Emma Middleware: An Application-level Multicast Infrastructure for Multi-party Video Communication

Takeshi Yamashita[1] Hirozumi Yamaguchi[2] Keiichi Yasumoto[3] Teruo Higashino[2] Kenichi Taniguchi[2]

[1]Graduate School of Engineering Science, Osaka University
[2]Graduate School of Information Science and Technology, Osaka University
{h-yamagu, higashino, taniguchi}@ist.osaka-u.ac.jp
[3]Graduate School of Information Science, Nara Institute of Science and Technology
yasumoto@is.aist-nara.ac.jp

ABSTRACT

In this paper, we propose new middleware written in Java based on an application level multicast protocol. This middleware is designed for multi-party video communication systems such as video chatting systems where multiple video streams are multicast simultaneously on overlay networks. The main contribution of our middleware is that it provides two types of QoS control mechanisms on overlay networks suitable for such an application. One is *inter-stream QoS control*, which controls the number of video streams on overlay links based on priority given to those streams. Another is *intra-stream QoS control* which provides congestion control on each overlay link. Also, our middleware has a quick recovery mechanism against end hosts' failure/leave. Our experimental results have shown that our middleware could realize a video chatting application without causing serious delay and jitter, which is considered inevitable in application level multicast protocols.

KEY WORDS

Application Level Multicast, Java Middleware, Multi-party Video Communication

1 Introduction

Recently, a new kind of communication paradigm which realizes multicast communication in the application layer (called ALM: Application-Level Multicast) has had much attention, and a lot of researches for ALM have been proposed [1, 2, 3, 4, 5, 6, 7, 8]. Our research group has also proposed a new application level multicast protocol called Emma (End-user Multicast for Multi-party Applications) [9] suitable for communication systems where video data are exchanged in real-time among end-hosts. In Emma, multiple multicast streams on overlay networks are controlled in a decentralized manner based on priority given by users, in order to maximize the receivers' satisfaction.

From the application developers' viewpoints, application level multicast is an attractive solution. However, those developers are required to implement into each user application various mechanisms for multicast routing, packet relaying, membership (join/leave) management and so on. In particular, for video communication systems where multiple video streams are exchanged among users, they have to control multiple video streams which often compete with each other for limited system resources such as end-to-end bandwidth. Therefore, an infrastructure which undertakes such control on behalf of applications facilitates the development of multi-party video applications and also brings them better performance.

In this paper, we propose new middleware based on Emma protocol [9]. This middleware (called *Emma middleware*) is designed for multi-party video communication systems such as video chatting systems where multiple video streams are distributed simultaneously on overlay networks. Emma middleware consists of Java components running on end-hosts. Those components maintain overlay networks and multicast distribution trees, and relay packets from hosts to hosts. Not only those basic functionalities of application layer multicast, Emma middleware provides two different levels of QoS. One is *inter-stream QoS control* based on priority given to video streams. An application user on Emma middleware is allowed to specify priority called a *preference value* to each video stream. Then the dynamic preemption of bandwidth by the prioritized streams from the existing streams is carried out by the cooperation of components on end hosts. Note that the preemption can minimize the sum of the preference values given to the existing streams from which bandwidth is preempted, *i.e.* the total loss of preference values in case of preemption is optimal. Also *intra-stream QoS control* is supported in Emma middleware. Intuitively, this means rate control on each overlay link. If congestion is detected on overlay links, the transmission rates of video streams on the links are degraded at end-hosts which are forwarding the video streams. If the congestion still continues, the inter-stream QoS control is activated, *i.e.*, a stream with lower priority is stopped to be delivered on the overlay link. Finally, we would like to address that Emma middleware has a quick recovery mechanism against end hosts' failure/leave.

We have implemented a prototype application using Emma middleware and evaluated its performance. The experimental results have shown that each end-host could

play and forward several Motion JPEG video streams with small delay and jitter, and that main functionalities of Emma middleware such as QoS control and tree recovery could be done in reasonable time.

