Concast: Design and Implementation of a New Network Service*

Kenneth L. Calvert, James Griffioen, Amit Sehgal, and Su Wen
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046

Abstract

This paper introduces concast*, a new network service. Concast is the inverse of multicast: multiple sources send messages toward the same destination, which results in a single message being delivered to the destination. The received message appears to come from the concast group rather than any particular receiver. Different forms of concast service can be defined by varying the mapping from the set of sent messages to the received message. The service is useful for preventing implosion and reducing bandwidth consumption in cases where many senders transmit to the same receiver—for example in aggregating (or suppressing) positive (or negative) acknowledgements*

We define the semantics of a simple concast service that is the inverse of multicast, as well as a more general custom concast, which allows users to define certain aspects of the service's semantics. We describe how to implement the service so that it scales approximately as well as IP multicast. We also present results from a simulation study showing that concast provides significant benefits in a layered-video application.

1. Introduction

Multicast service is now a reality in many parts of the Internet. The multicast abstraction uses a single network address to represent a group of receivers. When a host sends a packet to a multicast group address, the network makes its best effort to deliver a copy to all receivers in the group. The sender need not know the identities of the receivers, and indeed cannot learn them via the multicast service itself. Internet multicast is a scalable service because it hides the number and identity of receivers behind a single address and allows a sender to communicate with any number of receivers as a single entity. Efficient multicast service implementations duplicate packets in the network as needed, thereby reducing bandwidth consumption (as compared to unicast implementations).

An increasing number of applications now require a communication service in which a receiver collects messages from many senders. This raises the question of whether a scalable service similar to multicast can be defined for messages flowing in the opposite direction—essentially an "inverse multicast" service. In this paper, we propose and describe such a service, which we call *concast*. With concast, a single network address represents a group of *senders*. When multiple group members send packets addressed to a single destination, only a single packet is delivered to that destination (Figure 1). Where a multicast datagram has a unicast source address and a group destination address, a concast datagram contains a group source address and a unicast destination address. Thus, concast too is scalable: it abstracts away the reality of multiple senders and allows a receiver to avoid *implosion*—that is, processing a number of incoming packets that grows with the size of the group. Like multicast, a good implementation will also conserve bandwidth by reducing or eliminating redundant transmissions.

However, the precise definition of an "inverse multicast" service is non-obvious. In particular, two interesting questions arise. First, *what* packet is delivered to the receiver as a result of the multiple senders' transmissions? Second, *when* is this single packet delivered to the receiver? We refer to the answers to these questions as the *merge semantics* and the *timing semantics*, respectively. Various merge and timing semantics are possible, and give rise to different

*Effort sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-1-0514. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force Research Laboratory, or the U.S. Government.

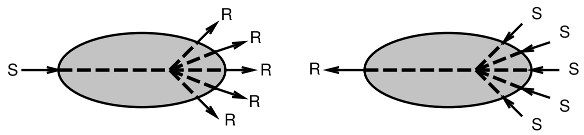

Figure 1. Multicast (left) and concast services.

forms of concast service.

In what follows we present two concast service models. Like multicast, both are best-effort services. The first, *simple concast*, provides generic, application-independent *merge semantics* that are the exact opposite of multicast: the network "fuses" identical packets from different senders into one copy that is delivered to the receiver. It also provides application-independent *timing semantics*: delivery of the "fused" packet occurs as if it were the *first* transmitted copy. Observe that this simple service can be used to implement *nack suppression*, which is useful in implementing reliable multicast: the network forwards the first nack and discards all other nacks for the same message.

Our second model is called *custom concast*. This form allows the user of the service to choose the merge and timing semantics, either by selecting them from a predefined set, or by installing code in some form. Custom concast is thus an excellent match for *active networks* [5].

As a trivial example of how a concast service might be customized, consider alternatives for the timing semantics. One possibility is that delivery to the receiver is triggered by the *last* send. The definition of "last" is problematic here, especially in view of the best-effort nature of the service. However, it is easy to envision applications desiring delivery of a message as soon as a certain threshold number of sends has occurred; the receiver could specify this threshold via signaling when subscribing to the service. Another possibility is to transmit a merged message at fixed time intervals.

The rest of this paper is organized as follows. Section 2 describes some classes of applications that could benefit from a concast service; in Section 3 we discuss prior work related to concast. Section 4 reviews the characteristics of the Internet multicast service. Section 5 describes the simple and custom concast service semantics, while Section 6 describes how the service can be implemented by placing concast-specific capabilities in routers. Section 7 presents results of a simulation study involving a particular multimedia distance-learning application. Section 8 offers conclusions and some directions for future work.

