Autonomic Reliable Multicast

Application-Level Group Communication Using Self-Organization Principles

Edzard Höfig
Fraunhofer Institute for Open Communication Systems
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
edzard.hoefig@fokus.fraunhofer.de

ABSTRACT

We present a mechanism for reliable multicast based on autonomic principles (AutoRM). So-called Beamon nodes exchange information in a peer-to-peer manner, deriving a subjective view of the environment. Applications connect to a Beamon network to participate in reliable communication within groups they declare to be joined to.
This short paper describes the general AutoRM concepts, the architecture and protocols used between application and Beamon, as well as between the nodes themselves.

Keywords

Multicast, Group Communication, Autonomic Communication, Self-Organization

1. INTRODUCTION

Group Communication (GC) is a major concept when dealing with situations that require the exchange of data in a 1-to-many fashion, for example service discovery, load-balancing, or distributed control purposes. It is often implemented using standard UDP-based IP multicast, which matches most underlying transport mediums well, but exhibits unfavourable properties, e.g. packets may disappear or appear out of order. The size of a UDP packet is also limited to the Maximum Transmission Unit (MTU) of the transport technology, e.g. making it impossible to transmit more than $1\frac{1}{2}$ Kbyte of consecutive data over Ethernet. There are several solutions to cope with the limitations but these are either implemented on network-level [5, 4], limited to only a certain language [1], or configured in a way that contradicts autonomic design concepts [6, 3]).

2. CONCEPTS

A major characteristic of autonomic systems is their ability to properly operate on their own accord in a changing environment (often referred to as *homeostasis*). It also dictates that a system needs to be able to infer its specific view of an operational context in a dynamic and continuous manner. We therefore refrain from employing principles that rely on a universally consistent, shared global state. Instead autonomic systems should communicate to reach a common agreement or goal, always subjecting exchanged information to an interpretative analysis. Enabling an active exchange of information forms another cornerstone of the design principles: autonomic systems are assumed to be communicating constantly. If a system does not participate in the exchange of information it is considered non-existent. Together, these principles lay the foundation for a self-organization of the AutoRM components.

2.1 Architecture

Our implementation of an AutoRM infrastructure consists of three main components: A shared communication medium, several so-called **Beamon** nodes and several applications (also referred to as frontends). As depicted in

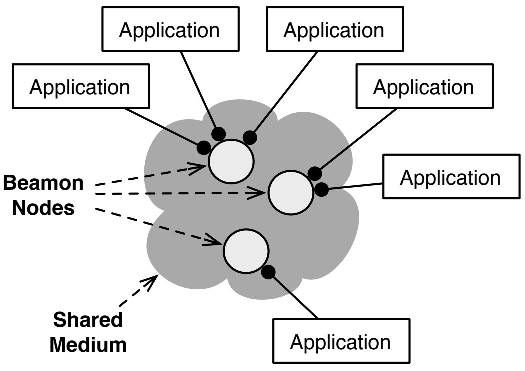

Figure 1: General Architecture

Fig.1 the application components are connected to the Beamon nodes in an 1-to-many manner: normally only a single Beamon is deployed at a host, which is then facilitated by several local applications. The nodes themselves are exchanging information via a shared communication medium. Applications may connect to any reachable Beamon – in most cases we envision this to be a local one – and participate in message transmission using groups.

2.2 Autonomic Group construction

Following autonomic principles, groups are a purely local concept: A Beamon declares membership in a group by including it in a list that is repeatedly broadcasted on the shared medium. The group declaration messages serve as

Bionetics '07, December 10-13, 2007, Budapest, Hungary
Copyright 2007 ICST 978-963-9799-11-0.

a heartbeat mechanism, enabling other Beamons to decide upon the dead of a node by absence of the regular heartbeats. Every Beamon not only tracks timestamps for every incoming message, but also membership declarations of every node in its environment[1], forming a subjective view of the environment. By performing analysis on the history of timestamps it is decided if another Beamon is still active. Currently we are only employing a comparison of the last timestamp value against a so-called **stale-time** (proportional to the number of nodes in the environment), but we plan to adapt this to more sophisticated means, e.g. statistical analysis.
The frequency with which a node sends heartbeat messages is determined by the number of overall nodes in the environment: the more Beamons in an environment, the lower the frequency, the more inaccurate the subjective views of each Beamon. Without this frequency adaptation, an environment would be swamped with heartbeat messages. An approximation of the overall number of nodes is calculated by using the local node tracking table and an average of the number of nodes as reported by each Beamon as part of its heartbeat. A node uses a so-called **trust** value to decide the ratio with which both values (local and global average) are combined to give the approximation of overall nodes in the environment.

