the Availability Digest

Reliable Multicasting

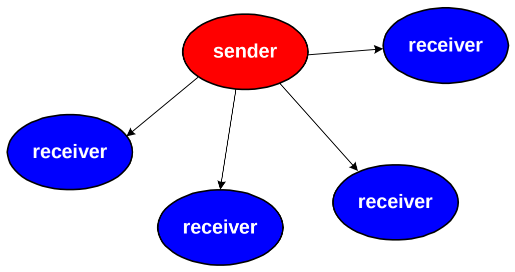
January 2008

What is Multicasting?

Many applications exist in which messages must be sent to a group of readers. Replicating database changes to the database copies in an active/active network is one such example with which we deal frequently in the *Availability Digest*. However, there are many other applications that require this capability. The distribution of actionable events, of stock market activity, and of news streams are other examples.

The straightforward approach is for the sender to send its message to each receiver over a point-to-point connection using a protocol such as TCP/IP (a unicast). However, this approach is not scalable and breaks down with a large number of receivers.

A more practical approach is to *multicast* a single message to a group of receivers in a multicast group. The group contains all of the receivers that have an interest in the message.



A Multicast Network

Multicasting in its simplest form is unreliable. Messages are sent via a one-way best-efforts protocol such as UDP. There is no guarantee that a message will be received by a specific receiver. If a receiver misses a message, it may not only be unaware that it has missed that message, but even if it is aware, it has no way to recover that message from the sender.

In many applications, this is not acceptable. It must be guaranteed that every receiver in a multicast group receives every message. This is known as reliable multicasting. Building reliable multicast networks that can scale to a large number of receivers across a wide-area network is a difficult problem. No single best solution exists, and each solution introduces new problems.[1]

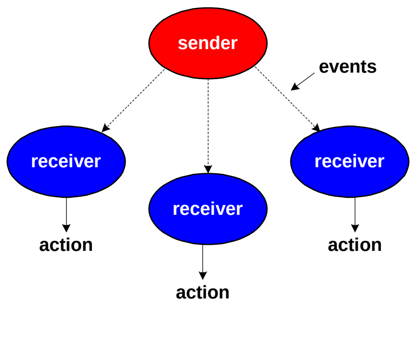
The problem of reliable multicasting has different solutions for local area networks and wide area networks. The problem is further complicated if there are multiple senders and if proper message ordering is required among the senders. These considerations are discussed below.

Classes of Multicasting Applications

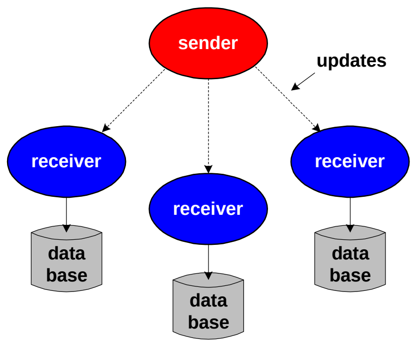
There are two primary types of multicasting applications. One is the distribution of actionable events. For instance, processes in a distributed environment may need to know when control parameters have changed. An event can be multicast to all concerned processes so that they can read the new parameters from a central source. Alternatively, an event could be broadcast to

[1] A. S. Tanenbaum, M. Van Steen, *Distributed Systems: Principles and Paradigms*, Pearson Prentice Hall; 2007.

all processes of a class indicating that an updated version of the process is available or that all processes should begin or end processing.



Event Multicasting



Update Multicasting

The other type of application is the maintenance of multiple copies of a database. The database might be, for instance, the application database in an active/network, a securities database accessed by the customers of an information distribution service, or news stories available over the Internet. Database multicasting applications are further complicated by the fact that new database copies joining the group need to be initialized and by the possibility that subtle errors could cause the databases to diverge over a period of time.

Either of these types of multicast networks may be preconfigured, or they may be dynamic. An example of a dynamic multicast network is a publish/subscribe environment. In such an environment, a *subscriber* subscribes to certain message types, such as certain classes of events or certain database changes. The *publisher* then sends an event or database update message only to those subscribers who have expressed an interest in the content of that message. In dynamic multicast networks, members may join (subscribe) or leave (unsubscribe) the group at any time.

