Implementation of a Middleware for Hybrid Adaptive Multicast (H∀Mcast)

Dominik Charousset

Summer semester 2010

Gliederung

Motivation

Problem Statement

- Multicast is most efficiently implemented on the lowest layer available
- Deployment status largely varies throughout the Internet
- Native (IPv4/v6) multicast is not globally reliably deployed

Motivation

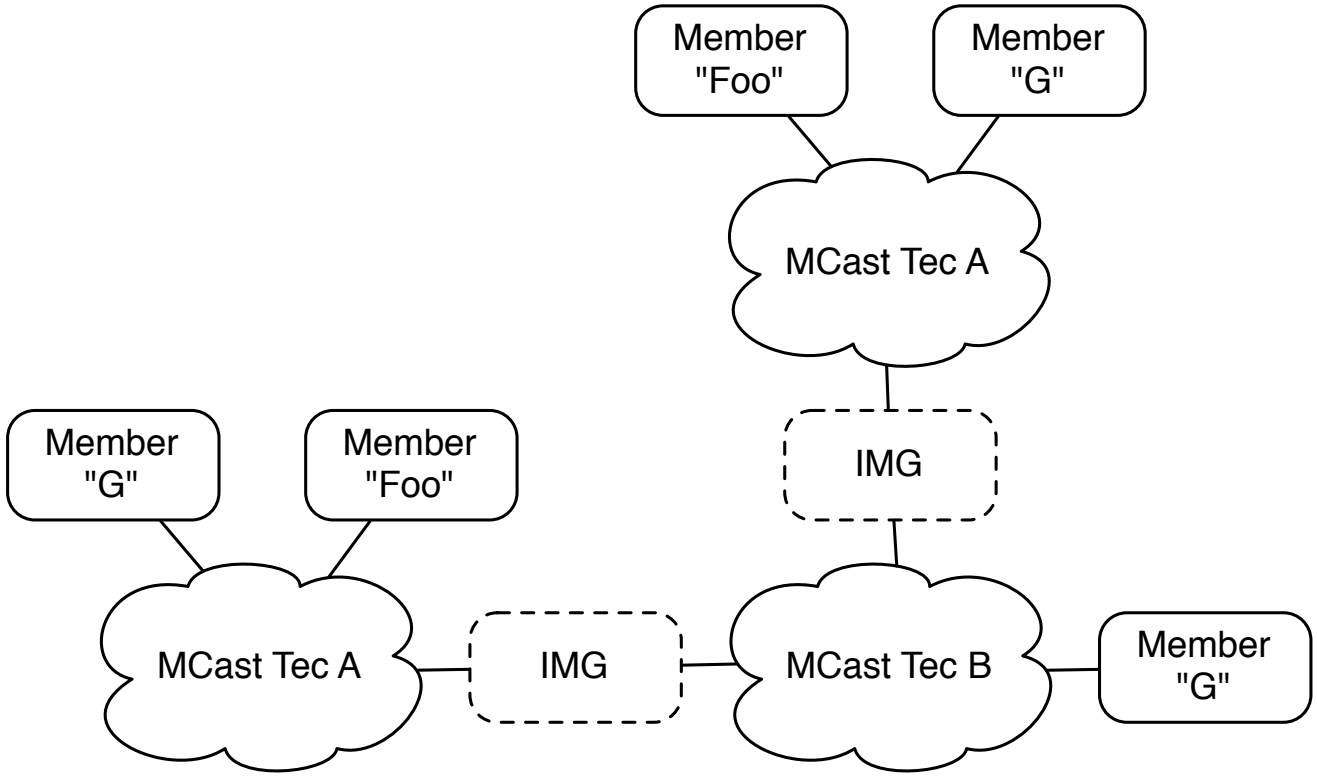
Problem Statement

- Multicast is most efficiently implemented on the lowest layer available
- Deployment status largely varies throughout the Internet
- Native (IPv4/v6) multicast is not globally reliably deployed

