XORP Multicast Forwarding Engine Abstraction

Version 0.2

XORP Project
International Computer Science Institute
Berkeley, CA 94704, USA
feedback@xorp.org

March 10, 2003

1 Introduction

1.1 Overview

This document provides an overview of the XORP Multicast Forwarding Engine Abstraction (MFEA). It is intended to provide a starting point for software developers who wish to modify this software.

The main purpose of the MFEA is to abstract the underlying system and the multicast forwarding engine, and to provide a consistent interface to multicast-related modules such as PIM and MLD/IGMP. Thus, if we want to use PIM and MLD/IGMP on different OS platform or hardware, we would have to modify the MFEA only. In addition, all the complexity associated with network sockets, sending and receiving packets, and so on are moved away from the protocol modules. This eliminates code duplication, and reduces greatly the development overhead associated with protocol-independent issues when implementing a new protocol. In addition, if we want to use PIM and MLD/IGMP in a simulation-like environment, it will be sufficient to add the simulation environment support only to the MFEA.

Currently (March 2003), the MFEA supports abstraction for the following systems:

- {Free,Net,Open}BSD OS (tested only on FreeBSD-4.5).
- Linux OS (tested only on Red Hat Linux 7.2 with kernel 2.4.18)

In the future, the MFEA will support simulation environment, and abstraction for Click [1] forwarding path [1]. Support for other systems will be added on-demand if there are available resources.

For simplicity, currently (March 2003) the MFEA is a separate process from the unicast FEA. One of the reasons for that separation is because multicast routing/forwarding should have minimum impact on the unicast routing/forwarding. If the FEA and MFEA were a single process, then if the PIM-SM daemon for example starts sending a large number of requests to the unified FEA/MFEA process, then it may block the operation of that process, and it may affect not only multicast, but unicast routing/forwarding as well. If the FEA and the MFEA are separate processes, the impact will be localized to multicast only.

[1] Currently (March 2003) Click does not support multicast yet.

1.2 Acronyms

Acronyms used in this document:

- **MFC**: **M**ulticast **F**orwarding **C**ache: another name for an entry in the multicast forwarding engine (typically used on UNIX systems).
- **MFEA**: **M**ulticast **F**orwarding **E**ngine **A**bstraction
- **MLD/IGMP**: **M**ulticast **L**istener **D**iscovery/**I**nternet **G**roup **M**anagement **P**rotocol
- **MRIB**: **M**ulticast **R**outing **I**nformation **B**ase
- **PIM-SM**: **P**rotocol **I**ndependent **M**ulticast–**S**parse **M**ode
- **RIB**: **R**outing **I**nformation **B**ase

