# Integrating Concast and Multicast Communication Models

Su Wen[a], James Griffioen[a] and Rajendra Yavatkar[b]

[a]Department of Computer Science, University of Kentucky, Lexington, KY 40506

[b]Communication Architecture Lab, Intel Corporation, Hillsboro, OR 97124

## ABSTRACT

This paper defines a new group communication model called *concast communication.* Being the counterpart to multicast, concast involves multiple senders transmitting to a single receiver. Concast communication is used in a wide range of applications including collaborative applications, report-in style applications, or just end-to-end acknowledgments in a reliable multicast protocol. This paper explores the issues involved in designing concast communication services. We examine various message combination methods including concatenation, compression, and reduction to reduce the traffic loads imposed on the network and packet implosion at the receiver. Group management operations such as group creation/deletion, joining/leaving, and concast routing are discussed. We also address transmission issues such as reliable delivery, flow control, congestion control, and QoS. We conclude the paper by presenting a concast communication model that we have been developing in the context of TMTP[1] (a reliable multicast protocol). The model uses concast communication to implement reliable multicast and it shares concast trees with the multicast group whenever possible to reduce overhead costs.

**Keywords:** Concast, multicast, network protocols, group communication

## 1. INTRODUCTION

The number of network applications that depend on group communication has grown rapidly in recent years. Moreover, this growth is likely to continue in the future. Collaborative applications such as tele-conferencing, virtual-offices, shared whiteboards, chat rooms, application sharing, and multi-player games all involve group communication models. Information dissemination applications such as stock price updates, network news feeds, mailing lists, and webcasting also require communication to a group of receivers. As the world moves toward IP telephony and other video service, conference calls and television services will required group communication support.

Group communication models have been in use for many years.[2–4] A group consists of a set of processes (machines) that need to send or receive data to or from all the other machines in the group. Historically communication systems have supported two group communication models, both loosely referred to as *multicast.* In the first case, a group consists of a single sender and multiple (say $N$) receivers (i.e. $1 \times N$ communication) (Fig. 1). In the second case, a group may consist of multiple (say $M$) senders and multiple (say $N$) receivers (i.e., $M \times N$ communication), where some machines are both senders and receivers. In most cases, $M \times N$ communication is simply implemented as $M$ distinct $1 \times N$ sessions.

This paper describes a new group communication paradigm called *concast* communication. Concast is the counterpart to conventional multicast communication services and is important to a wide range of applications. Specifically, a concast group communication model consists of $N$ senders and a single receiver (i.e. $N \times 1$ communication)(Fig. 2). Like multicast, the basic concast group model can also be extended to an $N \times M$ communication model. In this model, $M$ different $N \times 1$ concast communication sessions exist simultaneously with some machines acting as both senders and receivers. It is worth noting that $M \times N$ multicast and $N \times M$ concast can be combined in a single model where a message is replicated (multicast) at certain network nodes and combined (concast) with other messages at other network nodes.

In this paper we explore the issues involved in the design of efficient concast communication protocols such as group membership, message combination operations, and QoS issues such as reliability, flow control, and congestion control. We conclude by describing how concast communication can be integrated into a reliable multicast communication model, namely TMTP.

Other author information: (Send correspondence to Su Wen)
Su Wen: E-mail: suwen@dcs.uky.edu
James Griffioen: E-mail: griff@dcs.uky.edu
Rajendra Yavatkar: yavatkar@ibeam.intel.com

