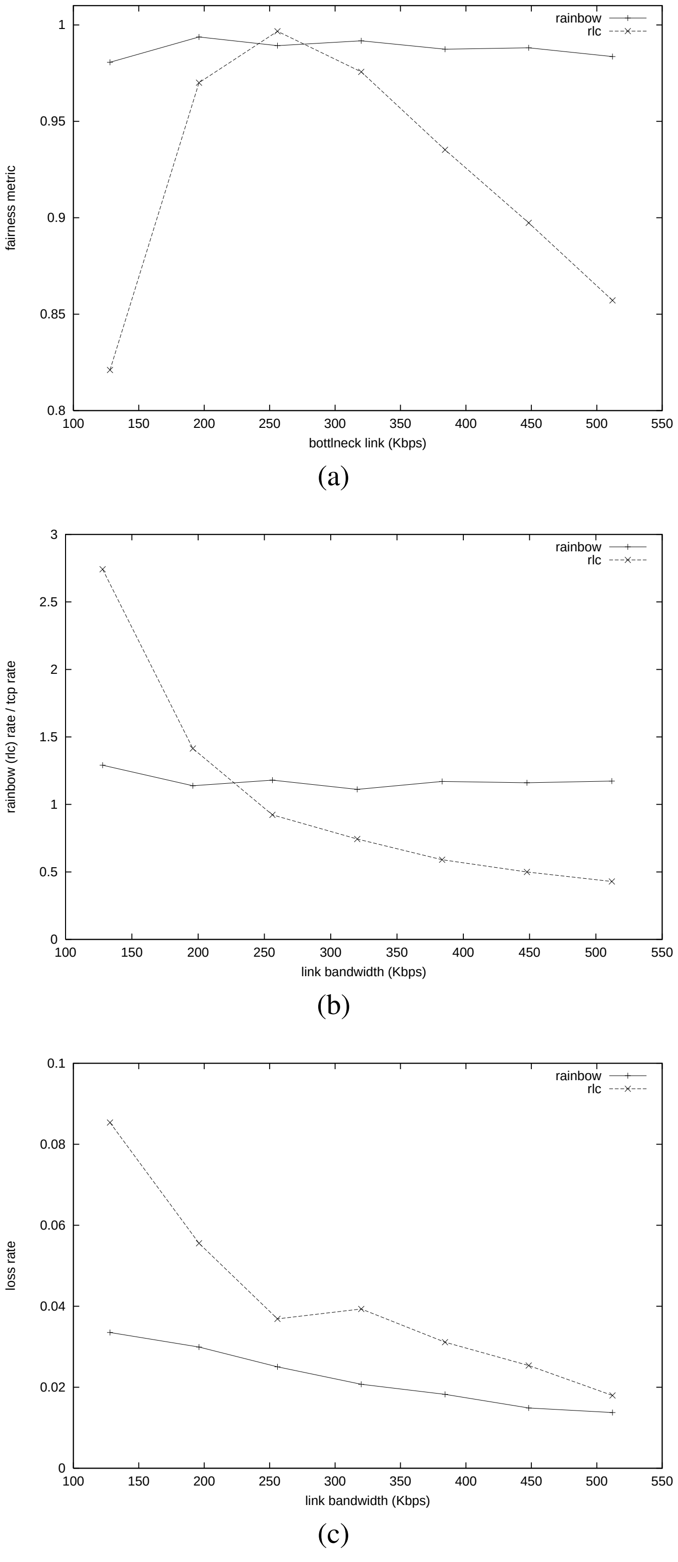


Fig. 12. Simulation Result (scenario B-(iii)).

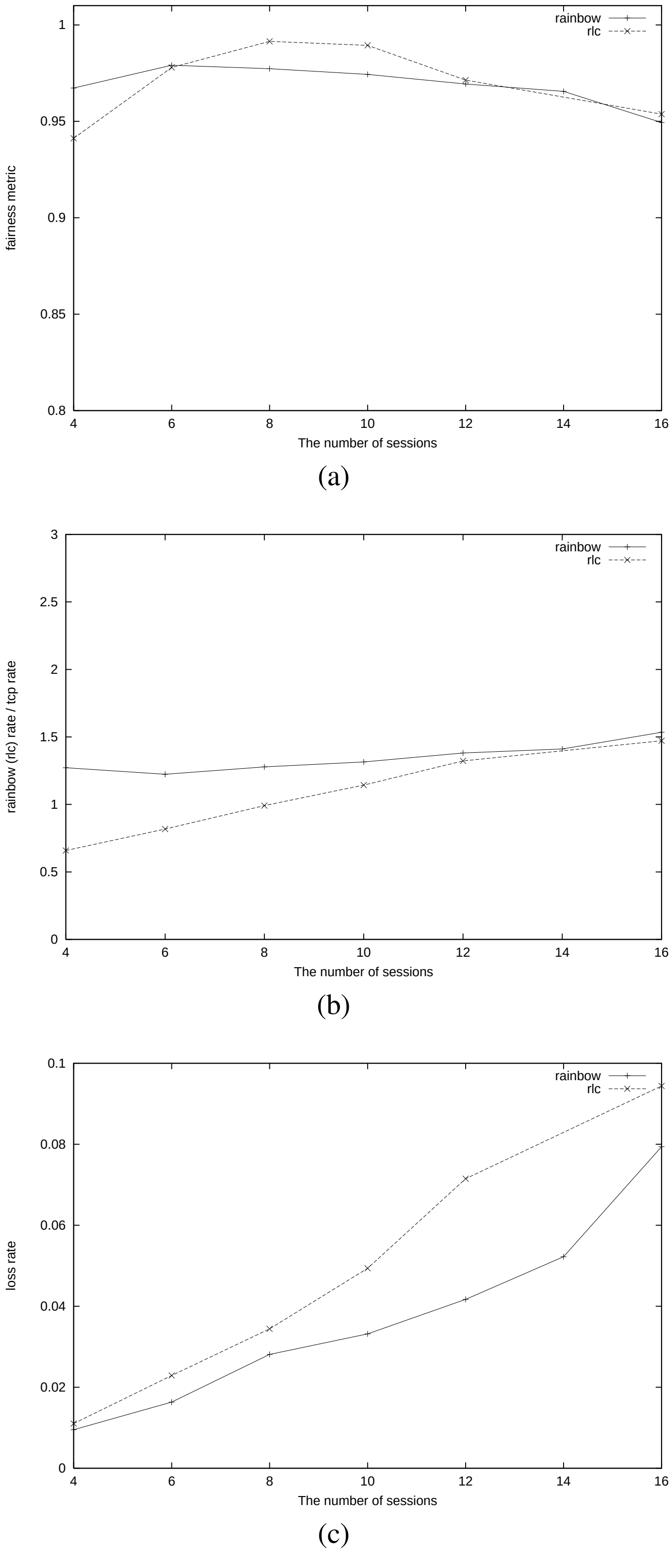


Fig. 13. Simulation Result (scenario B-(iv)).