1.3 MFEA Design Architecture Overview

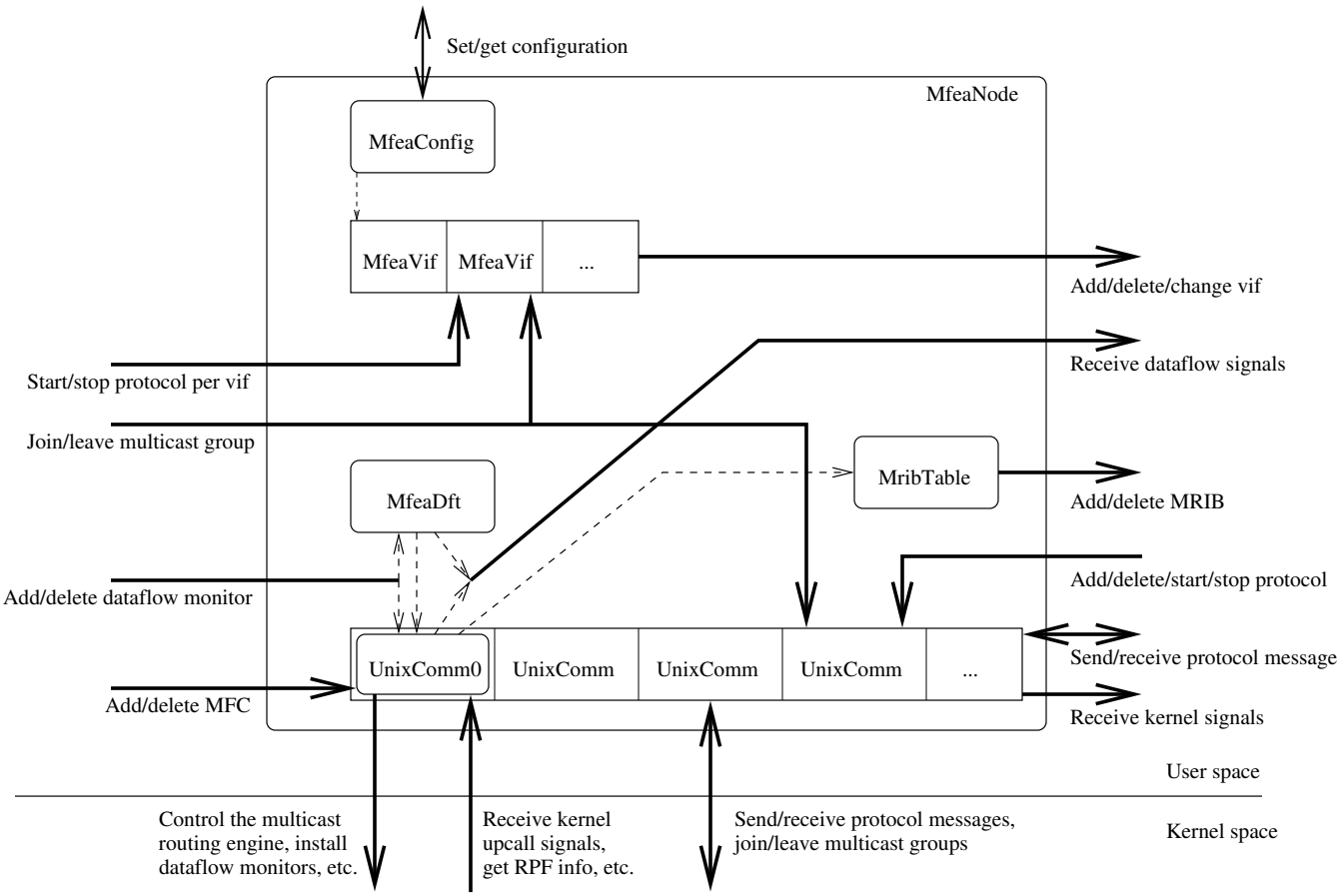

Figure 1: MFEA design overview

Figure 1 provides a general overview of the MFEA components. For each component there is a C++ class with exactly the same name. The main components are briefly described below:

- **MfeaNode:** a representation of a single MFEA unit (*e.g.,* as part of a virtual multicast router). Typically, there would be a single MfeaNode per multicast router.
- **MfeaVif:** MFEA-specific virtual (network) interface that is used to keep state per network interface.
- **MfeaConfig:** contains MFEA-specific configuration.
- **MfeaDft:** table with dataflow-related information for estimating the bandwidth per dataflow.
- **MribTable:** the table with MRIB information.
- **UnixComm:** per-protocol UNIX-specific unit for communication with the underlying system.
- **UnixComm0:** special protocol-independent UnixComm unit for communication with the underlying system.

Those components are described in details in Section 2. For information about the interaction between the MFEA and other modules see [2].

2 Components Description

2.1 MfeaNode Description

MfeaNode is a representation of a single MFEA unit (*e.g.,* as part of a virtual multicast router). Typically, there would be a single MfeaNode per multicast router. However, in some cases a multicast router may have more than one MFEA units. For example, it could have one MfeaNode for IPv4, and another one for IPv6 multicast routing. Further, if we want to run MFEA in a simulation environment, each multicast router within that simulation will have a single MfeaNode.

From a developer's point of view, MfeaNode contains all the state related to the MFEA unit, and exports the front-end interface to interact with that unit. For example, MfeaNode contains the methods to start/stop or configure the MFEA, or to send/receive protocol control packets (*e.g.,* PIM or MLD/IGMP) to/from the unit. Those methods are described in the following files:

- `mfea/mfea_node.hh`
- `libproto/proto_node.hh`
- `libproto/proto_unit.hh`

MfeaNode provides the following abstraction to the multicast-related modules such as PIM and MLD/IGMP:

- Interface to add/delete/start/stop a protocol within the MFEA and the underlying system.
- Interface to send or receive protocol packets through the underlying system.
- Interface to receive (kernel) upcalls/signals from the underlying system, and to forward those signals to the multicast-related modules that are interested in receiving those signals. Examples of such signals in case of UNIX kernel are NOCACHE or WRONGVIF/WRONGMIF: the former one is sent when the underlying multicast forwarding engine has no multicast forwarding entry for a multicast packet; the latter one is sent when a multicast data packet has arrived on an interface that is not the expected incoming interface to forward that data packet.