Reliable Multicasting over Local Area Networks

Multitasking over LANs is fairly straightforward. A multicast group may be established with preconfigured nodes as receivers or as a dynamic multicast group in which members join or leave at their will using, for instance, a publish/subscribe mechanism. The Internet Group Management Protocol (IGMP) supports such multicasting.[2] A sender can then send a message to the multicast group members. The message will be delivered only to those members of the group but not to other nodes on the LAN that are not included in the group. The multicast message can propagate through the various subnets on the LAN network via routers connecting the subnets. In order to limit subnet loading, the routers should be configured to block a multicast message to a subnet that serves no members of the multicast group.

Messages to multicast groups on a LAN are sent via UDP (the Internet's User Datagram Protocol). UDP is a best-efforts only protocol that tries to deliver the message but has no feedback as to which receivers actually got the message. It is the goal of reliable multicast to correct this situation.

Lost Message Detection

The first problem is that a receiver must be able to identify a lost message. This is straightforward. The sender simply provides a sequence number in each message that it sends.

[2] H. R. Stevens, *TCP/IP Illustrated, Volume 1*, Addison Wesley Longman; 1994.

Each receiver can then monitor the sequence number and can determine not only that it missed a message, but it can identify for the sender which message it missed.

Message Buffering

Fundamental to reliable multicasting is the capability for a sender to resend a lost message. This means that the sender must buffer all of its recently-sent messages. If a receiver requests the retransmission of a lost message, the sender can retrieve that message from its buffer and resend it.

One issue with message buffering is the determination of the time at which a message can be discarded. This depends upon the means for retransmission requests, as discussed next.

Positive Acknowledgements

The one certain way for a sender to verify that each receiver has received the multicast message is for each receiver to acknowledge that it has received the message. These acknowledgements can be sent over unicast connections between each receiver and the sender, such as provided by TCP/IP. Only when the sender has received an acknowledge message from each member of the multicast group can the message be deleted from the sender's message buffer.

A receiver will know that it has missed a message when it receives a message with a later sequence number. At this point, the receiver will return a negative acknowledge (NAK) to the sender; and the sender will retransmit the lost message, as described later.

The obvious problem with this solution is the load imposed on the sender by the acknowledgement traffic. Though acknowledge messages can be piggybacked on top of other messages, the advantage of multicasting is that only one message need be sent by the sender regardless of the number of receivers. Now, however, the sender must receive an acknowledge message from all receivers. If the number of receivers is small, this load may be acceptable. However, this technique is not scalable and will rapidly break down as the number of receivers increases.

Negative Acknowledgements

The scalability problem is solved by eliminating the positive acknowledgements. With this technique, receivers return only negative acknowledgements. If the sender does not hear from a receiver, it assumes that the receiver has received all messages to date.

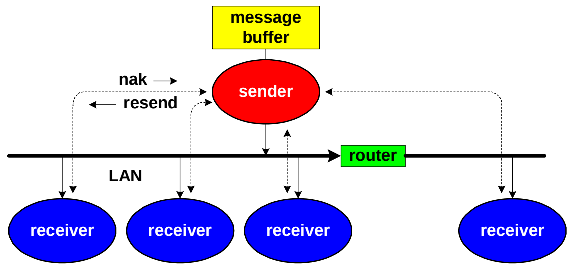
In this model, the sender has no specific information as to how long it must retain messages in its buffer. Therefore, it must depend upon timeouts. A message is held long enough so that the sender can be ensured that all receivers have received it and that there will be no further retransmission requests.

There are two ways in which negative acknowledgements can be sent from a receiver to the sender – unicast and multicast.

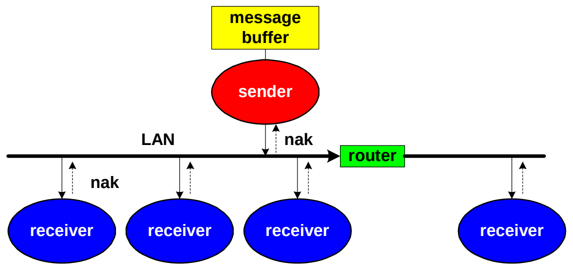
- Unicast: Negative acknowledgements can be sent from each receiver to the sender over point-to-point connections such as TCP/IP. The missing message may be returned to the receiver over this channel.

- Multicast: A receiver can multicast a negative acknowledgement to the multicast group including the sender. The missing message is multicast to the multicast group.

