A Dynamic Multicast Tunnelling Architecture for Multimedia Applications

Pathik Gupta, Chai Kiat Yeo, Bu Sung Lee, *Meng Hwa Er
School of Computer Engineering
*School of Electrical and Electronics Engineering
Nanyang Technological University
Nanyang Avenue, S639798
SINGAPORE

Abstract: With the growth in the development of the multicast technology, there is also a need to provide a solution that can bridge a unicast host to the multicast network. This architecture provides a dynamic and scalable solution that can be used to tunnel the multicast packets on a unicast network at the application level. This is achieved via a network of dynamic peers which allows the hosts to switch their sources dynamically to achieve better QoS. The architecture described herein provides an insight into the protocol as well as the implementation details. It uses the gossip protocol to achieve better QoS.

Key-words: multicast, unicast, tunnel, bridge, QoS, gossip, peer

1 Introduction

For the past few years, there has been constant research and development into multicast technology. However it is still in a stage where it is not accepted commercially worldwide as many of the commercial Internet Service Providers (ISP) still do not support multicast. This means that majority of the Internet users are deprived of the sessions running on the multicast networks. Moreover many of the big organizations use firewalls on the gateway and might not deploy multicast on the gateway routers due to lack of security. This creates a need to develop a framework that can provide a complete solution to bridge multicast and unicast network.

There have been applications like mTunnel [1] and livegate [2] that provide such kind of services but both of them create static tunnel which means that a host can receive a MBONE session via some static reflector.

The proposed architecture does not rely on static reflectors; rather it provides techniques for a host to discover the appropriate hosts that can source them the sessions. These hosts may not be directly connected to the multicast network but are receiving the session from a host that is connected to the multicast network. This leads to the creation of a hierarchical tree-like structure. This structure has a major advantage of saving a large amount of bandwidth especially in the case when multiple users on the same network are trying to obtain the same session. In this case, instead of getting multiple streams from a source, multiple recipients on the same network will discover each other dynamically and share the session. This is also flexible in the sense that if there is a big network which is multicast-enabled within it, then a host can obtain a unicast session from the external host and directly multicast within the network. The other hosts in the same network can receive it without going out to the external host. Thus the bandwidth saved can be enormous in these situations. This leads to the creation of multicast-unicast bridges where a host who receives a unicast stream can multicast it or vice versa.

To improve upon the Quality of Service (QoS), every host receiving a session keeps track of the various other hosts sourcing that particular session. In the event that the QoS of the current stream goes down, the host can immediately switches to a new source providing a better QoS. This leads to the idea of GOSSIP protocol in which a host continues to gossip with other sources from time to time so as to get an idea of the QoS that the other sources are providing.

This architecture therefore creates a dynamic structure of hosts sharing a particular session. Every host can act as a peer as it can receive a session as well as source the session. Section 2 covers the protocol needed to establish an application level tunnelling between a unicast and a multicast network and section
3 discusses the implementation of the protocol. Section 4 concludes the paper and discusses the future scope of the project.

2 Protocol Overview and Design
Fig. 1 shows the overview of the proposed architecture. There are two main entities in this architecture. The peer itself that provides the bridging capabilities and a Central Directory Service (CDS) that acts as a central repository for the dynamic host detection. The CDS keeps track of the various hosts and the sessions they are receiving. To make it more scalable, the CDS can be arranged in a hierarchical fashion with multiple servers located at different places. The CDS is designed to be a totally stateless system and does not maintain any state for a host. It just keeps a record of the various hosts.

The protocol has three main operations, namely, source Discovery, peer joining the session (peer join) and peer leaving the session (peer leave). Each of these operations requires the peer to follow certain rules and are discussed briefly below.

2.1 Source Discovery
When the peer wants to receive a multicast session, it has the knowledge of only the group IP and group port. The group IP and port can be obtained by using the Session Directory Service which uses Session Description Protocol [3].

- The host who wants to receive the session sends a QUERY message to the CDS. The query includes the group IP and port of the multicast session.
- The CDS performs a lookup in its database to find out the peers receiving that particular session. For each of the peers, CDS also maintains some information about the QoS that they are receiving. In case of multiple sources, the CDS arranges each of them in priority based on QoS and sends it to the requesting peer. If there is no active source, the peer will perform a ‘proxy source join’.
- After obtaining the source, the peer performs a join.

