Implementation of a montoring framework for the HAMcast project

Sebastian Zagaria

Ausarbeitung Projekt 1

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Contents

1 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 applications. In such applications a large number of users participate in a network. The usual way to transmit data is to use a unicast. Since unicast transmits data from a source to a single receiver, it is inefficient for the use of group communication applications. Another downside of unicast in terms of group communication is that all receivers have to be known to the data source. Even tough most group communication applications use a unicast transmission protocol to exchange data.

Unlike unicast, multicast protocols are optimized for group communication. Multicast exists in many flavours. There is IP-layer multicast [5] and application layer multicast [18]. If we compare both IP-layer and application-layer multicast, IP-layer multicast is more efficient in terms of data delivery but also less deployed, because of the dependency on router support. Due to the lack of native multicast support in today's Internet [6], the most widely used multicast algorithm are Server-based. In recent years, there has been approaches to combine native and application-layer-multicast to use the benefits of both. These approaches are called hybrid-multicast-networks [19] [7] [16]. In hybrid-multicast-networks, IP-layer technologies will be used where they are available, and application-layer-multicast is used to get a widely deployed multicast network via the Internet.

Due to the the complexity of networks, it is necessary to have tools that help network 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 monitoring these networks. In the case of IP-layer multicast, there exists a number of tools that mimic the functionality of ping and traceroute, e.g, mcping [11] and tracetree [15]. Also a number of monitoring and visualization tools exist, e.g, mHealth [8] and MUVI [14]. However, to the knowledge of the author, there exist no tools supporting monitoring of hybrid multicast networks. This project work focuses on the development and implementation of a monitoring and visualization framework for a hybrid-multicast-network. The framework will be based on HAMcast [10], a hybrid multicast network developed by the Internet Technologies Group at HAW-Hamburg. The target of the monitoring framework is to collect an visualize data from the HAMcast multicast network using the HAMcast-API [17]. The framework will provide functionalities to display the captured data in a graphical and intuitive way. This graphical representation shall show multicast forwarding trees as well as detailed node and group information.

The reminder of this paper is structured as follows, in section two we start with related work to this project explaining how HAMcast works. After that we give an overview of existing multicast monitoring tools, focusing on tools that both monitor and visualize. In section three we describe

the concept of the monitoring framework. Section four explains the implementation. Finally in section five, we give an outlook and conclude this project work.

2 Related work

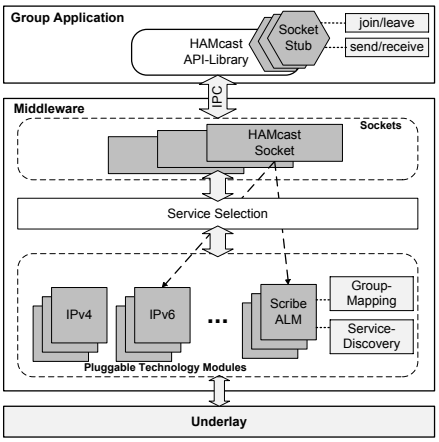
2.1 HAMcast

The HAMcast project [1] focuses on the development and analysis of a hybrid group communication Internet architecture. This hybrid architecture is realized by a transparent application API and a middleware. It provides a universal service for group communication. The architecture (fig. 1a) is an evolutionary model using a middleware at each system and gateways to forward multicast data between technologies [10]. This middleware does not require any development by the users, application developer or service provider. The middleware manages a number of multicast protocols divided into technologies modulus. This approach makes HAMcast highly flexible and adaptable to the needs of the application programmer as well as easy to deploy via Internet.

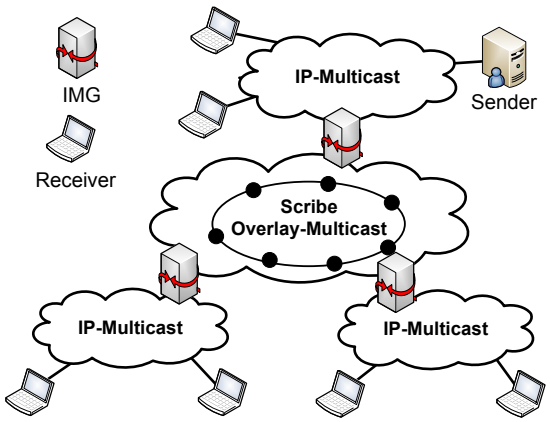
For the development of group communication software, the HAMcast project offers a universal layer-transparent API [17] implemented as a C/C++ and Java library. This API encapsulates the underlying middleware and multicast technologies. The developer has full freedom in developing group communication software without the need to code the software for a specific multicast technology. Simple socket configuration and creation makes HAMcast an easy to use and rich framework for the development of group communication Software. In addition to that the API provides further functions called service calls. These service calls are designed to offer detailed informations about the current network status of the system.