The use of negative acknowledgement multicasting can be put to good use to reduce negative acknowledgement traffic. First, each receiver imposes an arbitrary delay on its transmission of a negative acknowledgement message. Each receiver monitors the negative acknowledgements of the other receivers. If it detects a negative acknowledgement from another receiver that is the same as one that it wishes to make, then it does not have to send its negative acknowledgement. It knows that the message it needs will be coming shortly (if, in fact, it has not already been retransmitted). Consequently if multiple receivers have missed a particular message, only one (or a very few, depending upon timing) negative acknowledgements need be multicast.



Reliable LAN Multicast with Unicast NAKs



Reliable LAN Multicast with Multicast NAKs

Message Retransmission

When the sender receives a negative acknowledgement, it must resend the missed message to the requesting receiver. This can be done in one of several ways:

- It can send the missing message directly to the requesting receiver over a point-to-point connection between it and that receiver. This requires that the sender maintain a permanent or transient connection to each receiver. However, this technique precludes the use of negative acknowledge multicasts as described above since only the requesting receiver will receive the missing message.

- The missing message can be multicast to all receivers. Receivers can determine if this message is a duplicate of a previously-received message based on the message sequence number, and they can discard duplicate messages. Though processing duplicate messages is an increased overhead for the receivers, this overhead is generally very small.

- The multicast stream can be restarted at the point of the missing message. Receivers will discard duplicate messages. This technique is useful on noisy channels that are not heavily loaded.

Reliable Multicasting over Wide Area Networks

Multicasting over wide-area networks (WANs) differs from that over LANs because there is typically no way for a sender to multicast a single message over a WAN without some specialized infrastructure. Therefore, a custom WAN multicast network must be configured. Such a network is often organized as a tree in which a message sent by a sender propagates down the tree to its receivers.

The Multicast Tree

A multicast tree is a typical tree structure with a root node and several child nodes. Each node in the tree can serve its own receivers and can also serve as a parent to its child nodes. Each parent can have several child nodes. Each child node has a single parent node.

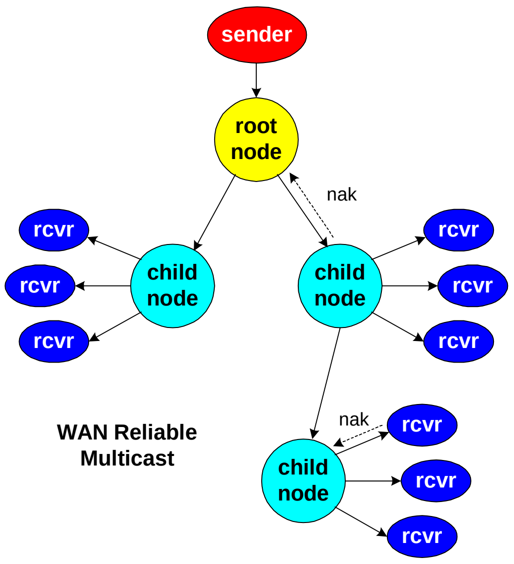
The network serving a node and its children can be either a WAN or a LAN. If the children are distant from the node, a WAN network must be used. However, if the children are local, then the parent node can multicast to its children as described previously for reliable multicasting over a LAN.

Node Buffering

One problem with WAN multicasting is that the links in the tree can have very large communication latencies, often extending into the tens or hundreds of milliseconds. Therefore, it can take not only a long time for a message to propagate from the sender to a receiver, but it also can take a long time for the receiver to receive the retransmission of a lost message.

To alleviate this problem, the nodes in the tree can buffer recent messages. Then, if a receiver detects that it has missed a message, it need only ask its parent for a retransmission of that message.

This technique also serves to reduce the retransmission load on the sender.



WAN Reliable Multicast

Distributed Databases

As mentioned earlier, one application for multicasting is to support distributed databases. As opposed to events that are multicast, distributed databases must be persistent. However, database copies do not have to be complete.

This is not true of application databases used in active/active application networks as each database copy has to reflect the current state of the application. However, this completeness requirement can be relaxed for information distribution databases as might be used to distribute stock quotes or news items.

Node buffering is very useful in these cases. To use node buffering to make information distribution more efficient, an update is not propagated to a receiver unless the receiver has requested that this particular data item be made available to it.

