SOMECAST
A Paradigm for Real-Time Adaptive Reliable Multicast

JAEEH YOON
jaehye@cs.bu.edu
AZER BESTAVROS
bestavros@cs.bu.edu
IBRAHIM MATTA
matta@cs.bu.edu

Computer Science Department
Boston University
Boston, MA 02215

Abstract
SomeCast is a novel paradigm for the reliable multicast of real-time data to a large set of receivers. SomeCast is receiver-initiated and thus scalable in the number of receivers, the diverse characteristics of paths between senders and receivers, and the dynamic conditions of such paths. SomeCast enables receivers to dynamically adjust the rate at which they receive multicast information to enable the satisfaction of their QoS constraints. This is done by enabling a receiver to join SOME number of concurrent multiCAST sessions, whereby each session delivers a portion of an encoding of the real-time data. By adjusting the number of such sessions dynamically, client-specific QoS constraints are met independently. The SomeCast paradigm can be thought of as a generalization of the AnyCast and ManyCast paradigms, which have been proposed in the literature to address issues of scalability of UniCast and MultiCast environments, respectively. In this paper we overview the SomeCast paradigm, describe an instance of a SomeCast protocol, and present simulation results that quantify the significant advantages gained from adopting such a protocol.

1. Introduction
The ubiquity of the Internet in our society has encouraged the development of many applications that are inherently of a real-time nature—or that deal with information that is inherently temporal in nature. The communication of real-time (RT) information over the Internet is challenging due to the inherent unpredictability involved in using such an open infrastructure. This unpredictability is documented in a number of studies that confirm the highly variable nature of Internet traffic over a multitude of time scales [9, 10, 11], and the futility of techniques such as buffering to eliminate such variability [25].

Motivation: An important class of RT applications requires the communication of the same content to a very large number of receivers. To cope with the highly-variable nature of network congestion, applications often (a) trade reliability for timeliness, (b) trade timeliness for reliability, or (c) trade resources for both timeliness and reliability.

Trading reliability for timeliness involves the use of a rate-based transport (e.g., using UDP for the communication of audio/video streams as with the MBone multicast protocol). This approach results in the deployment of congestion-insensitive applications, and is viewed by the Internet community as a bad practice [21]. Even if acceptable, such an approach would only be useful for RT Internet applications that are able to tolerate some degree of unreliability (i.e., packet losses). For many RT applications, such unreliability is intolerable. Examples include group simulations, live auctions, RT content replication for Web portals, and stock brokerage applications. Such applications require a multicast infrastructure that is both real-time and reliable [30].

Trading timeliness (i.e., temporal redundancy) for reliability is evident in all communication protocols that use retransmissions to recover from packet losses due to network congestion (including TCP). An example of this approach for multicast communication is the Scalable Reliable Multicast (SRM) [14], the Cyclic UDP Multicast [2], and the Digital Fountain Multicast [6] paradigms. Obviously, such techniques are not adequate for RT applications, for which “a late packet is a lost packet”.

Trading resources (i.e., spatial redundancy) for reliability is common for mission-critical systems that cannot tolerate recovery delays (e.g., using NMR in collision avoidance systems) or for system with irreversible failure modes (e.g., using mirror disks to protect against a disk crash). While a widely accepted practice for hard RT systems, the trading
of resources for reliability and/or timeliness is not common for systems with “softer” or no deadline constraints—over the Internet, for example.

**Paper Overview:** In this paper, we argue that the use of redundant resources to improve the reliability and timeliness of Internet applications in general (and multicast communication in particular) is appropriate due to the already existing multiplicity of resources in such systems—a multiplicity that is required for performance and scalability purposes. To that end, we present SomeCast—a multicast paradigm that enables the satisfaction of timing constraints without sacrificing communication reliability. This is done by enabling receivers to dynamically adjust the rate at which they receive multicast information to guarantee the satisfaction of RT QoS constraints. This rate adjustment is made possible by enabling a receiver to join some number of concurrent multiCAST sessions, whereby each session delivers a portion of an encoding of the RT data. By adjusting the number of such sessions dynamically, client-specific QoS constraints can be met independently. SomeCast is a generalization of the AnyCast and ManyCast paradigms, which address issues of scalability of UniCast and MultiCast environments, respectively. SomeCast is receiver-initiated and thus scalable in the number of receivers, the diverse characteristics of paths between senders and receivers (e.g. RTT and bandwidth), and the dynamic conditions of such paths (e.g. congestion-induced delays and losses).

