XORP Forwarding Engine Abstraction

Version 1.2

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

March 8, 2006

1 Introduction

The role of the Forwarding Engine Abstraction (FEA) in XORP is to provide a uniform interface to the underlying forwarding engine. It shields XORP processes from concerns over variations between platforms. As a result, XORP processes need not be concerned whether the router is comprised of a single machine, or cluster of machines; or whether the network interfaces are simple, like a PCI Ethernet adapter, or are smart and have processing resources, like an Intel IXP cards.

The FEA performs four distinct roles: *interface management*, *forwarding table management*, *raw packet I/O*, and *TCP/UDP socket I/O*. Those are described briefly in Section 1.1, Section 1.2, Section 1.3, and Section 1.4 respectively. Section 2 presents the design and implementation of the FEA components. FEA status summary is in Section 3.

In a standard XORP system, the Multicast Forwarding Engine Abstraction (MFEA) is part of the FEA. The MFEA is conceptually distinct from FEA and is used for multicast-specific abstraction of the underlying system. Combining the MFEA with the FEA reduces the load on the system. For information about the MFEA architecture, see [1].

1.1 Interface Management

In the normal course of interaction, the RouterManager process is the principal source of interface configuration requests to the FEA. The RouterManager constructs the interface configuration from the router configuration files and the input it receives at the command line. The type of requests the RouterManager sends to the FEA are to enable interfaces, create virtual interfaces, set interface MTU's, and so forth. The FEA interprets and executes these requests in a manner appropriate for the underlying forwarding plane.

Processes can register with the FEA to be notified of changes in interface configuration. The registered processes are notified of changes, and may query the FEA on the receipt of an update notification to determine the change that occurred. These notifications are primarily of interest to routing protocols since these need to know what the state of each interface is at a given time.

Both of the above interactions are depicted in Figure 1.

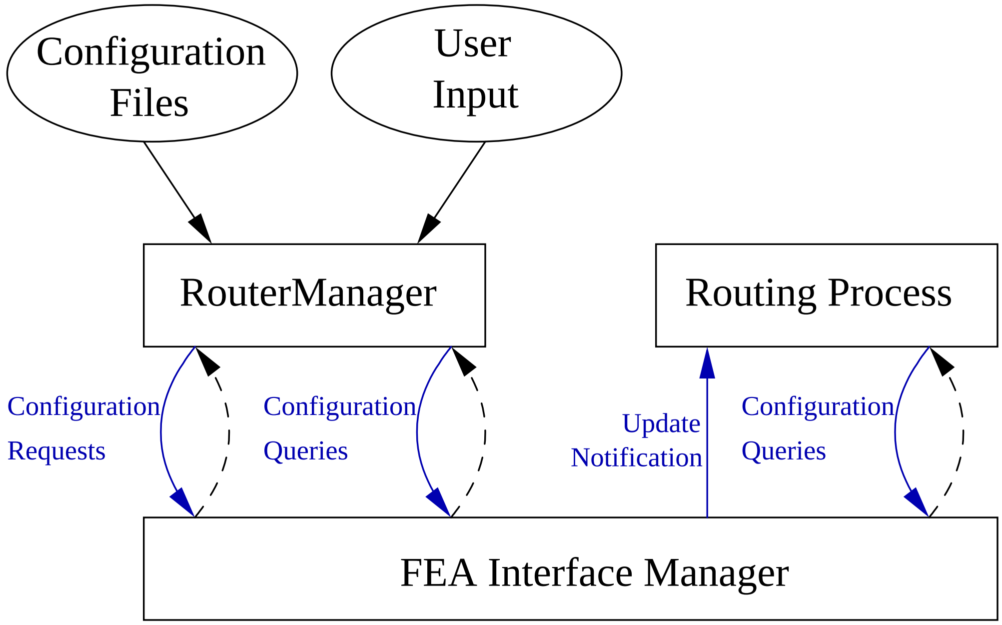

Figure 1: FEA Interface Management interaction with other XORP processes

1.2 Forwarding Table Management

The FEA primarily receives forwarding table configuration information from the RIB process. The RIB arbitrates between the routes proposed by the different routing processes, and propagates the results into the FEA's forwarding table interface. The FEA accepts requests to insert and remove routing entries and propagates the necessary changes into the forwarding plane. The FEA also supports queries on the current contents of the forwarding table. Finally, processes can register with the FEA to receive update notifications about changes to the forwarding table.

1.3 Raw Packet I/O

Routing protocols, such as OSPF, need to be able to send and receive packets on specific interfaces in the forwarding plane in order to exchange routing information and to determine the liveness of connected paths. Since the forwarding plane may be distributed across multiple machines, these routing protocols delegate the I/O operations on these packets to the FEA. The FEA supports sending and receiving raw packets on specific interfaces.