2. Uses for Concast

Concast services can benefit a growing number of applications involving group communication, including teleconferencing, virtual offices, shared whiteboards, chat rooms, application sharing, and multi-player games; information dissemination applications such as stock price updates, network news feeds, mailing lists, and webcasting. Historically these types of applications also benefit from multicast, but have had to achieve concast style services via N independent unicast channels or hierarchy of unicast channels. Consider the following broad classes of applications which can be expected to benefit from concast services:

Group-request-Group-reply: In these applications, a single machine issues a request to a group of machines (typically via multicast) and then waits for a response from all group members. This model of interaction occurs both at the network-level and application-level. For example, reliable multicast protocols [12, 22, 13, 20] can use concast to provide ACK/NACK supression. Similarly, many distributed applications require end-to-end (application-level) ACK messages to ensure reliability [15]. For example, a multicast file transfer program (even when implemented using a reliable multicast protocol) may require application-level end-to-end ACKs to ensure the data has been stored.

Report-in: These applications arise in systems of distributed sensors, robots, or other devices that periodically transmit data to a centralized monitoring or control system. For example, cameras may transmit signals used for security/tracking purposes, manufacturing quality control systems, stereo reconstruction of 3D objects, etc. In many cases, the signals only become meaningful when they are combined and processed. Concast's many-to-one communication model is ideal for such processes.

Client-Multiserver and **Multiclient-Server**: To ensure reliability or availability, *client-multiserver* applications replicate server functionality across multiple machines. Client requests are multicasted to the servers who send simultaneous responses to the client. The client often needs only one, or K out of N, responses. For example, a distributed database may require K out of N ACKs when storing replicas of a newly written record. We use the term *multiclient-server* to represent the classical client-server model, in which a single server responds to request from any number of client machines. Concast offers benefits whenever concurrent requests are similar or even the same (e.g., URL request to a web server).

Collaborative: Real-time conferencing and application sharing can benefit from concast communication when they require synchronization (e.g., floor control) and/or resolution of inputs into a single stream or program. Several systems already provide audio mixers that merge the voice streams of conference call participants. Similarly, shared control of whiteboards or other shared applications often requires merging and mixing user inputs to provide floor control.

3. Related Work

As noted above, concast communication patterns arise in a variety of application domains, including reliable multicast, where receivers must all transmit ack/nack messages to the sender [22, 9, 13, 12].

Multimedia gateways exemplify the application-level approach to providing concast-style communication services. Such gateways perform mixing and transcoding on multimedia flows to match the receive capabilities of specific audio/video conference participants [1, 2, 3]. They must be explicitly deployed as application programs running on strategically-placed hosts in the network. On the other hand, by placing functionality "in" the network layer, concast frees both senders and receivers from having to determine the location of the gateway functionality—it is automatically deployed in the "correct" location.

Other researchers have identified the problem of small-packet overhead and some have suggested that small packets be combined into larger packets at routers in the network. One example is the *gathercast* approach [4]: an aggregation mechanism based on transformer tunnels [17]. The basic idea is to delay small packets slightly as they go through a router in hopes that other small packets heading in the same direction will arrive. After a certain time, all such packets are then encapsulated in a single large packet and forwarded toward the destination. Because messages are packed and unpacked transparently *en route*, senders and receivers don't necessarily know that aggregation is occuring. Thus gathercast appears to be a semantics-independent mux/demux mechanism, delivering packets to the receiver as if they were unicast packets; as such, it does not prevent message implosion at the receiving *application* (e.g., web server). Concast, on the other hand, is a *group* communication service and treats the senders as a group. As such it is *not* transparent to the application, which must be designed to take the concast semantics into account.

In previous work [7, 18], two of this paper's authors presented a general-purpose group communication model that can be used to construct special-purpose group communication services. This paper builds on that work, focusing on a general-purpose concast service—a framework for implementing application-specific services as well as semantics-independent services such as gathercast.

4. Review of IP Multicast

In designing the concast services, we have been guided by the goal of maintaining symmetry with IP multicast where it was reasonable to do so. Therefore we present a brief summary of the salient features of Internet multicast (both service and implementation) before defining our concast services.

Multicast datagrams are identified by a multicast IP address in the destination field and a unicast address in the source field. Endpoints (e.g. sockets in Unix) may be bound to a particular multicast address to ensure that they receive only datagrams sent to that group. Any host can send a datagram with a multicast address in the destination field. The multicast service is best-effort: the network tries to deliver the message to all receivers in the group, but makes no guarantees. In particular, some members of the group may receive the message while others do not.

A host "joins" a multicast group by signaling the network using the Internet Group Management Protocol (IGMP) [6]. Specifically, routers periodically poll their local hosts to find out which groups have members on the local subnetwork. If a group has no members on a local subnet, messages addressed to that group are not transmitted on the subnet.

The state information maintained by multicast-capable routers depends on the multicast routing protocol used. In *shortest-path* multicast routing, state is maintained for each (sender, group) pair, and each packet is forwarded based on both its source and destination IP addresses. In *shared-tree* routing, state is maintained for each group, and routing is independent of sender. In each case the state includes a list of interfaces on which packets should be forwarded, plus other protocol-specific information.