In order to establish a hybrid multicast service, HAMcast uses Interdomain-Multicast-Gateways (IMG). Mainly an IMG serves as a gateway between two multicast domains using different or the same multicast technologies. IMGs forward multicast traffic from one technology to another (fig. 1b). To do so, it is necessary to relate different technologies with one another. Therefore HAMcast uses a universal multicast group addresses (HAMcast URI). Listing 2.1 shows how the URI is constructed. Each URI starts with a scheme, that refers to a specified multicast name space. The second argument identifies the group e.g host and port. The third part of the URI is instantiation, it serves to identify a resource that generates data, e.g, the source in source specific multicast. The sec-credentials argument is optional, it can be used for security purposes. For example, *ip://232.0.0.1:1234@141.22.26.54* a HAMcast URI with ip as the namespace and a multicast group address with a specification of source address.

```
1 scheme "://" group "@" instantiation ":" port "/" sec-credentials
```



(a) HAMcast Architecture



(b) HAMcast example network

Listing 1: HAMcast URI

2.2 Multicast Monitoring

There are several tool for multicast monitoring that discover the tree topology or analyse multicast traffic. For example, RTPmon [4] is a tool to collect and display packet loss and jitter. This tool can be used to find multicast distribution problems. RTPmon uses RTCP (Real-Time-Control-Protocol) to gather these information, and is therefore boud to applications using RTP protocol. Another tool is Mrinfo, it displays the configuration informations from multicast routers. Mrinfo [3] uses IGMP messages to acquire informations like routing neighbors, metrics and threshold. Also tools like tracetree [15] can be used to discover multicast forwarding trees. Tracetree is an extends multicast routers, this extension allows them to resolve trees. In this section we want to focus on tools that do both, tree discovery and visualizations.

Mtrace [9] is a multicast traceroute program. It basically mimics the functionality of unicast traceroute. It resolves the tree from the receiver to the source. In order to resolve the entire multicast forwarding tree, the subtrees of all existing receivers have to be merged into one. That implies that the monitoring tool has to know all receivers involved in the tree. Another disadvantage of this process is that it can only work if routers are running the Mtrace program. The behaviour of this traceroute algorithm starts by sending a query message to the first hop router of a receiver. The query message contains the address of the source, a multicast address and a response address. After that, the receiver converts this message into a request message by adding a request data block. The request block contains the information when the packet arrived, all outgoing interfaces, the number of packets previously sent by the router and the TTL. After creating this request message, the router sends this message to his previous

hop router. All messages are send via unicast. After receiving a request message, the router will add an request block and forwards the message as described before. If the router is responsible for forwarding the message to the source, the request message will be converted into a response message. Afterwords this message will be send to the response address contained in the message.

Mhealth [9] combines application and routing information to provide a detailed view of a multicast network that, runs RTP applications. The routing information will be obtained by Mtrace. This tool is not part of Mhealth and has to be installed separately. Also data like jitter, packet loss, delay and group relations can be monitored with Mhealth. These data is determined using the RTCP protocol. All information acquired with Mtrace and RTCP will be collected and visualized. Mhealth visualizes the multicast forwarding tree, routers are represented by rectangles and the connection between them by lines. More information like packet loss is displayed using color filled rectangles.

MUVI [14] is a Java-based multicast monitoring 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. Thus MUVI needs access to the management of routers that will be monitored. The tree discovery can be limited by setting an hop count, so the software will only trace all routers within this hop count. MUVI represents routers as icons and the interface connections are represented by lines connecting the icons. The filter option enables the user to see all interfaces that are used for forwarding multicast traffic for a specific group. The connection lines between the router will be painted in green while all connections that are not involved will be painted black. It is also possible to browse and save the MIB table of a router.

3 Concept of the monitoring framework

3.1 Architecture

The monitoring framework consists of three independent modules, each having its own purpose. The main functionalities of the framework are accessing data of distributed nodes, collect and visualize it. To access the data, every HAMcast node has to run a daemon process that serves as a remote interface to the HAMcast-API, the monitoring daemon. A collector will use these daemons 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 us to change functionalities without interfering with other modules. Most important the design and development of User-Interfaces can be separated from the monitoring task, allowing us to implement different kind of views. Since the API-interface used by HAMCast is

on track to standard, the framework can be used to monitor all kinds of applications that will implement this API.

Figure 1 gives an overview of the monitoring framework. On every HAMcast node, a daemon process is running. A single collector fetches the information from the nodes and provide it to the different views. As shown in the architecture overview 1, the viewers are not part of the HAMcast network and do not need to run any kind of library. The information provided by the collector can be accessed using a simple http based messaging protocol to ensure all kind of different views.

