Audio Video Conferencing in Distributed Brokering Systems

Ahmet Uyar[1,2], Shrideep Pallickara[2] and Geoffrey Fox[2]
(auyar@syr.edu), Department of Electrical Engineering and Computer Science, Syracuse University.[1]
(spallick,gcf@indiana.edu) Community Grid Labs, Indiana University.[2]

1.0 Introduction

Systems supporting multimedia conferencing and those based on the generalized publish/subscribe paradigm have both been around for quite some time. At its very core, both these systems tend to solve the same problem viz. the delivery of the content from the producers to the interested consumers. Performance is easily perceived in multimedia systems, and as the scale of the system increases the performance issue becomes a pivotal one. To mitigate these problems, multimedia systems have concentrated efforts on tightly integrating the content distribution problem with specific transport protocols, such as RTP [1], Multicast [2] and UDP. Efforts have also tended to focus on optimally encapsulating media content in a variety of codec formats.

In publish/subscribe systems, though timing constraints have been considered an important one (especially in systems deployed for synchronous collaboration), the emphasis has generally been on facilitating richer interactions between communicating entities. This difference in approach vis-à-vis multimedia systems, are exemplified in the sizes of the messages that encapsulate content. These messages generally tend to have several headers pertaining to content description, reliable delivery, priority, ordering, and distribution traces among others. These headers play a crucial role in implementing various Qualities of Service (QoS) strategies. As alluded to earlier, codecs used to encapsulate multimedia content tend to be far more compact.

Inevitably, these competing approaches to the content distribution problem will draw closer together to provide an interaction rich efficient content distribution solution. As opposed to relying on multicast for communications, publish/subscribe systems rely on software multicast. This allows routing solutions to work across realms and network boundaries where MBONE, necessary for multicast, is disabled or does not exist.

In this paper we suggest that the deployment of systems based on the publish/subscribe paradigm in the context of audio/video conferencing is a very good one. We base our investigations in the context of the NaradaBrokering [3-8] system, which provides support for peer-to-peer (P2P), distributed and centralized interactions. In the approach outlined in this paper, we encapsulate multimedia content in specialized events that facilitate fast intelligent routing while incorporating headers relevant only to problem of content distribution. This approach attempts to draw upon features that enhance performance while seeking to avoid features that result in performance degradation.

Besides our earlier work [9], which explored multimedia conferencing in the context of NaradaBrokering's JMS solution [8], the other work in the area of event based multimedia distribution can be found in [10] where a CORBA broker is used to route events encapsulating audio/video content.

Our current work eliminates certain drawbacks from our earlier approach, while making three distinct contributions. First we have eliminated the dependency on the large set of headers that are inherent in JMS messages, and the accompanying network/CPU cycles expended in the processing of the same. Our first improvement has been in the context of a compact representation of content that facilitates fast efficient routing and processing. Second, we have incorporated a strategy that allows us to deal with legacy multimedia applications. This is an important contribution since in this case the two systems/applications are completely decoupled from each other – obviating the need for specialized initialization, registration and any pre-processing that might be necessary. In our previous work we needed to write a specialized client, resulting in interactions only between these specialized clients.

Finally we include comprehensive benchmarks and our observations from a real time audio/video conferencing that was performed on our prototype system. Our benchmarks are based on the newly incorporated modifications in the calculation of destinations associated with the specialized events, the event's representation and support for legacy systems. Our observations from the real-time test combined with our benchmarks, enable us to explore deployability of systems such as NaradaBrokering in the context of real-time multimedia systems.

This paper is organized as follows. In section 2.0 we present an overview of the related work in this area. Section 3.0 provides a brief overview of NaradaBrokering and its applicability in the context of multimedia systems. Section 4.0 provides a discussion of the improvements, and changes these improvements entailed, while routing multimedia content. Section 5.0 outlines our experimental setups and incorporates results under various setting. We then include a discussion of the work that would augment the work outlined in this paper and set of conclusions derived from this work.

2.0 Related Work

Currently RTP is the most commonly used transport protocol to transfer audio/video traffic over the Internet and is the most favored approach to implementing conferencing solutions. RTP is designed to be used by those applications that require real-time end-to-end delivery services and is usually implemented on top of UDP or multicast. Sometimes it is also implemented on top of TCP or HTTP, particularly when traversing through firewalls. RTP achieves its best performance when implemented over either UDP or multicast, which do not incur delays pertaining to ordering, error correction and reliable delivery overheads that exist in TCP.