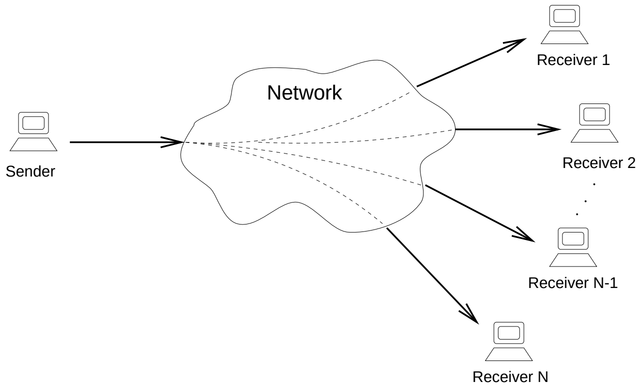


**Figure 1.** Example multicast communication $1 \times N$

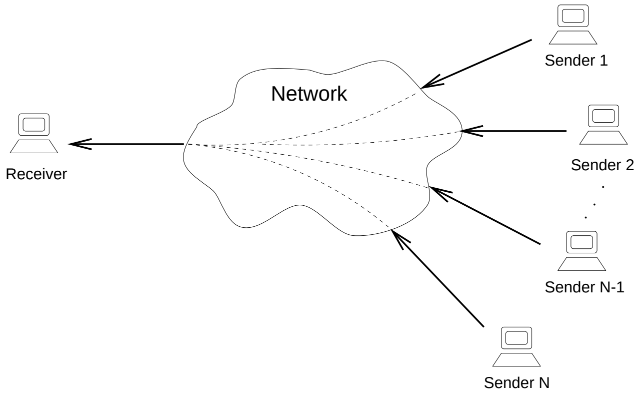


**Figure 2.** Example concast communication $N \times 1$

## 2. MOTIVATION

It is obvious that multicast communication ($1 \times N$ or $M \times N$) should be an integral part of any group communication model. However, it is not as obvious that concast ($N \times 1$ or $N \times M$) is also a valuable component of a group communication model. Therefore, before we explore the issues that must be addressed by a concast communication service, we must first identify (1) applications that might use a concast service, and (2) the benefits of concast as opposed to a unicast communication model.

### 2.1. Who Needs Concast?

One class of applications where concast style communication occurs is the class of *group-request-reply* applications. In group-request-reply 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 (historically implemented via unicast). This model of interaction is used in network transport protocols, and in application-level communication. For example, sender-initiated reliable multicast protocols[5,1,6,7] transmit data to receivers via multicast and then wait for receivers to send ACK messages to the sender. Similarly, application-level concast ACK messages are required by a variety of applications to ensure end-to-end reliability.[10] For example, a multicast file transfer program (even when implemented using a reliable multicast protocol) may require application-level end-to-end ACKs from each of the remote ftp servers to ensure the data has successfully been written to disk. Distributed shared memory systems that multicast changes to shared pages must wait for the recipients to install (not just receive) the page before initiating the communication to unblock processes waiting for the data. Other applications need to gather distributed state information before making a decision. For example, a load-balancing algorithm may request load information from all participants who concast their load information back to the sender. Distributed consensus algorithms may request a vote from all members. Most applications that use multicast communication fall in the group-request-reply model, and are characterized by concast response messages to a single receiver.

Two other classes of application that exhibit concast communication patterns are *client-multiserver* and *multiclient-server* applications. 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 multiple, $K$ 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. In this model, $N$ clients transmit request messages, often simultaneously, to a single server. In many cases, the requests are similar or even the same. Consider a web server at a popular web site. It can service hundreds or thousands of requests each second, often for the same page. Logically these requests can be viewed as concast communication.

Another class of applications that are characterized by concast communication patterns is the *report-in* style applications in which distributed machines, sensors, robots, or other devices periodically transmit data to a centralized control or monitoring system. In some cases, transmission from the concast senders may be continuous such as stereo video feeds sent to a centralized video processing engine that may reconstruct 3D models, extracts depth, perform motion detection, target tracking, etc.

In short, most applications built around group communication require a model that supports both multicast communication and concast communication services. Although a variety of protocols have been developed to support multicast communication,[5] concast communication has typically been left to applications that implement ad-hoc solutions based on multiple unicast channels. Our goal is to explore protocol designs that support the concast communication services required by applications.

### 2.2. Benefits of Concast

Like multicast, concast can be implemented with $N$ (1-to-1) communication channels. However, a concast model that can combine messages from multiple senders at intermediate network nodes offers several advantages. Before describing the potential benefits, it is useful to consider the types of combination that can occur. We broadly categorize combination techniques as either *merge techniques* or *reduction techniques*:

**Merge Techniques:** As the name implies, the class of *merge* operations attempt to merge the data from multiple senders into one (or more) potentially larger packets. Merge techniques themselves can be classified as either *concatenation techniques* or *compression techniques.*