$\Rightarrow$ Overlay multicast globally available but of lesser efficiency

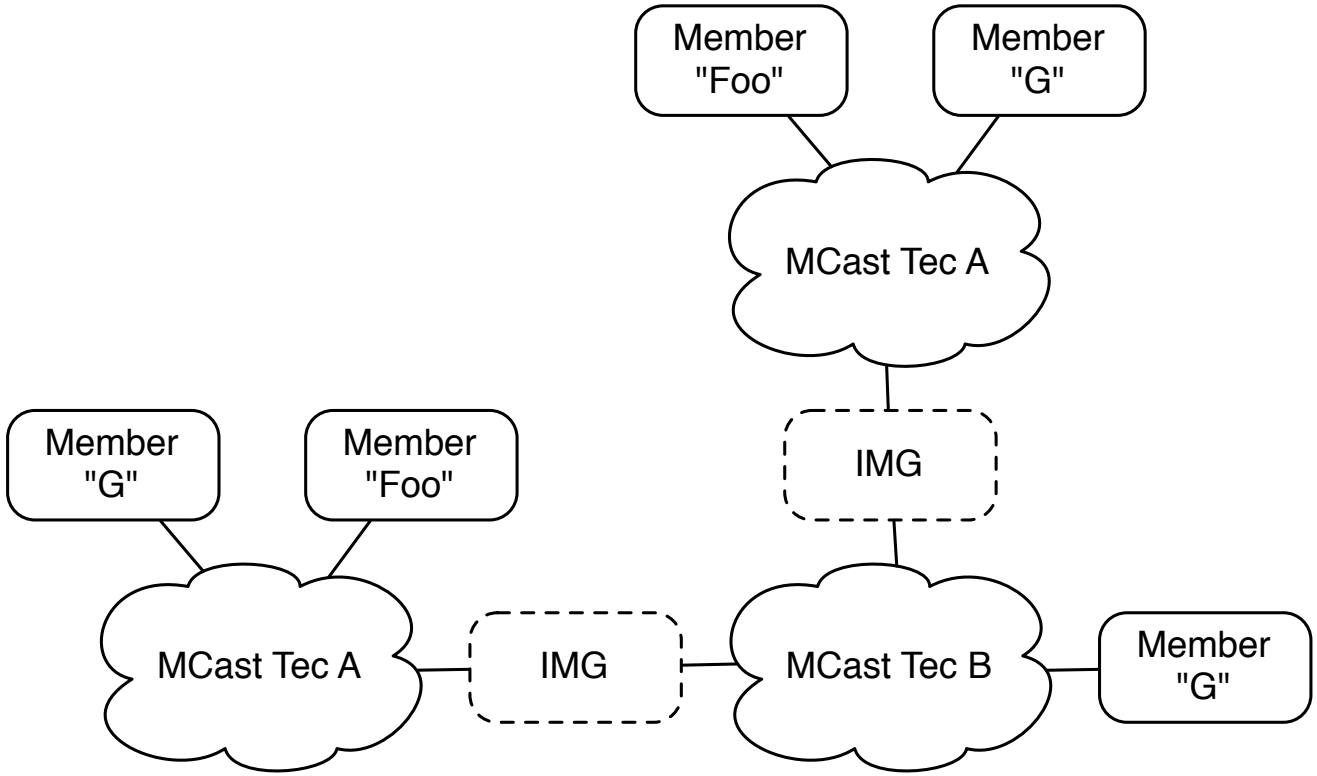
Motivation

H∀Mcast Reference Scenario



Motivation

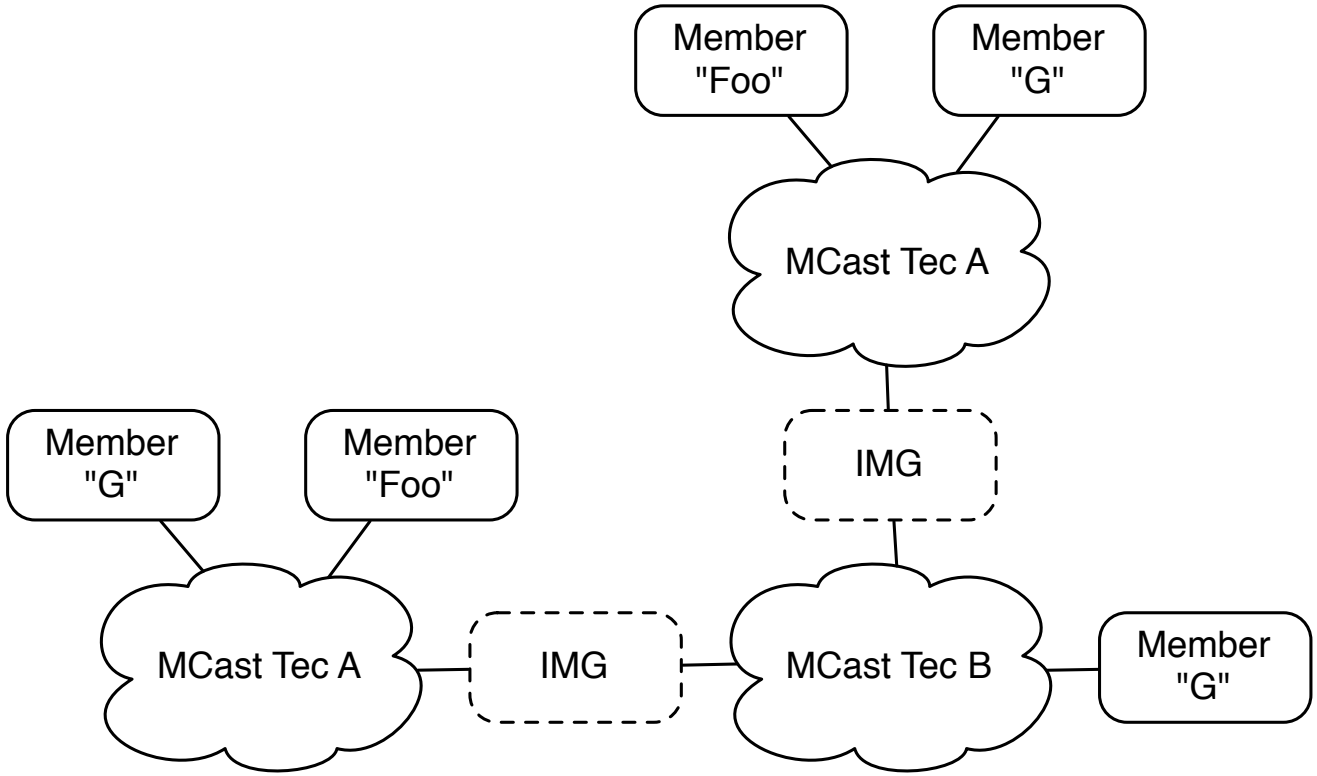
H∀Mcast Reference Scenario



- Hosts are members of the **same group** but use **distinct technologies**

Motivation

H∀Mcast Reference Scenario



- Hosts are members of the **same group** but use **distinct technologies**
- Hosts of the same group use **identical technologies**, but in **separated domains** (walled gardens)

Motivation

Goals of H∀Mcast

- Provide a common, efficient and easy-to-use multicast API

Motivation

Goals of H∀Mcast

- Provide a common, efficient and easy-to-use multicast API
- API describes a virtual multicast layer whose capabilities are not known at compile time

Motivation

Goals of H∀Mcast

- Provide a common, efficient and easy-to-use multicast API
- API describes a virtual multicast layer whose capabilities are not known at compile time
- Forward multicast data between different technologies
  - Deploy Interdomain Multicast Gateways (IMGs)

Motivation

Goals of H∀Mcast

- Provide a common, efficient and easy-to-use multicast API
- API describes a virtual multicast layer whose capabilities are not known at compile time
- Forward multicast data between different technologies
  - Deploy Interdomain Multicast Gateways (IMGs)
- Late binding
  - Technology selection at runtime
  - Locator/ID split (generic group addressing)

  $\Rightarrow$ make applications future-proof and technology independent

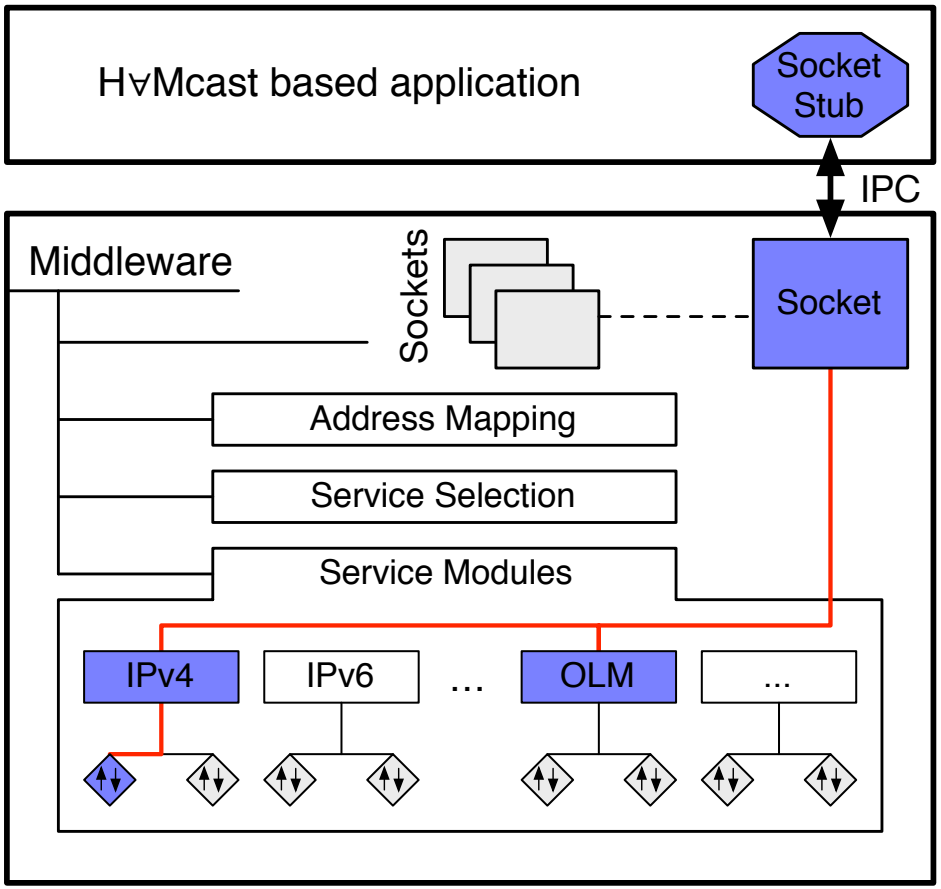
- API
  - Interface for application developers
  - Forwards all calls to the middleware via IPC

- API
  - Interface for application developers
  - Forwards all calls to the middleware via IPC
- Middleware
  - Implements the virtual multicast layer
  - Dispatches send/receive/join calls to *Service Modules*

- Service Modules
  - A Service Module is a technology specific implementation of the API
  - Modules are loaded at startup and perform a Service Discovery
  - Implemented as C interfaces
    - Fixed ABI
    - No name mangling (no compiler dependencies)

