

Functional Unit Oriented Middleware for Application-Level Multicast Services

Nodoka MIMURA^{†a)}, Student Member, Kiyohide NAKAUCHI^{†*}, Hiroyuki MORIKAWA^{††}, Members, and Tomonori AOYAMA^{†††}, Fellow

SUMMARY Application-level multicast (ALM) is a novel technology for multipoint applications, such as large scale file distribution, video and audio streaming, and video conferencing. Although many ALM mechanisms or algorithms have been proposed, all the multicast functions have been independently developed and integrated into individual applications. In such a situation, the development of ALM applications includes a lot of redundancy. Our goal is to improve the efficiency of developing ALM applications by reducing the development redundancy and to provide application developers with a middleware on which various ALM applications can be efficiently developed with minimum efforts. To this end, we develop a functional unit oriented ALM middleware, namely *RelayCast*. *RelayCast* provides a minimum but fundamental set of functionality as a *functional unit*, and constructs the basis on which additional and specific functions (i.e. codec, video capture, etc.) for each application are implemented. Some functional units contain several *components* with different algorithms, and *RelayCast* meets the requirements of various applications by choosing the appropriate component. In this paper, we propose *RelayCast* architecture, and present the implementation and experiments of a prototype.

key words: application-level multicast, middleware, Peer-to-Peer network, overlay network

1. Introduction

Efficient and scalable multicast technology is indispensable for multipoint applications, such as large-scale file distribution, video-on-demand, and video conferencing. Multicast could be achieved either in a network layer (switches and routers) or in an application layer (end-hosts). The former is called “IP multicast” and the latter is called “application-level multicast (ALM).”

The concept of IP multicast is accepted widely, and its functionality is currently implemented in most routers. However, IP multicast has many fundamental problems inherent in its original architecture, including inter-domain routing, group management, multicast address assignment, reliability, congestion control, flow control, security, and lack of business model [1]. These issues have prevented

the deployment of IP multicast on the Internet. While there are attempts to partially address some of these issues, they further complicate a network layer.

On the other hand, the recent rapid growth of computation and network resources for end-hosts has enabled Peer-to-Peer (P2P) computing and networking, and multipoint communication using ALM [2]–[7]. In ALM, an application layer provides multicast functionality, that is, packet duplication, multicast routing, and group management. End-hosts self-organize into an overlay network, and transfer data along the overlay network using unicast transport services.

In the past, ALM has been actively researched and many ALM mechanisms (or algorithms) have been proposed [8]–[18]. Those mechanisms and algorithms can meet almost all the fundamental requirements of various applications. We believe that, in the future, it will be necessary to consider how to develop and deploy ALM systems from the practical perspective, and a crucial issue will be how to efficiently implement ALM applications. It is redundant and time-consuming that all the multicast functions have been independently developed and integrated into individual applications in spite of sharing an application design and several common functions.

Our goal is to improve the efficiency of developing ALM applications by reducing the development redundancy and to provide application developers with a middleware on which various ALM applications can be efficiently developed with minimum efforts. To this end, we develop a functional unit oriented ALM middleware, namely *RelayCast*, which provides not only a minimum but fundamental set of functionality as a unit but also capability of flexibly introducing additional algorithms. Application-specific functions, such as video and audio capture, codec, and multi-windows, are implemented on the basis which *RelayCast* constructs.

The *RelayCast* design is originally motivated by our key observation that in previous ALM systems, we can mainly abstract two functions; “overlay network construction” and “multicast routing.” We define each of the functions as a *functional unit*. Each previous ALM system differs from others in the operation of functional units. For example, End System Multicast (ESM) [3] constructs a mesh overlay network adaptive to end-to-end delay with NARADA protocol, and then builds a multicast tree using the shortest widest path algorithm. In Scattercast [4], a mesh

Manuscript received March 31, 2005.

Manuscript revised June 28, 2005.

[†]The authors are with the School of Engineering, The University of Tokyo, Tokyo, 113–8656 Japan.

^{††}The author is with the School of Frontier Sciences, The University of Tokyo, Tokyo, 113–8656 Japan.

^{†††}The author is with the School of Information Science and Technology, The University of Tokyo, Tokyo, 113–8656 Japan.

*Presently, with National Institute of Information and Communications Technology.

a) E-mail: mimura@mlab.t.u-tokyo.ac.jp

DOI: 10.1093/ietcom/e88-b.12.4442

overlay network is constructed according to delay and link stress, and a multicast routing protocol like DVMRP (distance vector multicast routing protocol) [19] runs on the overlay network.

Some functional units contain several *components*, each of which corresponds to individually proposed algorithm. For instance, we can regard ESM as applying the component to calculate the link cost based on end-to-end delay and the component of NARADA protocol to decide an overlay network topology in the overlay network construction function. And in the multicast routing function, ESM applies the component of the shortest widest path algorithm. Such an approach is also used in Building Blocks [20] for the reliable multicast discussed in IETF RMT WG (Reliable Multicast Transport Working Group) [21]. Conceptually, our components are equal to Building Blocks.

Basic design of functional units and components described above enables application developers to flexibly add application specific functions or algorithms to ALM applications. However, further division of the main two functional units could make the middleware more smart and eliminate redundancy, because these units share some modular functions. Those are the communication function to send/receive packets to/from other end-hosts, the database function to store the information of other end-hosts, and the estimation function to measure the network state used as metrics. So we separate these shared modular functions from the main functional units, and also define each of them as a functional unit. The functions of the separated units are operated through the interfaces defined between the functional units.

In this paper, we implement the prototype of RelayCast on the basis of the above design concept, and experiment with the prototype on a local area network. One of the purposes of our experiments is to study whether the unpractical performance bottleneck is caused by the division of functionality. Our motivation stands on a practical viewpoint, and so we believe that it is advisable to demonstrate the practicability of our middleware architecture through the implementation and experiments.

The main contributions of this paper are the followings. Firstly, we abstract two fundamental functions, divide shared modular functions, and design an ALM middleware, RelayCast, which consists of five functional units; that is, the overlay network construction function, the multicast routing function, the communication function, the database function, and the estimation function. Secondly, we implement the prototype of RelayCast based on our design concept. Finally, we verify the practicality of RelayCast through our experiments.

The rest of this paper is organized as follows. In Sect. 2, we abstract ALM functionality and discuss basic components required for each function. In Sect. 3, we describe the RelayCast architecture. In Sect. 4, we show the implementation of a prototype. In Sect. 5, we report preparatory results of our experiment and discuss the practicality of RelayCast. In Sect. 6, we mention related work, and we conclude this

paper in Sect. 7.

2. Abstraction of ALM Functionality

2.1 Application-Level Multicast

Previous ALM systems are mostly classified into two groups, mesh-first approaches [2]–[4] and tree-first approaches [5], [7], [9], [10]. This paper assumes the mesh-first approach. In the mesh-first approach, end-hosts construct a mesh overlay network autonomously, and perform a multicast routing protocol for building a multicast tree on the overlay network. On the other hand, in the tree-first approach, end-hosts directly build a multicast tree without constructing a mesh overlay network. We believe that the mesh-first approach avoids repeated group management across multiple trees, enables the adoption of simple multicast routing protocols, maintains the quality of entire groups because of the ease of tree optimization, and provides a structure more resilient to end-host's failure.

The operation of ALM consists of three fundamental functions: 1) initial end-host discovery function, 2) overlay network construction function, and 3) multicast routing function. Firstly, in the initial end-host discovery, an end-host to attempt participating in a multicast session finds other end-hosts in the session. The initial end-host discovery is achieved by any of three approaches, that is, the bootstrap end-host advertised by external mechanisms like web pages, the directory server to store the identifiers of participating end-hosts, or the flooding search using broadcast channels.

Secondly, in the overlay network construction, each end-host gets the list of participating end-hosts, selects some neighbors from the list, and establishes logical links with them. A set of these logical links constitutes an overlay network. Then, each end-host optimizes the overlay network to seek a better connection environment. End-hosts periodically calculate the logical link costs to other end-hosts, add new useful logical links according to the costs, and drop logical links perceived as uselessness. Each previous ALM system has the individual algorithms for the choice of neighbors and the calculation of logical link costs.

Finally, in the multicast routing, a logical multicast tree for data delivery is built by a multicast routing protocol on the optimized overlay network. Figure 1(a) depicts a logical delivery tree, and Fig. 1(b) shows actual flow on a physical network. Previous ALM systems have different multicast routing protocols.

2.2 ALM Middleware

ALM may complicate the process of application development. One reason is that application developers must take care of the end-hosts' behavior of routing elements. In other words, it is necessary to repair the partition of an overlay network and a multicast tree caused by end-host's failure, such as congestion at access link, application errors and

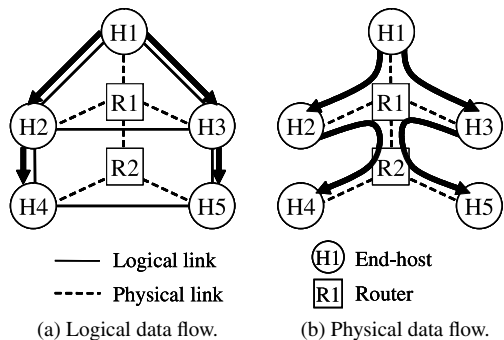


Fig. 1 Delivery tree on application-level multicast.

computer crashes. The partition prohibits end-hosts to receive application data. Another reason is that the developers must fully understand the algorithms applied to ALM systems for the implementation. However, this means that ALM can meet the requirements of applications flexibly.

Such a situation disturbs the deployment of ALM in the Internet. In case of IP multicast, which includes many problems as we argued in Sect. 1, application developers can use its functionality only by setting socket options, and they need not pay attention to router’s failure. We believe that ALM should have almost the same usability as IP multicast in the stage of application development.

In view of above consideration, the introduction of an ALM middleware will be helpful for developing and deploying ALM applications. It is necessary that the middleware provides the three functions mentioned in Sect. 2.1. However, an external discovery mechanism is generally required in the initial end-host discovery. We assume that end-hosts to attempt joining in a multicast session have already known bootstrap end-hosts. The integration of the initial end-host discovery function into the middleware is a future work. We therefore abstract the overlay network construction function and the multicast routing function.

2.3 Overlay Network Construction Function

Table 1 denotes the basic components contained in the overlay network construction function. In this function, algorithms to decide optimization metrics and to form overlay network topology organize components. The overlay network topology has influence on the choice of neighbors, and the optimization metric becomes criteria for the calculation of logical link costs.

Bandwidth and delay are usually candidates for the optimization metrics. Bandwidth is estimated by active measurement (pchar/packet pair, etc.) or passive measurement (by measuring a receiving rate, etc.). Delay is measured as RTT (Round Trip Time) to other end-hosts [2] or landmarks [22].

The choices for the overlay topology are full flat topology and hierarchical topology. The full flat topology, in which all end-hosts have equal relationships, is suitable for the case where the resources of end-hosts are nearly level.

Table 1 Components of overlay network construction function.

Component		Applied situation
Optimization Metric	Delay	To guarantee delay
	Bandwidth	To guarantee bandwidth
	Delay and bandwidth	To guarantee delay and bandwidth
Topology	Full flat	Resources of end-hosts are nearly equal
	Hierarchical	Resources of end-hosts are unequal / Group is relatively large

Table 2 Components of multicast routing function.

Component	Applied situation
DVMRP	A few source end-hosts
PIM-SM/CBT	Many source end-hosts
Multi-path routing protocol	Continuous communication
Shortest widest path algorithm	Bandwidth guarantee and delay guarantee

The hierarchical topology is suitable for the case where the resources of end-hosts are unequal and the group size is relatively large.

2.4 Multicast Routing Function

Table 2 presents the basic components contained in the multicast routing function. In this function, algorithms to build multicast trees are components. Shortest path tree, shared tree, multi-path tree, and shortest widest path tree (SWPT) are enumerated as kinds of multicast trees.

The shortest path tree that minimizes a logical hop count from a source end-host on an overlay network is built by a protocol like DVMRP [19], and fits the multicast session of a few source end-hosts. The shared tree, which uses a super node (one of participating end-hosts) as a rendezvous point and organizes only one tree, is built by a protocol like PIM-SM (protocol independent multicast-sparse mode) [23] or CBT (core based trees) [24]. This is suitable when every end-host can be source end-hosts. However, the process of electing super nodes is complicated, since unreliable end-hosts work as rendezvous points. The multi-path tree is built by the multi-path routing protocol [25]. The redundant tree paths which are substituted when tree partition is caused by end-host’s leave or failure are beforehand given to each end-host. This means the fast rebuilding of a delivery tree at the time of end-host’s failure, and so the multi-path routing protocol is suitable for continuous communication. In the SWPT, the tree route that keeps bandwidth above a certain value and minimizes a logical hop count is chosen [3]. The tree route can guarantee the bandwidth and, to some extent, delay.

3. RelayCast

3.1 Architecture

We adopt as the design principle the following approach; we define the fundamental functions of ALM as functional units, and then we divide shared modular functions from those units. The functional units of the fundamental functions are the overlay network construction function and the multicast routing function. These units share some modular functions, that is, the communication function to send/receive packets to/from other end-hosts, the database function to store the information of other end-hosts, and the estimation function to measure the network state. We divide the shared modular functions from the two main functional units, and define them as the functional units of P2PCom, HostList, and MetricEstimator respectively.

We redefine the overlay network construction function and the multicast routing function from which we separated the shared modular functions as LogicNet and McastTree respectively. LogicNet and McastTree contain several components, each of which corresponds to individually proposed algorithm. RelayCast can meet the requirements of various applications by selecting appropriate components.

Moreover, we define the interfaces between functional units which are programming interfaces to use the functions of separated units and socket API seamlessly. The interfaces help application developers to introduce additional components into RelayCast.

The architecture of RelayCast is illustrated in Fig. 2. The area within the rectangle of dashed lines represents RelayCast. As mentioned above, RelayCast consists of five functional units and interfaces between them. Single-headed arrows represent the operations of the interfaces between functional units, and double-headed arrows show the data exchange caused by the operations of the interfaces.

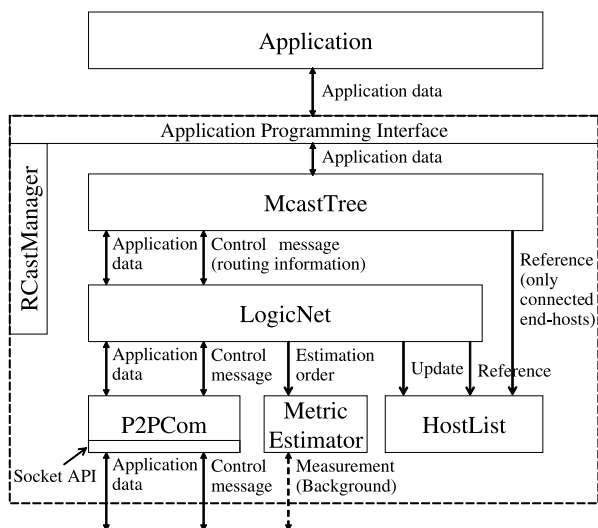


Fig. 2 RelayCast architecture.

RCastManager loads the configuration file which contains the information, such as the selection of components, and manages functional units according to the file. RCastManager is generated for every multicast session.

3.2 Functional Units and Interfaces

We describe the role of each functional unit, and define the interfaces which each functional unit has.

McastTree: McastTree builds a multicast tree and operates packet routing. McastTree specifies downstream end-hosts, and throws application data and control messages (routing information), to P2PCom via LogicNet. If an end-host itself is a source end-host, McastTree adds a sequence number to the application data. Each end-host checks the sequence number to avoid a loop.

McastTree has *SendMsg/RecvMsg* to send/receive routing information to/from connected end-hosts, *SendDat/RecvDat* to send/receive application data to/from connected end-hosts, and *GetCnnHostInfo* to refer the information of connected end-hosts.

LogicNet: LogicNet maintains and optimizes an overlay network. The most important task is to establish logical links on the basis of the results measured by MetricEstimator. LogicNet selects correspondents from HostList for sending control messages including list exchange, join/leave and link connect/disconnect, and passes the messages to P2PCom. When LogicNet receives control messages from other end-hosts via P2PCom, it transacts the messages or passes the messages to McastTree. If LogicNet deals with application data, it forwards the data to P2PCom or McastTree.

LogicNet has *SendMsg/RecvMsg* to send/receive control messages to/from end-hosts on an overlay network, *SendDat/RecvDat* to send/receive application data to/from end-hosts on an overlay network, *GetHostInfo/PutHostInfo* to refer/update the information of end-hosts, and *EstimateStart* to order the estimation of the network condition in the background.

MetricEstimator: MetricEstimator estimates network condition (bandwidth/delay). Usually, the estimation process is executed in the background. It is possible to incorporate various measurement methods of RTT and bandwidth, or external programs, such as ping and pchar, into MetricEstimator independently from other functional units. Estimation results are passed to LogicNet and are kept into HostList. MetricEstimator has *Estimate* to estimate network condition.

P2PCom: P2PCom communicates with other end-hosts directly in a P2P network. P2PCom contains sockets to other end-hosts, and transmits control messages and application data to the end-host specified by LogicNet. Moreover it is possible to incorporate NAT traversal technologies, such as STUN [26] and Teredo [27], into P2PCom independently from other functional units.

P2PCom has *SendMsg/RecvMsg* to send/receive control messages to/from the network, and *SendDat/RecvDat* to send/receive application data to/from the network.

HostList: HostList manages the list of end-hosts' information including IP address, GUID, estimation result, and state (alive/dead, relation on an overlay network). The list is updated by LogicNet and is referred to by LogicNet and McastTree.

HostList has no interface.

3.3 Combination Policy

RelayCast combines appropriate components and meets the requirements of applications. At the start of a multicast session, its owner decides the combination, and other users follow the decision.

The combination of components is one of the factors which determine the performance of the ALM system. Thus, it is necessary to select a combination suitable for applications' requirements. In addition, the confliction of the selected components has to be avoided. For example, when the component adopted in the LogicNet unit sacrifices delay even though the requirement of the component selected in the McastTree unit is to guarantee delay, the confliction will occur inside RelayCast. Although a simple approach is that a user determines the components one-by-one, the user who does not understand their algorithms might choose an ineffective combination.

Another approach is to prepare a qualitative combination policy. That is, a user selects properties including communication model, group size and application, and then an appropriate combination is applied in RelayCast. Table 3 gives some combinations considered to be effective. For example, in the case of the streaming media that requires a bandwidth guarantee and delay guarantee, both bandwidth and delay are applied to optimization metrics, and DVMRP or the shortest widest path algorithm is applied to the multicast routing protocol. Moreover, if a user requires the continuous communication, that is to say, he wants to minimize the influence of the tree partition caused by end-host's failures, the multi-path routing protocol is applied. We need to examine the way to provide a combination policy in the future.

Table 3 Examples of combination policy.

Application	Overlay Network Construction	Multicast Routing
Streaming media	Bandwidth and delay (metric)	DVMRP, SWPA
Interactive application	Delay (metric)	Multiple DVMRP, Multiple SWPA, PIM-SM/CBT
Bulk-data transfer	Bandwidth (metric)	DVMRP, SWPA

(SWPA = Shortest widest path algorithm)

4. Implementation

4.1 Implementation Overview

We have implemented the prototype of RelayCast with C++ on Linux (kernel-2.4). We define the five functional units and RCastManager as classes, and give the interfaces between functional units as methods in the classes.

MetricEstimator can measure delay (smoothed RTT) between end-hosts. LogicNet has the components to make the overlay topology full flat and to use the delay as the optimization metric. Consequently, LogicNet can construct a mesh overlay network based on delay. In addition, LogicNet exchanges IP address lists using a variant of the NameDropper algorithm [4] which circulates end-host's information in gossip style. McastTree has the components of DVMRP and the multi-path routing protocol. In order to support many-to-many delivery, McastTree can build multiple source trees.

4.2 Implementation Model

For examining the operation of RelayCast, we use VIC (video conferencing tool) [28] as an application. VIC enables communication between two end-hosts using UDP unicast. We build a many-to-many video conferencing system with RelayCast and the unicast function of VIC. We don't directly integrate RelayCast into VIC but we implement RelayCast as a local proxy.

Figure 3 illustrates the relationship between an application and RelayCast in the local proxy model. In this model, the application only communicates with the middleware directly within a localhost. The middleware builds a delivery tree and transfers application data to all end-hosts in a multicast session. The advantage of this model is that we can completely separate the debug from an application, and we can concentrate the development of RelayCast. On the other hand, the disadvantage is that the performance is restricted because packets pass through socket API twice.

At the time of starting the program, a user inputs the following commands with the IP address of a known bootstrap end-host.

```
% > rcast [bootstrap IP]
% > vic localhost/[middleware port]
```

We also argue the implementation framework inside the middleware. For the design of the framework, we can choose a queue-control-based router model or an

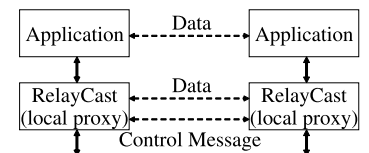


Fig. 3 Implementation model as a local proxy.

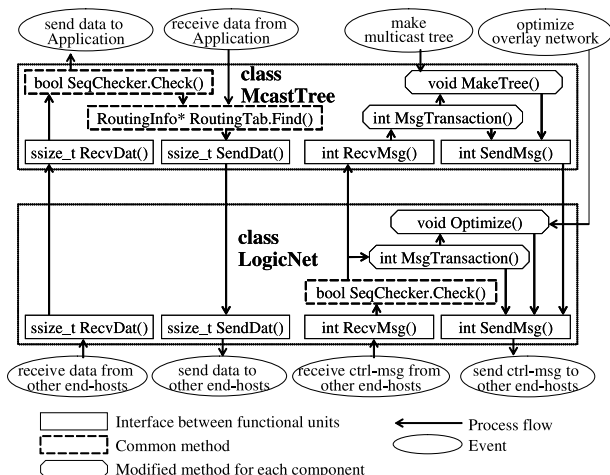


Fig. 4 Processing flow of event transaction in class LogicNet and class McastTree.

event-driven switch model. In the router model, packets are buffered in a queue once, and individual threads send/receive packets and read/write the queue. In the switch model, when a packet arrives, a chain of the methods of receiving the packet, looking up a routing table, and forwarding the packet is immediately operated. RON [29] adopted the switch model, and this model usually achieves higher throughput. Therefore, RelayCast adopts the switch model as the implementation model.

4.3 Component Selection

In our implementation, selecting appropriate components corresponds to modifying specific methods. It means switching the proper procedures in the methods. Figure 4 illustrates the basic framework of event transaction, that is, the processing flow which starts from event generation and is handled by a chain of methods in class LogicNet and class McastTree.

In Fig. 4, octagonal blocks represent the modified methods according to selected components. In the case of selecting the component for the optimization metric or the overlay network topology, *void LogicNet.Optimize()* is chiefly modified. This method calculates logical link costs and makes judgments to connect or disconnect logical links. If the algorithm for deciding the overlay network topology uses specific control messages, it is necessary to add the description to transact the control messages to *int LogicNet.MsgTransaction()*.

In the case of selecting the component for the multicast routing protocol, *void McastTree.MakeTree()* is chiefly modified. This method exchanges routing messages with other end-hosts and updates routing information (*class RoutingInfo*) in a routing table (*class RoutingTab*). If the multicast routing protocol uses specific control messages, developers have to add the description to transact the control messages to *int McastTree.MsgTransaction()*, too.

We describe the number of lines of the source code of

Table 4 Number of the lines of the source code for each component.

Total	4523 lines
Basic Framework	3656 lines
Full-flat topology	97 lines
Delay Metric	*347 lines
DVMRP	131 lines
Multi-path Routing Protocol	292 lines

(*including the RTT measurement method in the MetricEstimator unit.)

the basic framework and each typical component in Table 4. Analyzing an amount of the source code cannot become a complete evaluation. We believe, however, that it could become a rough criterion and indicate the architectural characteristics of the proposed middleware for software developers. Table 4 expresses that an amount of the source code for each component is at most 10% of the entire source code in our implementation. Note that the current source code contains only the minimal error handlers, and so this is just a criterion.

5. Experiments

5.1 Experimental Setup

We report preparatory results obtained by an experiment on a local area network, and verify the practicality of our middleware. In the experiment, we assume a one-to-many delivery as a service model. Although a many-to-many delivery should be assumed strictly, we reduce the number of source end-hosts to one in order to simplify the analysis of multicast tree building.

Our experiment is composed of 9 end-hosts, H_i ($i = 1, 2, \dots, 9$) as shown in Fig. 5. The specifications of H1 to H4 are Pentium III 700 MHz of CPU and 512 MB of memory. The specifications of H5 to H9 are Pentium IV 2.0 GHz of CPU and 512 MB of memory. Realistically, each end-host often belongs to a different domain, and so we set up 3 domains. We introduce a dummynet router which generates delay between the domains into the experimental topology.

Table 5 shows the applied components in the experiment. The optimization metric of an overlay network is delay, and the overlay network topology is full flat. Each end-host restricts the maximum number of logical links to 5, and only the source end-host, H1, limits it to 3. The applied multicast routing protocols are both DVMRP and the multi-path routing protocol. The time interval of DVMRP routing message exchanges is 60 sec. The minimum gradient [25] in the multi-path routing protocol is 0.1. The routing metric in both protocols is a hop count from H1 on an overlay network.

In the experiment, programs are started sequentially from H1. H1 sends the video stream of CBR traffic at 1.5 Mbps. We measure throughput at each end-host and analyze multicast tree topology.

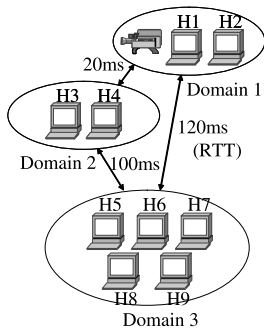


Fig. 5 Network topology in our experiment.

Table 5 Applied components in the experiment.

Functional unit	Component
LogicNet	Optimization metric: Delay Topology: Full flat
McastTree	DVMRP & Multi-path routing protocol

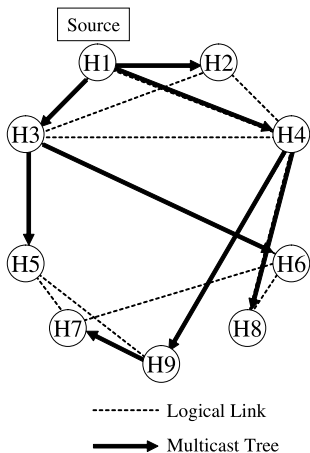


Fig. 6 Overlay network and multicast tree in a stable state.

5.2 Experimentation Results

Figure 6 illustrates the overlay network and multicast tree in a stable state, and Fig. 7 shows the receiving rate measured at H4 and H7 to H9. End-hosts could sufficiently receive application data in a few seconds after establishing the first logical link. H2 to H4, which had always been located next to the source end-host (H1) on an overlay network, could receive application data at an adequately stable rate, but other end-hosts which were located downstream on a multicast tree sometimes fell into an unstable state between 230 and 267 sec. This is because tree partition is frequently caused by overlay network optimization. Note that the overlay network does not always form the most optimal topology because each end-host calculates the cost of logical links autonomously. In the state of Fig. 6, the overhead for forwarding a packet is about 10 msec at H3 and H4.

Next, we force middleware programs in H2 and H4 to

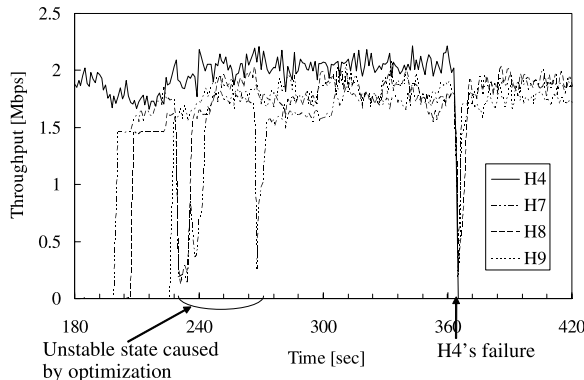


Fig. 7 Throughput (receiving rate) measured at some end-hosts.

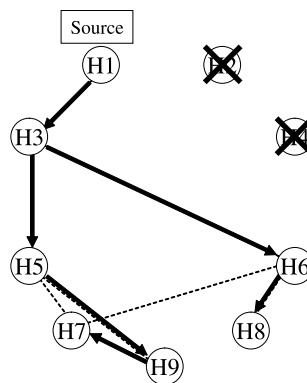


Fig. 8 Rebuilt multicast tree just after end-hosts' failure.

terminate at 363 sec. Although H7 to H9 which were located downstream on the multicast tree could not momentarily gain the enough throughputs, this state was restored within 3 to 5 sec by the multi-path routing protocol. The rebuilt multicast tree is shown in Fig. 8. H8 and H9 changed the upstream end-host of the multicast tree to H6 and H5 respectively. The receiving rate of end-hosts did not fall to zero but each end-host received the video stream of traffic between 1.5 to 2.0 Mbps at a maximum.

5.3 Discussion

In this section, we discuss the practicality of RelayCast. In our implementation, we first constructed the basic framework shown in Fig. 4, and then incorporated the components, such as DVMRP and the multi-path routing, into the framework by modifying the specific methods as mentioned in Sect. 4.3. In our experiments, we verify the application behavior in the case of adopting multi-path routing protocol as a multicast routing component in McastTree. We can also make sure that the performance bottleneck does not reside in RelayCast. Here, the performance bottleneck means that the overhead of forwarding a packet by way of an end-host is large, the operation of a component is blocked, and the transfer rate of the middleware falls below the rate of application data.

Note that as our goal is designing and developing ALM

middleware rather than improving the performance of ALM using the middleware, the example application development and the single experiment scenario described above are enough for verifying an application development process and application behavior on the RelayCast.

Through the above discussion, we believe that our proposed middleware architecture has general versatility and practicality in application development, so that RelayCast will reduce the development redundancy and will enable developers to efficiently implement various ALM applications with minimum efforts.

6. Related Work

In many ALM systems [2]–[18], all the multicast functionality have been independently developed and integrated into individual applications. On the other hand, RelayCast provides the abstracted common functions of ALM as a middleware, and separates ALM functionality from applications. Yoid [5], PeerCast [30], and Overlay Socket [31] are also the ALM systems which mounts ALM functionality in the outside of applications.

Yoid adopts the tree-first approaches and builds a multicast tree using YTMP (Yoid Tree Management Protocol). Yoid is implemented as the local proxy of *wrappers* for supporting existing applications. The implementation of the prototype in this paper took the same model as the implementation of Yoid. PeerCast masks tree transience inherent in tree optimization and end-host's leave from an application by introducing a peering layer between application and transport layers.

Yoid and PeerCast aim at offering as the middleware the individual algorithms which they propose, and don't consider dealing with various fundamental requirements from applications. On the other hand, we proposed the architecture for meeting the various fundamental requirements.

Overlay Socket abstracts ALM functionality at the same level as RelayCast, whose abstraction is presented in Sect. 2. Overlay Socket incorporates only single ALM technique [8] into the middleware on Socket API. In this paper and [25], we define the fundamental functions as functional units, and investigate to deal as the components with the individual algorithms proposed to each functional unit. RelayCast can meet the fundamental requirements from various applications by selecting the appropriate components.

7. Conclusion

In this paper, we proposed RelayCast, a middleware for ALM services, and focused on its architecture, implementation and experiments. RelayCast has the functional unit oriented middleware architecture which provides a minimum but fundamental set of the functionality of the overlay network construction and the multicast routing. We implemented the prototype of RelayCast as a local proxy model, and verified the practicality of RelayCast through our exper-

iments. Although a current target application is live streaming distribution, we believe that RelayCast can be applied to a wide range of applications by incorporating components of retransmission and caching mechanisms. Our future work includes the integration of the initial end-host discovery function, the description of a multicast session, and the introduction of the advanced API for monitoring an overlay network.

References

- [1] C. Diot, B.N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *IEEE Netw.*, vol.1, no.14, pp.78–88, Jan. 2000.
- [2] Y.H. Chu, S.G. Rao, and H. Zhang, "A case for end system multicast," *Proc. ACM SIGMETRICS 2000*, pp.1–12, June 2000.
- [3] Y.H. Chu, S.G. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay multicast architecture," *Proc. ACM SIGCOMM 2001*, pp.55–67, Aug. 2001.
- [4] Y. Chawathe, "Scattercast: An adaptable broadcast distribution framework," *ACM Multimedia Systems Journal*, vol.9, no.1, pp.104–118, July 2003.
- [5] P. Francis, "Yoid: Extending the internet multicast architecture," <http://www.icir.org/yoid/>, April 2000.
- [6] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "Almi: An application level multicast infrastructure," *Proc. USENIX USITS 2001*, pp.49–60, March 2001.
- [7] J. Jannotti, D.K. Gifford, K.L. Johnson, M.F. Kaashoek, and J.W. O'Toole, "Overcast: Reliable multicasting with an overlay network," *Proc. USENIX OSDI 2000*, pp.197–212, Oct. 2000.
- [8] J. Liebeherr and T.K. Beam, "Hypercast: A protocol for maintaining multicast group members in a logical hypercube topology," *Proc. NGC'99*, pp.72–89, July 1999.
- [9] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," *Proc. IEEE Infocom 2002*, vol.3, pp.1366–1375, June 2002.
- [10] W. Wang, D. Helder, S. Jamin, and L. Zhang, "Overlay optimizations for end-host multicast," *Proc. NGC 2002*, pp.154–161, Oct. 2002.
- [11] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," *Proc. ACM SIGCOMM 2002*, pp.205–217, Aug. 2002.
- [12] S.Y. Shi and J.S. Turner, "Routing in overlay multicast networks," *Proc. IEEE Infocom 2002*, vol.3, pp.1200–1208, June 2002.
- [13] V.N. Padmanabhan, H.J. Wang, P.A. Chou, and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking," *Proc. NOSSDAV 2002*, pp.177–186, May 2002.
- [14] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," *Proc. NOSSDAV 2001*, pp.124–133, June 2001.
- [15] A. Rowstron, A.M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," *Proc. NGC 2001*, pp.30–43, Nov. 2001.
- [16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," *Proc. NGC 2001*, pp.14–29, Nov. 2001.
- [17] S. Banerjee, C. Kommareddy, K. Kar, S. Bhattacharjee, and S. Khuller, "Construction of an efficient overlay multicast infrastructure for real-time applications," *Proc. IEEE Infocom 2003*, vol.2, pp.1521–1531, April 2003.
- [18] Z. Xu, M. Mahalingam, and M. Karlsson, "Turning heterogeneity into an advantage in overlay routing," *Proc. IEEE Infocom 2003*, vol.2, pp.1499–1509, April 2003.
- [19] T. Pusateri, "Distance vector multicast routing protocol," *Internet*

- Draft, draft-ietf-idmr-dvmrp-v3-10.txt, Aug. 2000.
- [20] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby, "Reliable multicast transport building blocks for one-to-many bulk-data transfer," IETF RFC 3048, Jan. 2001.
- [21] "Ietf reliable multicast transport working group," <http://www.ietf.org/html.charters/rmt-charter.html>
- [22] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," Proc. IEEE Infocom 2002, vol.3, pp.1190-1199, June 2002.
- [23] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast-sparse mode (pim-sm): Protocol specification," IETF RFC 2362, June 1998.
- [24] A. Ballardie, "Core based trees (cbt) multicast routing architecture," IETF RFC 2201, Sept. 1997.
- [25] N. Mimura, K. Nakauchi, H. Morikawa, and T. Aoyama, "Relaycast: A middleware for application-level multicast services," Proc. IEEE GP2PC 2003, pp.434-441, May 2003.
- [26] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "Stun - simple traversal of user datagram protocol (udp) through network address translators (nats)," IETF RFC 3489, March 2003.
- [27] C. Huitema, "Teredo: Tunneling ipv6 over udp through nats," Internet Draft, draft-huitema-v6ops-teredo-05.txt, April 2005.
- [28] "Vic - video conferencing tool," <http://www.mice.cs.ucl.ac.uk/multimedia/software/vic/>
- [29] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris, "Resilient overlay networks," Proc. ACM SOSP 2001, pp.131-145, Oct. 2001.
- [30] M. Bawa, H. Deshpande, and H. Garcia-Molina, "Transience of peers and streaming media," Proc. HotNets-I, pp.107-112, Oct. 2002.
- [31] J. Liebeherr, J. Wang, and G. Zhang, "Overlay socket: Programming overlay networks," Proc. NGC 2003, pp.242-253, Sept. 2003.



Nodoka Mimura received the B.E. degree in Information and Communication Engineering and the M.E. degree in Electrical Engineering from The University of Tokyo, Tokyo, Japan, in 2001 and 2003, respectively. He is currently a Ph.D. student of Electrical Engineering at the University of Tokyo. His research interests are in the areas of overlay networks, network collaboration, and network services. He is a member of IEEE and IPSJ.



Kiyohide Nakauchi received the B.E., M.E., and Ph.D. degrees in Information and Communication Engineering from The University of Tokyo, Tokyo, Japan, in 1998, 2000, and 2003, respectively. He is currently a researcher at The National Institute of Information and Communications Technology (NICT), Tokyo, Japan. His current research interests include high-speed transport protocols, peer-to-peer networks, and multicast communications. He is a member of IEEE, ACM, and IPSJ.



Hiroyuki Morikawa received the B.E., M.E., and Dr. Eng. degrees in electrical engineering from The University of Tokyo, Tokyo, Japan, in 1987, 1989, and 1992, respectively. He is currently an Associate Professor of the Department of Frontier Informatics at The University of Tokyo. From 1997 to 1998, he stayed in Columbia University as a visiting research associate. His research interests are in the areas of computer networks, ubiquitous networks, mobile computing, wireless networks, and network services. He serves on the technical program committees of IEEE/ACM conferences and workshops, and sits on numerous telecommunications advisory committees and frequently serves as a consult to government. He is a member of IEEE, ACM, IPSJ, and ITE.



Tomonori Aoyama received the B.E., M.E., and Dr. Eng. from The University of Tokyo, Tokyo, Japan, in 1967, 1969, and 1991, respectively. Since he joined NTT Public Corporation in 1969, he has been engaged in research and development on communication networks and systems in the Electrical Communication Laboratories. From 1973 to 1974, he stayed in MIT as a visiting scientist to study digital signal processing technology. In 1994, he was appointed to Director of NTT Opto-Electronics Laboratories, and in 1995 he became Director of NTT Optical Network System Laboratories. In 1997, he left NTT, and joined The University of Tokyo. He is currently Professor in the Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo. His research activities cover the next generation networking technologies from layer 1 (e.g. photonic networks) to higher layers including middleware for network collaboration, P2P routing, mobile networking, and ubiquitous networking. Dr. Aoyama is involved in several governmental projects such as Japan Gigabit Network II (JGN II) and the Ubiquitous Networking Forum, and in some non-profit organizations and consortiums such as the Photonic Internet Forum (PIF), the Digital Cinema Consortium of Japan (DCCJ) and the Digital Cinema Technology Forum in which he is serving as Chairman. Dr. Aoyama is IEEE Fellow and was a Members-at-Large of the IEEE Co mSoc Board of Governors. He serves Vice President of IEICE.