- **Concatenation:** Concatenation is the most basic of the combination operations. Incoming messages are simply concatenated together to form a larger message that is transmitted to the receiver. Concatenation can occur in a "hierarchical fashion" whereby network nodes merge the already-merged messages created by the previous node in the concast tree. In this case, the receiver must hierarchically unpack the merged message. Hierarchical merging may be particularly useful when the hierarchy matches the structure of the processing. Alternatively, concatenation can occur in a "flat fashion", logically unpacking merged messages at each node and remerging them with other unpacked messages. Concatenation is particularly useful when transmitting small packets.
- **Compression:** The basic objective of a compression merging operation is to find redundancy in messages from the senders and eliminate it to reduce the amount of data transmitted to the receiver. This may be as simple as removing the (obviously redundant) destination address from each message and then applying concatenation. Other approaches might include diffing the incoming messages and then transmitting a base message along with a series of diffs, or concatenating messages and then applying a compression algorithm, thereby reducing the time and/or space overhead of compressing individually. Clearly, the receiver must decompress the data to reconstruct the original message sequence. Compression is particularly useful for large data sets, images, or common requests (such as web URL requests).

**Reduction Techniques:** Reduction techniques distribute part of the processing that would normally be done at the receiver to the combining-nodes in the concast tree. The objective is to preprocess the data close to the source. Thus only the processed (thus reduced) data is sent on, thereby reducing the bandwidth consumption and the number of messages transmitted. Because reduction-processing is application-specific, it requires the ability to install application-specific code at intermediate nodes in the network and thus is ideally suited for active network environments. Reduction operations may be as simple as merging ACKs into a single representative (or selective) ACK, or as complex as performing stereo reconstruction from multiple image streams and only transmitting the wiremesh to the receiver. Other reduction operations might make QoS decisions such as forwarding a message early without waiting to combine it with other messages in order to meet delay constraints. We expect reduction functions will be selected from a predefined set of common reduction operations (such as ACK combination) or will be dynamically loaded when application-specific codes are needed.

There are several benefits that result from merging communication flows together inside the network (much like multicast benefits from replicating flows inside the network). Although the benefits depend on the application, the following describes common advantages of the combination operations listed above:

**No Implosion:** *Packet implosion* occurs when a large number of packets must be received and processed by the destination over a short interval.[11] As the number of senders increase, the buffer space requirements at the receiver and the processing load on the receiver increases. This may ultimately result in lost packets at the receiver and an overall drop in throughput. By preprocessing and merging messages at intermediate nodes in the network, a concast communication service can reduce the workload at the receiver.

**Reduced Bandwidth:** Merging messages together near their point of origin (i.e., near the senders), using any of the previously defined methods, can substantially reduce the traffic load imposed on the network. This is particularly important in WAN settings such as the Internet where bandwidth is shared, and thus is a valuable resource.

**Reduced Congestion:** Reducing the bandwidth required lessens the load on the network and reduces the risk of congestion.

**Larger Packets:** Packet switches and routers are often limited more by the number of packets they can switch each second than by the amount of data they can handle. By concatenating or merging multiple small messages into a larger single message, concast may effectively increase the amount of data network routers and switches can process.

**Early Error Detection:** In certain applications, concast can provide early error detection by identifying missing packets at the merge points rather than at the final destination (i.e., the receiver). For example, a merge point that receives multiple streams of data from its nearby senders can often detect missing packets in one of the streams long before the receiver would learn about the lost messages, and can request retransmission much earlier. Merge points might be allowed to issue positive ACKs on behalf of the receiver, effectively increasing the sender's window size (i.e., the amount of dating in transit but not yet ACKed by the receiver) without requiring additional buffer space at the sender.

**Simplified Receiver Code:** Just like multicast protocols simplify the sender code (i.e. use a single send() rather then multiple send() operations), a concast transport service may simplify the receiver code to use a single receive() operation when it's expecting messages from multiple senders. This can also reduce the state information required at the receiver.

### 2.3. Drawbacks to Concast

Unfortunately, the above advantages come at a price. Processing loads on intermediate nodes within the network is increased from the extra processing that needs to be performed at merge points. To alleviate receiver from handling and processing all data in a short amount of time, concast distributes the work across multiple nodes which may already be heavily loaded. Routers are the logical location for combining sender packets, yet they are often most performance critical and have the fewest cycles to spare. Combining messages may force the receiver to demultiplex messages, consuming receiver cycles. As a result, any concast model must be designed to limit the load imposed on any single network element. Our goal is to limit drawbacks so that concast models result in overall improvement, or, in the worst case, perform no worse than unicast implementations. The remainder of this paper explores these and other issues related to the requirements and implementation of a concast protocol.