5. Concast Service

We now define the semantics of two new internet services: *simple concast* and *custom concast*. Like multicast, both are best-effort services; both implement a form of message merge. Our description is couched in terms of Internet Protocol datagrams. Because we are describing the *service*, we describe the behavior of "the network" as if it were a monolithic "cloud" whose internal structure is hidden. Steps taken by individual routers are described in Section 6. We also assume some part of the IP address space represents *concast group addresses*.[1]

Table 1 highlights the correspondence between multicast and concast with respect to service abstraction and signaling characteristics. In multicast, receivers (group members) signal the network via IGMP in order to receive. Concast

[1] One possibility is to use Class E addresses for concast.

is similar in that signaling is performed by the (single) receiver, which must indicate its desire to receive messages *from* a particular concast group. This causes the network to set up the state necessary for messages to be "merged" on their way to the receiver. In multicast, any host can send to a multicast group without signaling; concast is again similar in that any host can send a datagram (containing any concast group as source address) without signaling first. Alternative definitions of the "merge" operation performed by the network give rise to the two flavors of concast, which we describe next.

IP Multicast	Concast
Receivers signal to receive.	Receiver signals to receive.
Individual receivers are unknown to senders.	Individual senders are unknown to receivers.
Packets have group destination address, unicast source address	Packets have unicast destination address, group source address
Multicast addresses may only be used in the destination field.	Concast addresses may only be used in the source field.
Any host can send to the mulicast group.	Any host can send from the concast group.
Packets are duplicated in the network.	Packets are "merged" in the network.

Table 1. Multicast vs. Concast

5.1. Simple Concast

Simple concast is an application-independent service. Its *merge semantics* are the opposite of "duplication": identical datagrams transmitted by different senders result in the delivery of at most one copy to the receiver. For the purposes of the simple concast service, IP datagrams are considered to be "identical" if they have the same source (concast) address, destination address, protocol number in the IP header, and payload.[2] Thus, to make packets eligible for merging, a sending application must arrange for them to come from the same source port on each sending machine, and ensure that payloads are the same (e.g. they contain the same ack sequence numbers).

Signaling for concast can be achieved by a slight modification to the IGMP. Instead of a multicast group address, the "membership report" message sent by the receiver would contain a concast address. The interpretation of a message containing a concast address by the concast-capable router(s) on the local network would be different from that of messages containing multicast addresses. Instead of indicating a group that the sender wants to join, the message indicates a group from which the sender wants to *receive* messages.

To send a concast datagram, the sending program supplies its local IP implementation with the concast address to be placed in the source address field.[3] The local machine then routes and transmits the packet just like a unicast packet.

To keep track of whether a message has already been delivered, the network must keep track of the messages sent on each active concast *flow*. A flow is identified by a pair (R, G), where R represents the receiving host's IP address, and G represents the concast group address. For each flow, the network keeps a list of (protocol, payload) pairs

When a sender transmits a concast datagram to the network, the network checks to see whether a list of "current" messages already exists for the (R, G) pair. If no such list exists, the network silently discards the datagram, because the receiver has not yet signaled its intention to receive from that group. If a list of messages exists for flow (R, G), the network checks whether the (protocol, payload) pair from the incoming datagram matches an entry already in the list. If not, it adds the pair to the list, and the original datagram is delivered to the receiver. If it is already in the list, the incoming datagram is discarded.

The amount of state maintained by the network for each concast flow needs to be bounded. A straightforward approach is to keep only a hash of the payload, and to place a bound on the number of message list entries kept for each flow; when a flow's list becomes full, new messages for that flow push old entries out, in FIFO order. Clearly a number of other approaches are possible.

Note that multiple receivers may be located on the same host. Messages destined for different receivers on the same host will not be merged, provided they use different higher-level addresses (e.g. UDP port numbers), which will cause their IP payloads to differ. Multiple senders may also be located on the same host, and need not be aware of each other.

5.2. Custom Concast

More sophisticated forms of concast can be defined by specifying other *merge functions*, *timing semantics*, and *packet identification functions* (definitions of "identical" datagrams). Such *custom concast* services provide the opportunity for applications to tailor the service to their specific needs. However, they also raise the issue of control.

[2]Other definitions of "identical" might be useful, e.g. considering only a fixed-length prefix of the payload.

[3]In terms of the "sockets" API, this could be achieved via a socket option that marks the socket so that datagrams sent from it have a particular concast source address. Alternatively, the socket might bind to the concast address, but in that case it cannot be used to receive.

How much can the application customize the service? One option is to provide one or more parameterized services, and allow applications to select a service and its parameters. Another approach is to allow the user much greater freedom in specifying the semantics—say, by providing a description of the merging algorithm function and other aspects of the service. This enables application to define their own form of the service.