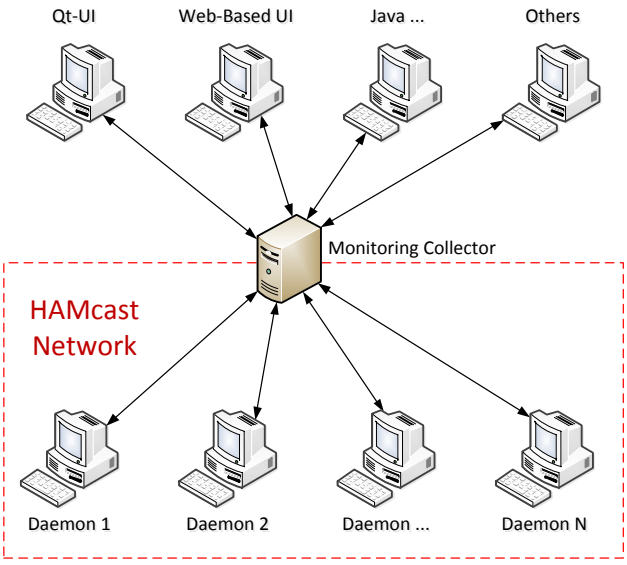


Figure 1: Architecture overview

The communication between monitoring modules is realized using a RESTful-Web-Service. REST [13] is a lightweight remote procedure call (rpc), using a client-server architecture. The server provides a number of methods that can be called by the client. For this purpose the client and server are connected via TCP using HTTP as 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.

3.2 Monitoring Framework modules

Daemon

The daemon process serves as remote access to the HAMcast-API. A Node running this program can be connected to one or more collectors. There exist a number of HAMcast-API calls called service-calls, these calls provide information about the current network status of a node. Basically the daemon wraps the service-calls in methods that convert the returned values into a XML document. A collector connected with the daemon can use the REST-Interface to execute the calls. The sequence diagram in figure 2 shows the communication between the collector and daemon, include the node discovery process 3.2.

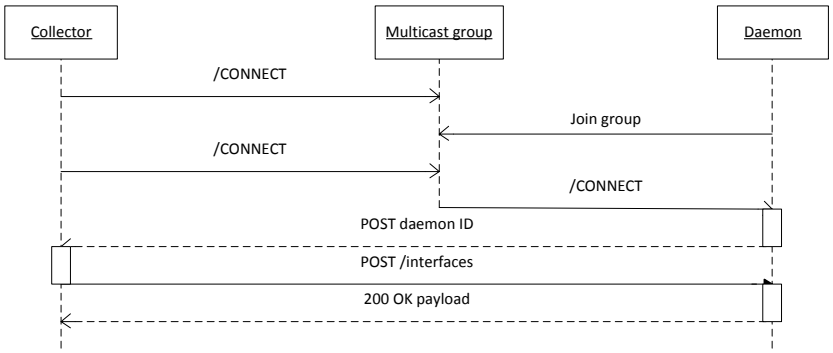


Figure 2: Communication between collector and daemon

Callable methods of the daemon are as follows [17] :

get_interface returns a list of communication interfaces and their properties, equal to the get_interfaces call of the HAMcast-API.

group_set returns a list of groups for the specified interface, equal to the group_set call of the HAMcast-API.

neighbour_set returns a list of routing neighbours for the specified interface, equal to the neighbour_set call of the HAMcast-API.

parent_set returns a list of routing parents for a specified interface and group, equal to the parent_set call of the HAMcast-API.

children_set returns a list of routing children for a specified interface and group, equal to the children_set call of the HAMcast-API.

Collector

This program collects all data from connected daemons, feeding the view with information about the network. In a frequent time interval, the collector will update and store the node information. The node information is accessed using the REST-Interface of a daemon. The information gained from a daemon is stored within a data model.

To provide the collected data to a view, the collector implements a REST-Interface. The methods made available are directly adapted for monitoring reasons and visualization. The sequence diagram (fig 3) shows how the viewer and collector communicate with one another. Important is that the collector is reachable via a public IP-address allowing daemons and views to open a connection. This reduces the NAT connection problem to a single entity.

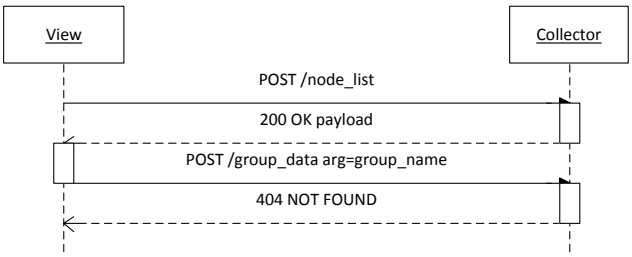


Figure 3: Communication between view and collector

Callable methods of the collector are as follows :

group_list returns a list of HAMcast groups merged from the group sets of all known nodes.

node_list returns a list of daemon ids from all known nodes. Every daemon has an id to identify the node.

group_data returns a list of all daemon ids for the specific group. This call needs a HAMcast group URI as argument

node_data returns a list that contains detailed information about a specific node. This call requires a valid daemon id as argument.

group_tree returns a list of edges specific to a group. The list contains pairs of daemon ids representing edges between nodes. This call requires a HAMcast group URI as argument.

In order to exchange data, the collector and the daemons, need a discovery mechanism to find one another. Normally, the entity that opens a unicast connection has to know the communication. Due to the nature of a dynamic network user 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.