# 3. CONCAST GROUP MANAGEMENT

A concast group consists of multiple senders, a receiver, and interior nodes in a *concast tree* that combine sender packets as they move toward the root (the receiver). Each interior node in the concast tree must execute a *concast manager (CM)* (Fig. 3). A *CM* performs the desired combination operation on the data received from its children, where a child is either a sender or a CM. The CMs are collectively responsible for concast group membership and maintenance of the concast tree structure. All the CMs in a concast tree must be injected with the appropriate processing code (e.g., similar to active routers in an active network[12]), so CMs can execute application-specific combinations operations. In this paper we assume that all CMs in the concast tree execute the same processing code, although this is not a requirement. The CM code for a concast tree is typically specified when the concast group is created. Ideally CMs execute on network routers and switches to ensure the CM lies on the shortest path from all senders to the receiver, but this is not a requirement.

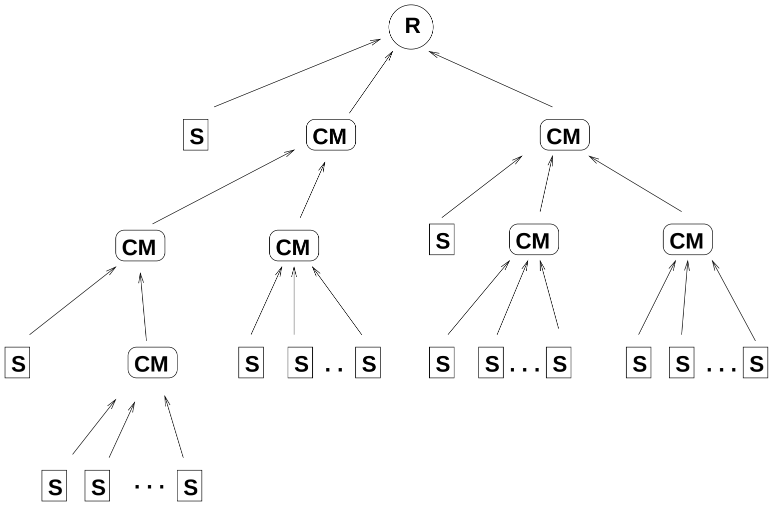


**Figure 3.** Example concast tree (CM = concast manager, S = sender, R = receiver)

Unlike multicast groups where members must join the group to participate, a concast group may be *explicitly* or *implicitly* joined. An *explicit group* consists of collaborating senders that realize they are part of a concast group and thus their messages will be combined by CMs according to the application-specific code for the group. In other words, senders are not part of the concast group unless they execute a group_join() operation. *Implicit groups* consist of senders that are unaware they are part of a group. Senders implicitly join the group simply by sending to the same destination. For example, two or more web clients that are surfing the same or similar pages of a web server might be implicitly merged into a concast group. In this case CMs must dynamically identify group members based on the machine to which packets are destined. It is interesting to note that multicast groups have no counterpart to the implicit concast group. That is, members must explicitly join a multicast group to participate.

Because CMs in the concast tree must be initialized with the desired processing code, *group creation* is a necessary operation in a concast model (whereas it can be omitted in a multicast model assuming group IDs are allocated via some other mechanism). Consequently, the concast group_create() operation must minimally specify the processing code to execute at group CMs. This is true of both explicit and implicit groups. As a result, we assume that all concast groups are created by the receiver and also terminated/deleted by the receiver.

## 3.1. Group Management Issues

A concast group management protocol must address the following issues:

1. Can senders join and leave at random times or is the group size fixed (i.e., all members must join before any data will pass up the concast tree)?
2. Does the receiver need to know how many senders there are in the group? Does it need to know the identity of the senders?

3. Do the senders need to know how many, or the identity of, other senders in the group?
4. Does the application need to be able to specify the concast tree structure? If so, how it is specified, and by whom (the receiver or the senders)?
5. If senders can join/leave at random times, does the tree need to be restructured?

Issues 1-3 are not new to concast and must be addressed by multicast models as well. However, issues 4-5 are unique to concast because the concast tree structure may affect the concast results. For example, if each CM is collecting ACKs, sorting data items, or concatenating messages, the concast tree structure will not affect correctness (although it might affect performance). On the other hand, if the CMs are performing differencing, arithmetic operations (add/subtract/multiply/divide), or stereo reconstruction, the application needs to influence the concast tree formation so that tree structure matches the application's organization, otherwise incorrect result will occur.