2. Related Work

**ARQ-based Techniques:** Using (Automatic RepeAT eQuest), the sender retransmits lost data upon request from the receiver. A straightforward application of ARQ in a multicast setting results in the NACK implosion problem, when every receiver sends a NACK message to request retransmission of the same packet. To prevent NACK implosion, the Xpress Transport Protocol (XTP) [31] requires a receiver to multicast control packets to the entire group. A receiver waits for a random time before sending a NACK packet, and refrains from sending a NACK if it sees a NACK from another receiver for the same packet. SRM (Scalable Reliable Multicast) [14] uses similar mechanisms to control the sending of request (NACK) and repair (retransmitted data) packets. As we hinted earlier, retransmission-based approaches trade timeliness for reliability and hence are not useful for the class of multicast applications that require both reliability and timeliness.

**Deadline-Cognizant Techniques:** Most reliable transport protocols (unicast or multicast) are cognizant of the temporal semantics of the data being communicated. Thus, a reliable transport protocol may end up wasting resources attempting to recover from the loss of a packet that is of no value to the receiver (because it is late). Such wasteful resource utilization may result in delaying more packets, resulting in further violations of timeliness constraints. The work of Li, Ha, Varghavan [18] is an example of an approach that attempts to address such a scenario. While this technique was proposed primarily for unicast communication (namely TCP), it is conceivable that similar techniques could be used to avoid unnecessary retransmissions in a multicast environment (e.g. SRM). Deadline cognizance is likely to improve the timeliness of a reliable communication by preserving network resources, but it is unable to mask (or hide) the delays resulting from congestion between a sender and a receiver.

**FEC-based Techniques:** Using FEC, the original data is encoded to obtain additional repair packets that are used to recover from data packet loss. Examples of this approach include the SHARQFEC protocol of Kermode [17], the RT reliable multicast of Rubenstein, Kurose, and Towsley [28] and our recently proposed Adaptive Reliable Multicast (ARM) protocol [32]. FEC-based techniques enable a more efficient/timely recovery from packet losses, but they are not deadline-aware.

**Multi-Layer-based Techniques:** One approach to providing scalable reliable multicast is the use of multiple channels (or layers), whereby receivers experiencing a higher degree of losses join more channels to recover lost packets. An example of this approach is the work of Kasera, Hjalmtýsson, Towsley, and Kurose [16]. While effective in dealing with the variability of loss characteristics across a large set of receivers, this approach does not allow receivers experiencing high loss rates to recover from such losses in a timely fashion. In other words, the reliability of a multicast is guaranteed, but not its timeliness.

**AnyCast-based Techniques:** AnyCasting speeds up access by enabling the selection of “the best” server to fulfill a client’s request. Such selection could be done at the server (e.g. the AnyCast paradigm of Fei, Bhattacharjee, Zegura, and Ammar [12]) or at the client (e.g. the Dynamic Server Selection protocol of Carter and Crovella [8]). While these techniques were proposed primarily for unicast communication, a similar AnyCast paradigm could be
used to select the “best” multicast group for a given receiver (out of many possible alternatives offering the same service) [13]. Given the high variability in network conditions, AnyCast-based techniques are unlikely to be effective for applications that involve a prolonged communication session (e.g. VOD).

**ManyCast-based Techniques:** *ManyCasting* speeds up access to popular content by enabling clients to access replicated sources concurrently. This is exemplified in the work of Byers, Luby, and Mitzenmacher [7], which uses Tornado encoding to ensure the efficiency of the retrieval and reconstruction processes. While not addressing the problem of RT reliable multicast specifically, this work is similar to ours in that it enables a receiver to communicate concurrently with many senders.