RTP provides a host of services such as payload type identification, sequence numbering, timestamping, source identification and monitoring for audio/video communications. RTP provides these services through a 12 byte header which is added to each audio or video package. RTP uses RTCP (RTP Control Protocol) to monitor the timely delivery of real-time data. It provides control and identification functionality. RTCP packages are very similar to RTP packages but they do not contain any media information, rather they contain information about the identity of the sender and the quality of media transfer.

UDP based RTP conferencing servers are either implemented in hardware or software and their meeting management concepts are based on a session management protocol such as H.323 [11] and SIP [12]. Currently, most of them are based on H.323. Recently there have been efforts in the SIP community pertaining to the development of conferencing servers [13]. Although these systems provide a good quality audio and video, they are very complex, expensive and hard to maintain. Development and maintenance of distributed conferencing servers can be very time consuming and challenging.

Multicast provides a very powerful, elegant and flexible framework for implementing audio/video conferencing solutions. The biggest obstacle for using Multicast in videoconferencing applications is the lack of widespread support. Private corporations usually choose not to support it, with some universities denying this support too. Cable modem companies providing broadband connections to homes and small offices do not support it either. For dial-up users there is no hope of getting a multicast service in the near future. It is thus rather difficult to deploy multicast for applications that are expected to serve all Internet users. Furthermore, for low bandwidth entities, difficulties in participations stems from the volume of audio and video streams. Such sessions are of course vulnerable to denial of service attacks from a malicious user. We believe that, currently, multicast is not a solution to be widely used by entities with varying bandwidth constraints.

Brokering systems can provide a unified framework for a wide gamut of applications, from media communications to application sharing. The Garnet [14] collaboration environment based on JMS is an example of such a system, which provides application-sharing, whiteboard, and shared-display support.

3.0 NaradaBrokering and the Rational for Multimedia Support

NaradaBrokering is a distributed brokering system, which provides support for centralized, P2P and distributed interactions. The smallest unit of the messaging infrastructure, which can run on a network of cooperating nodes, is the broker. Each broker is responsible for processing events (specialized messages with additional headers), computing destinations and making decisions to facilitate efficient routing.

In NaradaBrokering the broker nodes are organized in a cluster-based architecture. The cluster based architecture allows the system, to scale, to support clients of arbitrary size, while allowing individual broker nodes to compute alternate routes in response to node failures. NaradaBrokering provides intelligent routing of events within the system by selectively deploying brokers and communication links to aid disseminations.

We may enumerate reasons in support of deploying NaradaBrokering to route multimedia content –

1. *Availability* – Since it is based on a distributed architecture, there is no single point of failure within the system. Additional broker nodes may be added to support large heterogeneous multimedia client configurations.
2. *Scaling* – NaradaBrokering's cluster based architecture allows the system to scale. The number of broker nodes may increase geometrically, but the communication *pathlength*s between nodes increase logarithmically.
3. *Efficient routing and bandwidth utilizations* – NaradaBrokering computes destinations associated with an event efficiently. The accompanying routing solution deploys links efficiently to reach these computed destinations. The routing solution conserves bandwidth by not overload links with data that should not be routed on them. Under conditions of high loads the benefits accrued from this strategy can be substantial.
4. *Software multicast* – Since it relies on software multicast, entities interested in conferencing with each other need not set up a dedicated multicast group for communications. Problems associated with setting of multiple unique multicast groups are exacerbated in settings with large number of clients.
5. *Communication over multiple transports* – In distributed settings, events may traverse over multiple broker hops. Communication between two nodes may be constrained by the number and type of protocols supported between them. NaradaBrokering incorporates support for TCP, UDP, Multicast and SSL. HTTP support will be available soon. Multi-protocol support increases possibility of communications between two nodes. Furthermore, depending on the state of the network specific transports can be deployed to achieve better performance under changing network conditions.
6. *Communication over firewalls and proxy boundaries* – A lot of times two nodes/entities may be in realms separated by firewall and proxy boundaries. Irrespective of how elegant the application channels are, communications would be stopped dead in their tracks. NaradaBrokering incorporates strategies to tunnel through firewalls and authenticating proxies such as Microsoft's ISA and iPlanet's proxy.
7. *Ability to handle clients with varying bandwidth constraints* – Specialized links can deployed to filter, and possibly process, the volume of information funneled over links with slow, low-bandwidth connections.