2.2 Peer Joining
The peer joining process is as follows:

- Before performing the source discovery, the peer attempts to obtain the multicast session directly by creating a socket and joining the desired multicast group. A certain time out value is set and upon the time out, the peer assumes that it is not connected to a multicast enabled network. It closes the socket and tries to obtain a peer that can source that session.
- If the direct join is successful (the peer is connected to the multicast network), the peer informs the CDS that it is ready to be a potential source.
- If however the direct join is unsuccessful, the peer then performs the Source Discovery. Upon receiving the potential sources, the peer then tries to find whether any of the sources is located in its own network. This is simply implemented by matching the network portion of the address based on the classification of IP addresses. If there exists a source in the same network, then the peer directly contacts the source and exchanges the set up information.
- In other case when there is no match of the same network, the peer contacts the source based on the priority assigned by the CDS.
- The source on receiving the peer’s request adds the peer into its local table. The source initially provides the peer with a test multicast group IP and port that the source is sending data on. The peer tries to open a socket and receive data on this test IP. This is done to test the local multicast connectivity. Local multicast connectivity means that the routers between the peer and the source are multicast enabled, however the peer does not have a direct connectivity to the MBONE. In this case, the source will locally multicast the data and the peer can receive it.
- If the local multicast fails, the source then streams the unicast data.
- The architecture tries to create the most optimum solution starting from trying out the global multicast connectivity to the local multicast connectivity to the unicast connectivity. In the course, it creates a tree that is rooted at the first peer receiving the MBONE session directly.
- Once the source establishes the connection with the peer, the peer continues to send Alive message to the source. This is to let the source know that the peer is alive and receiving the session. Also, the source sends out its Alive message to the peer. This is necessary to let the peer know that the source is still active. In the event that the parent of the source crashes, neither the source nor any of the peers below it will be receiving the data. This can lead to each of the peer mistakenly believing that
their own source has crashed and trigger a flood of source switching among the children of the source. The constant ‘alive’ messages from the source make sure that the source is active even though data may temporarily be unavailable.

2.3 Peer Leaving
There are two possible scenarios when a peer leaves the session. The first can be the peer leaves the session by notifying its source and its receivers. The other scenario can be that the peer fails and just crashes. The protocol handles both these scenarios.

- When a peer wishes to leave the session, it sends a PEERLEAVE message to its source as well as to its receivers. The source then shuts off the stream to this peer. The receivers then follow the rules of the Peer joining to receive the session. This is the case when the peer leaves the session normally.
- If however, the peer crashes or suddenly shuts down, the source finds out that its receiver is dead because it does not receive any ‘Alive’ message from it. The source then queries the peer to check whether it is responding or not. If it fails to respond, it assumes that the peer is dead and shuts off the stream. It also notifies the CDS that a peer has crashed and CDS marks that peer dead. All the receivers of the peer will know that their source is dead because there is no data flowing as well as there is no ‘alive’ message from the source. This makes all the receivers perform the join again.

The various steps required in building up the session is shown in Fig. 1.

2.4 Proxy Source Join
- The proxy source is a kind of “reflector” that is directly connected to MBONE. There can be multiple reflectors around the whole MBONE and all the reflectors are registered with CDS.
- When the peer does not obtain any source for a requested session, it then performs a proxy join. The CDS provides the peer with a proxy IP. This is basically a static reflector that is connected to the multicast network directly. The peer will then contact the proxy and the proxy will receive the session on behalf of the peer and relay the data.
- The reflectors periodically communicate with the CDS to update their status. Hence the CDS can reply to the host with the best possible reflector that can service it.