In this paper we pursue the latter option and describe a "programmable" custom concast service. In this service the receiver gives the network a *merge specification*, which consists of code defining the following application-specific functions:

getTag(m)**:** a *tag extraction* function returning a hash or key identifying the message. Messages m and m' are eligible for merging iff *getTag*(m) = *getTag*(m').

merge(s, m, f)**:** the function that combines messages together. The first parameter is the current *merge state* (i.e., information representing messages that have already been processed). The second parameter is the new message to merge into the saved state s. The third parameter is a "flow state block" containing information about the concast flow to which m belongs (see Section 6.1).

done(s)**:** the *forwarding predicate* that checks s, the current merge state, and decides whether a message should be constructed (by calling *buildMsg*) and forwarded to the receiver.

buildMsg(s)**:** the *message construction* function, which takes the current merge state, s, and returns the payload to be forwarded toward the receiver.

The custom concast signaling interface must enable the receiver to provide the network with descriptions of these computations. This might be accomplished using Java or another mobile-code language, or a more restrictive special-purpose language could be defined. We do not consider the language further in this paper, except to note that a number of research projects are developing languages for *active networks* [8, 19, 16].

The generic processing applied by the network to a concast packet with source address G and destination address R is shown in Figure 2. The network first retrieves any state associated with the given receiver and group address. If there is none, the packet is quietly dropped. If state exists, then the identifier (tag) of the message being processed is computed using the user code, and any associated message state is retrieved. A new message state block is computed from the old state and the message payload, and the result is stored back with the tag. If the *done* predicate indicates that merging is finished, the *buildMsg* function is called to generate a payload, which is forwarded toward the receiver

```
void ProcessMessage(Receiver R, Group G,
                    IPdatagram m) {
 FlowStateBlock fsb;
 MsgKey k;
 MergeStateBlock *s;

  fsb = lookUpFlow(R,G);
  if (fsb != NULL) {
    t = fsb.getTag(m);
    s = fsb.findMergeState(t);
    s = fsb.merge(s,m,fsb);
    /* replace prev state value */
    fsb.saveMergeState(s,t);
    if (fsb.done(s))
      forwardMsg(R,G,fsb.buildMsg(s));
  }
} /* ProcessMessage */
```

Figure 2. Network per-packet processing.

with the concast source address and the IP address of the receiver for this flow.

6. Implementing Concast

The concast service described in Section 5 can be implemented and deployed incrementally in a manner that is backward-compatible with the present Internet. The service is usable and provides benefits to the application even if only the end systems are concast-capable, although it is clearly most effective in terms of resource conservation when concast nodes are located throughout the network. The components of the implementation are (i) the *soft state* maintained at each router for each concast flow (Sect. 6.1); (ii) the steps carried out at concast-capable routers to implement the service using the soft state (Section 6.2 and Section 6.4); and (iii) the *Concast Signaling Protocol* (CSP), which can be used to implement either simple or custom concast service (Sect. 6.3).

In the following description, "downstream" means toward the receiver, while "upstream" means toward the senders.

6.1. State Required for Forwarding

Each concast-capable node maintains a table of *concast flow state blocks* (FSBs) indexed by pairs (R, G), where G is a concast group address and R is the receiver specification, which is either a socket identifier (a unicast IP address, protocol, and port number) or just a unicast IP address.[4] Each FSB contains the following information:

[4] Concast flows heading for different application programs on the same receiving host can be distinguished either by requiring them to use different concast (i.e. network-level) addresses, or by including higher-level information in R. Space limitations force us to remain agnostic about the question here.

***Concast Flow State*:** The state is one of: DEAD, FORWARDING, PENDING, MERGING.

***Upstream Neighbor List* (UNL):** Each item in the UNL represents a concast-capable node or a local sender (i.e. an application on the same node) for which this node is the next concast-capable hop on the way to R. Each entry in this list contains a node identifier and a Time-to-Live value. For routers, the identifier is an IP address; for local applications, it is a handle for a local signaling channel. The TTL decreases monotonically over time; when it reaches zero, the entry is removed (Section 6.4).

***Merge Specification*:** as described in Section 5.2.

***Per-message State List*:** A list indexed by message tags; each entry contains the state of an in-progress "merge".

Each FSB is created in response to a forwarded concast datagram or signalling message, and is deleted when its UNL becomes empty, i.e. when no nodes upstream of it are sending to the receiver[5]

6.2. Processing and Forwarding

Concast datagrams are defined to be those containing a a concast (group) address in the source address field. Because the normal (fast path) forwarding process in a router may not examine the source address, another mechanism is needed to identify packets for concast processing. The IP *Router Alert* option [10] is intended specifically for the purpose of "flagging" packets for examination by routers to see if they require special hop-by-hop processing. Concast-capable routers, on detecting Router Alert and examining the packet, will recognize the source as a concast address and process the packet accordingly; others will forward it as usual, leaving the option unchanged.