There are other advantages that become more obvious when considered in the context of other features in NaradaBrokering. NaradaBrokering incorporates a security infrastructure [15] that also incorporates schemes to foil certain denial of service attacks. Similarly its archiving support and performance monitoring could be used to record sessions and deploy better transports for dissemination respectively. Multimedia content routed using this solution can harness these features to provide a better solution.

4.0 Incorporating support for Audio/Video conferencing

There were two drawbacks associated with our earlier approach, discussed in [9], which stemmed from the use of JMS messages to encapsulate multimedia content. First, each of the different JMS message [16] types has at least 10 different headers (all of which are redundant in the context of multimedia content) that can add up to 200 bytes of data in their serialized transfer over the network. When this is viewed in the context of the size of audio/video packets encapsulated the costs seem substantial. For example, a ULAW audio package for 20 ms has a size of 172 bytes including the RTP header and entails a 64kbps network bandwidth. Padding an extra 200 bytes of header to each audio package results in the bandwidth requirement of up to 148kbps. Then, there is the cost associated with serializing and de-serializing the multimedia content. Second, we did not support for legacy clients. This meant that the system could be used only by those applications that had been ported to incorporate specific initialization and registration schemes.

4.1 Designing the RTPEvent

We designed a special event, the RTPEvent, to encapsulate media content that comprises of 4 elements. There is a header (1 byte) identifying event type, followed by a topic name encapsulating information about the meeting that this content was generated in. To eliminate echo problems arising from the system routing content back to the originator of the content, information pertaining to the source is also included. This information can be represented in an integer, which amounts to 4 bytes. Finally, there is the media content itself as the payload in the event.

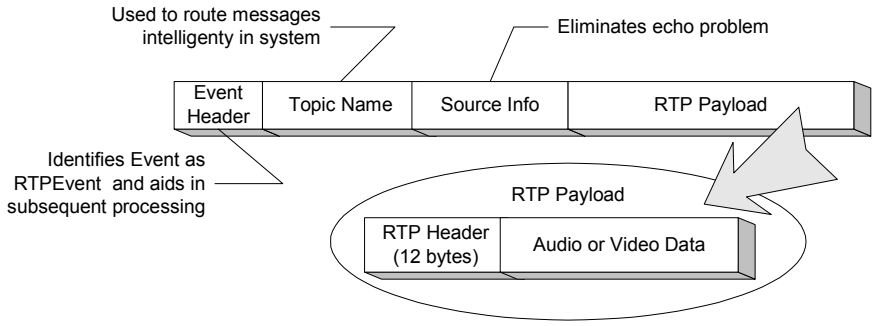

Figure 1: Anatomy of the RTPEvent

Depending on the type of the topic name, associated with the RTPEvent the number of extra bytes padded onto the RTP payload varies. If as in most publish/subscribe systems we choose to have String topic names, the number of extra bytes padded to the RTP payload is – five bytes for the header and source *plus* one byte for each character in the topic name. To avoid incorrect routing decisions caused by collisions in topic name space, we require that the topic names, corresponding to meetings, be unique. The cost of this choice could end up being substantial considering that each character adds one additional byte. Furthermore, since the topic names are of variable length one would also need to include information pertaining to the length of the topic name.

To obviate problems inherent in String based topic names, we decided to use integer topic names. This eliminates two different problems. First, the topic length does not vary and it will be represented in 4 bytes. This representation results in a total of 9 bytes being padded to the RTP payloads which is acceptable for almost all codecs. Second, using 32-bit integers allows us to uniquely distinguish between 2^{32}=4,294,967,296 different concurrent meetings. Also, when we compared the performance of serialization/de-serialization times for RTPEvents with String and integer based topics, the latter one was twice as fast.

Entities would thus subscribe to integer topics. This calls for an integer based matching engine which computes destinations from topics contained in the RTPEvent. This matching engine would also be

efficient since the memory requirements for integer topics are lower than those required for String based topics.