### 3. Overview of the SomeCast Paradigm

Under the *(client-based)* SomeCast paradigm, clients are empowered to exploit the multiplicity of resources so as to meet specific reliability and RT QoS constraints. It enables multiple resources available in an inherently best-effort environment to be leveraged to achieve a prescribed QoS. A client (or receiver) contacts “some” providers of service as needed. The delegation of QoS management to clients is attractive because it enables the receivers to have very diverse QoS requirements without resulting in a state-explosion problem at the sender (or network).

**SomeCast Content Delivery Architecture:** Figure 1 illustrates the general architecture of a SomeCast system. We assume that “content” is to be delivered from a *source* to a potentially very large number of *receivers* (or clients) through a number of *senders*, each of which acts as a proxy of the source (i.e. as an outlet for the content).\(^1\) Under SomeCast, each sender sets up a multicast group and a receiver joins as many multicast groups as necessary to satisfy its QoS constraints, in an adaptive fashion.

It is important to emphasize that there are *two* “distribution” problems in the architecture depicted in Figure 1: (1) the source must distribute the content to its proxies (the senders), and (2) the senders must relay that content to the receivers. These two problems are quite different. In the first, the system is “closed” in the sense that the set of proxies are all known *a priori*, and are (most likely) within the confines of a single organization or network (e.g. content replication services on the Internet [1, 29]). In the second, the system is “open” in the sense that the (potentially very large) set of receivers are not known *a priori*; they operate independently and may require significantly different QoS. The SomeCast paradigm addresses the challenges posed by the second of the above two distribution problems.\(^2\)

Another important consideration in the architecture depicted in Figure 1, is the nature of the content being distributed. Two possibilities exist: (1) the content comprises a single object (e.g. current bids) that is updated frequently by the source, or (2) the content comprises a live feed (e.g. prices on the stock exchange) that is constantly generated at the source. The SomeCast paradigm can be used to support both of these models.

Under the first model, senders act as *repeaters*. They continuously and repeatedly multicast the most-up-to-date content on their respective multicast groups. Receivers join as many such multicast groups as necessary to retrieve such content in a timely fashion. The continuous, periodic retransmission of content by senders is similar to the Broadcast Disk techniques proposed in [3], the Digital Fountain techniques proposed in [6], and the Cyclic Best Effort UDP Protocol proposed in [2]. Thus, under this model, content flows in a store-and-forward fashion from the source to the senders and then from the senders to the receivers. Example applications that fit that model would be the multicast of radar information, or the multicast of the status of an on-

\(^1\)This architecture—comprising a large number of servers acting as proxies for a single content source—is quickly being accepted as inevitable for scalable Internet systems [1, 29, 15].

\(^2\)There are many products in the market-place that address the distribution of content from source to proxies [15, 30].

![Figure 1. The SomeCast Paradigm](image-url)
line auction. Under the second model, senders act as relays. They receive a segment of the stream, which they multicast once on their respective multicast groups. Receivers join as many such multicast groups as necessary to be able to recover such segments in a timely fashion (i.e. before the senders switch to the next segment). Thus, under this model, content flows in a pipelined fashion from the source to the senders to the receivers. Example applications that fit that model would be the multicast of live feeds from a stock market exchange, or the multicast of sensory information or live video.

From the perspective of distributing content from the proxies (i.e. senders) to the receivers, the problem is identical under both models. Thus, to simplify our presentation, for the rest of this paper (and without loss of generality), we assume that the first of the above two models is in play.

Reed-Solomon Encoding in SomeCast: In SomeCast, receivers receive the multicast content from multiple senders. Thus, a key component of the SomeCast paradigm is the use of a mechanism that ensures that the various segments of content (received from the various senders) are independent, and thus can be combined efficiently to obtain the original content. To that end, SomeCast assumes that content is encoded using a Reed-Solomon encoding mechanism. Reed-Solomon Codes (RSC) [22] are a popular FEC coding technique, which is used in many FEC-based reliable multicast protocols [17, 28]. RSCs are based on the arithmetic of finite (Galois) fields [27]. RSC-like codes have been proposed and used in a number of projects for efficient information retrieval. Examples include (1) the Information Dispersal Algorithm (IDA) [26] used for efficient, secure, and fault-tolerant parallel data access [20], (2) the Adaptive IDA protocol [4] used in TCP Boston to address the fragmentation of IP over ATM [5], and (3) the Tornado codes [19] used in Digital-Fountain multicast [6].