2.3 Group Communication

An application manages group participation by sending a **join** or **leave** command to a certain Beamon, identifying a group by name. After joining a group, the Beamon will start to announce the group in its next heartbeat message (if not already part of the groups it announces) and the application will be subsequently informed of any information exchanged within the group.
There are two ways for an application to send messages to a group: either by sending to everyone, or by sending to someone. In the former case, a message will be send to the whole group. In the later case, the Beamon responsible for sending picks a single node of that group (currently at random) from its tracking table and directly transmit a message. The target Beamon subsequently selects a single registered application and dispatches the message. Similar GC primitives have also been implemented in the Group Event Notification Protocol [2].

2.4 Reliability

Messages may be larger than the current MTU and the sending of messages is done in a reliable fashion, whereas the reliability model for GC in autonomic system is different than existing ones used for traditional distributed systems [6], as we postulate that no universal global state may be used. Reliability is defined as follows: As long as a receiver is participating in a group some messages are send to, it is guaranteed that the messages will be delivered to the receiver and that they arrive in the order they were sent. If transmission of a message is not possible (for example due to excessive packet loss on the shared medium) the receiver will be dropped from the senders tracking tables and therefore vanish from the sender's environment; in other words: the Beamon implementation guarantees that either a message arrives completely and in order at a destination or the destination node is removed from the sender's environment tables. The determination of a node's reachability is dependent on the numbers of re-transmissions, employing the timestamping mechanism as discussed in section 2.2.

[1]The environment is understood as consisting of all hosts reachable via the shared medium

3. IMPLEMENTATION

Decomposing the AutoRM architecture in a so-called **backend** (a Beamon node) and **frontend** (a lightweight component bound by an application) enables the employment of different implementation languages for the separate parts. In our case the backend is designed as a UNIX daemon and currently implemented in C++, whereas the frontend is currently implemented using three Java classes. Developing new language bindings for Beamon is done by creating new frontend implementations in a language of choice, allowing many potential applications access to the AutoRM GC facilities. This two-level architecture lead to the design of two different communication protocols: the protocol between frontend and backend, and the protocol for information exchange between Beamons.

3.1 Frontend-Backend Protocol

Communication between front- and backend follows a client-server paradigm, employing TCP for reliable, connection-oriented data transfer. Messages that are transmitted using this protocol are following a simple common layout as shown in Fig. 2. The diagram shows the packet layout on a byte

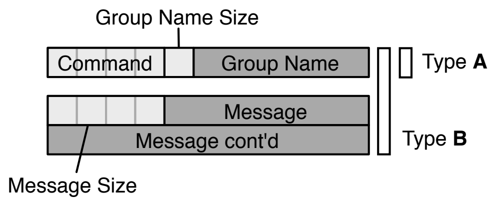

Figure 2: Layout of Frontend-Backend Protocol

level: Fixed size values are depicted in a lighter shade of gray, variable size values are shown in a darker colour and without gridlines. There are two types of packets: Type **A**, consisting of a 4 byte command name in ASCII, followed by a one byte value for the size of the group name and the variable length group name itself (encoded in UTF-8). Type **B** extends this structure by adding a 32bit message size field[2] and a variable number of bytes containing a message.
The protocol itself is stateless and simple: A type A packet with the command JOIN send from the front- to the backend instructs the Beamon to join a group, the command LEAV does the opposite. Using type B packets, the Beamon is directed to send messages with either the SNDE (send to everyone) or SNDS (send to someone) commands. The only packet transmitted from the Beamon back to the application is type B RCVE containing a message that the application ought to receive.

3.2 Beamon Protocol

Communication that adheres to the inter-Beamon protocol follows a peer-to-peer paradigm, utilising the 1-to-many broadcasting capabilities of the shared medium. In our current implementation the shared medium is a single IPv4

[2]Wire format: all values are encoded in network byte order

multicast group, joined at the start of each Beamon and left when a node exits. Addressing of packets is based on non-persistent Universally Unique Identifiers (UUID) which each node assigns to itself. The protocol is stateful, employing a binary format to encode information in a tightly manner, and consists of only three packet types: heartbeats, data packets, and acknowledgements. The layout of heartbeat packets is shown in Fig. 3. Packets of this type are