4.2 Support for Legacy Applications

The other drawback of our previous effort was the issue of support, or the lack thereof, for legacy RTP clients such as VIC, RAT and JMStudio. To circumvent this problem we incorporated a specialized implementation of the NaradaBrokering transport framework [17]. This process entailed an implementation of the Link interface which abstracts the communication link between two entities. The RTPLink, which we implemented can receive raw RTP packages over UDP from legacy system, wrap these packages in RTPEvents and propagate these events to the protocol layers in the broker node. Once it reaches the protocol layers at broker node, the event is routed within the distributed broker network.

The RTPLink deals with the initialization, registration and data processing on the communication link. For initialization purposes when a RTPLink is created, one should provide the port-number on which the RTPLlink should listen to RTP packages, and finally the IP-address/port-number pair at which the legacy system is listening to data. For registration purposes, the RTPLink is assigned a NaradaBrokering-ID and the RTPLink also subscribes to topic corresponding to its meeting. In the data processing part, the RTPLink when it receives media packages it constructs the RTPEvent for processing within the broker network. When an RTPEvent is ready to be sent to the legacy application the RTPLink retrieves the RTP payload from the RTPEvent and routes it to the legacy application based on the parameters specified during initializations.

For every legacy RTP audio or vide client, one corresponding RTPLink needs to be set up at a broker, within broker network. In our current implementation we are initializing these RTPLinks statically from a configuration file, but future work will involve a Conference Manager responsible for dynamically creating and destroying these links.

4.3 Some Implementation Details

There is a unique meeting-ID associated with each media type (audio/video) in a multimedia meeting, thus if there is both audio and video conferencing (depicted in figure 2), there would be unique meeting-ID for the audio conferencing and one for the video conferencing. Furthermore, since we use RTP for individual audio/video streams, we need two unique topics for each meeting — one for RTP packages and the other for RTCP packages (discussed in section 2.0). To this effect, individual meetings have unique even-numbered integers assigned as topics. We then used this even-numbered integer as the topic for RTP packets and the odd-numbered integer, immediately following that even-number, as the topic for RTCP packets. This is very similar to the RTP protocol in which RTP packets are exchanged on an even number port and RTCP packets are exchanged on the odd-numbered (RTP port + 1) port.

We have also developed a NaradaBrokering audio/video client using JMF. To summarize briefly, this client is capable of sending (and receiving) audio/video streams as RTPEvent to (and from) the broker network. The client subscribes to relevant topics and is then capable of exchanging audio/video streams with other clients (and even legacy clients subscribed to the same topic). We use the JMF RTP library to packetize and depacketize media streams. JMF has a RTP connector architecture, which allows us to provide our own transport module. Through this module, while sending the RTP packages, we encapsulate them in RTPEvents while assigning relevant topics to them. Upon receipt of the RTPEvents we extract the RTP payload from the RTPEvent. The raw RTP package is then fed to the JMF RTP library, which then constructs the audio and video streams. We also use JMF libraries to read a media file and play the received audio/video stream.