The SomeCast paradigm is independent of the specific Reed-Solomon coding technique chosen for an implementation. However, to make our presentation concrete—and for purposes of illustration and derivation of specific realizations—we will adopt one such coding technique, namely the Information Dispersal Algorithm (IDA) of Rabin [26]. To understand how IDA works, consider (a segment of) a data object to be multicast. Let that object consist of $K$ blocks (or packets). Using IDA’s dispersal operation, this object could be processed to obtain $s \times K$ blocks, where $s > 1$ is the stretch factor. Recombining any $K$ of these blocks, using IDA’s reconstruction operation, is sufficient to retrieve the original data object. Figure 2 illustrates data dispersal, communication, and reconstruction using IDA.

4. RT Reliable Multicast using SomeCast

In this section, we present an instance of a SomeCast-enabled protocol that empowers a set of receivers to satisfy diverse RT QoS constraints in a reliable multicast setting. Consider the reliable multicast of an object of size $K$ packets. We assume that the object is encoded so that each sender $S_i$ has $u \times K$ packets, where $u = s/S$, $s$ is the stretch factor of the encoding and $S$ is the total number of senders. $S_i$ starts a multicast group over which its packets will be sent only if there is at least one receiver that is a member of its group. At any point in time, if $S_i$ finds that its group is empty (i.e., with no members), $S_i$ stops transmitting packets.

Let $S_0$ be the primary sender. If there is no deadline requirements or the delay bounds are very loose, then a receiver may only join the multicast group of $S_0$ to receive at least $K$ packets and reliably recover the original data by the deadline. Initially, a receiver may join one or more multicast groups associated with one or more senders. Those senders send packets over their multicast groups. By allowing a receiver to join all groups, the protocol can effectively handle stringent delay bounds. For less stringent delay bounds, the receiver may leave all but the first group associated with the primary sender $S_0$. Peri-
Table 1. SomeCast-enabled protocol (Notation)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td>Number of original data packets. Each receiver should receive $K$ packets by the deadline.</td>
</tr>
<tr>
<td>$S$</td>
<td>Total number of senders.</td>
</tr>
<tr>
<td>$S_i$</td>
<td>Sender $S_i$.</td>
</tr>
<tr>
<td>$s$</td>
<td>Stretch factor of the Reed-Solomon-like encoding. The $K$ original data packets are encoded to obtain up to $s \times K$ packets.</td>
</tr>
<tr>
<td>$u \times K$</td>
<td>Maximum number of packets transmitted by a sender, where $u = s/S$. We take $u = 2$.</td>
</tr>
<tr>
<td>$\text{maxseqno}_i$</td>
<td>Maximum sequence # of data to be transmitted by $S_i$.</td>
</tr>
<tr>
<td>$\text{seqno}_i$</td>
<td>Sequence # of packet (probe) sent by/received from $S_i$.</td>
</tr>
<tr>
<td>$\text{PTS}_i$</td>
<td>Number of packets yet-to-be-transmitted at the time $S_i$ sends a probe.</td>
</tr>
<tr>
<td>$\text{RPC}_i$</td>
<td>The Received Packet Counter denotes the number of packets received thus far from $S_i$.</td>
</tr>
<tr>
<td>$\text{TRPC}$</td>
<td>The Total Received Packet Counter denotes the number of packets received thus far.</td>
</tr>
<tr>
<td>$g$</td>
<td>Current number of groups to which a receiver is subscribed.</td>
</tr>
<tr>
<td>$D$</td>
<td>A receiver’s deadline.</td>
</tr>
<tr>
<td>$R_i$</td>
<td>Estimated throughput from $S_i$.</td>
</tr>
<tr>
<td>$L_i$</td>
<td>Estimated loss rate on the path from $S_i$.</td>
</tr>
<tr>
<td>$\text{RTT}$</td>
<td>Maximum Round-Trip Time between the sender and a receiver.</td>
</tr>
<tr>
<td>$\text{expPkt}_i$</td>
<td>Expected number of packets received by the deadline from $S_i$.</td>
</tr>
</tbody>
</table>

our SomeCast-enabled protocol. We describe our SomeCast-enabled protocol by presenting the steps undertaken by the Sender(s) and Receiver(s) at various stages of the protocol.