To do so, we decided to use that same infrastructure that we want to monitor, as a tool to discover nodes. Since all nodes participate in a multicast network it is possible to contact the nodes without any prior knowledge. All daemons involved in the network join a well known multicast group. The collector is now able to send a connect messages into the well known group (fig. 4). This messages contains the IP-address of the collector. Every daemon who receives this message will open a TCP-connection to the collector. This enables the collector and daemons to exchanged messages. In addition to that daemons have no interest in receiving responses from one another, so we avoid to pollute the network with monitoring messages. To track additionally joined nodes, the collector frequently sends connect messages into the multicast group.

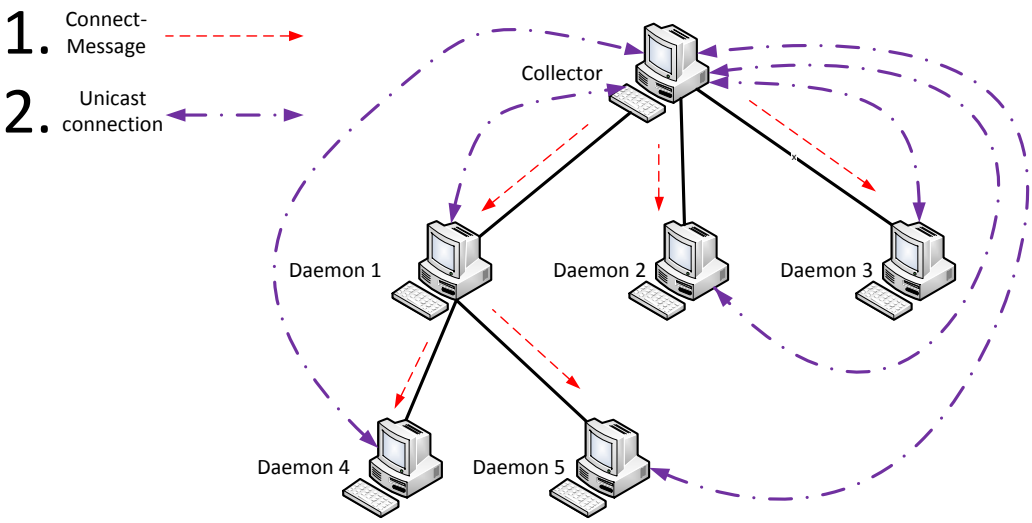


Figure 4: Node discovery

Viewer

This program visualizes data provided by the collector. The viewer can obtain information data via the REST-Interface of 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 require the view to update the data displayed in near real time to give the user an accurate representation of the network. Also we want the view to provide an intuitive, group focused visualization of the network. All groups existing in the HAMcast network should be clearly displayed, to get an overview of the network. This group overview will be shown in a list view.

There should also be a more detailed view on the specific groups. The detailed view should list all nodes and their properties, for example interface and routing information. This detailed group window can be opened by clicking on one of the groups listed. The node information in this window are displayed in a tree view. This tree view uses a hierarchical representation, every root in this window is the daemon id of a node. By clicking on a root the user can expand the tree showing more detailed information about the node.

As a second visualization, we want the viewer to draw multicast group forwarding trees that show the relation between nodes in a graphical manner. This graph will be displayed in a new tab so that the user can switch between different graphs. The visualized nodes should display information about the used technology and their identity, so nodes can be related to the none graphical representation of the group. All nodes are represented by icons, the look of the icon depends on the active technologies modules of the node. By clicking on an icon a small freely movable window will open. This window contains all the node information available for this node, equal to the detailed group view, but reduced to this single node. This allows the user to view node information without the need to switch between the text and the graphics view.

4 Implementation

In this section we present the implementation of the discussed presented in section 3. All programs are written in C++. The daemon and collector are implemented using the boost library and the viewer is implemented using the Qt-UI framework. All of these libraries are multi platform and can at least be operated under Windows and Linux distributions. We tried to minimize the use of external libraries to make sure the framework easy to operate on many different systems.

4.1 Shared utilities

Software components like the data model and the REST-Interface are shared among the monitoring modules. All modules use the a method_caller class and http_message class to exchange data via a REST-Interface. Also the Collector and View use the same data model to save network information.

The data model (fig. 5) represents a HAMcast node and all of his properties. A HAMcast node has a name wich is the daemon id of a node. This id serves to identify the node. It is used by the collector and the view. Every node can have one to n active interfaces.

The interface has a name which is the name of the used technology, an id assigned by the middleware, a network address, a technology which describes the type of technology used, e.g, application-layer- or IP-layer-multicast, a set of routing neighbours and joined groups. The group has a name (HAMcast group address), a parent and children set.

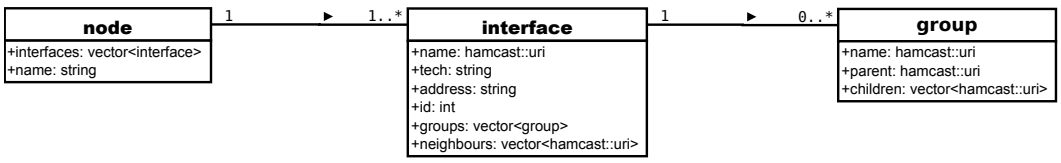


Figure 5: The HAMcast node data model

For the implementation of the REST-Interface, a class or library that supports the http protocol for both client and server side, is needed. Most of the libraries are only supporting the client side. Thus we decided to implement a class which supports all messages we need for the REST-Interface. There are basically two messages needed HTTP POST and HTTP REPLY. The HTTP REPLY is defined by the error code in the message header. The most common error codes are 200 OK, 404 NOT FOUND and 500 internal server error. The 200 OK code in

the message header represents a successful call and the 404 NOT FOUND an unsuccessful call. Which mean that either the call could not be found or a arguments were missing/wrong.