2 Related Work

A lot of researches about ALM have been investigated so far, and some of them have been implemented as middleware. In Ref. [5], an application level multicast communication library called ALMI has been implemented in Java. ALMI supports both reliable communication and datagram communication, and provides basic operations for constructing and maintaining shared trees among end-hosts. On the other hand, in Yoid (Your Own Internet Distribution) Project [3] which promotes unification of unicast/multicast communication and protocol stability, wrapper scripts are provided as Yoid Software for Mbone tools such as vic. The research group in Carnegie Mellon University has developed ESM, a native code toolset based on End System Multicast methodology[2]. This tool was used for distributing live video in SIGCOMM2002 conference. HyperCast 2.0 is the Java implementation based on two methodologies, HyperCast[7] and Delaunay Triangulation[8] and provides socket-like Java APIs. RelayCast[10] is middleware to aim at adapting to various applications that require different metrics (bandwidth, delay or both), by component-based design and implementation.

Emma middleware is different from any of the above approaches. Its main contribution is that it supports *multi-party video communication systems where multiple video streams are exchanged and users priority should be considered*. For such a purpose, not only providing basic functionalities of application level multicast protocol, Emma middleware also provides two different types of QoS as stated in Section 1. To our best knowledge, none of the other ALM middleware has considered such a functionality. We believe that Emma middleware facilitates the development of multi-party video applications and brings application users better performance and satisfaction.

3 Basic Functionalities

In this section, the functionalities of Emma middleware are explained briefly.

3.1 Join Management

We assume that there exists a server called a *lobby server* which keeps and manages the profiles (IP addresses, port numbers, node IDs, node affiliations etc.) of all the nodes which have already joined a session. It also keeps the profiles of video sources (resolutions, rates, codecs and content information etc.) sent by those nodes.

A node who wants to join a session first contacts the lobby server to obtain the profiles of the existing nodes, and also registers its own profile. After obtaining the profiles, the new node measures RTTs between those nodes, selects a few nodes with the smaller delays, and then tries to establish overlay links (a TCP connection for control messages, and a UDP virtual connection) with them. Consequently, the overlay networks become mesh-like ones. Note that at the time of establishment, an *overlay link capacity* (the number of streams allowed on the overlay link) is negotiated and determined between the end nodes of the overlay link.

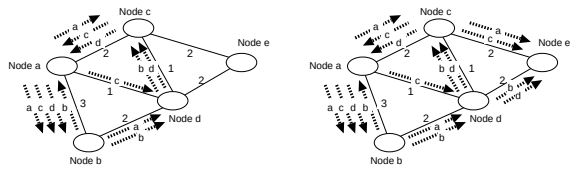


Figure 1. Routing tree expansion by a new node on overlay network.

In order to avoid to handle too many video streams, each node can specify the total sum of overlay link capacities of all its own overlay links as *capacity constraint*, depending on its own LAN interface bandwidth and machine power. For example, a node on 10Mbps Ethernet may specify the capacity as 10 (or usually less) if each stream consumes 1Mbps, since all the streams arriving at or through the node are delivered via the LAN interface even if they are delivered on different overlay links. In practice, this value should be adequately small, considering overhead and bandwidth outside LAN. The new node trying to establish overlay links negotiates with the peer nodes to determine the capacity of overlay links within the capacity constraints.

3.2 Routing Tree Construction

In Emma middleware, on an overlay network, a spanning tree rooted at each node is used as a multicast routing tree, and it is expanded whenever a new node joins the overlay network. This expansion is done depending on the constraint of maximum delay (or hop count) from the root node and overlay link capacities. Also message flooding is used to construct a routing tree for the new node.

In Fig. 1, we show a snapshot of the tree construction process when a new node joins a session. In Fig. 1(a), each of nodes a, b, c and d has its own spanning tree as the routing tree. Now, let us suppose that node e joins the overlay network by establishing overlay links to nodes c and d, and that more than two hops from root nodes are not allowed for the routing trees. Under this hop count constraint, node e can select only link c-e to expand the nodes a and c's routing trees. Similarly, node e can select only link d-e to expand the node b's routing tree. Here, node e can select either link c-e or d-e to expand node d's routing tree. However, considering the link capacity, d-e is the better choice since capacity competition is not likely to occur on d-e. Fig. 1(b) shows the result of the above expansion. The routing tree of node e is constructed by message flooding

and it is not shown in the figure.