Upon receiving a concast packet for processing, a router first checks for a FSB matching the packet's flow ID (R, G). If none exists, the router creates a FSB, drops the packet, and begins the signaling process described in the next section. (Note that typically this situation will arise only at end-systems, because routers should see a signaling message before any data messages are forwarded through them.)

If a matching FSB is found, the action taken depends on the flow state:

DEAD *state*: Drop the packet. (Receiver has gone away.)

PENDING *state*: Drop the packet.

MERGING *state*: Follow the procedure described in Section 5.2 for packet processing. Note that at the receiving node, "forward toward the receiver" means passing the datagram to a higher-level protocol or directly to the receiving application.

[5]The concast implementation in an end-system (part of the IP layer) considers sending application(s) to be its upstream neighbors.

FORWARDING *state*: Forward the packet unchanged toward the receiver.

Note that the user-specified computations occur off the fast path, and so might be carried out on an auxiliary processor dedicated to the purpose.

6.3. Concast Signaling Protocol

The *Concast Signaling Protocol* (CSP) deals with the establishment of concast-related state in the nodes of the network. All CSP messages are sent as regular IP unicast datagrams with CSP identified in the protocol field, and the Router Alert option included to stimulate hop-by-hop processing. Concast-capable nodes process every CSP packet as described below. Whenever a CSP message is sent downstream, the IP destination address is R (the flow's receiver); CSP messages sent upstream have the IP address of a next upstream router in the tree as destination. The source address of a CSP packet is always the (unicast) IP address of the packet originator, which may be a router.[6]

Each CSP message contains a *flow id* (R, G), where R is a receiver specification and G is the concast group address. In addition, each message contains an indication of the message type and type-specific information. The message types along with their type-specific information are:

Request for Merge Specification (RMS): This message is always sent downstream. It indicates to a downstream node the existence of at least one sender upstream of the source of the packet. RMS messages contain a **Refresh Flag**, which indicates whether message is being sent to keep an entry alive in the downstream UNL. If the flag is cleared, the sender expects the flow's merge specification in response.

Merge Specification (MS): The MS message is sent upstream, in response to an RMS message (with the Refresh Flag clear), and in the case of custom concast, carries the merge specification (Sect. 5.2) for the requested receiver and group. For simple concast, the MS message acks the validity of the concast flow.

Error: An error message may be sent directly to the originator of a RMS message (e.g. if the receiver has not yet defined a merge spec for the concast group), or to all the members of a node's Upstream Neighbor List, to convey error information relating to the concast data flow (e.g. network unreachable). The Error message contains a **Reason Code**, one of: *Merge not defined*, *Receiver dead*, *Receiver unreachable*.

The normal progression of messages in the CSP protocol is illustrated in Figure 3. Initially receiver R indicates to

[6]This is crucial because it enables connectivity problems in the concast tree to be detected via ICMP. (ICMP messages must not be sent in response to packets with concast source addresses.)

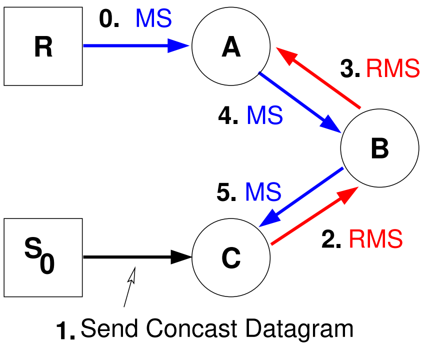

Figure 3. Initial CSP Processing.

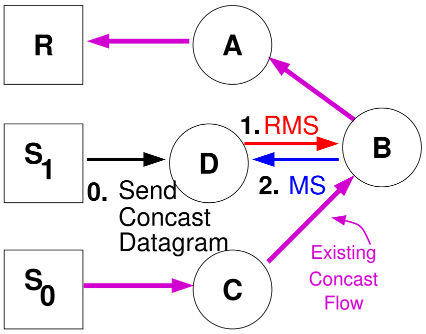

Figure 4. Tree Branching.

the local concast implementation (A) its desire to receive from concast group G. In the case of custom concast, it also provides a merge specification. The local implementation creates the FSB for (R, G) with an empty UNL, and sets the state to FORWARDING.

Subsequently, sender S0 transmits a datagram with source address (concast group) G and destination R. The first concast-capable node between S0 and R (C in the figure), creates a FSB with state PENDING and UNL containing only S0. Node C then creates and sends a Request for Merge Spec message toward R. At the next hop (B), again a FSB is created and another RMS message is sent. When it reaches A, B is added to the UNL for (R, G) and a Merge Specification message is sent back to B, the source of the RMS. When B receives the MS message, it installs the merge specification, sets the flow state to FORWARDING, and send the MS message on to C, which does the same thing. Processing of concast datagrams is now being performed at A, B, and C but there is only one sender, so messages sent by S0 are forwarded unchanged to R.

Figure 4 shows a sequence of messages that result in another branch of the concast flow being added at B. Note that RMS messages propagate only until they encounter the merge specification for the flow. After processing these messages, B's state is MERGING.