**Sender:** Start

**SS.1** Each sender $S_i$ ($i = 0, 1, \ldots, S - 1$) sets the initial $\text{seqno}_i$ of its first packet and its $\text{maxseqno}_i$.

$$\text{seqno}_i = 2iK \quad \text{and} \quad \text{maxseqno}_i = 2iK + K - 1$$

**SS.2** If a sender has receivers in its multicast group, it starts to transfer the first $K$ data packets.

**SS.3** Concurrently with step SS.2, the sender applies coding to the first $K$ data packets to obtain $2K$ packets.

**Sender:** Probing

**SP.1** Periodically, a sender transmits a probe piggybacked on a data packet. The probe consists of a timestamp that identifies the time at which the probe is sent and $\text{PTS}_i$. Namely, $\text{PTS}_i = \text{maxseqno}_i - \text{seqno}_i$, where $\text{seqno}_i$ is the sequence number of the packet transmitted with the probe.

**Sender:** NACK Processing

**SN.1** Upon receipt of a NACK, sender $S_i$ updates $\text{maxseqno}_i$ to the maximum requested by all receivers in response to the same probe. This ensures that all receivers subscribed to the multicast group of $S_i$ receive the additional packets they need by their deadlines.

**SN.2** Upon receipt of a NACK, sender $S_i$ also updates its estimate of the maximum Round Trip Time (RTT).

**SN.3** While transmitting the last RTT worth of packets, sender ignores NACK requests for decreasing $\text{maxseqno}_i$ to avoid delays due to possible underestimation of losses in the last stage.

**Sender:** End

**SE.1** Sender $S_i$ stops its transmission once all members (receivers) in its group leave.

**Receiver:** Packet Processing

**RP.1** Whenever a receiver receives a packet from sender $S_i$, it increments the Received Packet Counter ($\text{RPC}_i$). Also, it increments the Total Received Packet Counter ($\text{TRPC}$).

**RP.2** If $\text{TRPC}$ is greater than or equal to $K$, the original data can be reconstructed from the packets received so far. The receiver then decodes the received data and leaves all multicast groups it is a member of.\(^4\) If the

\(^4\)Our protocol may overlap data transmission with encoding/decoding, hence dramatically reducing latency for all receivers to receive the number of packets needed for recovering the original data by the deadline.

\(^5\)By allowing receivers to leave multicast groups once they receive the $K$ packets needed, we significantly reduce the bandwidth consumed over the network.
deadline has expired, the receiver drops all (useless) packets it has received so far and leaves all multicast groups it is a member of.

**RP.3** If TRPC is less than K and if the packet received is a probe, then the receiver proceeds as follows:

**RP.3.1** If \( (\text{seqno}_i + \text{PTS}_i < \text{maxseqno}_i) \) of receiver), then the forthcoming packets from \( S_i \) are not enough to recover the original data. The receiver sends a NACK that includes the new lower bound on maxseqnoi calculated in step RE.4.

**RP.3.2** If \( (\text{seqno}_i + \text{PTS}_i \geq \text{maxseqno}_i) \) of receiver) and \( (\text{seqno}_i + \text{PTS}_i) \) equals a maxseqnoi the receiver had sent to \( S_i \) in a previous NACK, then the receiver was the bottleneck and is now unnecessarily requiring \( S_i \) to send more packets than needed. The receiver sends a NACK that includes the new lower maxseqnoi calculated in step RE.4.