3.3 QoS Control in Video Distribution

In Emma middleware, a node has a certain amount of *preference values* (integer values) and is allowed to assign the amount to its receiving and requesting video streams by dividing it into arbitrary portions. A preference value assigned to a video stream by a node means how strongly the node wants to receive the video. The *satisfied preference* of a node is the sum of the preference values given by the node to its *receiving streams*. The goal of QoS control provided by Emma middleware is to maximize the sum of satisfied preferences of all the nodes in a session.

For this purpose, the *inter-stream QoS control* mechanism tries to accept a request with a large preference value. When a node u requests to receive node s's video stream, node u sends a request message toward node s. The message is forwarded along the multicast routing tree of node s, checking the capacity on each overlay link on the route. When a message arrives at node s' which has already received node s's video stream, node s' knows whether each overlay link on the path from node s' to node u has a capacity to accommodate the requested stream. If so, node s' and the nodes on the route from node s' to node u start forwarding node s's video stream and node u can receive it. This is a receiver-oriented normal join procedure. If there is no extra link capacity, Emma middleware can determine, during request message forwarding, whether the preference value given to the requested stream is larger than the satisfied preference lost by preempting a capacity from the existing video streams or not. Moreover, in the former case, the optimal preemption to minimize the loss can be calculated.

Also the *intra-stream QoS control mechanism* is provided for rate adaptation. Both QoS control mechanisms are described in the next section.

3.4 Leave and Failure Management

In Emma middleware, when a node leaves a session or becomes silent (due to failure or ill mannered leave), overlay networks and multicast trees are repaired so that satisfied preference can be kept as high as possible. If a node u's leave is detected by neighbor nodes (each is denoted as v), node v determines the nodes to which node v connects in order to continue receiving video streams which had formerly been forwarded via node u. This process will also be exemplified in the next section.

4 Middleware Design and QoS Support

We have designed and implemented Emma middleware in Java (J2SDK 1.3.1 and Java Media Framework (JMF) 2.1). Emma middleware consists of a host controller running on each end host and a lobby server. The lobby server manages node profiles. We do not discuss the detail design of the lobby server in this paper since its functionality is not essential and can easily be implemented (even by a web server with CGI). In this section, we focus on the host controller configuration. Moreover, two different types of QoS control (inter- and intra-stream QoS control) are provided by Emma middleware. They are also presented in this section.

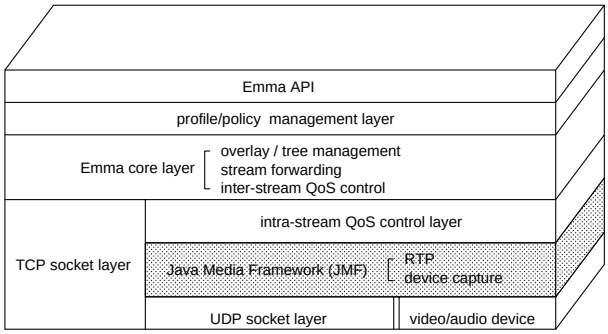


Figure 2. Host controller configuration.

4.1 Host Controller

The configuration of the host controller is shown in Fig. 2.

Profile/policy management layer keeps and manages (a) the policy of a session (*e.g.* delay constraints of a routing tree, maximum of the preference values and so on) and (b) the profiles of nodes and video sources (called node profiles and media profiles). These are obtained from the lobby server. The component of this layer monitors the user's behavior (such as requesting a new video) and checks whether the behavior follows the session's policy or not. On receiving video streams, it uses the media profiles that indicate the parameters such as bit-rate, codec and resolution.

Emma core layer is the core engine of Emma middleware. This layer manages overlay links and a multicast routing table, by exchanging control messages explained later. Especially, this layer provides the inter-stream QoS control. The inter-stream QoS control is invoked by the user's video requests or the signaling from **intra-stream QoS control layer** explained below, and decides which streams should be stopped or newly delivered.