The creation of a HTTP POST and HTTP Reply is implemented by the http_message class. This class can be used to crate new messages or parse received messages into an object. The construction of a HTTP POST message needs two arguments a URL defining the method name and a list of arguments, if needed. To construct a REPLY message the error code and payload is needed.

To call a method defined in the header of a message we implemented a class called method_caller. The programs inherit from this class to implement the REST-Interface. One method of this class is the register_method call. This call takes a name to identifying the method and a function pointer to the method. The function pointer and the method name will be saved into a map. All methods have to have the same method body with only one argument a string array. The arguments passed by an HTTP POST message will be parsed into strings and hand over to the method. The conversion and verification of arguments into the right data types is in the responsibility of the method, as well as the construction of a valid XML document for the reply. To create and parse the XML documents we used the boost property tree library.

After a method is registered it can be executed by the invoke_method call. This call takes two arguments, the name of the function and a vector containing the arguments for that function. On calling this method the function pointer map will be searched for a registered method matching the specified name. If no method is found an empty string will be returned, so the caller knows that the method call failed, otherwise the method will be invoked with the given arguments. An example call is shown in figure 6.

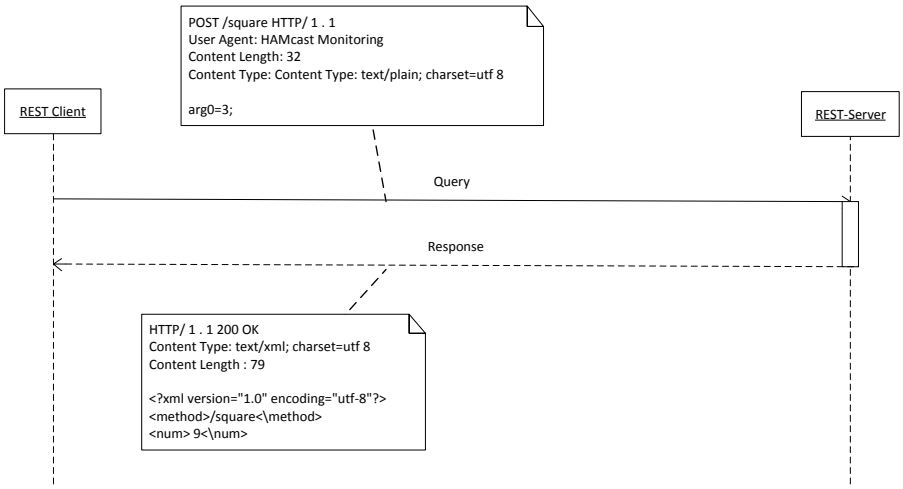


Figure 6: REST message exchange

4.2 Monitoring Daemon

The implementation of the monitoring daemon has two main behaviours. A multicast module waits for connect messages from a collector to invoke a TCP connection. The TCP-Connection class that connects to the collector and waits for incoming calls that invoke methods using the implemented REST-Interface. The figure 7 shows a class diagram with the most important classes and attributes of the daemon program.

Command line arguments At start up, the daemon takes two command line arguments, the daemon id and a HAMcast multicast group. All arguments are optional, the program can also run with default options. The first argument defines the daemon id that will be used by the collector and view. If not set, the daemon id will be set to the IP-address of the node. The second argument is a HAMcast multicast group, this group address will be used for node discovery. If not set, the default value ip://230.0.0.1:1234 will be used.

Program behaviour If all arguments are correctly set, the daemon will create a multicast module class that receives all incoming messages on the specified group. On receiving a connect message, the daemon will open a TCP-connection, the collector uses a new thread to handle the incoming calls received via this socket. If a daemon is already connected with this collector, the connect message will be ignored. On receiving a message, the received data will

be used to create a http_message object. The URL and arguments contained in the message will be used to invoke a method using the invoke_method call of the function_wrapper class. This call returns a string, if the string is empty a HTTP 404 message will be created using the http_message class. Otherwise a HTTP 200 OK message will be created with the returned value as payload.