The transmission of raw packets through the FEA is straightforward, the routing process simply hands the FEA a raw packet and indicates which interface it should be sent on. The reception of raw packets is handled through a register-notify interface where the routing process registers which types of packets on which interfaces it is interested.

1.4 TCP/UDP Socket I/O

Routing protocols, such as BGP or RIP, need to be able to send and receive TCP or UDP packets to/from a specific IP address in order to establish peering connectivity and to exchange routing information. Similar to the raw packet I/O delegation, the FEA can be used to delegate the TCP/UDP socket I/O operations.

The handling of TCP or UDP operations is done by simply extending the UNIX TCP/UDP socket interface such that all relevant socket operations have XRL front-end interface.

2 Design and Implementation

2.1 Overview

The FEA fulfills four discrete roles: Interface Management, Forwarding Table Management, Raw Packet I/O, and TCP/UDP Socket I/O. The Interface Management and Forwarding Table Management roles follow a similar design pattern since both relate to the setting and getting of configuration state. The Raw Packet I/O and TCP/UDP Socket I/O have little in common with the other two roles.

The Interface Management and Forwarding Table Management roles use transactions for setting configuration state. The transactions are a collection of grouped operations that are queued until committed or aborted. Transactions provide atomic updates to the forwarding plane, which has the virtue of ensuring a consistent state at any particular instant of time. In addition, forwarding plane updates may incur per update costs, and grouping operations may help to reduce these. Queries of the configuration state happen on the immediate state, and are independent of any transactions that are in progress.

The FEA, as with other XORP processes, uses the XRL mechanism for inter-process communication and each role of the FEA is represented by a distinct XRL interface. The Interface Management, Raw Packet I/O and TCP/UDP Socket I/O roles support the notion of clients that notified when event occur and client processes are expected to implement known interfaces. The FEA XRL and FEA XRL client interfaces are shown in Table 1.

Role	XRL Interface file	Client XRL Interface
Interface Management	`fea_ifmgr.xif`	`fea_ifmgr_client.xif`
Forwarding Table Management	`redist_transaction{4,6}.xif`	`fea_fib.xif`
Raw Packet I/O	`fea_rawpkt{4,6}.xif`	`fea_rawpkt{4,6}_client.xif`
TCP/UDP Socket I/O	`socket{4,6}.xif`	`socket{4,6}_user.xif`

Table 1: FEA XRL Interfaces (defined in `$XORP/xrl/target/fea.tgt`)

The XRL handling code is found in `$XORP/fea/xrl_target.{hh,cc}`. Each XRL interface is handled by an XRL-aware helper class. The helper class understands the semantics of the implementation, and maps errors and responses to the appropriate XRL forms. The helper classes and their relations to the interfaces are depicted in Figure 2.