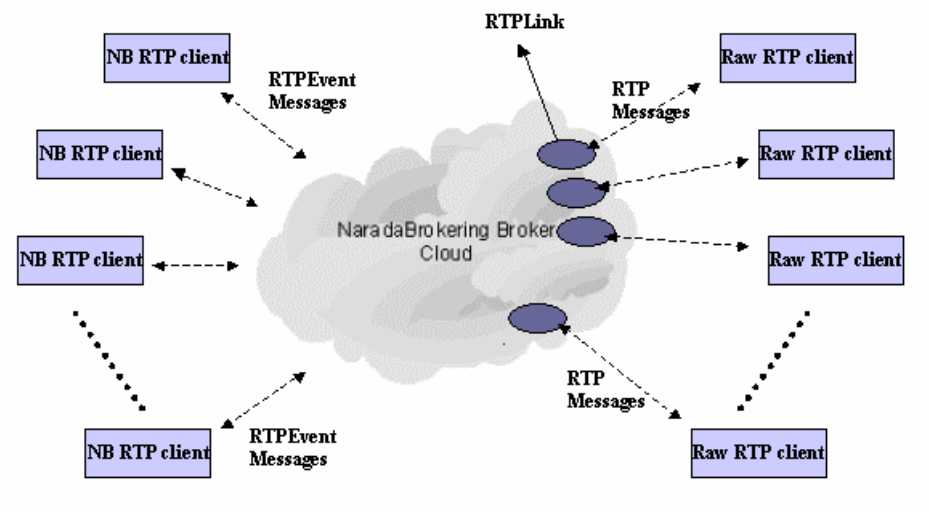

Figure 2: Conferencing in NaradaBrokering

5.0 Performance Measurements

Since NaradaBrokering is a Java-based messaging system, we contrast the performance of the NaradaBrokering broker with a conferencing server written in Java using JMF [18] libraries. This server creates an audio or video session and delivers the audio or video streams it receives to all participants except the sender itself.

In our measurements we create 1 audio or 1 video conference at a time. In each case, we test the results for different number of participants. One participant sends an audio or video stream to the server, and the server delivers it to all the other participants. In each test we have send 2000 audio or video packages. To compute transit delays for each delivered package, we assign a package-id to every package and track the send/receive times for each package. For NaradaBrokering we used the RTP sequence number as package id, but for JMF we could not use this since JMF conferencing server modifies the sequence number and timestamp. We instead compute a unique package-id from the audio/video data that is sent out.

Although we have hundreds of audio/video receiver clients in a conference, we gathered results only from 12 video and 30 audio clients. These clients and the sender client ran in the same machine and rest of the receiver clients ran in another machine. This way we avoid the clock synchronization/drift issues that arise when we incorporate results from receivers running on different machines. On the other hand, having all receivers running on the same machine as the sender, would introduce application overhead that would cloud the metrics we wish to measure. Having only 12 video or 30 audio receivers ensures that the overhead alluded to earlier are minimum. In addition, to make sure that the results which we gather reflect the true performance of the server, we gather video results from first 4, middle 4 and the last 4 clients that were added to the conference. In the audio case results are retrieved from the first 10, middle 10 and last 10 clients. Finally, when we performed one of our benchmarks on 4 machines by hosting one-half of the non-measuring receivers on the fourth machine, the results were similar to the ones we report in this paper.

5.1 The Video Test

We have created a video conference on the server machine – 1.2GHz Intel Pentium III dual CPU, 1GB MEM, RedHat Linux 7.3. One of the participants sends a H.263 video stream to the server, which delivers it to all the other participants in the session. The sender client and the 12 receiver clients, from whom we gather results, were running on a 2.4GHz Intel Pentium 4 CPU, 512GB MEM, RedHat Linux 7.3. The video stream had an average bandwidth of more than 600 kbps. We calculated the transit delay and jitter values for each video package. The sender application sends 2000 packages in each test. We used the same video stream for each test. The 3 machines involved in this test reside on a gigabit subnet. For every

package we calculated the transit delay, (receivedTime – sentTime), for all 12 clients. We then get the average of these 12 delay values for that package in milliseconds. We also calculate jitter for each package based on the formula explained in RTP RFC [1]. We then get the average jitter for the 12 clients.

Number of clients	NBrokering Avg Delay	JMF Avg Delay	NBrokering Avg Jitter	JMF Avg Jitter	NBrokering loss rate	JMF loss rate	Total bandwidth
50	2.23ms	3.08ms	0.95	1.10	0.0%	0.0%	30 Mbps
100	7.20ms	10.72ms	2.57	3.34	0.0%	0.0%	60 Mbps
200	20.38ms	27.69ms	6.18	7.56	0.0%	0.4%	120 Mbps
300	42.61ms	60.86ms	9.93	11.84	0.0%	0.7%	180 Mbps
400	80.76ms	229.23ms	13.38	15.55	0.2%	6.0%	240 Mbps

Table 1: Video streaming performance results

Table 1 summarizes the video tests which were conducted. It shows the average transit delays, average jitter values and average loss rates for the NaradaBrokering broker and JMF conferencing server. It also shows the total bandwidth used to transfer the test video stream to participants involved in the tests. The tests clearly demonstrate that the NaradaBrokering broker out performs the JMF conferencing server in every aspect, and is capable of delivering a video stream to more than 400 participants.