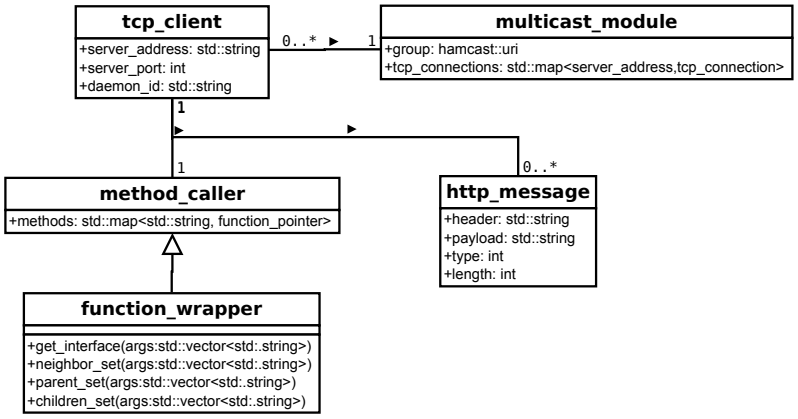


Figure 7: Daemon class diagram

4.3 Monitoring Collector

The collector has two main behaviours, a multicast module that is used for the node discovery and a TCP-Server that handles all incoming TCP-Connections. Figure 8 shows the class diagram of the collector.

Command line arguments The collector needs three command line arguments in order to execute. The first argument specifies a HAMcast multicast group that will allow him to execute the node discovery. The second argument is a time interval that indicates how frequently the collector sends connect messages into the multicast group. The third argument is a time interval that indicates how frequently the collector will update the data model for connected daemons. All arguments are fully optional, if they are not set the default values will be used. The default value for the group is set to ip://239.0.0.1:1234, the port for the tcp-server is set to 35000, the time value for the node discovery is set to 30 seconds and the time interval to update the node data is set to 10 seconds.

Program behaviour At start up the monitoring collector frequently sends connect messages into the multicast group using the multicast module. The connect message contains the IP-address and port of the collector. After a monitoring daemon joins the multicast group, it will receive the message and open a TCP-connection with the collector. After a connection is successfully established, the daemon will send his daemon id immediately to the collector. The collector memorises the daemon id and TCP-connection object in a map. For every TCP-connection the collector holds a data model, this data model will be frequently updated depending on the time interval set. Every TCP-Connection object runs a timer set to the specified time value. If the timer is triggered the data model for this connection will be updated. To do so, the collector executes the method a daemon offers over the REST-Interface using the http_message class to create the requests. All incoming messages will are handled as describes in the program behaviour of the daemon. If the message is an HTTP 200 OK the payload of the message contains which call created this payload and how it should be handled. This is done by checking the method entry of the XML document.

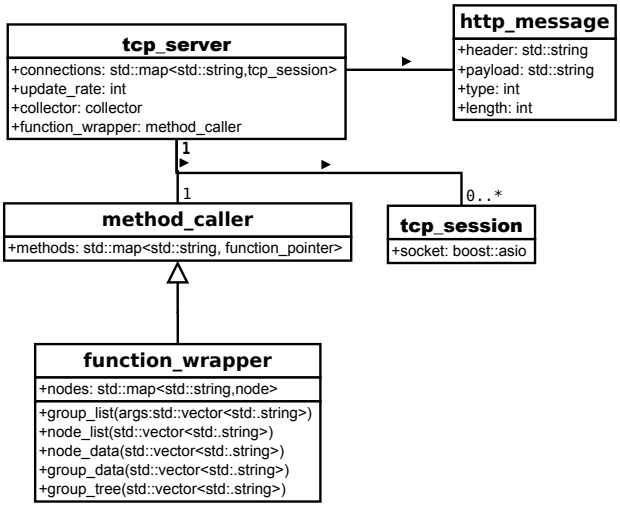


Figure 8: Collector class diagram

4.4 Monitoring Viewer

The view provides an intuitive, groupfocused visualization of the network. All groups existing in the HAMcast network are clearly displayed in a list view, to get an overview of the network. There is also a second view that lists all nodes and their properties in a tree view. The tree view

will be set with the group information when a user clicks on a group address in the list view. On Double click of a group address a new tab opens where the forwarding tree is rendered (fig. 11). The group nodes are represented by icons connected with lines. Every line represents the parent children relation between nodes. If the user scrolls over the icon a tool tip window opens showing the interface address of the node, the interface technology and the number of children this node has. A left click on a node icon opens a small movable window containing detailed information about the node in a tree view, equal to the information of the tree view. The UI and graphical trees are created using the Qt-framework [12]. All items in the graphics tree are graphic object created using the the QGraphicsView framework. The class diagram (fig. 9) shows the most important classes of the viewer.

Program behaviour The program behaves as follows, on start up a configuration window will show up (fig. 10). In this configuration window the user has to set the IP-address and port of the collector. Optional a time interval that indicates how frequently the data will be updated, can be committed to the software. All entries are set with default values, the IP-address of the collector is set to localhost, the port is set to 35000 and the time interval is set to 10 seconds. Thereafter a connection to the collector will be established. A list of all multicast groups will be fetched and displayed using the group_list method from the REST-interface of the collector. If a user requests to get detailed information about a group, all needed data will be retrieved by executing the group_data call and the node_data call of the REST-Interface. The group_data call returns a list of group members for the group, for each entry in this list the node_data call will be executed to get the node details. If the user wants to display the graphical group tree, the viewer calls the group_tree call from the collector, this call returns a list of all edges in the tree. For every node in the edge list a image_node graphics object will be created. The image_node class defines the paint method and mouse listeners for a graphical object. To connect the graphics objects, edge objects will be created from the the edge_class. The edge_class draws a line between two image_nodes. The image_nodes. will be connected as defined by the edge list. At last the node positions will be calculated and the tree can be displayed. For the calculations of the node positions we use the igraph library [2]. This library supports data types for graphs as well as functions for generating and manipulating graphs. Also it provides layout algorithms for different types of graphs. We use the Reingold and Tilford layout algorithm. This Layout algorithm is directed downwards, children are centered below their parent. In parallel an update thread is running to keep the displayed graphs current. If this data changes, all currently opened views that are afflicted by that will be updated.