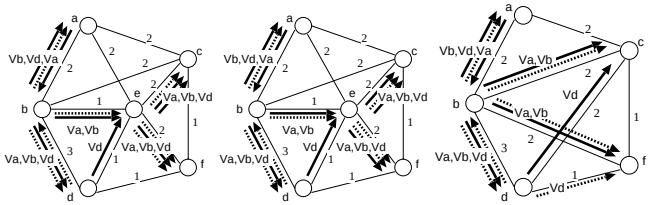
Intra-stream QoS control layer provides congestion monitor and rate control on each overlay link. Emma middleware uses RTP/RTCP[11] for the quality management of video streams. In JMF version 2.0 and later, RTP/RTCP implementation is provided and a sender node can obtain loss ratio at a receiver node by RTCP receiver reports. Moreover in JMF, the streaming rate can be controlled by the sender by allowing him/her to specify the ratio, called a *scaling ratio*, to the original bitrate in motion JPEG (thus 1.0 means the original quality). We have implemented the intra-stream QoS control layer to control this ratio dynamically according to the loss ratio feedback from the receiver.

If the congestion lasts for a long time, and a scaling ratio becomes lower than an allowable threshold, this layer sends a signal to Emma core layer. Emma core

Table 1. Control message set.

type	subtype	src	dst	payload	description
SES	JoinRequest	HC	LB	NProfile	join a session
	JoinReply	HC	LB	NID, list of NProfile	get NID and Nprofiles
	Leave	HC	LB		leave a session
	DetectLeave	HC	LB		notify unexpected leave
	GetMemberList	HC	LB	list of NProfile	get Nprofiles
	GetMediaProfile	HC	LB	NID, MProfile	get Mprofile
	SetMediaProfile	HC	LB	NID, MProfile	set Mprofile
CTRL	JoinRequest	HC	HC	link cap. limit, RTT	
	JoinReply	HC	HC	[Accept,Reject], (negotiated) link cap.	
	Leave	HC	HC		leave a session
	ReJoinRequest	HC	HC	link cap. limit, RTT	link reconstruction
	ReJoinReply	HC	HC	[Accept,Reject], (negotiated) link cap.	
MEDIA	JoinRequest	HC	HC	NID, pref	video request
	JoinReply	HC	HC	[Accept/Reject], NID, UDP port	
	Keep	HC	HC	NID, pref	keep alive and pref collection
	Leave	HC	HC		
	Advertise	HC	HC	NID, metric, seq	routing tree construction
	Reverse	HC	HC	NID, seq	add a link to a tree
	Prune	HC	HC	NID, seq	do not add a link to a tree
	ReJoinRequest	HC	HC	NID	for recovery of delivery
	ReJoinReply	HC	HC	[Accept/Reject], NID, UDP port	

HC=Host Controller, LB=LoBby server, NID=Node ID, NProfile=Node Profile, MProfile=Media Profile, seq=sequence number, pref=preference value



(a) (b) (c)

Figure 3. Overlay network topology. Dashed lines represent routing trees and solid lines represent actual video streams.

node	V_a	V_b	V_d
a	–	7	x
b	5	–	x
c	(8)	5	2
d	x	7	–
e	0	5	5
f	(8)	3	x

(a)

node	V_a	V_b	V_d
a	–	7	x
b	5	–	x
c	8	[5]	2
d	x	7	–
e	0	[5]	5
f	8	[3]	x

(b)

x=not received, (i)=requested with pref. value i, [i]=decided to be stopped

Figure 4. Preference values in Fig. 3.

layer invokes inter-stream QoS control to reduce the number of streams on the congested overlay link, in order to resolve congestion and keep the quality of the other important streams.

4.2 Control Messages

We have defined a set of control messages as shown in Table 1. Basically, we categorize those messages into three types, according to their roles. The messages of type *SES* are used for communication with the lobby server and those of type *CTRL* are used for joining/leaving overlay networks. The messages of type *MEDIA* are related with video stream delivery.