- Interface to add/delete dataflow monitors, to monitor network bandwidth per dataflow, and to send the appropriate signals to the interested multicast-related modules. For example, the PIM-SM module might be interested to know when the bandwidth of a given dataflow becomes zero, or is above a threshold.
- Interface to inform the multicast-related modules about the available virtual interfaces (*e.g.,* network interfaces, tunnels, etc.) on the system, and to inform them about any changes on those interfaces (*e.g.,* interface going DOWN, network address change, etc.).
- Interface to join or leave a multicast group.
- Interface to add/delete MFC entries, *i.e.,* entries to the multicast forwarding engine.
- Interface to monitor the state of the underlying unicast forwarding table, to construct the MRIB table from it, and to send the MRIB information to the interested multicast-related modules.

MfeaNode itself does not implement the mechanisms to communicate with the multicast-related modules (*e.g.,* to send or receive control packets to/from the PIM module). Those mechanisms are outside the scope of MfeaNode, and must be implemented separately.

MfeaNode contains several pure virtual methods (*e.g.,* `send_add_mrib()` is used to add MRIB entry to a multicast-related module that is interested in tracking the MRIB) that must be implemented by a class that inherits MfeaNode. For example, XrlMfeaNode is a class that uses MfeaNode as a base class; XrlMfeaNode uses XRL-based communication mechanisms between MfeaNode and other XORP components such as the PIM and MLD/IGMP modules.

By default, MfeaNode is disabled; therefore, on startup it must be enabled explicitly.

2.2 MfeaVif Description

MfeaVif is a MFEA-specific virtual (network) interface that is used to keep various state per interface. Typically, there would be one MfeaVif per network interface such as physical interface, tunnel, or the loopback interface. In addition, there is one special MfeaVif: the PIM Register virtual interface that is used for sending and receiving PIM Register packets [2].

One of the purposes of MfeaVif is to keep various information about each network interface available on the system: network and subnet address, is the interface up or multicast-capable, and so on. This information is used by the MFEA to keep track of any changes to an interface (*e.g.,* an alias address has been added/deleted, etc). If there is a change to an interface, the MFEA informs all protocol modules that have enabled the specific protocol on that interface [3].

Another purpose of MfeaVif is to keep track of the multicast groups that have been joined per interface. For example, if a multicast-related module that uses the MFEA joins a multicast group on an interface, the MFEA uses the appropriate system call to join the group on the specified interface, and at the same time it would keep the appropriate state on the corresponding MfeaVif. Thus, if another module joins exactly same multicast group on that interface, but later one of those modules leaves that group, the MFEA would modify only the appropriate state in MfeaVif, but will not use a system call to leave the multicast group on the interface.

[2] In the future (after March 2003), the PIM Register MfeaVif interface may not be part of the MFEA anymore, because it is strictly PIM-specific.

[3] Currently (March 2003), the MFEA reads the interface status from the underlying system only once (on startup); instead, it should keep monitoring the interface status by periodically reading that status for example, and/or by considering the information provided by the XORP Router Manager (within the XORP framework).

Typically, from developer's point of view, all interactions with MfeaVif would be through MfeaNode [4].

The public interface for MfeaVif contains the methods to manipulate a virtual (network) interface. Those methods are to start/stop/enable/disable a virtual interface, and to configure it. The methods are described in the following files:

- `mfea/mfea_vif.hh`
- `libxorp/vif.hh`
- `libproto/proto_unit.hh`

By default, each MfeaVif is disabled; therefore, on startup it must be enabled explicitly.

2.3 MfeaConfig Description

MfeaConfig handles the MFEA-specific configuration [5]. This configuration is used to configure the following units:

- MfeaNode: how often to read the unicast forwarding table, default routing metrics and metric preferences to assign to each route, etc.

2.4 MfeaDft Description

Some protocols such as PIM-SM need the bandwidth of multicast data flows to be monitored: if the bandwidth of a specific data flow (defined by a source and a group address) is above or below a pre-defined threshold (defined per data flow), the protocol should be informed. For example, if the bandwidth of a data flow is zero for some predefined amount of time, the corresponding multicast routing entry in the multicast routing protocol module might be deleted (as well as the corresponding multicast forwarding entry in the multicast forwarding engine). Another example is the Shortest-Path Tree switc in case of PIM-SM: the SPT switch is triggered if the bandwidth from a specific source is above a pre-configured threshold.