2.5 Gossip Based Approach
Till now we have discussed the basic framework that is required to obtain the session. Now we take a look at the gossip based approach or also referred as ‘epidemic style’ approach [4]. Gossip is a new generation protocol that provides excellent scalability and stability of throughput under stress. Although most of the current research on gossip is based on providing a reliable and secure multicast, we are exploiting this approach to improve the QoS that a peer receives from a source. In the case when CDS obtains multiple sources providing the session, it sends all the source information to the peer. The peer first establishes a connection with one of the sources and once the session is established, it tries to gossip with the other potential sources. The gossip message is exchanged in a simple unicast manner. There is no reliability needed in exchanging the gossips because the failure for the message to reach the sources does not cause any problems. The gossip message contains the QoS that is received by a particular peer. So when the peer finds that there is other source that provides a better QoS, it will start a join on that source. Once the peer receives the new session, it gracefully closes the original session. It is also possible for the source to switch to the peer. This is because the peer includes its own QoS parameter (Round trip time) when it sends the gossip message to the source. So if the source finds that the new peer has a better QoS, it can perform a switch to make this peer the source. Hence the whole
architecture is dynamic providing an excellent QoS and also trying to minimize the bandwidth.

There are in fact two types of gossip that are under research. One is called the gossip-push and the other one is the gossip-pull [4]. Since the gossip has been used for studying the reliable multicast techniques, so each host sends a digest of the messages that it received to the other gossipers. The gossip-push occurs when one of the gossipers receives a digest from the other gossiper and then noting from the digest received realizes that the other gossiper has a packet not received by it asks the other gossiper to forward that particular packet. On the other hand gossip-pull occurs when one of the gossiper receives a digest from the other gossiper and notes that it has got a packet that the other gossips failed to receive forwards the packet to it. It has been proved that the gossip-push is in fact more efficient than the gossip-pull-push [4]. The idea of the gossip-pull and gossip-push is also incorporated into this architecture. Whenever the QoS parameter received by a potential source from the peer is worse than the QoS that the potential source is itself getting, it can ask the peer to switch from some existing source to this new potential source. This is an example of the gossip-push technique as the potential source has pushed the session to the peer. On the other hand, if a peer receives a QoS parameter from the potential source that is better than the original QoS that it is receiving, the peer will ask the potential source to start the session and shut off the original session. This is an example of the gossip-pull technique as the peer has pulled the session from the other source.

3 Implementation of the Architecture

This section discusses the various implementation issues involved in developing the framework for this architecture. Some of the salient features of the framework include:

- Provides support for Real Time Protocol (RTP) [5] as well as non-RTP based session.
- Provides in-built plug-in for transcoding MPEG, AVI, MOV, AU streams into RTP streams.
- Uses Java as an implementation language providing a cross-platform compatibility.
- Provides in-built plug-in to play the RTP streams. Also support external applications to directly use the framework without any configurations.
- Provides a Session Directory Client to view the information on the MBONE sessions even for non-MBONE clients.
- Provides a GUI for configuring the various configuration options.

The framework has been implemented in Java and uses Java Development Kit (JDK1.3) v1.3 [6] and Java Media Framework (JMF2.1) v2.1 [7]. As every peer in this architecture is a receiver as well as a source of the session, both client and server modules are implemented. The client module connects to the server module of another peer and pulls the session from it while the server module sources the session for the other client.

The pseudo codes used by a peer for connecting to the source is described below (client module):

```plaintext
Open multicast socket to receive the session directly;
If (data obtained)
  Inform the CDS about the connection type along with the group IP and port;
Else
  Connect to the CDS and provide the group IP and port for the desired session;
  Obtain the source information for the desired session;
  Connect to the source on the TCP port 4200;
  Obtain the test multicast IP from the source;
  Open multicast socket for the test multicast IP;
  If (data received)
    Send a success to the source and close the connection.
Else
  Inform the source that data is not received on the multicast and provides a local port to source to send the stream to.
  Source closes the multicast stream and provides the data on the port sent by the peer.
End if
Send the connection information to the CDS.
Start a thread to handle Alive messages to and from the source.
End if
```

As can be seen from the algorithm, the architecture relies on the TCP connection to send the necessary information. The use of TCP is necessary so as to ensure that the protocol does not break down due to loss of information packets. However once the necessary information is exchanged, the TCP connection is immediately closed to avoid any loss of bandwidth. The handshaking of sending and receiving
the ‘alive’ messages then relies on the use of UDP packets as the loss of some ‘alive’ messages does not cause the system to break down.