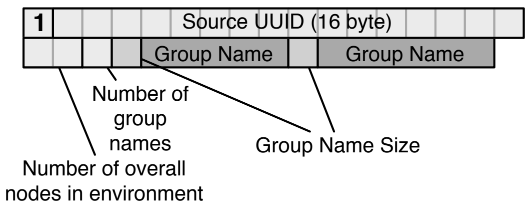

Figure 3: Layout of Heartbeat Packet

continuously send by each Beamon. They consist of a single byte packet type identifier with the value 1, a 16 byte source node address, a two byte value containing the number of nodes in the sending Beamon's environment table, a one byte value indicating the number of group names transmitted in this heartbeat, and a list of declared groups for the node in question. The group list is structured by using single byte values indicating the size of the group name, followed by the name itself, encoded in UTF-8. When a

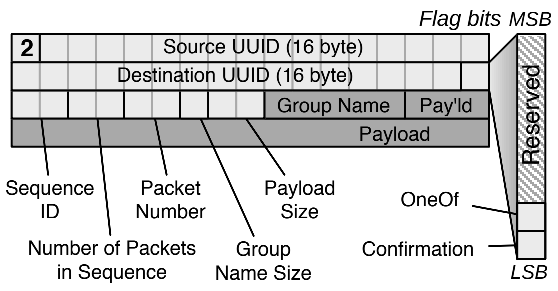

Figure 4: Layout of Data Packet

node wants to send a message it uses data packets, as shown in Fig. 4. Following the type identifer (2) these contain a source and a destination address. The destination might be set to all zeros, indicating that the packet is addressed to a group and not a single node. Besides containing a sequence identifier[3], the overall number of packets in the sequence, the current packet number, a group name, and the payload itself, the packet contains a set of flags. Currently only two of them are used: the **OneOf** flag instructs a receiver to choose exactly one application to forward the message to and the **Confirmation** flag requests an acknowledgement packet (see Fig. 5) from the receiver. Besides carrying information that enables the sender to identify the proper sequence, the acknowledgement packet transports all packet numbers missing in the receiver, limited by the number of the last received packet, so the sender will not re-transmit packets that had not arrived as the acknowledgement request was processed in the receiver. The last packet of a sequence always has to be confirmed, but a sender might introduce

[3]which needs to be unique in regard to the sending node

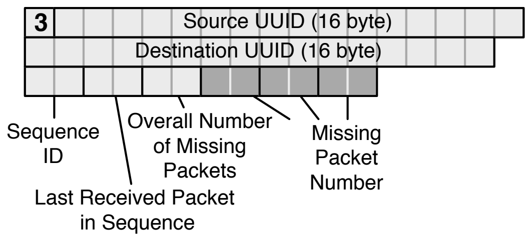
3
Source UUID (16 byte)
Destination UUID (16 byte)
Sequence ID
Overall Number of Missing Packets
Missing Packet Number
Last Received Packet in Sequence

Figure 5: Layout of Acknowledgement Packet

additional confirmation requests in any packet it sends, optimising confirmation traffic load when transmitting large messages.

4. CONCLUSIONS

We are currently employing a predecessor of the presented architecture as underlying infrastructure for the management of a distributed testbed using virtualization, enabling us to start or stop virtual machines on several hosts and to distribute runtime data to all participating hosts in parallel. Due to the autonomic principles used, the configuration on a per-host basis is obsolete, making testbed set-up trivial. The new protocols presented here are bound to replace the old version, because former lacked the reliability concepts and was not been implemented in a language agnostic way. There are several open issues that we would like to study in greater detail: For example we don't know how to control congestion at the shared medium, we are also using a fixed MTU which may not work well with transport mediums different than Ethernet or Wireless LAN, and we have no experience regarding robustness or scalability of the new protocol. Nonetheless, the current version is a matching solution for the requirements we have and – hopefully – a small, but valuable contribution to the Autonomic Communication (AC) community at large.

5. REFERENCES

[1] B. Ban. Javagroups – group communication patterns in java. Technical report, Cornell University, 1998.

[2] C. Reichert and D. Witaszek. An implementation of the group event notification protocol. Technical Report TR-2002-0301, Fraunhofer FOKUS, 2002.

[3] Sally Floyd et al. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, 1997.

[4] Sanjoy Paul et al. Reliable multicast transport protocol (RMTP). *IEEE Journal of Selected Areas in Communications*, 15(3):407–421, 1997.

[5] T. Speakman et al. PGM Reliable Transport Protocol Specification. RFC 3208 (Experimental), 2001.

[6] Yair Amir et al. Secure spread: An integrated architecture for secure group communication. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(3):248–261, September 2005.