For instance, in a securities quotation service, each node may contain a partial database containing the quotes for a certain set of securities. A node serves a set of brokers' terminals. When a broker requests of his node the price for a security, the node will send him the current price if it has it in its database, If it does not, the node asks its parent node for the data item. If the parent node does not have it, it asks its parent node and so on until the root node, which has the complete database, is reached.

Each parent node is aware of the data items that are being monitored by each of its children. Therefore, when an update for a data item is received by a node (which will receive it only if its parent is aware that it is monitoring that data item), it will send the update to those of its children nodes that are also monitoring that data item. Thus, a node will always maintain the latest state of all data items that it is currently monitoring.

This organization has several advantages. First, except for the first query for a data item, all queries are satisfied by the requester's parent node. Queries do not have to travel across the network to a master database somewhere distant.

Another important advantage is bringing a node online. When a node joins the network, either because it is new or because it is a recovered failed node, the node's database does not have to be initialized. Rather, it is self-initializing over a period of time as requests are made of it.

In addition, the database can be continually refreshed by sending its information as background traffic over an extended period of time (which might take hours). This provides two functions:

- It brings incomplete databases up-to-date even in the absence of query activity.
- It corrects database errors that might occur over time.

Dynamic Trees

LAN multicast groups can easily be dynamic as users join and leave the group. However, dynamic multicasting over a WAN is not quite so simple.

If a node that is not currently part of the multicast tree wants to join the tree, it must bind with some other node as a child to that node. But to which node should it bind?

A common solution is for the new node to contact the root node (or some other monitor node) and request that it be allowed to join the tree. The monitoring node can make the decision as to which node should be the new node's parent based on one or more criteria:

- Stress: the current load on existing nodes in the tree.
- Stretch: the ratio of the latency through the tree to the node as compared to the latency directly to the root node (i.e., the efficiency of the path).
- Cost: the total path time to reach the node through the tree.

Note, for instance, that connecting all nodes to the root in a star network minimizes the cost but maximizes the stress.

From time to time, it may be desirable to rebalance the tree by redefining the parent-child nodal relationships.

Node failure can be conveniently handled in a dynamic tree. Should a node fail, its children can be connected to other nodes according to the tree balancing algorithm in use. The child nodes will detect any messages that they had missed and will request retransmission once they are rejoined to the tree.

Multiple Senders

If multiple senders can multicast messages to the group, each simply maintains its own message sequence numbering. A receiver therefore can detect missing messages from any sender and can request transmission from that sender.

An interesting example of this is the multicasting of negative acknowledgements over LAN networks, as described above. In this case, any receiver can also be an occasional sender. However, in this case, the data stream sender receiving the negative acknowledgement messages will probably not request that a missed negative acknowledge message be retransmitted. It will instead wait for the receiver to retransmit that negative acknowledge message after the receiver has determined that it still has not received the retransmission of the lost message. In this case, there is no reason for the receiver to even add sequence numbers to its negative acknowledgement messages.

If the senders are all independent, there is no need to require total ordering of all messages. Ordering of messages received from each sender according to sequence numbers is sufficient.

However, if two or more senders are causally dependent – that is, messages from one sender depend upon the activities of another sender – then total ordering of messages between senders may be required. This is a particularly difficult problem   It can be solved via vector timestamps as discussed in the Tanenbaum/Van Steen reference cited previously.[3]

Summary

Reliable multicasting to several receivers is required in many applications in which events or databases must be distributed across a network. Reliable multicasting requires that receivers can detect missing messages and can efficiently request their retransmission. Lost message detection is easily accomplished by message sequence numbering.

However, the methods for distributing messages and message retransmissions to a group of receivers is different for LANs and WANs. With LANs, IGMP protocol support exists for multicasting. Retransmission requests can be via unicast to the sender or by multicast over the multicast network. Likewise, retransmissions can be sent via unicast to the requesting receiver or to all receivers over the multicast network.

WAN multicasting can be implemented via trees overlaid on the network. Receivers obtain their updates from nodes in the tree. Updates propagate down the tree on an as-needed basis. A distributed informational database can be created over time as it processes requests or by database refreshes carried out in the background.

[3] A. S. Tanenbaum, M. Van Steen, Chapter 6, Synchronization, *Distributed Systems: Principles and Paradigms*, Pearson Prentice Hall; 2007.