### 3.2. Concast Tree Construction

Some applications require a specific concast tree structure, and will define the location of the CMs, grouping of the senders, and the connections between CMs. In this case, constructing the concast tree is straightforward and is simply a matter of arranging the senders and CMs according to the specification. However, in many cases the tree organization is not important to the application and should be decided dynamically by the concast group management mechanism. Even when the application does specify the organization of the tree, it may leave placement of the CMs to the concast group management mechanism.

In such cases, the system must dynamically select a concast tree for the application. The concast tree will be built as senders join the concast group (assuming explicit groups). Many possible concast routing protocols could be used to construct the concast tree (just like there are many multicast routing protocols). However, unlike multicast routing protocols, concast routing usually attempts to perform combination processing at nodes that are near the senders whose messages are being combined. This location may or may not be on the shortest path from the senders to the receiver. Moreover, if the processing function requires a substantial CPU cycles, the routing protocol may avoid (or prohibit) placing CMs on the backbone network, where simple packet duplication (needed by multicast) maybe allowed.

One advantage concast routing protocols have over multicast routing protocols is that concast addresses do not necessarily require a new namespace, distinct from the unicast namespace. In particular, the unicast address of the receiver could be used to identify the concast group. This implies that routers cannot detect the difference between concast and unicast packets from the address alone. As a result, all packets would need to be "processed" to see if they are concast or unicast packets. Note that this addressing model assumes the CM to which a sender attaches is on the shortest (unicast) path from the sender to the receiver. Therefore, the unicast/concast decision only occurs at the first CM. Unicast packets would simply be forwarded, while concast packets would be combined and then tunneled to the parent CM.

However, even if concast addresses require a new namespace, the namespace can be constructed such that a well-known function $f$ can be applied to any concast address to return the unicast address of the receiver.

$$f(concast\_address) = unicast\_address\_of\_the\_concast\_receiver$$

This implies that joining a concast group (i.e., concast routing) may be more efficient than multicast joins because the search for a graft point in the concast tree can be restricted to the path from the sender to the concast receiver.

Reorganizing the concast tree structure is often non-trivial and may mean moving children (and any state information maintained by the processing function) from one CM to another CM. This assumes that the child's state information can be extracted from the current CM and then grafted back in at the new CM. This may not be possible. Even if it is, it may not be possible in a running system. Unlike multicast, a concast messages may not be independent of the other concast messages. For example, consider a system where senders periodically transmit messages that a CM periodically combines and forwards. If a sender is relocated to a new CM, the CM may have already combined and forwarded the most recent period's messages, and thus the relocated message are never forwarded. Consequently, the concast routing protocol should construct the tree in such a way that there will not be a need to reorganize later if members leave or join.

In the following, we propose methods for joining a concast group, assuming the tree structure is not restricted by the application.

**Nearest Neighbor:** The *nearest neighbor* method attempts to identify and attach to the point in the current concast tree that is the closest to the joiner. A new sender initiates the join process by contacting its local CM (on the local LAN's default router). The CM then searches for the closest CM in the concast tree by invoking an *expanding ring search.*[1] Join messages are multicast with a small time-to-live (TTL) value. If no CMs respond, the TTL value is increased and the message remulticasted. The first, and thus nearest, CM to respond is chosen as the attachment point.

The nearest-neighbor method has the advantage of CMs combining messages from nearby senders or CMs, thus achieves the goal of processing data near the source. It is also simple and requires no state information at routers that are not part of the concast tree. However, it is highly dependent on the order in which senders join the tree and can result in routes to the sender that are far from optimal. It does not require CM processing at any of the backbone routers which is good if the processing is computationally expensive. However, it may mean that some traffic traverses the backbone two or more times; once to get to the CM on a remote LAN, and then to get from the CM to the receiver (or other CMs).

**Shortest Path Routing:** The *shortest path* method makes use of the fact that the receiver and the shortest path to the receiver are known (or can be easily discovered). Given the list of routers on the shortest path to the receiver, the joining sender queries each router about its involvement in the concast tree beginning with the nearest router. If the router is not part of the concast group, the CM on the router adds itself to the concast group. Each router is queried in turn until a router that is in the concast group is found. This router becomes the attachment point and the search terminates. Note that all messages are unicast. Only routers between the sender and the receiver are bothered by join requests, and the final routes will be the shortest paths. On the other hand, to ensure that messages are combined as early as possible, every router in the network must execute a CM so it can handle concast group_join() queries.