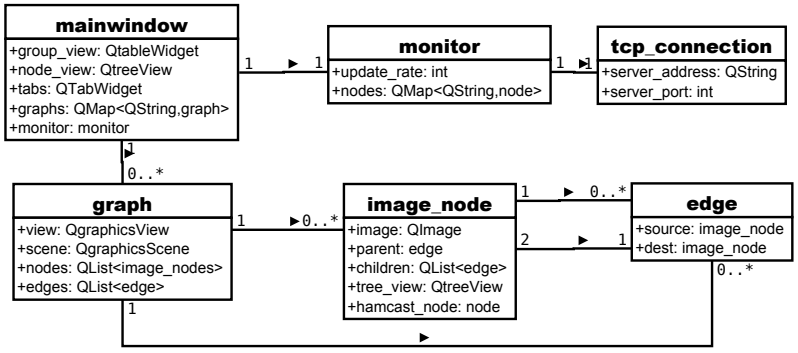


Figure 9: Collector class diagram

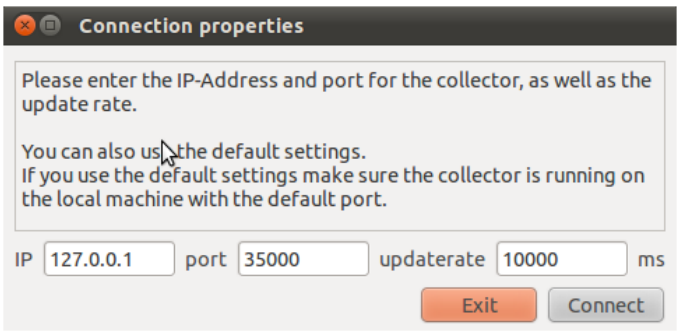


Figure 10: Configuration window

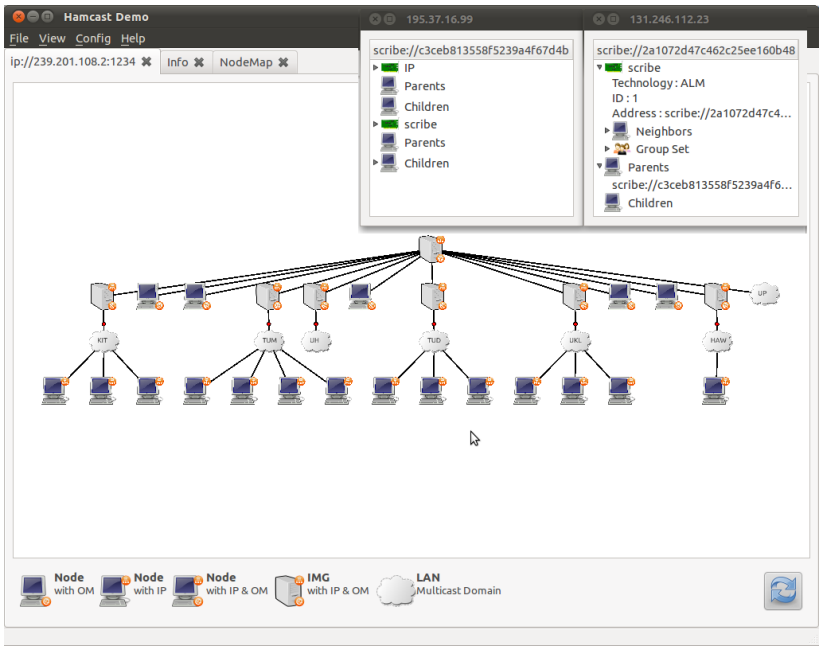


Figure 11: Tree visualization

5 Conclusion and Outlook

The motivation in developing a monitoring framework for HAMcast was to give the users a tool at hand that can be used to debug and analyse hybrid multicast networks. We wanted to develop a software that provides detailed information about multicast groups and tree visualization. The concept and implementation of the framework provides detailed informations about multicast groups and nodes as well as a graphical representations of multicast forwarding trees. Over the development cycle of this framework, we discovered several bugs that helped us enhance the HAMcast technology modules and middleware.

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 the 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 plan to implement statistics for HAMcast networks. We also want to extend the demons enabling the user to run scripts on distributed HAMcast nodes using the frameworks view. Also we want to extend the daemons to use mtrace giving network administrator the possibility to view native forwarding trees under the requirement that mtrace is installed on the routers. For the current implementation of the

view we use the Reingold and Tilford layout algorithm to position the nodes in our graphical representation. It is planned to provide more layout algorithms in the near future to give the user the possibility to arrange the graphs as preferred.