It should be noted that the bandwidth requirement of our test video stream is quite high and most real-time videoconferencing result in streams, whose bandwidth utilizations, fall in the 100-200kbps range. A single NaradaBrokering broker can thus deliver up to 1200 real-time video clients. It should be noted that the natural setting for NaradaBrokering is a distributed broker network, with at least one broker in every domain. Typical broker settings will thus handle significantly larger client concentrations and streams.

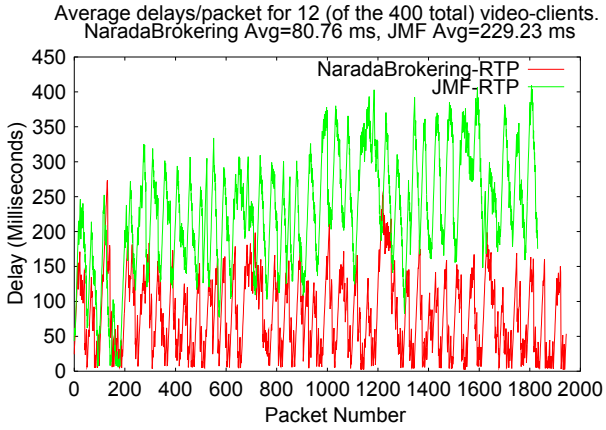

Figure 3: Delays for 400 video clients

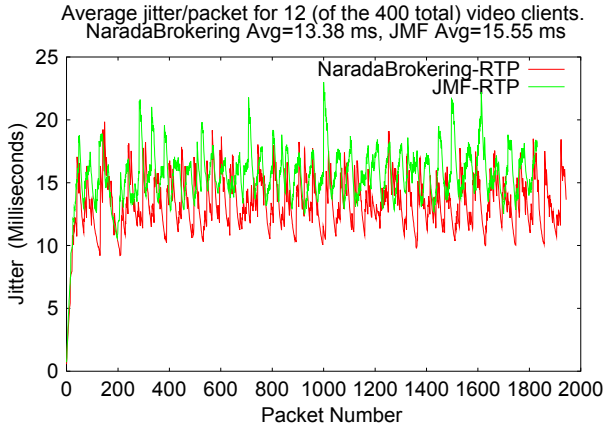

Figure 4: Jitter for 400 video clients

Figures 3 and 4 show the average transit delays and jitter values, of the 12 measuring clients for 1950 packages, respectively. We ignore the first 50 packages, where delays correspond to application start ups. From the graphs it is clear that NaradaBrokering delivers significantly better performance and is stable over time. The transit delay values are all in the acceptable range for video.

In the JMF conference server the first client that joins the session consistently gets the best performance with decreasing performance until the last client gets the worst performance. In the NaradaBrokering broker, we provide equal performance to all clients by rotating the delivery privilege among receivers for alternate packages. As a result, over time, all NaradaBrokering clients get equal performance.

5.2 Audio Test

We have created an audio conference on the server machine, 1.2GHz Intel Pentium III dual CPU, 1GB MEM, RedHat linux 7.3. One participant sent a ULAW audio stream to the server, which delivers it to all participants in the session. The sender client and 30 receiver clients, from whom we gather results, were

running on 2.2GHz Intel Xeon dual CPU, 1GB MEM, RedHat Linux 7.3. The audio stream has the bandwidth of 64kbps. Every 60ms, a 480 bytes audio package is sent. We calculate the latency and jitter values for each audio package. The sender client sends 2000 packages in each test. The three machines used in this test reside on a 100Mbps subnet.

Number of clients	NBrokering Avg latency	JMF Avg latency	NBrokering Avg Jitter	JMF Avg Jitter	NBrokering loss rate	JMF Loss rate	Total bandwidth
100	2.89ms	2.77ms	0.56	0.61	0.0%	0.0%	6.4 Mbps
500	12.02ms	11.80ms	0.59	0.56	0.0%	0.0%	32 Mbps
1000	23.44ms	23.01ms	0.79	0.47	0.0%	0.0%	64 Mbps

Table 2: Audio streaming performance results

Table 2 shows that the NaradaBrokering broker is capable of sending an audio stream to 1000 clients by providing a very good quality. The 23 ms delay introduced by the broker is well with the real-time constraints imposed on audio communications. Since we run this test in a 100 Mbps network, when we increase the number of clients we hit the network bottleneck. We are therefore providing results only for up to 1000 clients.