- Service Modules
  - A Service Module is a technology specific implementation of the API
  - Modules are loaded at startup and perform a Service Discovery
  - Implemented as C interfaces
    - Fixed ABI
    - No name mangling (no compiler dependencies)
- Address Mapping
  - Maps identifiers to locators
  - Might be ambiguous
    (e.g.: a hostname has both an IPv4 and an IPv6 address)

- Service Modules
  - A Service Module is a technology specific implementation of the API
  - Modules are loaded at startup and perform a Service Discovery
  - Implemented as C interfaces
    - Fixed ABI
    - No name mangling (no compiler dependencies)
- Address Mapping
  - Maps identifiers to locators
  - Might be ambiguous
    (e.g.: a hostname has both an IPv4 and an IPv6 address)
- Service Selection
  - Chooses the most efficient module available for a given identifier
  - Based on the results of the Service Discoveries

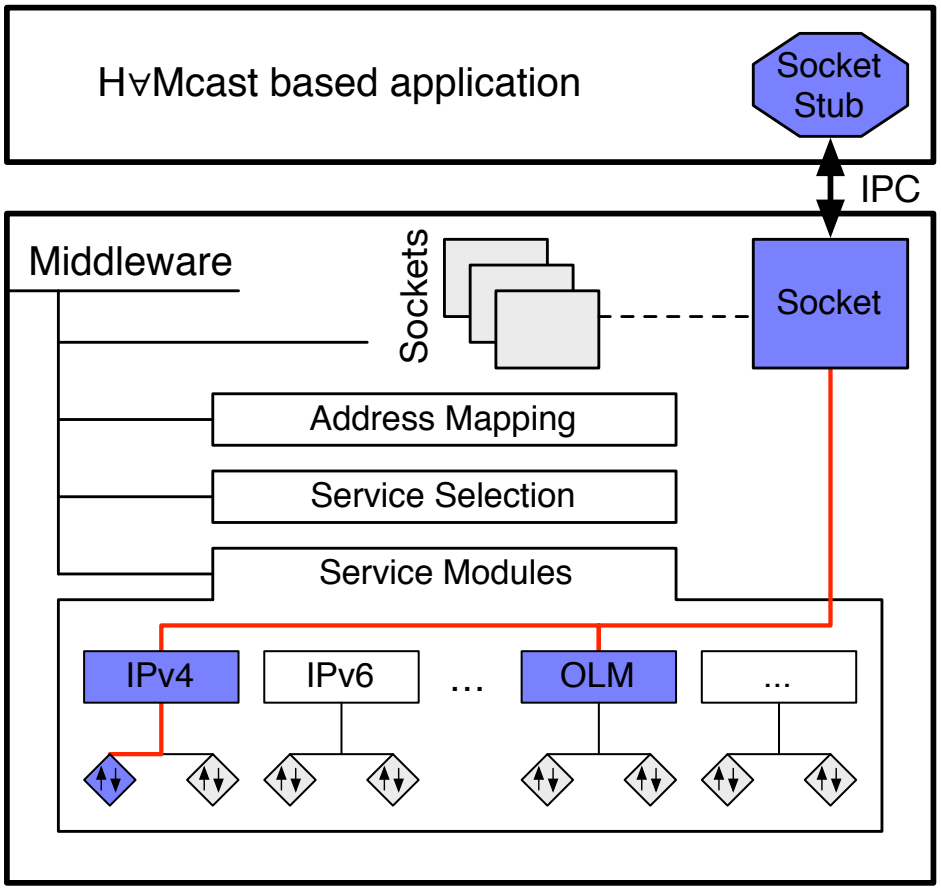


A H∀Mcast socket ...

- Unites any number of tech. specific sockets

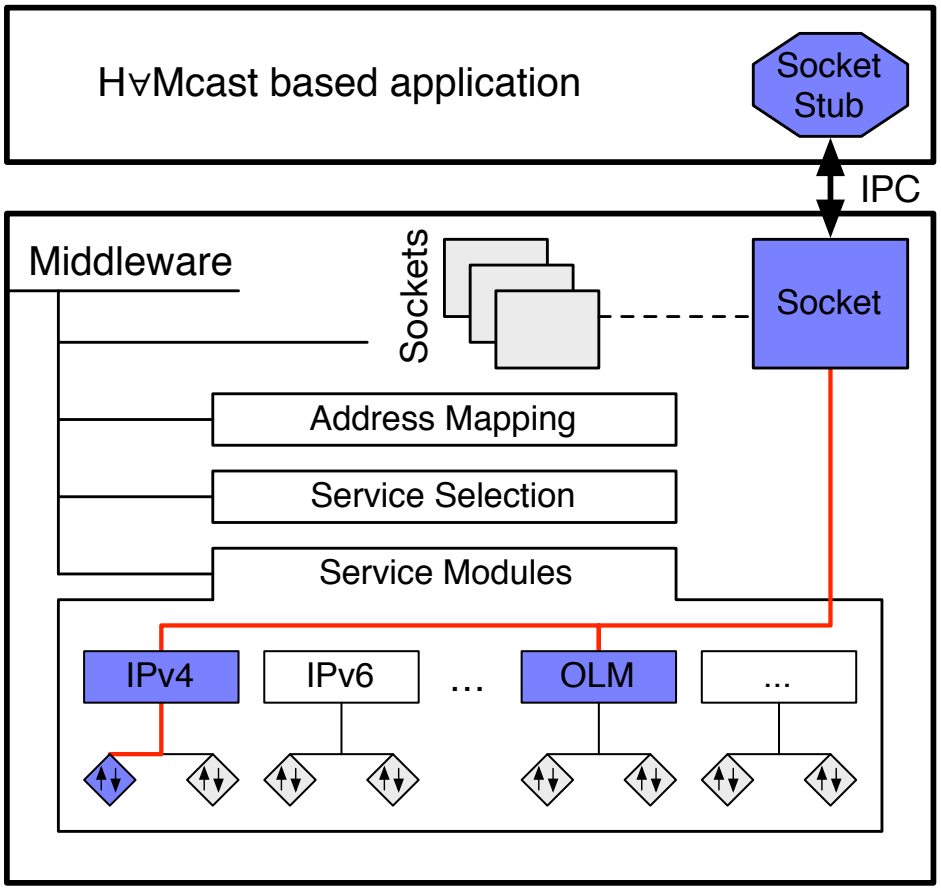
Architecture

Middleware Components – H∀Mcast Sockets



A H∀Mcast socket ...

- Unites any number of tech. specific sockets
- Send call:
  - Forward data to all assigned modules
  - Assign modules (if needed) by Service Selection