References

[1] : *HAMcast project page*

[2] *The igraph library*. Webseite. – URL http://igraph.sourceforge.net/. – Letzter Aufruf am 9. Jannuar 2012

[3] *OpenBSD System Manager's Manual*. Webseite. – URL http://www.openbsd.org/cgi-bin/man.cgi?query=mrinfo#AUTHORS. – Letzter Aufruf am 9. Jannuar 2012

[4] BACHER, David ; SWAN, Andrew ; ROWE, Lawrence A.: rtpmon: a third-party RTCP monitor. In: *Proceedings of the fourth ACM international conference on Multimedia*. New York, NY, USA : ACM, 1996 (MULTIMEDIA '96), S. 437–438. – URL http://doi.acm.org/10.1145/244130.244459. – ISBN 0-89791-871-1

[5] DEERING, Stephen E. ; CHERITON, David R.: Multicast routing in datagram internetworks and extended LANs. In: *ACM Trans. Comput. Syst.* 8 (1990), May, S. 85–110. – URL http://doi.acm.org/10.1145/78952.78953. – ISSN 0734-2071

[6] DIOT, Christophe ; NEIL, Brian ; BRYAN, Levine ; BALENSIEFEN, Kassem D.: Deployment issues for the IP multicast service and architecture. In: *IEEE Network* 14 (2000), S. 78–88

[7] JIN, Xing ; CHENG, Kan-Leung ; CHAN, S.-H. G.: Island multicast: combining IP multicast with overlay data distribution. In: *Trans. Multi.* 11 (2009), Nr. 5, S. 1024–1036. – ISSN 1520-9210

[8] MAKOFSKE, David B. ; ALMEROTH, Kevin C.: MHealth: A real-time multicast tree visualization and monitoring tool. In: *In Network and Operating System Support for Digital Audio and Vid eo (NOSSDAV*, 1999

[9] MAKOFSKE, David B. ; ALMEROTH, Kevin C.: Real-time multicast tree visualization and monitoring. In: *Softw. Pract. Exper.* 30 (2000), July, S. 1047–1065. – URL http://dl.acm.org/citation.cfm?id=350554.350572. – ISSN 0038-0644

[10] MEILING, Sebastian ; CHAROUSSET, Dominik ; SCHMIDT, Thomas C. ; WÄHLISCH, Matthias: 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, S. 913–917

[11] NAMBURI, Pavan ; SARAC, Kamil ; ALMEROTH, Kevin: Practical utilities for monitoring multicast service availability. In: *Comput. Commun.* 29 (2006), June, S. 1675–1686. – URL http://dl.acm.org/citation.cfm?id=1646665.1647082. – ISSN 0140-3664

[12] NOKIA: *Qt a cross-platform application and UI framework*. Webseite. – URL http://qt.nokia.com/products/. – Letzter Aufruf am 9. Jannuar 2012

[13] PRZYBILSKI, Michael: REST - REpresentational State Transfer. URL http://www.cs.helsinki.fi/u/chande/courses/cs/MWS/reports/MichaelPrzybilski_REST.pdf, ? – Forschungsbericht

[14] RADOSLAW KRZYWANIA AND ROMAN LAPACZ: *Multicast Visualisation Tool MUVI*. Webseite. – URL http://muvi.man.poznan.pl/index.php?id=01. – Letzter Aufruf am 9. Jannuar 2012

[15] SARAC, Kamil ; ALMEROTH, Kevin C.: Tracetree: a scalable mechanism to discover multicast tree topologies in the internet. In: *IEEE/ACM Trans. Netw.* 12 (2004), October, S. 795–808. – URL http://dx.doi.org/10.1109/TNET.2004.836123. – ISSN 1063-6692

[16] WÄHLISCH, Matthias ; SCHMIDT, Thomas C.: Between Underlay and Overlay: On Deployable, Efficient, Mobility-agnostic Group Communication Services. In: *Internet Research* 17 (2007), November, Nr. 5, S. 519–534. – URL http://www.emeraldinsight.com/10.1108/10662240710830217

[17] WÄHLISCH, Matthias ; SCHMIDT, Thomas C. ; VENAAS, Stig: A Common API for Transparent Hybrid Multicast / IRTF. URL http://tools.ietf.org/html/draft-irtf-samrg-common-api, January 2012 (04). – IRTF Internet Draft – work in progress

[18] YEO, C.K. ; LEE, B.S. ; ER, M.H.: A survey of application level multicast techniques. In: *Computer Communications* 27 (2004), Nr. 15, S. 1547 – 1568. – URL http://www.sciencedirect.com/science/article/pii/S0140366404001616. – ISSN 0140-3664

[19] ZHANG, Beichuan ; WANG, Wenjie ; JAMIN, Sugih ; MASSEY, Daniel ; ZHANG, Lixia: Universal IP multicast delivery. In: *Computer Networks* 50 (2006), Nr. 6, S. 781–806. – ISSN 1389-1286