There are two ways to transmit join messages in the direction of the receiver. The first method requires the intermediate routers to snoop all unicast messages (a similar technique is used in CBT routing[13]). A new sender joins the tree by sending a unicast join message to the receiver. As the unicast message passes through the routers toward the receiver, each router snoops the message to see if it is a concast join message. If so, it hands the packet to the CM process on the router. If the CM is already a member of the desired concast tree, the join request is not forwarded any further. Otherwise the join request is routed and forwarded just like any unicast packet. The obvious drawback is that the router must examine the contents of all unicast packets.

The second implementation uses an iterative approach to transmit the join request to routers on the shortest path. In this case the sender needes to know all the routers on the shortest path. In the first case, the unicast routing protocol provides the information (e.g., link-state routing). In the second case, the shortest path route must be discovered. For IP networks, algorithms such as those used in the *traceroute* facility can be employed. Traceroute iteratively transmits unicast messages to the receiver with increasing TTL values to elicit ICMP TIME_EXCEEDED responses from each router. As soon as a router is discovered, a join message can be unicast to the CM on the router. When a CM on the concast tree is found, the search terminates.

A concast service should also allow senders to leave the concast group at random times. If shortest path concast routing is used, leaving the group can be implemented simply by propagating the leave request toward the receiver until a CM with remaining members is encountered. However, in nearest neighbor routing, a CM may not be allowed to leave. It is possible that the CM is still the parent of other CMs, even if it has no senders as direct children. If tree reconstruction is supported, then CMs could be allowed to leave the tree.

## 4. COMMUNICATION ISSUES

Although a concast communication model defines the basic communication paradigm, it does not specify the services a concast communication protocol must provide. As a result, a wide range of concast services are possible. Obviously, the services offered depend, to some extent, on the application-specific processing that occurs at the CMs. In other words, the applications have some control over the services provides by the concast framework and can adapt the service to meet their needs. In the following we briefly examine typical services, what they mean in a concast model, and discuss possible implementations.

## 4.1. Message Combination

Logically, messages arrive at a CM, the CM applies the desired processing function to generate data to be sent to the CM's parent node. Unfortunately, messages from the children arrive at different times and thus must be buffered until the processing function is invoked. This raises the obvious question, "When should the CM invoke the processing function" or stated another way "How long will a CM delay a message before invoking the processing function". The answer depends on the input requirements of the processing function or the QoS requirments of the application. In other words, when to invoke the combination function should be controlled by the application.

We envision four message combination modes corresponding for four possible application classes:

1. **All-children:**

   We expect the most popular concast model to be the *all-children* combination model. A CM operating in all-children mode would delay processing until all the children have sent the desired message. This is useful for any processing algorithm that has a fixed number of inputs (i.e. all inputs must be present before the processing can be invoked). ACK processing uses this model.

2. **Any-child:**

   At the opposite extreme, a CM can invoke the processing function as soon as it recieves a message from any of its children. For processing operations that suppress redudant responses from servers (children), any-child mode provides the lowest latency delivery to the receiver.

3. **Periodic:**

   Report-in style applications are typically characterized by senders that send a burst of traffic at regularly scheduled intervals. As a result, the CM should invoke the processing operation with the same period used by the senders. Periodic mode can also be useed by applications whose senders transmit at random times. In this case, the period must be set carefully. If the delay period is set too high, messages will be unnecessarily delayed. If it is set too low, the chance of combining messages is reduced.

4. **Delay-based:**

   A CM might also use a delay factor $d$. It will hold a child's message for at most $d$ units of time and then carry out the operation. The delay factor provides the opportunity for messages from other senders to arrive but yet bounds the one-way latency added by the CMs. When the group is created, the total CM latency from any sender to the receiver can be specified. Given the total latency, each CM can dynamically calculate the appropriate delay based on the distances from the senders to the receiver.

Note that the delayed and period invocation modes could allow early invocation if the conditions for the all-children mode were met before the period or delay expired.