**Receiver: Periodic Estimation**

**RE.1** Periodically, a receiver updates its loss rate estimate \( l_i \) on the path from sender \( S_i \) to which it is subscribed.\(^6\)

\[
l_i = 1 - (\Delta RPC_i / \Delta \text{seqno}_i)
\]

where \( \Delta \text{seqno}_i \) is the difference in sequence numbers of packets received from sender \( S_i \) at the beginning and end of the update time interval. \( \Delta RPC_i \) is the number of packets received from \( S_i \) during the update interval. Thus, the ratio \( l_i \) gives the current proportion of \( S_i \) packets lost.

**RE.2** Periodically, a receiver updates its throughput \( t_i \) from sender \( S_i \), namely \( t_i = \Delta RPC_i / \Delta t_i \), where \( \Delta t \) is the length of the update interval.

**RE.3** Based on \( l_i \) and \( t_i \), a receiver maintains exponential moving averages and deviations of the loss rate and throughput for each sender \( S_i \). Specifically,

\[
\text{AvgR}_i = \alpha \text{AvgR}_i + (1 - \alpha) t_i
\]
\[
\text{DevR}_i = (1 - \delta) |t_i - \text{AvgR}_i| + \delta \text{DevR}_i
\]
\[
R_i = \text{AvgR}_i + \gamma \text{DevR}_i
\]

where \( R_i \) is the estimated throughput from sender \( S_i \), \( \text{AvgR}_i \) and \( \text{DevR}_i \) are the moving average and deviation, respectively.\(^7\) Similarly, the loss rate \( L_i \) from sender \( S_i \) is estimated.

**RE.4** Compute maxseqnoi for each sender \( S_i \), the receiver is listening to (i.e., receiver is currently a member of \( S_i \)'s multicast group) based on \( R_i \) and \( L_i \) computed as in RE.3

\[
\text{maxseqno}_i = \min(\text{seqno}_i + \frac{R_i}{1 - L_i} \times (D - t), 2(i + 1)K - 1)
\]

where seqnoi is the most recent sequence number from

\[^6\text{In our experiments, and unless otherwise specified, we take the update period to be 0.2 seconds.}\]

\[^7\text{In our experiments, we take } \alpha = \delta = 0.5, \text{and we set } \gamma \text{ to 1. } \gamma \text{ could be set to higher values to account for high variability in the case of stringent deadlines.}\]

**Sender Probing and RTT Estimation**

During data transmission, sender \( S_i \) transmits probes periodically. A probe packet includes the number of packets to be sent (PTS\(_i\)) and time-stamp for when the packet is sent. The purpose of using probes is two-fold: First, a probe is used to trigger NACKs from receivers. Upon receiving the probe containing PTS\(_i\), a receiver makes a local decision whether this is enough to sustain its current loss rate as we described earlier in RP.3. Second, a probe is used to estimate the round-trip time (RTT).

\[^8\text{In our experiments, we take } \text{relax} \text{ to be } 20 \text{ packets for } K = 1000 \text{ packets.}\]
When a receiver responds with a NACK, it sends the time-stamp for when the NACK is sent. The time-stamps are used to calculate RTT as suggested in [23]: the sender sends a probe at time \( t_1 \), and a receiver receives the probe at time \( t_2 \). If the receiver sends a NACK at time \( t_3 \), it includes \( (t_1, \Delta) \), where \( \Delta = t_3 - t_2 \). Once the sender receives the NACK at time \( t_4 \), it computes RTT as \( RTT = t_4 - t_1 - \Delta \).

In other reliable multicast protocols such as SRM [14] and SHARQFEC [17], the estimation of RTT is very critical for NACK suppression and repair. However, in our protocol, the RTT value is not critical. The sender only needs an estimate of the maximum RTT from receivers to set the probing period. As more data is transmitted, the feedback from receivers about their losses in response to probes becomes more critical. Thus, it is desirable to gradually decrease the probing period (to no less than the RTT to avoid sending duplicate probes).\(^9\)

**SomeCast Extensions**

Due to space limitations, we do not present results of SomeCast extensions for the dynamic selection of the “best set of server(s)” by a receiver, for balancing load across servers, and for a TCP-friendly transmission by the servers. Here, we briefly discuss such extensions.