5.3 Real-time Videoconferencing test

We had an online meeting in our 100Mbps network with a group of 30 participants for almost 2 hours. We used one NaradaBrokering broker to deliver the audio and video streams to all clients. Participants used VIC/RAT as the audio/video clients respectively. One person was speaking throughout the meeting and the rest were listening. His audio was delivered as 64kbps ULAW. Most of the participants had cameras and sent video streams to the meeting. At any given time there were 15-20 different video streams in the meeting. All video was in H.261 format and most of them had their bandwidth changing from 50-200kbps. The broker was running in a 1.2GHz Pentium III dual CPU, 1GB MEM, RedHat Linux 7.3 machine. We had excellent quality video and audio throughout the meeting.

6.0. Future Work & Conclusions

We plan to investigate the dynamic management of conferencing sessions within the broker network. Similarly the affects of incorporating audio-mixing capabilities as an extension to the broker needs to be researched further. Integration of the NaradaBrokering into the XGSP A/V Web-Services framework outlined in [19] is an ongoing effort.

More significantly, we plan to investigate the performance boosts that a migration to the JDK-1.4 New IO library would provide. This would be from an engineering stand point, where we would be utilizing the buffering and thread management capabilities provided by this high performance library. We expect the ability of individual brokers, to manage large client configurations, to improve substantially by deploying this solution.

In this paper we have shown that distributed brokering systems are suitable for transferring audio/video streams on the Internet. It is also very convenient to implement videoconferencing in these settings. Our tests have shown that this approach works and that our NaradaBrokering broker can handle real time sessions very well.

References

1. RTP: A Transport Protocol for Real-Time Applications (IETF RFC 1889) http://www.ietf.org/rfc/rfc1889.txt.
2. Almeroth, Kevin C., “The Evolution of Multicast: From the Mbone to Interdomain Multicast to Internet2 Deployment.”, IEEE Network, 2000.
3. The NaradaBrokering System http://www.naradabrokering.org
4. A Middleware Framework and Architecture for Peer-to-Peer Grids. Shrideep Pallickara and Geoffrey Fox (To appear) Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
5. NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Geoffrey Fox and Shrideep Pallickara. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April’03.

6. The Narada Event Brokering System: Overview and Extensions. Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, June 2002. pp 353-359.
7. A Scaleable Event Infrastructure for Peer to Peer Grids. Geoffrey Fox, Shrideep Pallickara and Xi Rao. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.
8. "JMS Compliance in the Narada Event Brokering System." Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.
9. Hasan Bulut, Geoffrey Fox, Shrideep Pallickara, Ahmet Uyar and Wenjun Wu, Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service. Proceedings of IASTED International Conference on Communications, Internet, and Information Technology, 2002.
10. D. Chambers, G. Lyons, J. Duggan, "Stream Enhancements for the CORBA Event Service", Proceedings of the ninth ACM international conference on Multimedia, 2001, Ottawa, Canada.
11. International Telecommunication Union, "Packet based multimedia communication systems", Recommendation H.323, Geneva, Switzerland, Feb. 1998.
12. J. Rosenberg et al., "SIP: Session Initiation Protocol", RFC 3261, Internet Engineering Task Force, June 2002, http://www.ietf.org/rfc/rfc3261.txt,
13. K. Singh, G. Nair, and H. Schulzrinne. Centralized conferencing using SIP. In Internet Telephony Workshop 2001, New York, Apr. 2001.
14. Geoffrey Fox et al. Grid Services For Earthquake Science. Concurrency & Computation: Practice and Experience. Special Issue on Grid Computing Environments. Volume 14:371-393.
15. A Security Framework for Distributed Brokering Systems. Pallickara et. al Available from http://www.naradabrokering.org
16. Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. Java Message Service Specification. 2000. http://java.sun.com/products/jms
17. A Transport Framework for Distributed Brokering Systems. Pallickara et. al (Under review)
18. Sun Microsystems, Java Media Framework 2.1, http://java.sun.com/products/java-media/jmf/2.1.1/index.html, 2001
19. Fox et. al. A Web Services Framework for Collaboration and Audio/Videoconferencing, Internet Computing(IC'02), June 2002, Las Vegas.