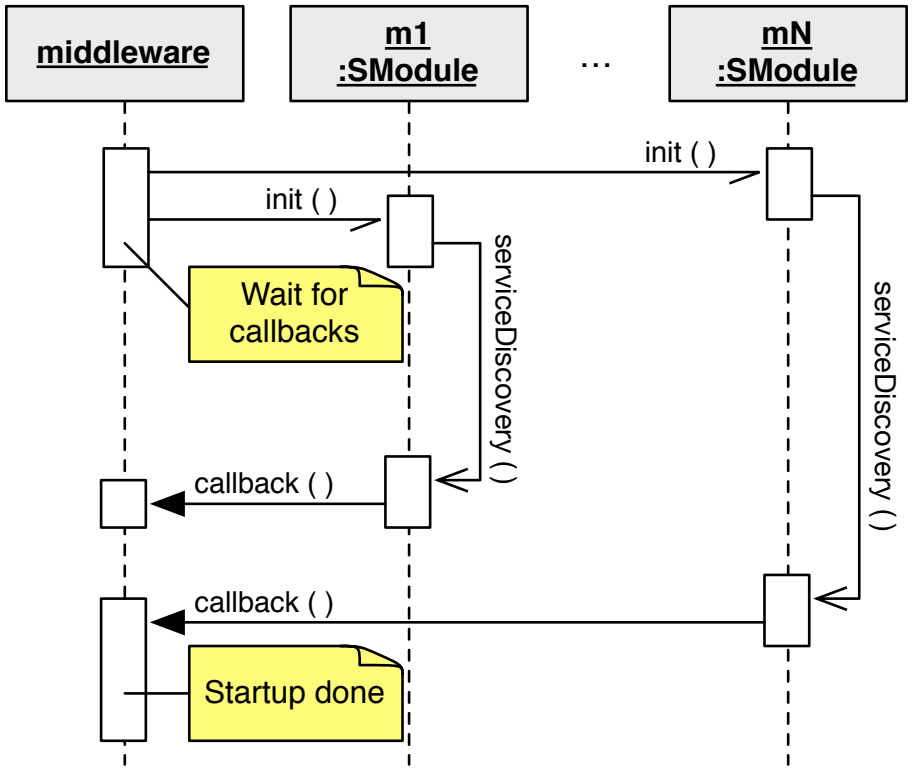


A H∀Mcast socket ...

- Unites any number of tech. specific sockets
- `Send` call:
  - Forward data to all assigned modules
  - Assign modules (if needed) by Service Selection
- `Receive` call:
  - Round-robin on all associated tech. sockets

Middleware

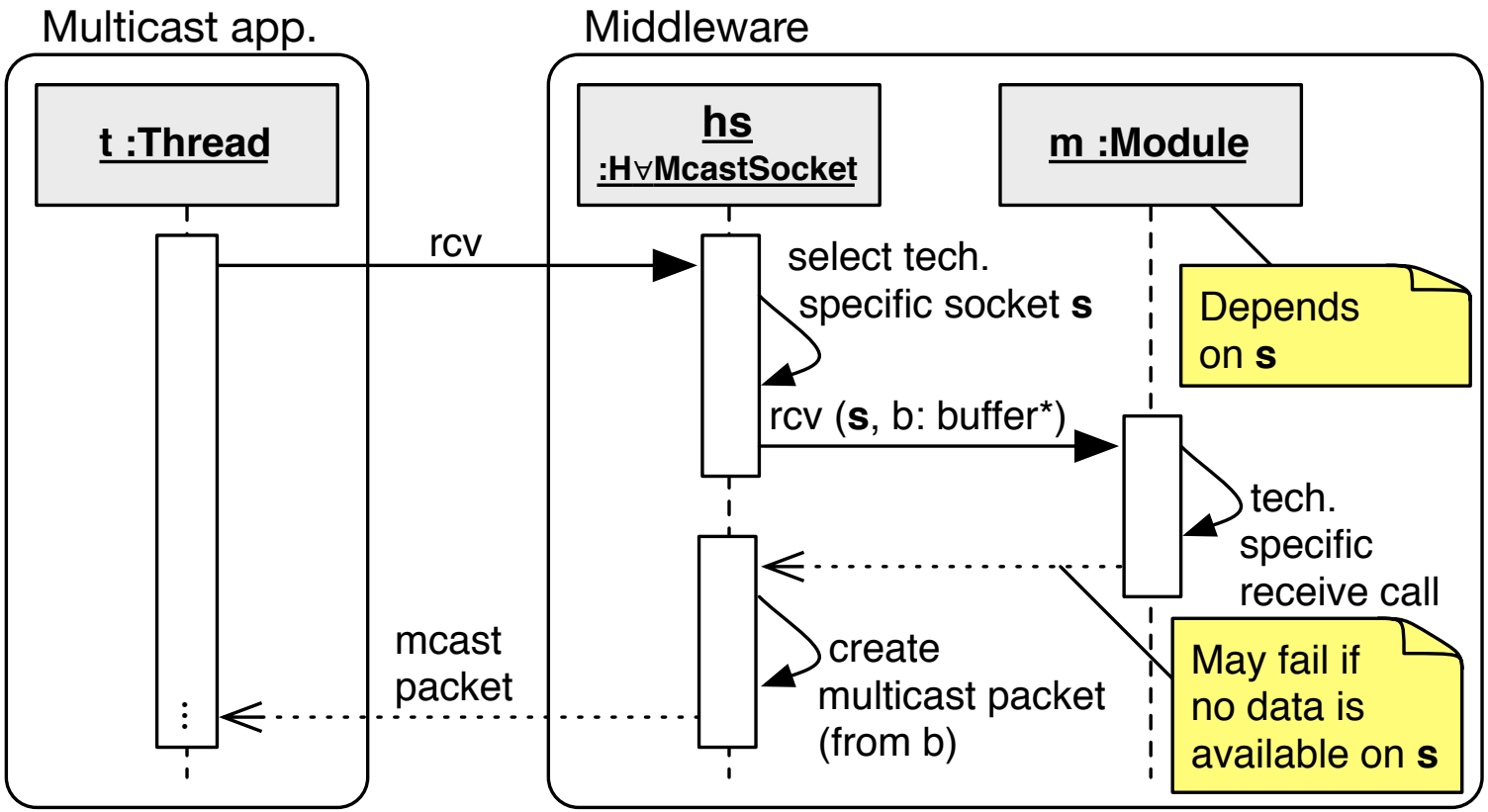
Startup Phase



- Service Discoveries (and/or Bootstrapping) runs asynchronously & parallell to minimize startup time
- The Service Discoveries are needed by the Service Selection