The specific actions taken to process each kind of CSP message are given below. In the descriptions, N is the identifier of the node processing the message, S is the source IP address of the CSP datagram, and (R, G) is the flow identifier.

Processing RMS Messages.

1. If no FSB exists for (R, G), create one and initialize its state to PENDING, its UNL to $\langle S \rangle$, and its merge spec to null. Create a RMS message for (R, G) and send it with N as source and R as destination.
2. Otherwise, a FSB already exists for (R, G). Do the following:
   (a) If the flow state is DEAD, create an **Error** message with reason "Receiver dead", send it to S, and discard the received datagram.
   (b) If the Refresh flag is not set *and* the local flow state is not PENDING, then construct a Merge Spec message containing the appropriate information, send it with S as destination and N as source, and continue with next step (c).
   (c) Check whether S is in the flow's Upstream Neighbor List. If S is not in the UNL, add it; if the size of the resulting UNL is 2 and the flow state was FORWARDING, change it to MERGING. If S is already in the UNL, reset its TTL to the initial value.

Processing Merge Spec Messages.

1. Locate the FSB for (R, G). If none exists, silently discard the packet.
2. Otherwise, If the flow state is PENDING, install the merge function using the information in the packet. If the size of the UNL exceeds 1, set the flow state to MERGING, otherwise set it to FORWARDING.

Processing Error Messages. The action depends on the Reason code in the message. If no flow state block exists for the indicated (R, G), the message is silently ignored.

Receiver Dead, Receiver Unreachable: Set the state of the flow to DEAD. Construct and send a CSP Error message with the same reason to each node in the UNL.

No Such Merge Spec: Send a similar message to every node in the UNL, then deallocate the FSB.

6.4. Timer-driven Processing

Periodic housekeeping is performed at each concast-capable router according to two timers. The *FSB Refresh*

Timeout periodically informs the downstream node that it has an upstream neighbor, and ensures that stale local FSBs are eventually flushed. For each local FSB whose state is PENDING, FORWARDING, or MERGING, the Refresh Timeout routine generates and transmits a RMS message toward R. If the state of the FSB is FORWARDING or MERGING, the *Refresh Flag* in this message is set, otherwise it is clear, to indicate that an MS message is expected in response. The Refresh Timeout routine also decrements the TTL of each entry in the FSB's Upstream Node List. If any entry's TTL becomes zero, it is removed from the list and the flow state is changed from MERGING to FORWARDING, or the FSB is deallocated if the resulting size of the UNL is one or zero, respectively.

Clearly the initial UNL TTL value controls how fast the network reclaims resources from inactive branches of a flow tree. A smaller value causes the tree to shrink more quickly, while a larger value provides greater robustness against lost RMS messages.

The *FSB Deallocation Timeout* deals with FSBs in the DEAD state. Such FSBs are kept around for some time to ensure that when the receiver of a flow goes away either intentionally or due to network partition, the information eventually propagates upstream to senders, in spite of lost RMS messages. Each DEAD FSB is either marked or unmarked. If it is unmarked, the handler marks it; if it is marked, the handler deallocates it. Thus a FSB persists for at most twice the period of the Deallocation timer after entering the DEAD state.

Note that this Deallocation processing implies that a flow becomes unusable for up to two Deallocation Timeout intervals after the receiver terminates or becomes unreachable. This means that the effect of brief network partitions on concast is magnified so that their minimum duration is the Deallocation Timer period. Thus there is a tension between ensuring that resources are reclaimed upon termination and ensuring robustness against temporary receiver unreachability.

7. An Example Application

To demonstrate the benefits of custom concast, we simulated a layered video application that uses concast to allocate network bandwidth fairly among competing streams. The scenario involves 16 senders each transmitting a layered video signal [11, 21] to a receiver. Such a scenario could arise in a variety of application domains, including distance learning (video feedback from students to teacher) teleconferencing, security systems, multi-camera TV coverage, roadway monitoring systems, etc. The receiver's objective is to achieve group max-min fairness [14] among competing video streams by regulating the number of layers transmitted by each sender. Ideally, when two or more senders share a common link, link bandwidth will be shared equally by the senders. Unfortunately, without knowledge of the network paths taken by the video streams, it is difficult for the receiver to know the optimal rate for each sender such that the overall objective function is maximized.

We simulated the above application using both a unicast and concast service model. In the unicast model, the receiver was unaware of the network topology, and used a heuristic algorithm based on network measurements taken at the receiver to adjust the senders' transmission rates. In the concast model, the receiver installed a merge function that "thinned" incoming streams to allocate link bandwidth equally among the active senders. We implemented all simulations using the ANSWER active network simulator. The simulator encodes each video frame into layers, packetizes the layer, and transmits the packets. Each sender can transmit up to four layers of constant bit rate video with layer bandwidths of 32, 64, 128, and 256 Kbps respectively. At the receiver, packets are considered decodable (and therefore useful) if every packet that belongs to the same frame and the same or lower layers is received. (The network does not reorder packets.)