If the multicast forwarding engine in the underlying system does support bandwidth dataflow monitoring, then any addition or deletion of a dataflow monitor to the MFEA translates to a system call to the underlying system, and the MFEA itself does not need to keep any state. However, if the underlying system does not support bandwidth dataflow monitoring, then the MFEA needs to implement that on its own. In case of UNIX kernel for example, the kernel supports an ioctl() system call to obtain the number of octets and packets forwarded so far on an existing MFC entry. Thus, if the MFEA reads this information twice, it can compute the data bandwidth between the two readings.

The purpose of the MfeaDft table is to keep state about the dataflows the MFEA is monitoring (only in the case the underlying system does not support bandwidth dataflow monitoring). For each entry in MfeaDft, the MFEA periodically reads the bandwidth forwarding statistics from the underlying system. If the forwarded bandwidth satisfies the pre-defined condition for that dataflow, the MFEA originates a dataflow signal to the module that has installed that dataflow monitoring entry. This signal is delivered every time the pre-defined condition is true (until the entry is explicitly deleted by the module that installed it).

[4] For simplicity, currently (March 2003) there are few occasions when XrlMfeaNode uses direct access to MfeaVif.

[5] Currently (March 2003), MfeaConfig is not implemented; rather, all state is kept inside MfeaNode instead.

2.5 MribTable Description

MribTable is the table with the MRIB information. Whenever this information changes, the change is propagated to all interested parties that have registered interest in it. For example, in case of PIM-SM the MRIB is used to compute the Reverse-Path Forwarding information toward the RPs or the senders. This information contains the next-hop router address and the interface toward that router, the routing metric and the metric preference.

Currently (March 2003), the MRIB information is obtained by periodically reading the underlying unicast forwarding table. In the future, the MRIB would be obtained by the interested parties directly from the RIB module instead of the MFEA module; the MFEA itself should be used only as a backup solution, and only if explicitly enabled for that purpose.

2.6 UnixComm Description

UnixComm is a per-protocol UNIX-specific unit for communication with the underlying system (the only exception is the very first unit, which is called UnixComm0 and which is described in Section 2.7).

UnixComm implements various UNIX-specific methods that use the appropriate system calls to open or close a socket per protocol, to send or receive protocol packets, to join or leave a multicast group (per protocol per interface), and so on. Typically, there is a single UnixComm entry per protocol; *e.g.,* one UnixComm for PIM, another one for IGMP (or MLD in case of IPv6), etc. When a protocol module registers a network protocol with MFEA, the corresponding UnixComm for that protocol is created (if it did not exist).

Each UnixComm entry is also used to keep information about various protocol preferences: *e.g.,* whether a protocol module instance is interested in receiving various kernel upcall signals, whether it needs to receive MRIB information, etc. [6]

2.7 UnixComm0 Description

UnixComm0 is a special protocol-independent UnixComm unit used for communication with the underlying system. For example, UnixComm0 is used for the following tasks (this list may not be complete). In other words, UnixComm0 is used for all purposes that do not require protocol-specific sockets:

- Start/stop the multicast forwarding engine.
- Add/delete an interface used for multicast forwarding by the underlying system.
- Add/delete a MFC entry within the multicast forwarding engine.
- Install dataflow monitors (only if the underlying system supports that feature).
- Read data bandwidth forwarding statistics (per dataflow).
- Read information about the available network interfaces in the system, and report any changes about it (the latter one is not implemented yet).
- Read the underlying unicast forwarding table, and report any changes about it.
- Send/receive protocol-specific packets (e.g., PIM or IGMP) to/from the network interfaces.

[6] Strictly speaking, this information should not be kept in the system-specific UnixComm entry, but in some system-independent entry. For simplicity, for now this state is kept in UnixComm; in the future (after March 2003), UnixComm may contain only the system-specific information and methods.

UnixComm0 is started when MfeaNode is started, and usually stops operation when MfeaNode is stopped.

A Modification History

- December 11, 2002: Version 0.1 completed.
- March 10, 2003: Updated to match XORP version 0.2 release code; cleanup.

References

[1] The Click Modular Router Project. http://www.pdos.lcs.mit.edu/click/.

[2] XORP Multicast Routing Design Architecture. XORP technical document. http://www.xorp.org/.