Middleware

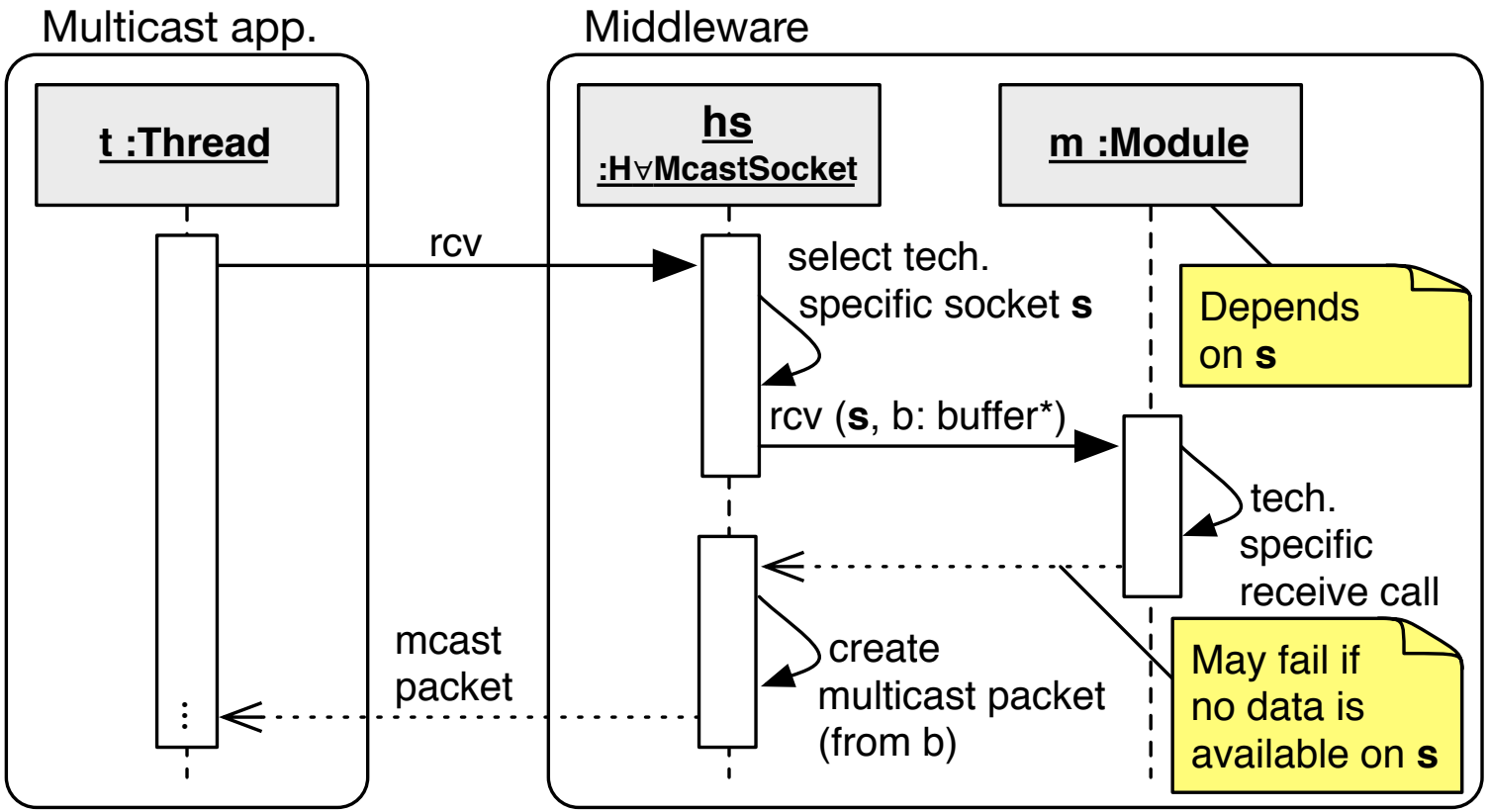
Receive Calls



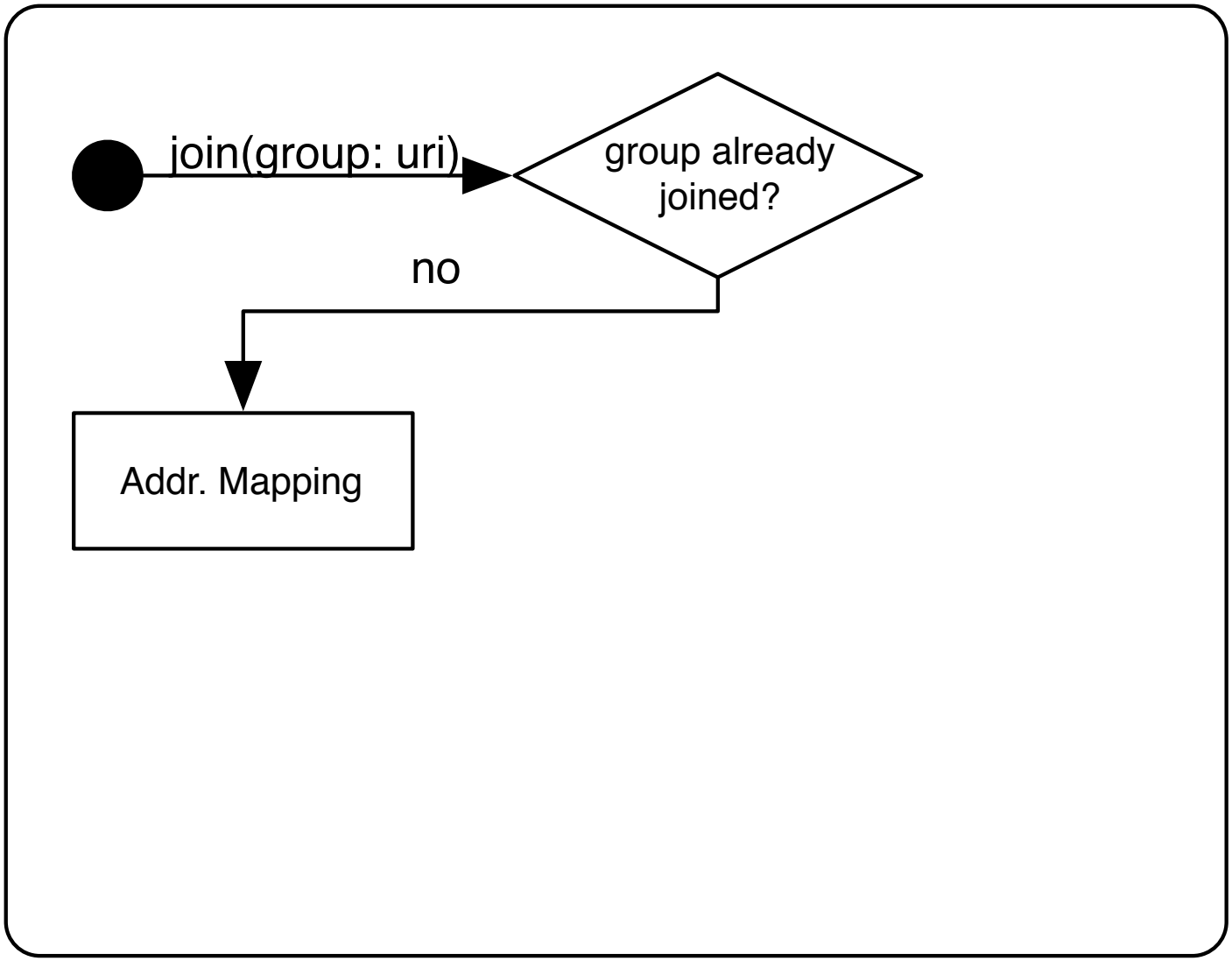
- **hs** selects tech. specific socket in round-robin order (avoid starvation)