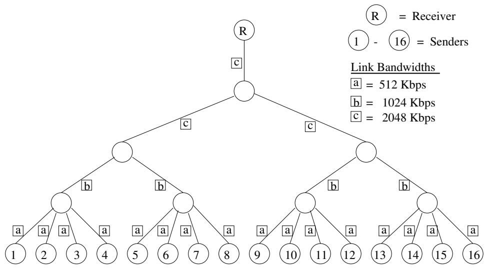

Figure 5. The topology used for the symmetric simulations

Our first experiment employed a "best-case topology" for the unicast approach. Specifically, we used the symmetric topology pictured in Figure 5. Given this topology, the optimal max-min fairness is achieved when all senders transmit at the same rate. In the unicast version, the receiver used an algorithm similar in spirit to RLM [11] and returned the same feedback to all senders, thus maintaining identical transmission rates. The receiver instructed senders to add a layer if the aggregate loss rate, measured over three consecutive RTTs, fell below a lower threshold (1%), and to drop a layer if the loss rate exceeded an upper threshold (15%). In the concast implementation, the receiver installed a merge function that was aware of the outgoing-link bandwidth and thus forwarded an equal number of layers from each sender up to the capacity of the link.

Figure 6 plots the aggregate *goodput* of both the concast and unicast approaches. We define *goodput* as the num-

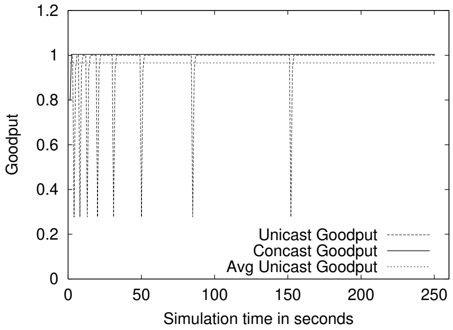

Figure 6. Simulation 1: Instantaneous (one-second) goodput, tree topology.

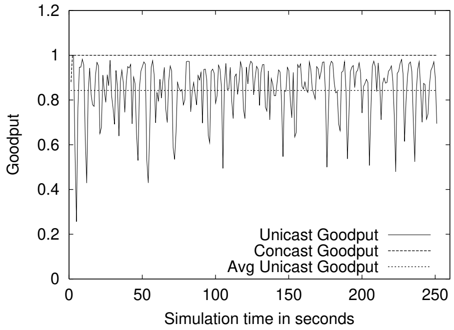

Figure 7. Simulation 2: Instantaneous (one-second) goodput with a random topology.

ber of usable bytes received divided by the total number of bytes received. The RLM-like nature of the algorithm is clearly visible via the exponential backoff where the receiver repeatedly attempts to add one more layer than the network can handle. Given enough time, the unicast algorithm will settle in on the correct transmission level. The concast approach, on the other hand, identifies the optimal transmission level immediately and achieves 100% goodput for the duration of the simulation. This is the best possible scenario for any approach in which senders are treated identically, because their requirements are in fact identical.

In most cases, senders' paths to the receiver have different characteristics, and are unknown to the receiver. In such a case, the best the receiver can do is to provide individual feedback and to adjust each senders' rate independently. In our second simulation, we constructed a random transit-stub graph with 100 nodes from which 1 node was randomly selected as the receiver and 16 other nodes were randomly selected to be senders. We modified the feedback algorithm for the unicast case so that each sender's transmission rate is set based on its current individual loss rate (again using thresholds of 1% and 15%). Thus senders differed in their distance from the receiver and the number of other senders they had to share bottleneck links with. The maximum line-speed of all links was 512 Kbps. Note that this approach clearly does not scale to large numbers of senders.

Figure 7 shows the goodput for concast and unicast in this simulation. Because the unicast approach must experimentally determine the optimal assignment of rates to senders, the algorithm oscillates, continually adding and dropping layers at senders in search of the perfect combination. On the other hand, concast ensures that max-min fairness is achieved and goodput remains at 100%. Figure 8 shows the number of layers received in each frame from one particular sender, located 10 hops from the receiver, for both concast and unicast. The concast curve is constant at 1, while the unicast curve fluctuates between zero and one.

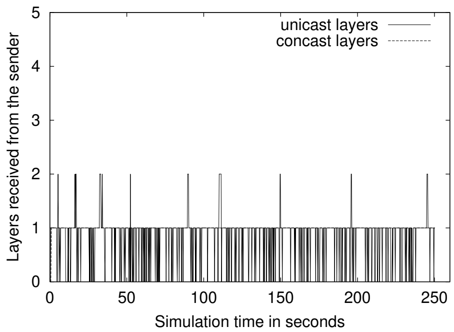

Figure 8. Simulation 2: Layers received from one sender

Other plots, not shown due to space limitations, show substantial variation across senders in the unicast case, with some getting one layer through consistently and thus frequently attempting to add another layer.