## 4.2. Reliability

Reliablility usually means that a message is guaranteed to be delivered to the receiver. Compression and concatination based concast model are effectively sophisticated multiplexing and demultiplexing techniques and thus the receiver will receive the same packets that were sent by the sender. Reliable concast in this setting implies that all packets sent by the sender are received by the receiver. Several reliability options exist including multicasting a selective ACK, or providing hop-to-hop reliability.

When reduction operations are used, the definition of reliable concast delivery becomes less clear. What does it mean to "reliably deliver" a sender's message if the receiver never actually receives the message. In the simplest definition, reliable delivery means that the sender's message cause a corresponding message to be received by the receiver. However, if the processing operation filters messages or discards redundant messages, the receiver may never actually receive a message, but yet the sender's message was reliably delivery as far as the processing would allow. Thus, a reliable concast protocol would have to clarify the definition of "reliable delivery".

### 4.3. Flow Control

Some applications may need flow control from the concast protocol to avoid over-running the receiver. The concast tree already eases the problem of packet implosion at the receiver, however it does not prohibit data from flowing up the tree and hitting the receiver faster than the receiver can process the incoming data. This is particularly true of trees that simply concatenate packets. To prevent the receiver from being overwhelmed, the group of senders must be requested to reduce their sending rates. A variety of flow control mechanisms are possible. Window-based or credit-based algorithms could be extended to multicast a vector of window-sizes to all the senders. Each vector entry would correspond to the amount of data a particular sender could send. Rate-based mechanisms could be extended to multicast a vector of rate information to all senders. Alternatively, back-pressure down the tree may be sufficient. In this case, the receiver would inform its children of the amount of available buffer space. If the children then became over-run, they could in turn inform their children, until finally the senders are requested to slow their transmission speed. This may be more appropriate for reduction-oriented communication where there is no direct correlation between the receivers available window size and the sender's transmission rate.

### 4.4. Congestion

Concast communication reduces the possibility of congestion in the network by combining messages to reduce bandwidth requirements and the number of packets switched. However, congestion can still occur along a concast path as a result of other nework traffic. In this case, the senders should reduce their sending rate to prevent further packet loss and alleviate congestion on the link. Protocols such as TCP[14] have no access to information within the network, thus can only guess that congestion is occurring. However, because CMs form a routing tree and work together to pass data from senders to the receiver, a CM that detects lost packets (or drops them itself) can communicate congestion information to downstream CMs, requesting that they slow their sending rate. As a result, concast senders will be informed when congestion arises and need not implement hueristic end-system solutions.

### 4.5. QoS

Providing QoS concast service is somewhat ambiguous. That is, it is not obvious what sorts of gurantees a QoS mechanism would provide in a concast setting. Because data is being combined (processed) at each node in the concast tree, the amount of data and its QoS requirments could vary across each link of the concast tree. Moreover, because the application-specific processing is not under the control of the concast service, it may not be able to guarantee things such as delay or jitter. There are also the obvious heterogeneous sender issues simliar to the heterogeneous reciever issues addressed by protocols such as RSVP.[15] The following briefly describes two common QoS concerns.

1. **Latency:**

   The timing of data transfer from the sender to the receiver is important to some applications. Naturally, the processing that occurs at each CM will add to the latency. If the processing latency is unknown, no sender-receiver latency guarantees can be made. However, in many cases, the CM processing time will be known (fixed) or insignificant, allowing the system to make guarantees. For all the combination modes except *all-children*, the maximum delay imposed by the CM will be known. Thus a maximum end-to-end delay can be calculated between every sender and the receiver and should not exceed the QoS latency specifications of the application.

2. **Bandwidth:**

   Like protocols such as RSVP,[15] bandwidth requests may be sender-specific. Moreover, because the data is combined and merged as it traverses the concast tree, the bandwidth requirements of each link in the tree depend on the amount of data transmitted by that CM after processing. In other words, bandwidth must be reserved on a link-by-link basis, with each link (possibly) reserving a different amount of bandwidth. Of course, this assumes the network link supports bandwidth reservations (or takes an approach like RSVP). As new senders join/leave, the link reservations may need to change. If the required bandwidth cannot be allocated on any one of the links to the receiver, the sender may be denied admission to the group.

## 5. CONCAST AND TMTP

In our past work, we developed a Tree-based Multicast Transport Protocol (TMTP)[1] that provides reliable dissemination to large number of machines across wide area networks such as the MBONE. We have begun integrating a concast model into TMTP at two levels. First, we employ concast services to implement TMTP's reliable delivery service. Second, we are in the process of adding concast services to TMTP's group communication services.