4.3 Examples: Inter-stream QoS Control and Leave Management

Here, we show two scenarios for demonstrating inter-stream QoS control and nodes' leave management.

The first scenario (scenario 1) demonstrates inter-stream QoS control. The topology of overlay network and video distribution before executing this scenario is shown in Fig. 3(a), and preference values given to video streams V_a, V_b and V_d are shown in Fig. 4(a). Note that dashed and solid arrows represent routing trees and video streams, respectively. Scenario 1 is as follows. Node c first requests V_a by using a *MEDIA/JoinRequest* message. This request is forwarded to node b but rejected due to a smaller preference value. Note that there may be a case that a request with a small value is acceptable if it is handled with its following requests of the same video stream by other nodes. Therefore, in Emma middleware, to avoid frequent submission of request retries and to increase request acceptance ratio, a rejected request is cached for a certain period at each node. In this case, the request by node c was cached at nodes b and e. Then node f requests V_a. When the *MEDIA/JoinRequest* message arrives at node e, the request is merged with node c's one which had been cached at node e and a new *MEDIA/JoinRequest* is forwarded to node b with the merged preference value 16. As a result, node b can determine whether V_a is accepted or not, and in this case, it is accepted since there is a way to keep a capacity with loss 13 by stopping V_b. This decision is informed to nodes c and f through the intermediate node e by *MEDIA/JoinReply* messages, and node e stops V_b and starts forwarding V_a to nodes c and f (Fig. 3(b) and Fig. 4(b)).

Scenario 2 considers the situation just after scenario 1. Node c is receiving two streams V_a and V_d and node f is receiving V_a from node e, as shown in Fig. 3(b). In this situation, node e leaves the session. Then nodes c and f send *CTRL/ReJoinRequest* messages to nodes which had forwarded those video streams to node e, in order to establish new overlay links with them. On these overlay links, routing trees are expanded in the way as we explained in Section 3.2 using *MEDIA/Advertise* messages. Finally, nodes c and f send *MEDIA/ReJoinRequest* in order to continue to receive V_a and V_d. The result of this scenario is shown in Fig. 3(c).

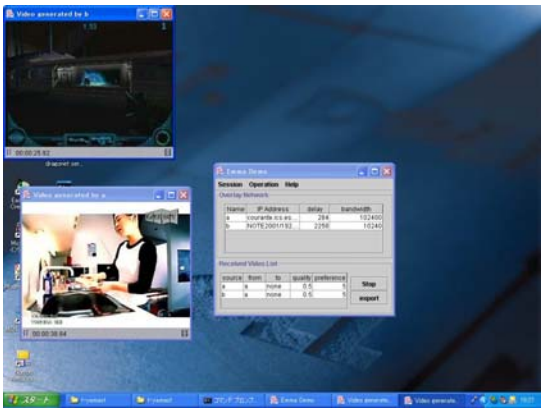
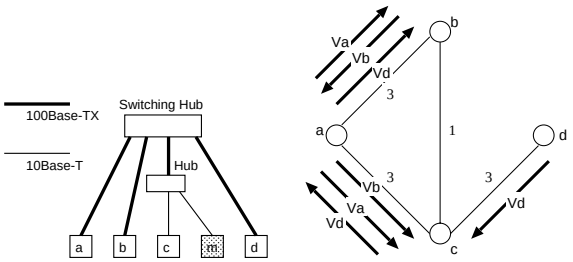


Figure 5. Sample application snapshot.



(a) underlying network (b) overlay topology

Figure 6. Networks in experiment 1.

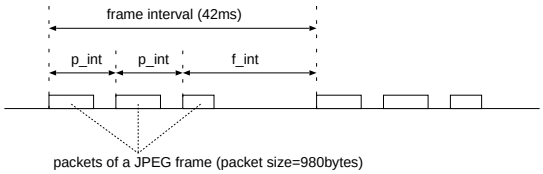


Figure 7. Video packets and frames.