8. Conclusions

Internet multicast service employs a scaling mechanism in which a single address represents an arbitrary number of receivers. In this paper we have shown how the same one-for-many mechanism can be applied to senders, to create a *concast* service.

Our *simple concast* service is intentionally symmetric with conventional IP multicast service, and is useful particularly for suppression of duplicate messages, e.g. negative acknowlegements in reliable multicast. The service can be implemented in a manner that scales approximately as well as IP multicast (in terms of per-flow state required in routers), and can be deployed incrementally in the present Internet without changes to existing IP routing mechanisms. We also defined a *custom concast* service, which allows the application to control the semantics of packet merging. We presented a simple and robust signaling protocol to set up and manage flow-specific state and code for both types of service.

We also demonstrated the value of concast in the context of a layered video application involving layered video, where it improved goodput and reduced wasted bandwidth significantly, even for modest numbers of senders, compared to non-scalable unicast-based approaches.

The varying needs of applications with respect to merge semantics and the timing of delivered messages makes it difficult to justify implementing a single merge semantics within the network. However, the ability to place application-specific processing just where it is needed in the network by downloading code makes this a much more attractive proposition. Indeed, concast seems to be an ideal match for active networks, and may motivate the deployment of active network capabilities in the infrastructure.

Acknowledgement: The authors acknowledge with thanks the contribution of Samrat Bhattacharjee, who wrote the ANSWER simulator and Raj Yavatkar who contributed to early versions of the abstraction.

References

[1] The UCL Transcoding Gateway (UTG). http://www-mice.cs.ucl.ac.uk/multimedia/projects/utg/.

[2] E. Amir, S. McCanne, and R. Katz. An Active Service Framework and its application to Realtime Multimedia Transcoding. In *Proceedings of the ACM SIGCOMM '98 Conference*, Sept. 1998.

[3] E. Amir, S. McCanne, and H. Zhang. An Application-Level Video Gateway. In *Proceedings of the ACM Multimedia Conference*, Nov 1995.

[4] B. Badrinath and P. Sudame. Gathercast: The Design and Implementation of a Programmable Aggregation Mechanism for the Internet, April 1999. (submitted for Publication).

[5] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in active networks. *IEEE Communications Magazine*, 36(10):72–78, October 1998.

[6] S. Deering. Host extensions for ip multicasting, August 1989. Internet Request For Comments 1112.

[7] James Griffioen and R. Yavatkar. Clique: A Toolkit for Group Communication Using IP Multicast. In *The Proceedings of the Workshop on Services in Distributed and Networked Environments*, June 1994.

[8] Michael Hicks, Pankaj Kakkar, T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks, 1998.

[9] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceeding of the ACM SIGCOMM'95 Conference*, November 1995.

[10] D. Katz. IP Router Alert Option, February 1997. RFC 2113.

[11] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-Driven Layered Multicast. In *Proceedings of the ACM SIGCOMM '96 Conference*, October 1996.

[12] Katia Obraczka. Multicast Transport Protocols: A Survey and Taxonomy. Technical report, University of Southern California Information Sciences Institute, November 1997.

[13] S. Paul, K. Sabnani, J. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *To Appear in the IEEE Journal on Selected Areas of Communication*, 1996. (see also the Proceedings of IEEE INFOCOM'96).

[14] D. Rubenstein, J. Kurose, and D. Towsley. The Impact of Multicast Layering on Network Fairness. In *Proceedings of the SIGCOMM '99 Conference*, September 1999.

[15] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments In System Design. *ACM Transactions on Computer Systems*, 2:277–288, November 1984.

[16] B. Schwartz, A. Jackson, W. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart Packets for Active Networks. In *1999 IEEE Second Conference on Open Architectures and Network Programming*, pages 90–97, March 1999.

[17] P. Sudame and B. R. Badrinath. Transformer tunnels: A framework for providing route-specific adaptations. In *Proceedings of the USENIX Annual Technical Conference*, pages 191–200, June 1998.

[18] S. Wen, J. Griffioen, and R. Yavatkar. Integrating Concast and Multicast Communication Models. In *Proceedings of the Int. Soc. for Optical Engr: Symposium on Voice, Video, and Data Communications*, November 1998.

[19] D. Wetherall, J. Guttag, and D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[20] B. Whetten, S. Kaplan, and T. Montgomery. A High Performance Totally Ordered Multicast Protocol, August 1994. ftp://research.ivy.nasa.gov/pub/docs/RMP/RMP_dagstuhl.ps.

[21] L. Wu, R. Sharma, and B. Smith. Thinstreams: An Architecture for Multicast Layered Video. In *Proceedings of NOSSDAV '97*, 1997.

[22] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *The Proceedings of the ACM Multimedia '95 Conference*, pages 333–344, November 1995.