Middleware

Receive Calls

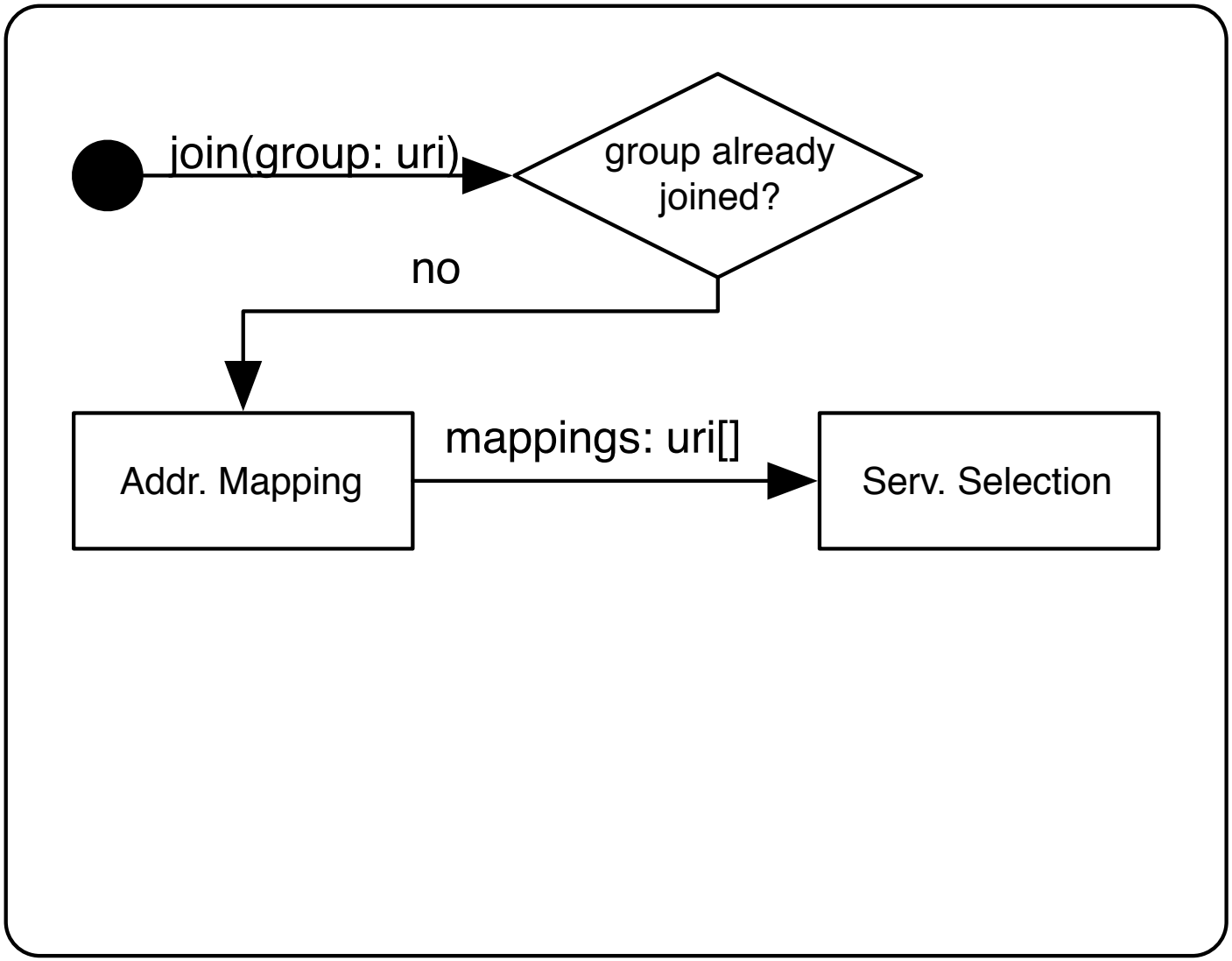


- **hs** selects tech. specific socket in round-robin order (avoid starvation)
- `Send` call is implemented similar