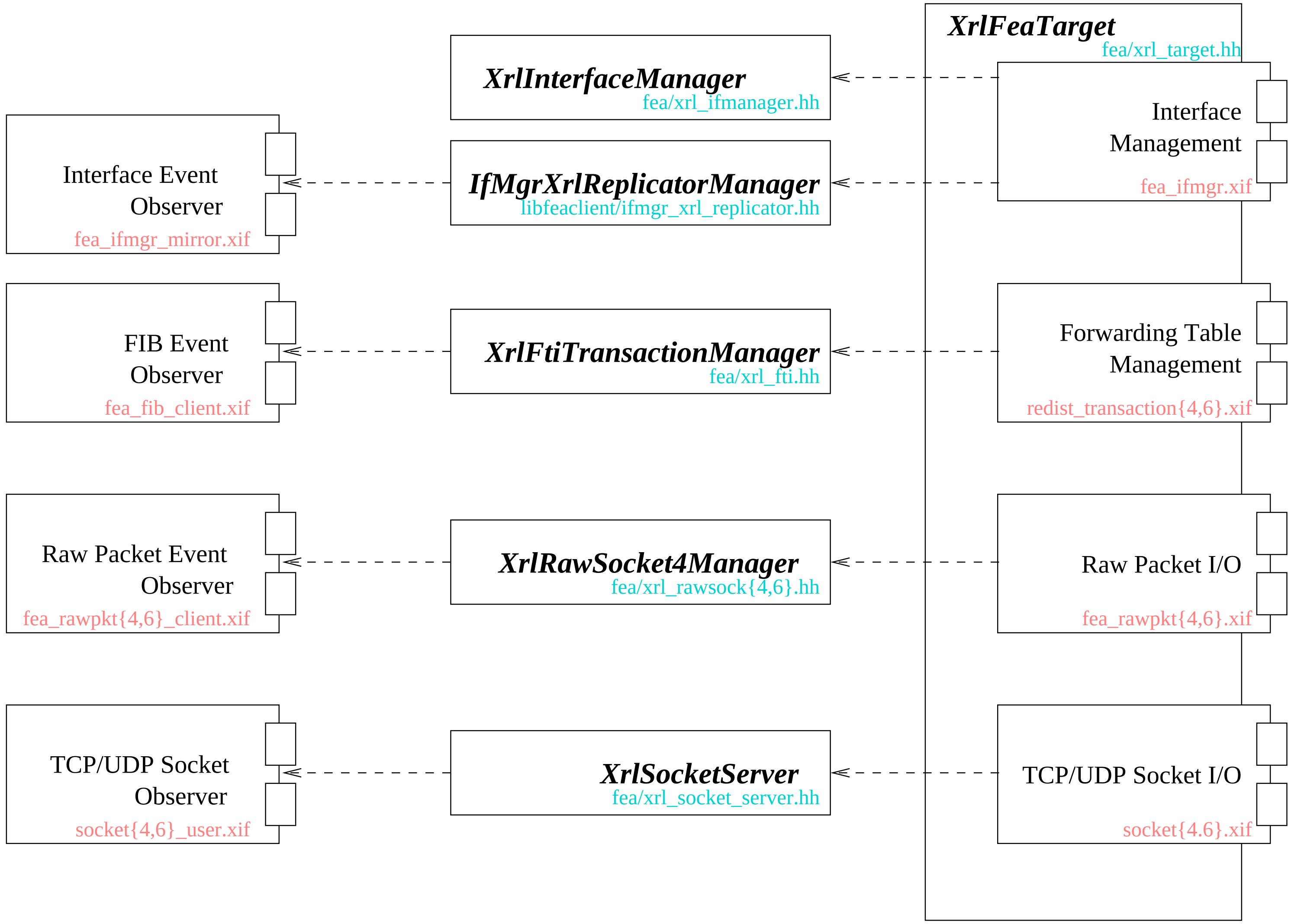

Figure 2: XRL Interfaces in relation to FEA classes

2.2 Interface Management

To succinctly explain the interface management classes and how they interact we first describe the representation of interface configuration state. Interface configuration state is held within `IfTree` class. The `IfTree` structure is used and manipulated by all of the the interface management classes. The `IfTree` class is a container of interface state information organized in a hierarchy:

`IfTree` contains:

- `IfTreeInterface` physical interface representation, contains:
  - `IfTreeVif` virtual (logical) interface representation, contains:
    - `IfTreeAddr4` Interface IPv4 address and related attributes.
    - `IfTreeAddr6` Interface IPv6 address and related attributes.

Each item in the IfTree hierarchy is derived from `IfTreeItem`. `IfTreeItem` is a base class to track the state of a configurable item. Items may be in one of four states: `CREATED`, `DELETED`, `CHANGED`, `NO_CHANGE`. For example, if an item is added to the tree it will be in the `CREATED` state. The IfTreeItem::finalize_state() method places the item in the `NO_CHANGE` state and items marked as `DELETED` are actually removed at this time.

The state labeling associated with `IfTreeItem` adds a small degree of complexity to the `IfTree` classes. However, it allows for one entity to manipulate an interface configuration tree and pass it to another entity which can immediately determine the changes from the state labels.

The interface management functionality of the FEA is represented by three interacting classes: `IfConfig`, `InterfaceManager`, `InterfaceTransactionManager`. The interaction of these classes is managed by the `XrlInterfaceManager`, which takes external XRL requests and maps them onto the appropriate operations. The interactions between these classes and related classes are shown in Figure 3. The `XrlInterfaceManager` is sufficiently aware of the semantics of the operations to pass back human parseable error messages when operations fail.

The `IfConfig` class is an interface configurator, and contains plug-ins for each supported forwarding plane architecture to access, set, or monitor the interface-related information. The functionality of the `IfConfig` is conceptually simple: it can push-down an `IfTree` to the forwarding plane or pull-up the live configuration state from the forwarding plane as an `IfTree`.

The `InterfaceManager` class contains the `IfTree` representing the live configuration, and a reference to the `IfConfig` that should be used to perform the configuration. The `InterfaceTransactionManager` class holds and dispatches transactions. Each operation within a transaction operates on an item within a `IfTree` structure. Each transaction operates on a copy of the live `IfTree` and when the commit is made, this structure is pushed down into the `IfConfig`.

The process of configuration is asynchronous, and two phase. Errors can occur whilst a transaction is being committed and operating on an `Iftree` (*e.g.,* because of a bad operation within a transaction), and errors can occur when the configuration is pushed down to the forwarding plane (*e.g.,* the configuration has an inconsistent number of interfaces). Errors in the first phase are reported by the `InterfaceTransactionManager`. Errors in the second phase are reported by the `IfConfig` through a helper class derived from `IfConfigErrorReporter`

The interface management role of the FEA is expected to report configuration changes to other XORP processes. Hence, the `IfConfig` class uses the `XrlIfConfigUpdateReporter` class to report configuration changes.

