HAMcast
An Implementation of a System-centric Middleware Component for Universal Multicast

Sebastian Meiling
Dominik Charousset
Fabian Holler
Sebastian Wölke
Sebastian Zagaria

{sebastian.meiling,dominik.charousset}@haw-hamburg.de
{holler_f,woelke_s,zagari_s}@informatik.haw-hamburg.de
Agenda

- Introduction
- Middleware Design
- Prototype Implementation
- Performance Evaluation
- Lessons learned
- Conclusion & Outlook
Introduction

Existing Problems

- Many multicast flavours and technologies
- Heterogeneous multicast deployment
- No general API to multicast services

<draft-irtf-samrg-common-api>

- Specification of a common multicast API
- Abstract naming scheme with Loc-ID split
- Concept for integrating different multicast technologies
Network Scenario Example

IPv6

Member of „G“

Member of „F“

IPv4

IMG

IMG

Member of „F“

Member of „G“

Scribe

w/o HAMcast stack

w/ HAMcast stack
Middleware Design

- Multicast service stack for end-systems
  - Dynamic discovery and configuration of network and system environment
  - Pluggable multicast technology modules
  - Late binding of multicast technologies at runtime

- Openness to future extensions and adaption of
  - Other application programming languages
  - New network technologies
  - API library and deployed HAMcast middleware remain unchanged
HAMcast Prototype Implementation

- Middleware
  - User-space process, runs once per host
  - Implemented in C++, using boost library

- API available for C++ and Java, conform to:
  - <draft-irtf-samrg-common-api>

- Service modules
  - Multicast technology specific, currently available:
    - IPv4, IPv6, and Scribe-ALM
  - Implemented in C++ and C
IPC Interface to Applications

- Connects applications that use multicast API with the middleware process
- Based on localhost sockets and self-designed IPC protocol
  - Open protocol specification
  - Simple adoption by programming language
- Synchronous and asynchronous transfers
- Single IPC session per application
IPC Communication

- IPC session combines all calls for one application
  - Multicast traffic is handled by streams (per group)
- Asynchronous:
  - Multicast send/receive calls
- Synchronous:
  - Join/leave calls
  - Service calls (e.g., group_set, parent_set …)
  - Set/get socket options
  - Update event calls
Multic和平cast Service Module

- Access to specific multicast technology
- Multiple instances (interfaces) possible
  - several IP interfaces or overlays
- Each module provides its own
  - Service discovery, to enable and configure module instance(s)
  - Technology dependent mapping function to translate group names (URI) to addresses (e.g., hash)
IP Service Discovery

- Investigate local system and network environment
- Passive approach:
  - Query multicast state tables
  - System calls to network card driver
  - Packet sniffing, e.g. IGMP/MLD queries or PIM messages
- Active approach:
  - Probe local network
  - Join specific groups
- React to system events
  - Device comes up or down
  - Network cable plug on/off
Performance Evaluation

- Measurements
  - Comparison of HAMcast middleware-stack and Linux standard IP-stack
  - Mean and std. deviation over 50 runs (60s each)
  - Analysis of different packet sizes (100–1500 B)

- Metrics
  - Throughput, and CPU usage

- Test Setup
  - Single source/receiver scenario
  - Quad-Core CPU, Ubuntu-Linux, 1Gbit network
Throughput in Mbit/s (Sender)
Throughput in #pkt/s (Sender)
Throughput in #pkt/s (Receiver)

![Graph showing throughput vs packet size]

- **MAX**
- **IP**
- **HAMcast**
CPU Usage (Sender)
CPU Usage (Receiver)

The graph shows the CPU usage [%] against the packet size [B] for IP and HAMcast. The CPU usage for IP increases with packet size, whereas the CPU usage for HAMcast decreases. The error bars indicate the variability in the data.
Lessons Learned

- IPC communication is critical bottleneck
- IPC optimizations for send/recv calls:
  - Asynchronous transmission
  - Aggregation of successive data packets
- Further optimizations:
  - Precached name-to-address mappings
  - Middleware runs with higher priority
Conclusion & Outlook

- Middleware prototype with multicast API
  - Supports native IP and Scribe multicast
  - Provides multicast API for C++ (and Java)
  - Runs on Linux and MacOS X

- Promising performance results

- Ongoing Work:
  - Extended IMG functionalities
  - Additional technology modules (e.g., spanning multicast tunnels)
Thank you …

Questions?

- Project Website:
  - http://hamcast.realmv6.org

- Prototype Release:
  - Friday next week (08.04.2011)