join(group: uri)
group already joined?
no
Addr. Mapping

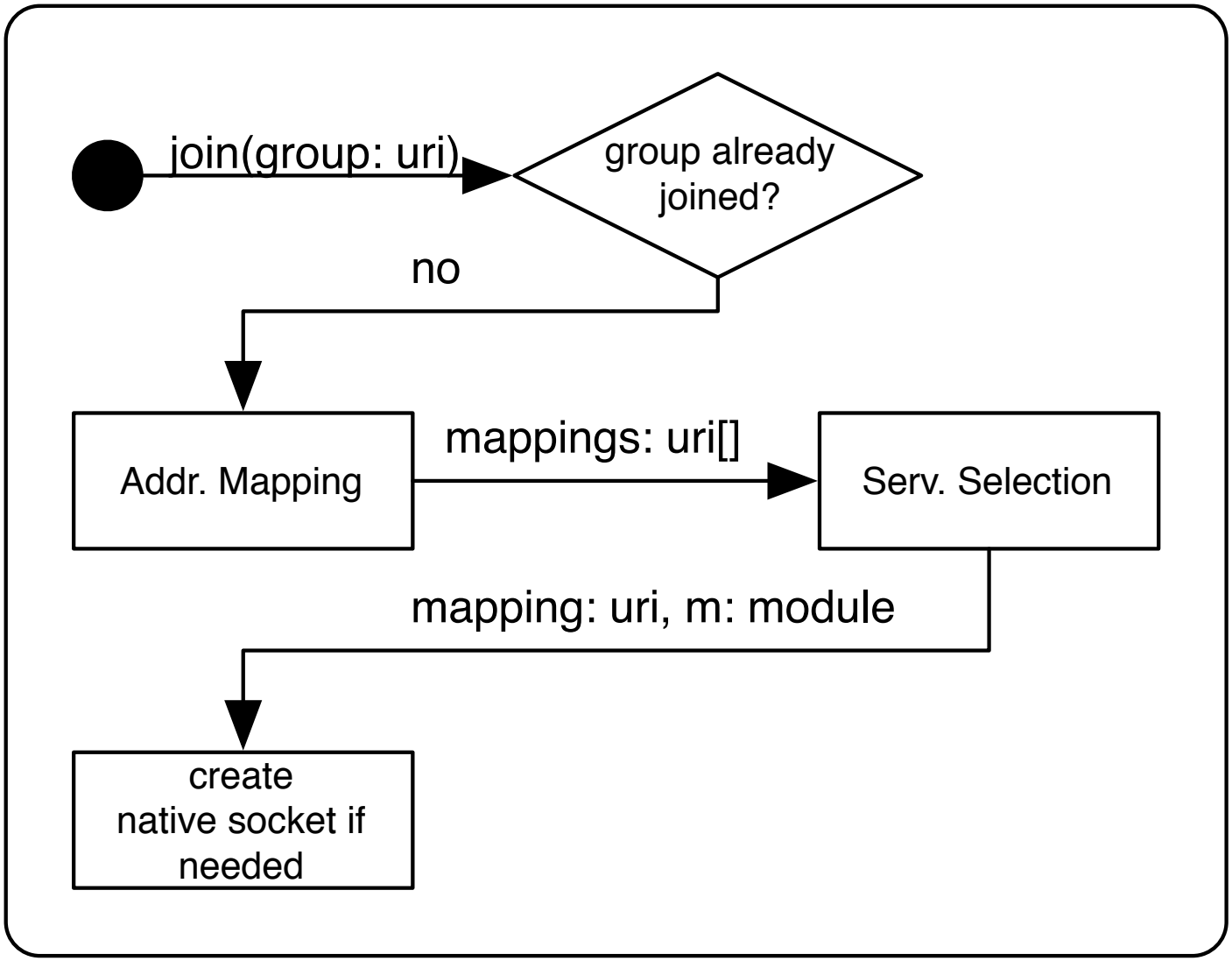
Middleware

Join Calls



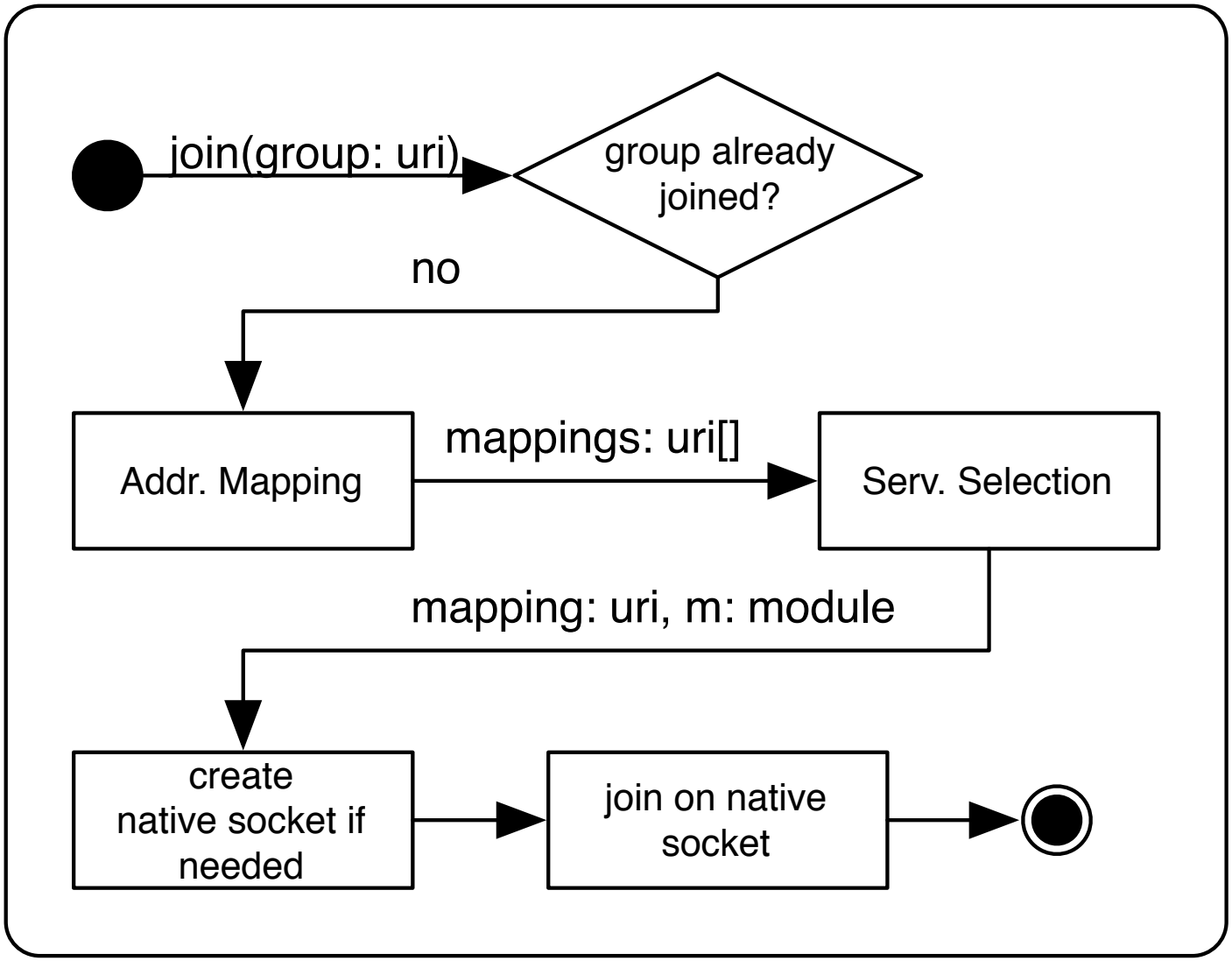
Middleware

Join Calls



Middleware

Join Calls



Middleware
IPC

H∀Mcast uses a localhost socket IPC implementation

- Data is serialized directly to sockets (no unnecessary overhead)
- IPC interface is independent of any platform/OS or language
- H∀Mcast API could be provided to every language that supports localhost sockets (almost every one: Java, C#, C, ...)

Middleware

Current state of the implementation

- IPC runs, but at this time there's only a C++ binding

Middleware

Current state of the implementation

- IPC runs, but at this time there's only a C++ binding
- IPv4/v6 and Scribe modules implemented; except for service discovery

Middleware

Current state of the implementation

- IPC runs, but at this time there's only a C++ binding
- IPv4/v6 and Scribe modules implemented; except for service discovery
- Service selection is mocked (always chooses the first available module)

Middleware

Current state of the implementation

- IPC runs, but at this time there's only a C++ binding
- IPv4/v6 and Scribe modules implemented; except for service discovery
- Service selection is mocked (always chooses the first available module)
- Mapping Service is an open issue

Outlook

Current ToDo List

- (Configurable) Logging to ease debugging of modules & middleware

Outlook

Current ToDo List

- (Configurable) Logging to ease debugging of modules & middleware
- Measurement of IPC overhead compared to native multicast apps

Outlook

Current ToDo List

- (Configurable) Logging to ease debugging of modules & middleware
- Measurement of IPC overhead compared to native multicast apps
- Service Discovery

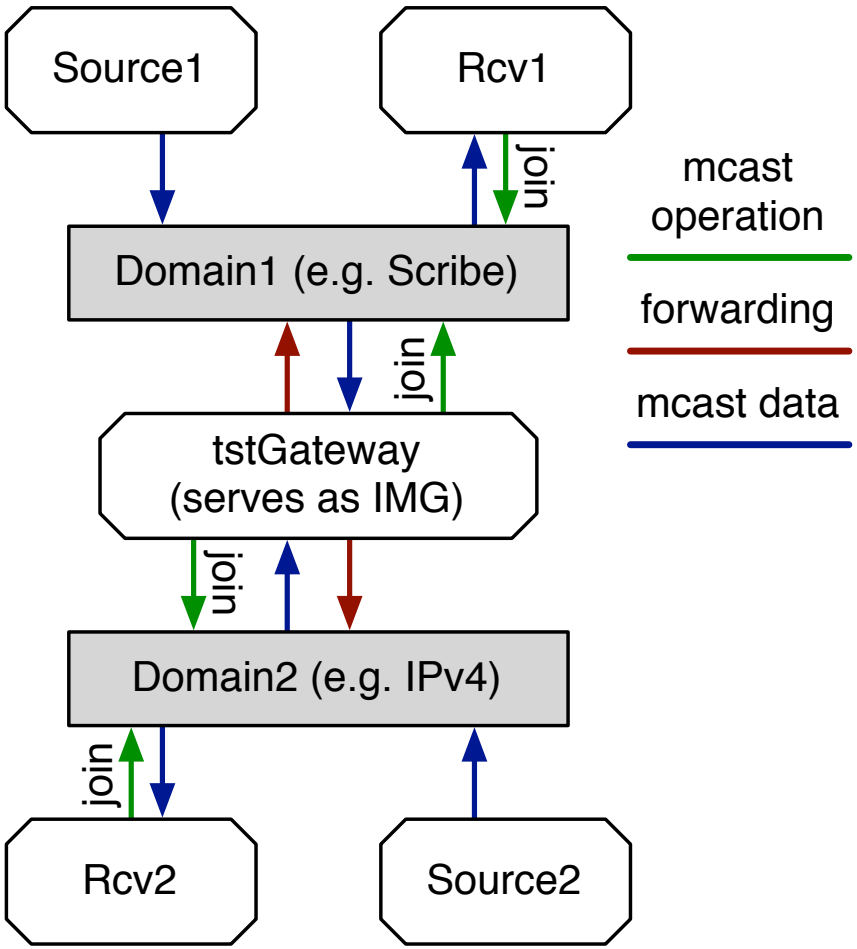
Outlook

Current ToDo List

- (Configurable) Logging to ease debugging of modules & middleware
- Measurement of IPC overhead compared to native multicast apps
- Service Discovery
- Implementation of an (IMG) test scenario (next slide)