### 5.1. Using Concast to Implement Reliable Multicast

To achieve scalability, TMTP employs a combination of sender-initiated and receiver-initiated techniques.[11] The receiver-initiated component is designed using a hierarchy of *domain managers (DM)*. Each DM receives multicast messages just like receivers. However, DMs also accept and respond to NACK requests from receivers and child DMs. If a child missed a packet, the DM will remulticast the missed data to the local domain using multicast messages with restricted scope. DMs, like receivers, may miss packets and can issues NACKs to their parent DM.

The sender-initiated component makes use of concast delivery. Receivers periodically concast a positive ACK to the sender. A concast tree of CMs is constructed using the *nearest neighbor* approach described in section 3. Each CM then applies an ACK reduction function that operates in *all-children* mode. When the CM receives an ACK messages from all its children, a single representative ACK is sent to the CM's parent node. In TMTP, the CMs and DMs are combined as distinct threads in the same process so that the DM can access state information in the CM to help optimize its buffer space management.

### 5.2. Adding a Concast Service

Adding generic concast support requires the ability for the application to inject and execute application-specific code into the CMs. Although we plan to move toward an active network implementation, our current design limits the CM code to a predefined set of common operations. For example, TMTP will support an ACK reduction method that applications can use to ensure end-to-end reliability (i.e., the same functionality used to implement reliable multicast at the transport layer). Other common (parameterized) functions such as concatenation, differencing, finding common subsets, and sorting will also be supported. The four message combination modes described in section 4.1 will be supported. The current design does not have any provision for reliable concast, flow control, or QoS beyond that provided by the application-specific CM processing functions.

Concast tree construction assumes there are no message ordering requirements and that senders can join or leave a group at any time. Also, TMTP is implemented at the user-level and requires no support from, or modification to, network routers. Given this environment, the current design uses the nearest neighbor approach to construct the concast tree. However, we take advantage of the fact that many applications require both reliable multicast and concast support. Therefore we allow the concast system to first check if the intended receiver is already a sender for a reliable multicast session to which the local host belongs. If so, a concast tree already exists and can be reused for general concast delivery.

## ACKNOWLEDGMENTS

We wish to thank Babu Yama for his work on concast implementation and management issues. This research was supported by the National Science Foundation under Grant CDA-9502645 and the Defense Advanced Research Projects Agency under Grant DAAH04-96-1-0327.

## REFERENCES

1. R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," in *The Proceedings of the ACM Multimedia '95 Conference*, pp. 333–344, November 1995.
2. D. R. Cheriton and S. E. Deering, "Host Groups: A Multicast Extension for Datagram Internetworks," in *Proceedings of the 9th Data Communications Symposium*, pp. 172–179, ACM/IEEE, September 1985.
3. K. Birman, A. Schiper, and P. Stephenson, "Lightweight Causal and Atomic Group Multicast," *ACM Transactions on Computer Systems* , August 1991.
4. D. R. Cheriton and W. Zwaenepoel, "Distributed process groups in the V kernel," *ACM Transactions on Computer Systems* **3**, pp. 77–107, May 1985.

5. K. Obraczka, "Multicast Transport Protocols: A Survey and Taxonomy," tech. rep., University of Southern California Information Sciences Institute, November 1997.
6. 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).
7. 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.
8. S. Floyd, V. Jacobsen, S. McCanne, C.-G. Liu, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," in *Proceeding of the ACM SIGCOMM'95 Conference*, November 1995.
9. H. Holbrook, S. Singhal, and D. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation," in *Proceeding of the ACM SIGCOMM'95 Conference*, November 1995.
10. J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-To-End Arguments In System Design," *ACM Transactions on Computer Systems* **2**, pp. 277–288, November 1984.
11. S. Pingali, D. Towsley, and J. Kurose, "A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols," in *Proceedings of the ACM SIGMETRICS '94 Conference*, pp. 221–230, 1994.
12. D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *Multimedia Computing and Networking '96*, January 1996.
13. A. Ballardie, "Core Based Trees (CBT) Multicast Routing Architecture," September 1997. RFC 2201.
14. D. Comer, *Internetworking With TCP/IP: Principles, Protocols, and Architecture*, Prentice Hall, Inc., 1988.
15. L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network* , September 1993.