The server module used by the peer is used for serving the clients thus sending the session to them. The pseudo codes for the server module is shown below:

```plaintext
Open a socket on TCP port 4200;
While (true)
    Wait for the request to come;
    When request arrives, spawn a new thread to handle the request;
End while;

-Request Handler Thread
Create a test multicast socket and send the packets received on the session to this socket;
Send the multicast test IP for the session;
If (Success returned)
    Exit;
Else
    Receive the port sent by the peer;
    Close the test multicast socket;
    Open a UDP socket and send the packets received on the session on to this socket;
End if
Start a thread to handle the ‘alive’ messages to and from the client
```

3.1 Central Directory Service (CDS)
The CDS is a central database that maintains a list of all the peers that are using the framework. The CDS provides necessary support for the new peers to join the network of peers forming a tree like structure. The new peer is added to the leaf of the tree. The CDS acts very much like a CORBA naming service providing service for namespace translation and obtain the object references. CDS provides a session information to source IP mapping. The new peer can then directly interact with the source and obtain the session.

3.2 Adding Gossip
The gossip is added to the system in the ‘alive’ message handler thread. Each of the peers sends an ‘alive’ message to their source. For this implementation, the QoS parameter used is the Round trip time (RTT) of the current source. The primary peer that is connected to the MBONE or directly receives the session initializes the values of QoS parameters to 0. The first level peer getting session from the primary peer calculates actual RTT. The first level peer passes this information to the second level peer and the second level peer calculates its own QoS values. The second level peer then adds the value it calculated and the value it received from its source (first level peer here). In this way each peer will have the QoS values that represent the quality they are receiving with respect to the primary peers. Here we do not consider any differences between the primary peers and take them all at par to make the discussion simpler. However, we can always initialize the primary peer’s QoS values to some other actually calculated value.

To implement gossip-push, each peer sends the QoS values it receives from its actual source to the other potential sources. Each of these sources compare their own QoS values with the one sent by the peer. If the QoS value sent by the peer is higher than their own QoS, then the source will ask the peer to perform a switch. However, this can lead to situation where the peer is always performing a switch. To avoid this situation, the peer will consider the history of the QoS values of the potential source asking for a switch if the history exists. If the average of the history values is better than the current QoS value it has and the peer has not been switching source within a certain predefined time period, the peer can perform a switch to the new potential source. This prevents the problem of source oscillations.

The gossip-pull is initiated by the peer rather than any of the potential sources. The peer receives the QoS values from each of the potential sources. The peer compares the QoS value it receives from its original source with the values from the potential sources. The peer also considers the history of the potential sources and also follows the requirement of some predefined time before making the actual switch to a new potential source.

As can be seen, the major difference between the gossip-pull and gossip-push is that the pull is initiated by the peer while the push is initiated by the potential sources.

3.3 Session Directory Service
When a peer wishes to join a session on the MBONE, knowledge of the group address and port is required. There are two ways to achieve this. One way is for the peer to directly use this architecture and subscribe to the group on which the Session Announcements [8] are made. The other way is to use the in-built Session Directory (SD) client included in the framework.
The session announcements are made on group address of 224.2.127.254 and port 9875. So if the first method is used, then a peer can directly subscribe to this group. If the second method is used, then SD client connects to the SD server to get the session information. The peer directly connected to the MBONE can listen to the session announcements. The primary peer connected to the MBONE starts a SD server. The peers on the unicast network can then access the SD server and get the session information from it.

4 Conclusion
The proposed bridge architecture is unique in that it provides a seamless bridge among multicast and unicast networks through application level multicast. Moreover, the distributed nature of the peering means that any client can act as a source dynamically which not only provides multicast accessibility to unicast networks but also enhances the QoS received by the clients.

The current implementation of the proposed architecture is aimed at streaming of multimedia information such as audio and video where loss of some packets is still tolerable. The architecture can be extended to support reliable multicast [9] for applications such as chat or whiteboard by extending the gossip-pull and gossip-push approach for the peers to update their lost data. The peer joins a session and starts receiving it. It also starts gossiping to the other potential sources. Using the gossip-pull or gossip-push approach, the peer can update itself of the loss data pulled from other peers or push data to update its peers.

References:


