A Monitoring Framework for Hybrid Multicast Networks

Sebastian Zagaria, Thomas C. Schmidt, Sebastian Meiling
{sebastian.zagaria,sebastian.meiling}@haw-hamburg.de, t.schmidt@ieee.org
iNET Research Group – Department Informatik
Hamburg University of Applied Sciences
Berliner Tor 7, 20099 Hamburg, Germany

Matthias Wählisch
waehlisch@ieee.org
Institut für Informatik
Freie Universität Berlin
Takustr. 9, 14195 Berlin, Germany

***Abstract*—Many popular Internet applications like IPTV, video or voice chat, social networks and massive multiplayer online games communicate in groups. While IP layer multicast remains hesitant in global deployment, applications implement their own group distribution techniques at the price of higher complexity and lower efficiency. Emerging hybrid multicast approaches become a promising alternative to fill that gap. Hybrid multicast networks bridge between application and IP-layer multicast and gain multicast deployment at a system level throughout the Internet. In this paper, we present a monitoring framework for such hybrid multicast networks based on a common API in the process of standardization. Monitoring tools are useful to identify network failures and to improve the performance. The target of our monitoring framework is to collect, analyze and visualize node and routing information, thereby making the complexity of hybrid networks accessible to network administrators for the first time.**

I. Introduction

In today's Internet, group communication software is an important part for people to interact and share data with one another. Applications like video-, audio-streaming and conferencing software are popular examples for group communication tools. In such applications, a large number of users participate in a network. An efficient and easy-to-use protocol for data delivery in groups is multicast. Multicast exists in many flavours. There is IP-layer multicast [1] and application layer multicast [2]. Due to the lack of IP-layer multicast support in today's Internet and the complexity of overlay schemes, the most widely used multicast algorithm is a simple reflection based on servers. In recent years, approaches arose to combine IP-layer and application-layer-multicast for transparently using the benefits of both. These approaches are called hybrid multicast networks. In hybrid multicast networks, IP-layer technologies will be used wherever available, and application-layer-multicast is chosen to bridge gaps in wide-area deployment of multicast network throughout the Internet.

In order to maintain complex networks, it is necessary to have tools available that help administrators to monitor these networks. Monitoring tools are useful to identify network failures and to improve the performance. There are well known tools for unicast like ping and traceroute that help managing standard networks. In the case of IP-layer multicast, there exist a number of tools that mimic the functionality of *ping* and *traceroute*, e.g., *mcping* [3], *tracetree* [4] and *SSM-ping* [5]. Also several monitoring and visualization tools exist, e.g., *Mhealth* [6] and *MUVI* [7]. However, to the best knowledge of the authors, there is no monitoring tools that supports hybrid multicast networks.

This paper focuses on the development of a monitoring and visualization framework for hybrid multicast networks.[1] The framework is based on the common API for transparent hybrid multicast, jointly developed by the Internet Technologies Group at HAW-Hamburg and Cisco. The target of the monitoring framework is to collect and visualize data from a hybrid multicast network using this common API. The framework does provide functionalities to display the captured data in a graphical and intuitive way. Its graphical representation visualizes multicast forwarding trees, as well as detailed node and group information.

The remainder of this paper is structured as follows. In Section II, we review related work with special dedication to H∀Mcast. Section III presents the monitoring framework. We conclude and give an outlook on future work in Section V.

II. Related Work

A. Hybrid Multicast

In recent years, a number of hybrid multicast approaches have been released providing solutions to overcome the lack of support for IP-layer multicast throughout the Internet. Approaches like Hybrid Shared Tree (HST) [8], Island Multicast (IM) [9], Scalable Hybrid Multicast (SHM) [10] and Universal Multicast (UM) [11] divide the network into two layers. The lower layer, consisting of IP-layer multicast domains, interconnected via the upper layer that consists of some global scheme of overlay multicast.

B. H∀Mcast

H∀Mcast [12] is a recent practical approach to hybrid multicast networks. The H∀Mcast project focuses on the development and analysis of a hybrid group communication Internet architecture. This hybrid architecture is centered around a common API (see Section II-C) and a universal middleware. The API is implemented as a C/C++ and Java

[1] Our monitoring software is publicly available at http://hamcast.realmv6.org/developers.

library. It provides a universal and technology-transparent service for group communication. The architecture follows an evolutionary model using a middleware at each system and gateways to forward multicast data between technologies. The middleware manages a number of multicast protocols divided into technology moduls. This approach makes H∀Mcast highly flexible and adaptable to the needs of the network status. The simple socket configuration and creation in H∀Mcast eases its use and offers a rich framework for the development of group communication software.

C. Common API for Transparent Hybrid Multicast

The aim of the common API [13] is to enable application programmer to develop applications that communicate in groups and run everywhere, independent of the state of network service deployment. Unlike the current multicast socket API, the naming and addressing is endpoint- and type-agnostic, using a universal multicast URI scheme to identify groups. The API calls group into four categories. The group management calls that provide creation of multicast sockets and management of the group memberships. The send/receive calls to send and receive multicast data. The socket options for the configuration of the multicast socket, e.g., to set the ttl or add/remove interfaces. The service calls provide means to access interface information such as routing neighbors or joined multicast groups.

D. IP Multicast Monitoring Tools

There are several tools for multicast monitoring that discover the tree topology or analyse multicast traffic. For example, *RTPmon* [14] is a tool to collect and display packet loss and jitter. *RTPmon* uses RTCP (Real Time Control Protocol) to gather these information. Also tools like *tracetree* [4] can be used to discover multicast forwarding trees based on extended router functionalities. *Mhealth* [6] combines application and routing information to provide a detailed view of a multicast network. The routing information is obtained by *Mtrace* [15], which is not part of *Mhealth*, *Mtrace* is a multicast traceroute program. It basically mimics the functionality of unicast *traceroute*. It resolves the tree from the receiver to the source. Also data like jitter, packet loss, delay and group relations can be monitored with *Mhealth*. These data is determined using the RTCP protocol, thus *Mhealth* is bound to programs using RTP. All information acquired with *Mhealth* are visualized, like the multicast forwarding tree and additional information. *MUVI* [7] is a tool that can monitor statistics and visualizes the multicast forwarding tree of a network. In order to trace the forwarding tree, *MUVI* uses the SNMP protocol to retrieve the routing neighbours of a router. It is also possible to browse and save the MIB table of a router. Thus *MUVI* needs access to the management of routers that will be monitored.

III. The Monitoring Framework

The monitoring framework consists of three independent modules, the *management agent*, the *collector*, and the *viewer*. The main functionalities of the framework are to first access the data of distributed nodes, to collect and finally visualize it. To access the data, every node needs to run a management agent that serves as a remote interface to the common API. A collector will use these agents to collect and process the acquired information. A view then displays the data provided by the collector. All components are completely separated. This modular architecture allows to change functionalities without interfering with other modules. Most importantly the design and development of user interfaces can be separated from the monitoring task, allowing to implement different kind of views. Since the common API used by the monitoring framework is on its track to experimental standard, the framework can be used to monitor group applications that implement this API. Figure 1 gives an overview of the monitoring framework. A single collector fetches and aggregates the information from the nodes and feeds into the different views. As shown in Figure 1, the viewers do not need to be part of the hybrid multicast network and do not require to run any kind of library. The information provided by the collector can be accessed using a simple HTTP based messaging protocol to ensure many different views.

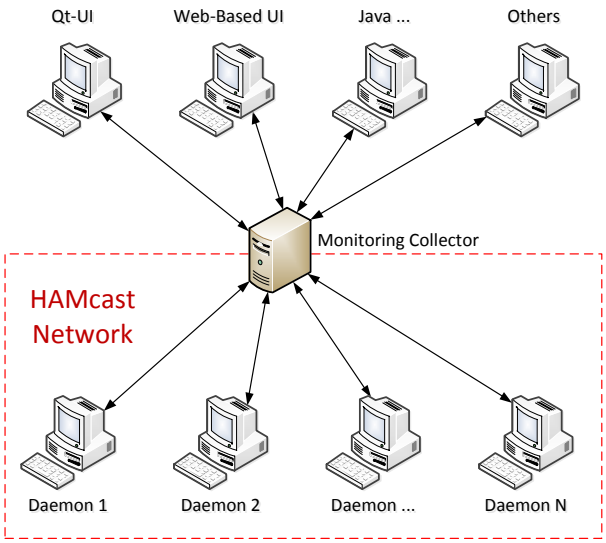

Fig. 1. Architecture overview

The communication between monitoring modules is realized using a RESTful-Web-Service. REST [16] is a lightweight remote procedure call (rpc), using a client-server architecture. The server provides methods that can be called by the client. For this purpose the client and server are communicating using the HTTP message protocol. Methods are identified by an URI in the HTTP header. The payload of a message and return values of a method are encoded in XML.

A. Monitoring Framework Modules

1) Management Agent: A Node running this program can be connected to one or several collectors. Basically the management agent wraps the service-calls (see Section II-C) in methods that convert the returned values into an XML

document. A collector connected with the agent can use the REST-Interface to execute the calls. All callable methods of the agent have a counterpart in the Common-API.
Callable methods of the management agent are as follows:

get_interface returns a list of communication interfaces and their properties.
group_set returns a list of groups for the specified interface.
neighbour_set returns a list of routing neighbours for the specified interface.
parent_set returns a list of routing parents for a specified interface and group.
children_set returns a list of routing children for a specified interface and group.

2) Collector: This program collects all data from connected agents, feeding the view with aggregated information about the network. In a periodic time interval, the collector updates and stores the node information within a data model. Node information is accessed using the REST-Interface of management agents. To provide the collected data to a viewer, the collector implements a REST-Interface. The methods made available are directly adapted for monitoring reasons and visualization.
Callable methods of the collector are as follows :

group_list returns a list of groups aggregated from the group sets of all known nodes.
node_list returns a list of agent ids from all known nodes. Every agent has an id to identify the node.
group_data returns a list of all agent ids for a specific group. This call needs a group URI as argument.
node_data returns a list that contains detailed information about a specific node. This call requires a valid agent id as argument.
group_tree returns a list of edges specific to a group. The list contains pairs of agent ids representing edges between nodes. This call requires a group URI as argument.

In order to exchange data, the collector and the management agents need a discovery mechanism to find one another. Due to the group dynamics of multicast, users may join and leave the network at any time. Thus in order to find all nodes involved in the network, the discovery process has to be as dynamic as the network itself. Since all nodes participate in a multicast network, it is possible to perform a group based rendevouz to contacting the nodes without prior knowledge. All agents involved in the network join a well known multicast group. The collector is now able to distribute *connect* messages within the well known group (fig. 2). This messages contains the IP-address of the collector. Every agent that receives this message will open a connection to the collector. This reduces the NAT connection problem to the collector and prevents the agents from polluting the multicast network with replies. To track additionally joining nodes, the collector periodically sends connect messages into the multicast group.

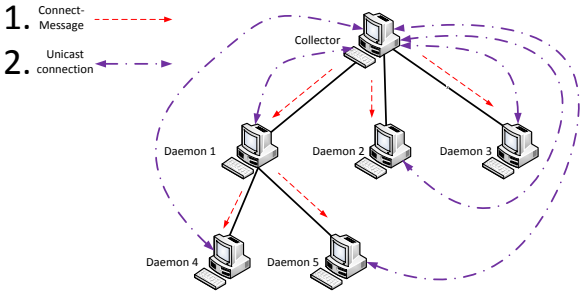

Fig. 2. Node discovery of the monitoring framework

3) Viewer: This program visualizes data provided by the collector. This part of the framework is the most exchangeable and can be adjusted to the needs of the user. For our implementation, we required the viewer to provide an intuitive, group focused visualization of the network. All existing groups in the network must be clearly displayed, to get an overview of the network status. In addition, a more detailed view for specific groups is given. This detailed view lists all nodes and their properties, e.g, interface and routing information. As a second visualization, the viewer can draw multicast group forwarding trees. This graph is displayed in a new tab, so that the user can switch between different graphs. The visualized nodes display information about the technology in use and their identity, so nodes can be related to the none-graphical representation. All nodes are represented by icons, the used icon depends on the available multicast technologies modules of the node. Every node in this tree can show detailed node information, equal to the detailed group view but for this single node. This allows the user to view node information without the need to switch between the text and the graphics view.

B. Interaction among Modules

The interaction among the modules is separated into two layer. At the lower layer, the management agents and collector communicate using hybrid multicast sockets and REST-Interfaces. This interaction consists of node discovery and monitoring. Figure 3 shows the sequence diagram of the communication between a collector and an agent. After the agent opens a TCP-connection, due to the node discovery, it instantly replies its agent id to the collector. Thus the collector can identify the agent for node specific requests. Also it is used by the viewer as a text representation of a node. After the node discovery the collector uses the open TCP-connection for the REST communication. At the upper layer the viewer and collector communicate via the REST-interface of the collector.

IV. Operating the Monitoring Framework

The framework is based on the H∀Mcast prototype. The management agent and the collector are build upon the C++ library of H∀Mcast to implement the discovery process via hybrid multicast sockets. Also the agent uses this API to access node information provided by the service-calls. The viewer is implemented using the Qt-UI framework [17] for the GUI

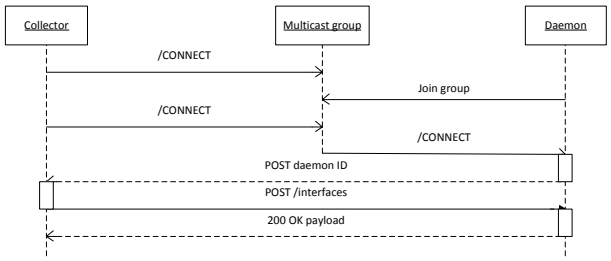

Fig. 3. Communication between collector and management agent

and tree visualization. The node positions of the graphical representation are calculated using the iGraph library [18]. This library supports data types, layouts and functions for generating and manipulating graphs. All of these libraries are multi platform enabled and may at least be operated under Windows and Linux distributions. We tried to minimize the use of external libraries to make sure the framework operates on many different systems.

There is no strict order to run the programs, but the viewer cannot work properly without a collector. After setting up the network or using an existing one, every node that should be monitored by the framework has to run a management agent that is properly configured. At start up, the agent takes two command line arguments, its agent ID and a multicast management group URI. If not set, the ID will be set to the IP-address of the node. The multicast group URI is used for the node discovery, this argument has to be set. After the program started, the management agent will join the multicast group given in the command line argument and wait for incoming */connect* messages from a collector.

After the agents are configured and running, the collector can be started on a system. As for the agent, the collector needs to be executed on a system with a running H∀Mcast middleware, which is part of the network that will be monitored. The collector can be configured by four command line arguments. The first argument specifies the multicast group URI that will be used to execute the node discovery process. The second argument is a time interval that indicates how frequently the collector will perform the node discovery. The third argument is a time interval that defines the lag between updates of node data from connected agents. The last argument defines the port of the TCP-Server which is used to communicate with the agents. After the collector is started, the program will send */connect* to the multicast group that was given as a command line argument. This message contains the IP-Address of the collector and the port of the TCP-Server. After the agents receive this message, they initiate a TCP-connection to the specified endpoint.

Once the connection is successfully established, the agent will immediately send its agent ID to the collector. On receiving the agent ID, the collector will save the ID and connection information into a map. This enables the collector to send requests to specific agents identified by the ID. This ID could also be auto generated by the collector, but it is better readable and more meaningful to the user if its an IP-address or a name set by the user.

The collector will periodically call the methods of an agent that can be accessed over the REST-interface to update a data model that will be created for every connected management agent. This data model represents all properties of a node. Figure 4 shows the data model and all its properties. A node consists of its name and a set of interfaces. Each interface has its own ID, name, address, technology, routing neighbours and a set of groups. A group consists of a group URI, parent and children.

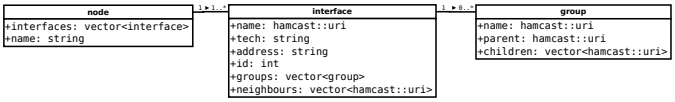

Fig. 4. Data model for a node

Once the collector is running, the viewer can be started as well. On start up the viewer takes no command line arguments. The viewer can be configured over an configuration window that will show up (see fig. 5). In this configuration window, the user can set the IP-address and port of the collector as well as a time interval that indicates how frequently the data will be updated.

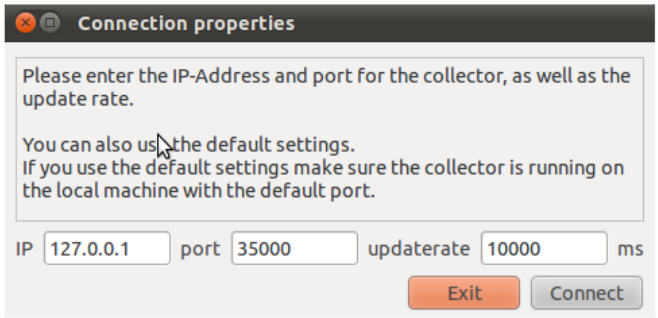

Fig. 5. Configuration window of the viewer

On confirming the configuration, the viewer will connect to the collector and a main window will show up. In this main window, there are two views, a group and a detailed group view. The first view contains a list of all multicast groups joined by the nodes. This information is provided by the *group_list* call of the collector. The second view is a tree of detailed group information. The information provided by that view include all nodes that have joined the group and their node specific informations, like active technology interfaces, routing neighbours, parent and children set. By clicking on an item in the group view the user can switch between the groups shown in the detailed view. On double clicking a group item in the group view, a graphical representation of the group tree will be opened in a new tab (see fig. 6). Every node in this tree view is represented by an icon, e.g, nodes with active application layer multicast interface, Inter-domain Multicast

Gateways, nodes with IP multicast interface or nodes with both native and ALM interface. In the bottom of a screen, a legend is displayed that explains the meaning of all occurring icons. All node icons are connected via edges representing the parent children relation of the multicast forwarding tree for the specific group. If the user moves the cursor over an icon, a tool tip window will appear that shows some quick informations about the node, e.g, the technology interfaces that are used, the name of the node and the number of parent and children for this group. By clicking on an icon an information window will pop up. This window shows detailed node information that can also be found in the detailed group view. The user can position the windows next to each other to get a direct comparison of nodes. Allowing the user to identify nodes that are important for routing. All tree views are displayed in different tabs so that the user can freely switch between tree vies or the textual representation.

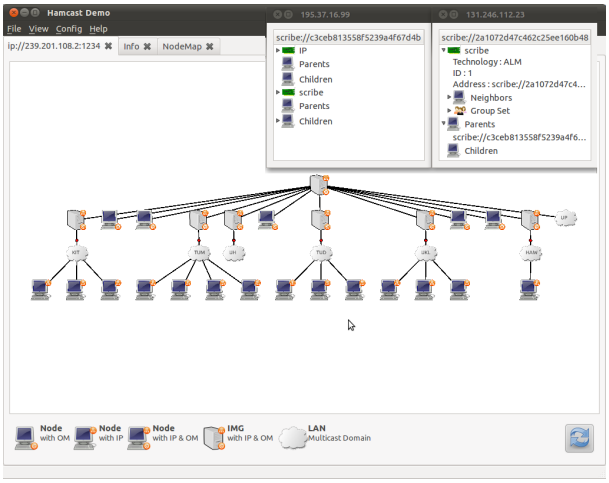

Fig. 6. Multicast forwarding tree visualization with detailed node information

V. Conclusion and Outlook

Hybrid multicast opens the realm of widespread, operator-independent deployment at the price of a virtualized network access of higher complexity. A monitoring and analysis framework is needed for large scale deployment. We developed a software that provides detailed knowledge and visual information about multicast groups and trees. The concept and implementation of the framework presented in this paper is generic in the sense that it extracts all base information from the common API calls and combines a complete view to provide a graphical representation of multicast forwarding trees.

Currently, we are working on a logging system for the collector that records the monitored networks and an offline mode for the view that visualizes recorded data. Furthermore, we are planning to develop a Web interface that uses the daemon and collector software of the framework. This Web interface should provide a second mobile view that can be used on smaller devices like smart phones as well. We also plan to implement statistic measurements for network performance, and to extend the daemons to run user-defined scripts on distributed nodes, using REST-interfaces of this framework.

Acknowledgment

We wish to thank Dominik Charousset and Sebastian Wölke for their contributions and development within the H∀Mcast project. This work is funded by the German Federal Ministry of Education and Research (BMBF) within the project H∀Mcast and the G-Lab initiative, see http://hamcast.realmv6.org.

References

[1] S. E. Deering and D. R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 85–110, 1990.

[2] C. K. Yeo, B. S. Lee, and M. H. Er, "A Survey of Application Level Multicast Techniques," *Computer Communications*, vol. 27, no. 15, pp. 1547–1568, 2004.

[3] P. Namburi, K. Sarac, and K. Almeroth, "Practical Utilities for Monitoring Multicast Service Availability," *Comput. Commun.*, vol. 29, pp. 1675–1686, June 2006.

[4] K. Sarac and K. C. Almeroth, "Tracetree: a scalable mechanism to discover multicast tree topologies in the internet," *IEEE/ACM Trans. Netw.*, vol. 12, pp. 795–808, October 2004.

[5] P. Namburi, K. Saraç, and K. C. Almeroth, "SSM-ping: A ping utility for source specific multicast." in *Communications, Internet, and Information Technology*, 2004, pp. 63–68.

[6] D. B. Makofske and K. C. Almeroth, "MHealth: A real-time multicast tree visualization and monitoring tool," in *NOSSDAV*, June 1999.

[7] R. Krzywania and R. Lapacz, "Multicast Visualisation Tool MUVI," http://muvi.man.poznan.pl, last called on January 09.2012. [Online]. Available: http://muvi.man.poznan.pl/

[8] M. Wählisch and T. C. Schmidt, "Between Underlay and Overlay: On Deployable, Efficient, Mobility-agnostic Group Communication Services," *Internet Research*, vol. 17, no. 5, pp. 519–534, November 2007.

[9] X. Jin, K.-L. Cheng, and S.-H. G. Chan, "Island multicast: combining IP multicast with overlay data distribution," *IEEE Transactions on Multimedia*, vol. 11, no. 5, pp. 1024–1036, 2009.

[10] S. Lu, J. Wang, G. Yang, and C. Guo, "SHM: Scalable and Backbone Topology-Aware Hybrid Multicast," in *16th Intern. Conf. on Computer Communications and Networks (ICCCN'07)*, August 2007, pp. 699–703.

[11] B. Zhang, W. Wang, S. Jamin, D. Massey, and L. Zhang, "Universal IP multicast delivery," *Computer Networks*, vol. 50, no. 6, pp. 781–806, 2006.

[12] S. Meiling, D. Charousset, T. C. Schmidt, and M. Wählisch, "System-assisted Service Evolution for a Future Internet – The HAMcast Approach to Pervasive Multicast," in *Proc. of IEEE GLOBECOM 2010, Workshop MCS 2010*. Piscataway, NJ, USA: IEEE Press, Dec. 2010, pp. 913–917.

[13] M. Wählisch, T. C. Schmidt, and S. Venaas, "A Common API for Transparent Hybrid Multicast," IRTF, IRTF Internet Draft – work in progress 04, January 2012.

[14] D. Bacher, A. Swan, and L. A. Rowe, "rtpmon: a third-party RTCP monitor," in *Proceedings of the fourth ACM international conference on Multimedia*, ser. MULTIMEDIA '96. New York, NY, USA: ACM, 1996, pp. 437–438.

[15] D. B. Makofske and K. C. Almeroth, "Real-time multicast tree visualization and monitoring," *Softw. Pract. Exper.*, vol. 30, pp. 1047–1065, July 2000.

[16] L. Richardson and S. Ruby, *RESTful Web Services*. Beijing: O'Reilly, 2007.

[17] Nokia Corp., "Qt : cross-platform application and ui framework," 2012. [Online]. Available: http://qt.nokia.com/

[18] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.