To enable receivers to choose the subset of multicast groups to which they subscribe, we have evaluated novel **multicast group selection** algorithms, which take into consideration knowledge of static network topology (e.g., based on distance between receiver and sender or closest router carrying the multicast group) or dynamic network conditions (e.g., whether or not paths to two multicast groups share a common congestion).

One of the salient features of the SomeCast paradigm is that it delegates the responsibility of QoS management to receivers. As we indicated earlier, such delegation is attractive because it boosts the scalability of the system by making the overhead incurred by senders independent of the number of receivers and/or the diverse characteristics of paths between senders and receivers. Another advantage of delegating the management of real-time QoS constraints to receivers is that it enables senders to respond to network congestion conditions without risking the violation of QoS constraints at the receivers (since receivers can recover from a reduction in the rate at which a sender transmits data on its multicast group). We have investigated such techniques, whereby SomeCast senders manage congestion by adopting TCP-friendly transmission policies (as opposed to the constant-bit-rate policy we adopted in the implementation presented in this paper).

5. **Performance Evaluation**

**Simulation Model:** To evaluate the performance of SomeCast against that of Unicast and ManyCast, we set up a simulated multicast network using the 19-node tree topology depicted in Figure 3. In this topology, a CBR (Constant Bit Rate) data source is attached to each of nodes 14 to 18 (the primary sender \( S_0 \) is attached to node 14). All other nodes (i.e. nodes 0 to 13) act as receivers. The packet interarrival time for the CBR source is set to 0.01 second. Each link in the network is subjected to a maximum of 32 on-off cross connections generated by a UDP-based agent. This UDP-based agent generates connections with an inter-arrival time uniformly distributed between 0 and 0.1 second. Each connection is an on-off source with Pareto distributed “on” and “off” periods with average durations of 0.1 second and 0.9 second, respectively. The Pareto distribution has a skew parameter of 1.35. During the “on” periods, packets are generated at a rate of 1000Kbps. This cross-traffic resulted in up to 30% loss rates observed at receivers. The bandwidth of the links in our simulated topology are set to 1.5Mbps. All links have a propagation delay of 15ms. The packet size is 1KB. We take \( K = 1000 \) packets, so the size of the data is 1MB.

**ns Prototype Implementations:** We prototyped an implementation of our SomeCast-enabled protocol using the UCB/LBNL/VINT network simulator, ns-2.1b4 [24]. A new agent, called *scast*, is created as a subclass of AgentClass and defined in `scast.cc` and `scast.h`. This agent implements the SomeCast-enabled protocol. The primary sender starts transmitting data at time 25.0 (a warm-up period during which cross traffic at all links are generated). Other senders start transmitting as soon as one or more receivers join their multicast groups. The simulation run is stopped once all receivers receive by the deadline the needed packets to recover the original data, or whenever the simulation clock exceeds the deadline. In the latter case, one or more receivers had missed their deadline.

---

\(^9\) In our simulations, sender \( S_i \) sends the first probe after sending the first \( K/5 \) packets, then the probing frequency is increased by sending one probe every \( RTT \).
Figure 3. Topology used in our simulations

In our experiments, we take the total number of senders/groups to be 5. Each sender is assumed to have $2K$ encoded packets. We consider two variations of our protocol: (1) **SomeCast-1** where a receiver starts by joining the multicast group of primary sender $S_0$ and then joins other groups as needed; and (2) **SomeCast-5** where a receiver starts by joining all multicast groups of all 5 senders and then leaves groups as needed as long as it is able to receive by the deadline the number of packets it needs for full recovery.

To compare against SomeCast, we also simulated the two special cases of **ManyCast** and **UniCast**. ManyCast is the same as SomeCast except that joins and leaves are static, i.e., every receiver is subscribed to all multicast groups all the time. In UniCast, there is only one sender, the primary one, that every receiver joins.

**Performance Metrics:** To compare SomeCast to the ManyCast and UniCast based protocols, we considered the following performance metrics:

- **Guaranteed Deadline Percentage:** Percentage of receivers which successfully receive by the deadline ($D$) all packets ($K$) needed for full recovery.
- **Goodput:** Ratio of number of packets received and used for data reconstruction (by all receivers) to the number of packets sent (by all senders).
- **Average Number of Groups Joined:** Average number of multicast groups that a receiver joins.