5 Experiments

Using Emma middleware, we have developed a simple video-chatting system where Motion JPEG live video (via USB video cameras) or stored video can be exchanged. A snapshot of the system is shown in Fig. 5. This system was used in the following two experiments.

Each host in Emma middleware, which may be a consumer level's PC, is required to forward and playback multiple video streams simultaneously as well as to manage overlay networks. Here, the basic requirement for Emma middleware is that each host can do such a forwarding job without causing serious forwarding delay and jitter at the host, even though all the programs are written in Java which is considered to have a performance disadvantage compared with the languages compiled to native codes such as C. For the purpose, in the first experiment (experiment 1), we have measured forwarding delay and jitter at an end host. Then, in the second experiment (experiment 2), we have demonstrated two scenarios explained in Section 4.3, (a) inter-stream QoS control and (b) nodes' leave management, in order to measure that they are done in reasonable time.

The hosts used in the experiments are Windows 2000 PCs with PentiumII 400MHz CPU and 256MB memory.

5.1 Experiment 1 : Forwarding Delay and Jitter at End Host

In Experiment 1, we have measured frame jitter and forwarding delay at an end host. We have activated all the functionalities in Emma middleware as well as video playout windows in the sample application.

The sample application was executed on the network shown in Fig. 6(a) and the overlay network shown in Fig. 6(b) was constructed. We have let nodes a, b and d be video senders. The measurement was carried out three times, using different transmission rates' Motion JPEG video sources (1Mbps, 1.4Mbps and 2.4Mbps) of 23.8fps (thus their frame sizes are 5.3KB, 7.5KB and 12.5KB, respectively).

Since each JPEG frame is fragmented into UDP packets of 980bytes (this is due to RTP implementation in JMF), we have measured the intervals between packets within a frame (p_int) and the intervals between the last packet of a frame and the first packet of the next frame (f_int) (see Fig. 7). The measurement was done by tcpdump at node m by monitoring video stream V_d incoming/outgoing through node c. We have also measured the delay required at node c to forward V_d. The results are as follows.

rate (Mbps)	p_int (ms)		f_int (ms)		$delay$ (ms)	
	in	out	in	out	ave.	max
1.0	0.34	0.35	40.43	40.46	10.3	23.0
1.4	0.34	0.35	39.52	39.53	11.7	39.0
2.4	0.33	0.33	38.11	38.13	15.0	32.0

We can see that the increasing rates of the intervals by node c's forwarding are very small. Also, delays are small enough compared with overlay link delay, which is in general, the order of tens or hundreds of milliseconds.

From the results above, Emma middleware does not cause serious delay and jitter even in a practical situation where actual video streams are forwarded and played back simultaneously.

5.2 Experiment 2: Emma Functionality Demonstration

In experiment 2, we have used 6 PCs and have demonstrated the two scenarios in Section 4.3. The underlying network is shown in Fig. 8. Throughout the scenarios, we have used video streams V_a, V_b and V_d of 1.0Mbps.

The result of the execution of scenario 1 is shown in Fig. 9. Again we briefly explain the scenario. The node c first requested V_a. This request was forwarded to node b but rejected due to a smaller preference value. This request was cached at nodes e and b. 2 seconds later, node f requested V_a. The request was merged at node e with the cached request of node c and was forwarded to node b. As a result,

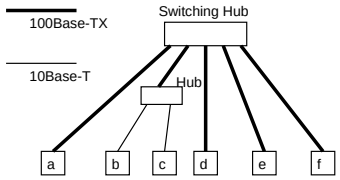


Figure 8. Underlying network in experiment 2.

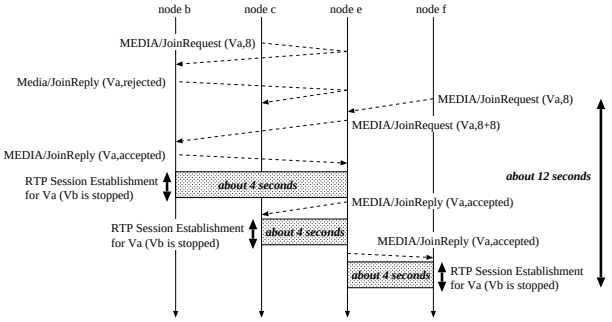


Figure 9. Timing chart for scenario 1.