Outlook

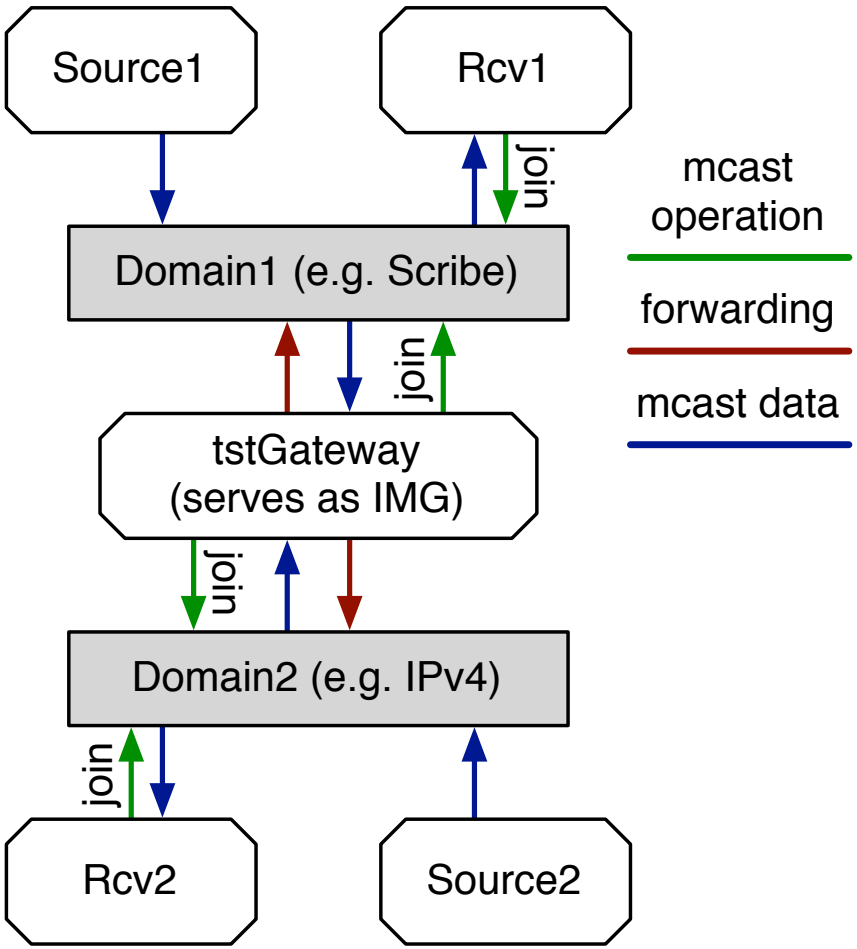
Test Scenario



- tstGateway as “hard wired” IMG

Outlook

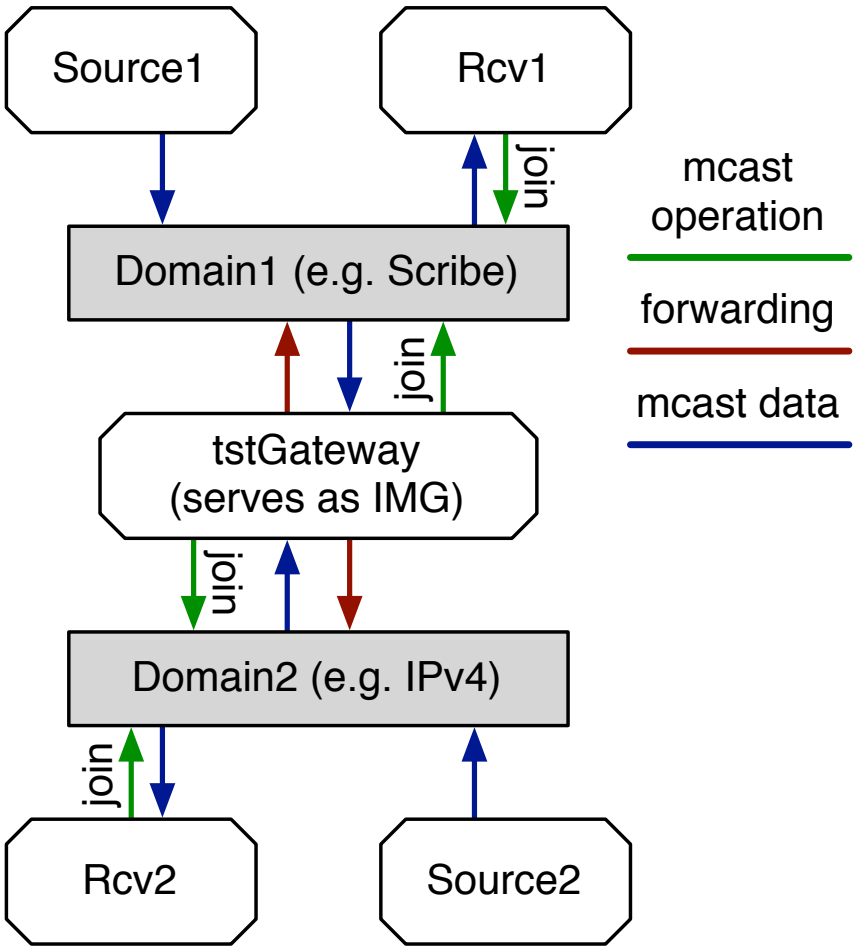
Test Scenario



- tstGateway as “hard wired” IMG
- Joins the multicast group in Domain1 & Domain2

Outlook

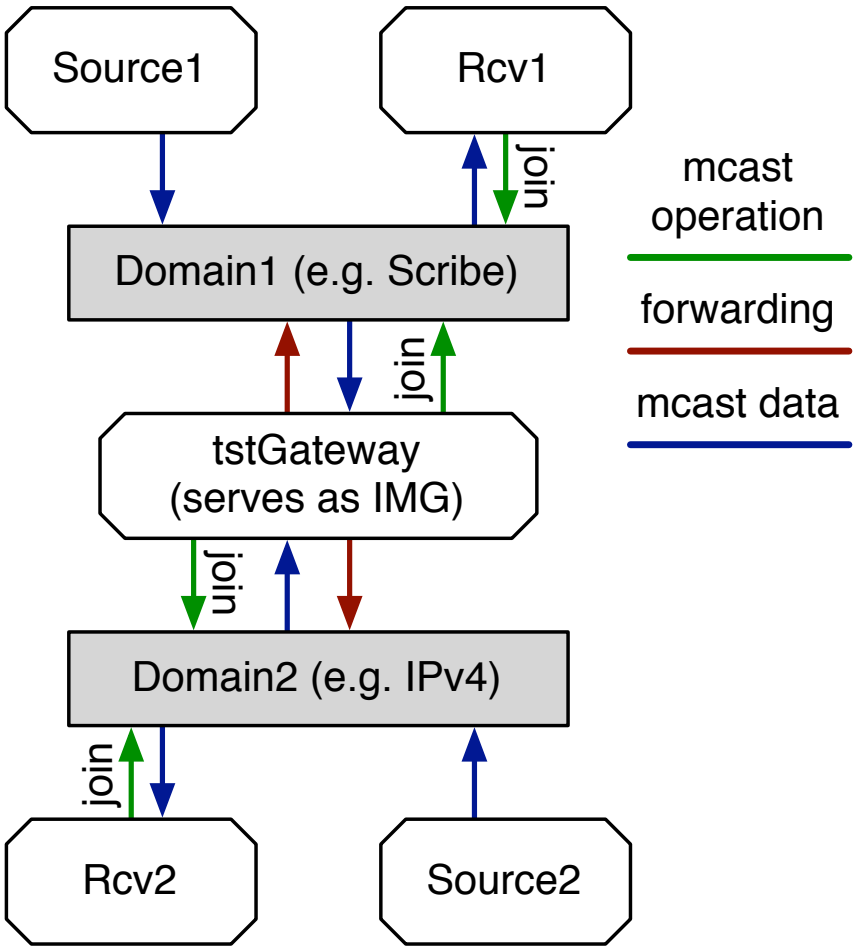
Test Scenario



- tstGateway as “hard wired” IMG
- Joins the multicast group in Domain1 & Domain2
- Forwards data from Domain1 to Domain2 and vice versa

Outlook

Test Scenario



- tstGateway as “hard wired” IMG
- Joins the multicast group in Domain1 & Domain2
- Forwards data from Domain1 to Domain2 and vice versa
- Test of middleware and modules in a realistic test case

Thank you for your attention!

Questions?