In our simulations, we assume (for simplicity) that all receivers are subject to the same deadline. We define the **laxity** to be the ratio between the requested deadline and the most stringent deadline that can be met (i.e., when there are no losses and when every receiver joins the multicast groups of all 5 senders).

**Simulation Results:** We present our performance metrics as a function of laxity. Figure 4 shows the Percentage of Guaranteed Deadlines, the Goodput, and the Average Number of Groups Joined.

Since in our SomeCast protocol, a receiver adjusts dynamically the number of groups to which it subscribes, SomeCast strikes a good balance between Percentage Guaranteed and Goodput since a receiver joins the minimum number of groups needed to receive $K$ packets by the deadline. UniCast, where every receiver only joins the primary sender, yields the lowest Percentage Guaranteed and Goodput for a wide range of laxity values. Finally, ManyCast, where every receiver joins all 5 groups, yields the highest Percentage Guaranteed at the expense of lower Goodput.

It is important to note that at lower laxities, ManyCast has the highest Goodput as it is able to make use of transmissions from all 5 senders to satisfy the requested deadline at many receivers. Other protocols suffer from lower Goodput as many receivers fail to meet their deadlines and transmissions, albeit from fewer senders, are wasted. Figure 4(c) illustrates how a SomeCast receiver adapts the number of multicast groups to which it subscribes depending on how stringent the deadline is.

Observe that the performance of SomeCast approaches that of UniCast under high laxities, and that of ManyCast under low laxities. Note the system is non-linear—as laxity increases beyond 2 seconds, performance in terms of Percentage Guaranteed and Goodput sharply increases as it becomes possible for several receivers to meet their deadlines. This behavior is typical of real-time systems.

Figure 5 shows the behavior of SomeCast-1 and SomeCast-5, respectively, for varying receiver update intervals. Recall from Section 4 that the update interval dictates how frequently a receiver updates its estimates of loss rate and throughput of its path from each sender, which then allows the receiver to decide whether to join groups of additional senders or leave some groups to which it is currently subscribed. Clearly, more frequent updates allow the receiver to adjust its level of service in a more timely fashion at the expense of higher com-

---

10 This is an artifact of the specific topology and link delays used in our simulations.
putational overhead. In SomeCast-1, more frequent updates are more critical at lower laxities so as to allow the receiver to join more senders in a timely fashion, hence increasing Goodput (cf. Figures 5(a) and (b)).11 In SomeCast-5, more frequent updates are more critical at higher laxities so as to allow the receiver to leave in a timely fashion additional senders beyond what is needed to fully recover the data by the (loose) deadline. These more frequent updates result in higher Goodput (cf. Figures 5(a) and (c)).12 These results suggest a dynamic update policy where the frequency of updates is dependent on how stringent the requested deadline is.

11Not shown is Percentage Guaranteed, which is also higher with more frequent updates.
12Not shown is Percentage Guaranteed, which is not much affected by the frequency of updates in SomeCast-5.

6. Summary
We have proposed SomeCast—a novel paradigm for the reliable multicast of real-time data to a large set of receivers over the Internet. SomeCast is receiver-initiated and thus scalable in the number of receivers, the diverse characteristics of paths between senders and receivers, and the dynamic conditions of such paths. SomeCast enables receivers to adapt dynamically to the unpredictability of network conditions. This adaptation enables receivers to meet specific real-time QoS constraints. We have presented an instance of a SomeCast-enabled protocol, which we have prototyped under the UCB/LBNL/VINT network simulator (ns-2.1b4). We have presented simulation results that show the superiority of SomeCast when compared to the previously proposed ManyCast and AnyCast paradigms.
Acknowledgments: We would like to thank John Byers for the many discussions on Tornado codes and Digital Fountains. This work was supported in part by NSF research grants ESS CCR-9706685, CAREER ANIR-9701988, and MRI EIA-9871022.

References