V_b was stopped and V_a was started to be forwarded to nodes c and f.

In this scenario, it took 12 seconds for node f to start playing V_a. Note that our current implementation of Emma middleware uses the RTP implementation in JMF for video transmission on each overlay link. Unfortunately, this RTP implementation blocks other operations for 3 or 4 seconds until the establishment is completed. Considering this fact, inter-stream QoS control itself was done very quickly. However, in order to shorten the response time for such an application that needs quick response for a request, we are now designing and implementing an original RTP module that is more lightweight.

In scenario 2, node e in Fig. 3(b) left the session. Then node c could continue to play V_a and V_d and node f could do V_a as shown in Fig. 3(c). The timing chart is shown in Fig. 10 where *MEDIA/Advertise* messages are omitted to make the figure more readable. Here, after node e's leave, it took about 11 seconds for node f to receive V_a again. This was also largely affected by session initiation time of RTP. The leave management process itself was done quickly.

6 Conclusion

In this paper, we have presented Emma middleware for multi-party video communication based on our application level multicast protocol Emma (End-user Multicast for Multi-part Application). Emma middleware has functionalities helpful for developing such an application where multiple real-time video streams are exchanged among users.

The current version of Emma middleware package can be found in: `http://www-tani.ist.osaka-u.ac.jp/software/emma-e.html`.

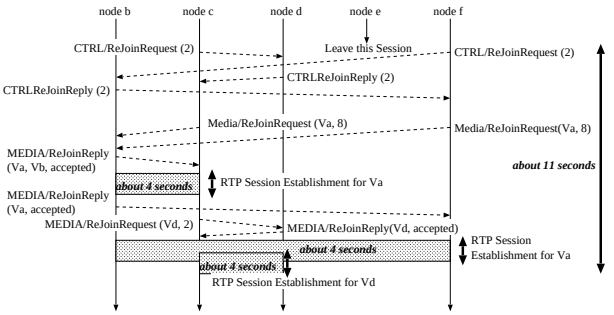


Figure 10. Timing chart for scenario 2.

References

[1] Y.-H. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proc. of ACM SIGCOMM*, 2001.

[2] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, 2000. Tools are provided: `http://www-2.cs.cmu.edu/~streaming/index.html`.

[3] P. Francis. Yoid: Extending the internet multicast architecture. `http://www.isi.edu/div7/yoid/`, 2002.

[4] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. of 4th Usenix Symp. on Operating Systems Design and Implementation (OSDI)*, 2000.

[5] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. of 3rd Usenix Symp. on Internet Technologies and Systems*, 2001.

[6] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of 3rd Int. Workshop on Networked Group Communication*, 2001.

[7] J. Liebeherr and T. K. Beam. HyperCast: A protocol for maintaining multicast group members in a logical hypercube topology. In *Proc. of 1st Int. Workshop on Networked Group Communication (NGC '99) (LNCS 1736)*, pages 72–89, 1999. Tools are provided: `http://www.cs.virginia.edu/~mngroup/hypercast/index.html`.

[8] J. Liebeherr and M. Nahas. Application-layer multicast with delaunay triangulations. In *Proc. of IEEE Globecom 2001*, 2001.

[9] Y. Nakamura, H. Yamaguchi, A. Hiromori, K. Yasumoto, T. Higashino, and K. Taniguchi. On designing end-user multicast for multiple video sources. In *Proc. of 2003 IEEE Int. Conf. on Multimedia and Expo (ICME2003)*, pages III497–500, 2003.

[10] N. Mimura, K. Nakauchi, H. Morikawa, and T. Aoyama. RelayCast: A middleware for application-level multicast services. In *Proc. of 3rd Int. Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems (GP2PC 2003)*, 2003.

[11] H. Schulzrinne, R. Frederick S. Casner, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, 1996. RFC 1889.