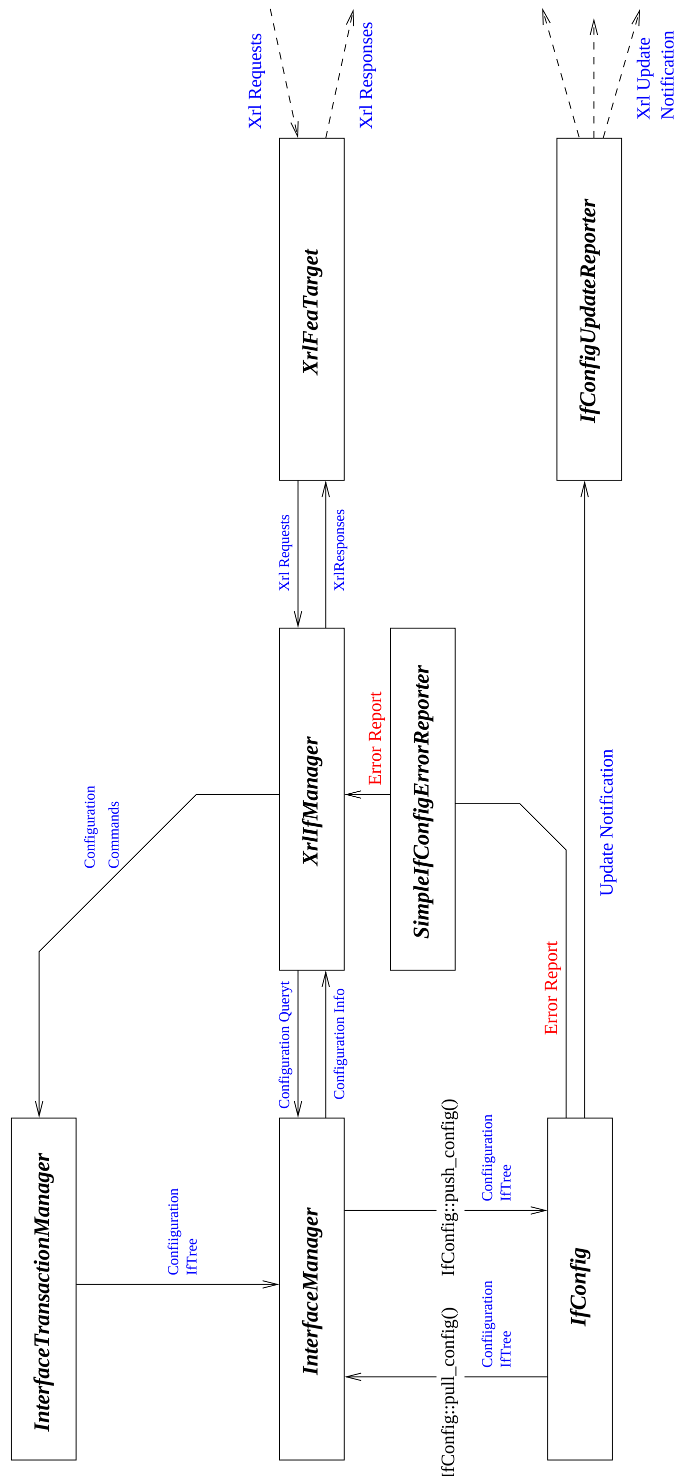

Figure 3: FEA Interface Management classes and their interactions

2.3 Forwarding Table Management

The Forwarding Table Management role propagates routes into the forwarding plane. The Forwarding Table Management role does not shadow the forwarding information outside of the forwarding plane itself; rather, it relies on the RIB to do this. As a result, it is considerably simpler than the Interface Management role.

The classes interacting to provide the Forward Table Management role are: the `XrlFtiTransactionManager` class, a class that adapts requests and responses from the subset of `XrlFeaTarget` methods that represent the forwarding table management externally; the `FtiTransactionManager` that builds and executes transactions to configure the forwarding table; and class `Fti` that understands how to program the forwarding plane.

The `Fti` class provides the interface for accessing the forwarding plane. It includes methods for adding and removing routes, as well as resolving routes in the forwarding table. Modifications to the `Fti` state are only permitted during a configuration interval. The configuration interval is started and stopped using `Fti::start_configuration` and `Fti:end_configuration`. The particular access to the forwarding plane is performed by plug-ins that are specific to that plane. For example, to read the forwarding table currently there are plug-ins that utilize the sysctl(3) mechanism (*e.g.,* in case of FreeBSD) or the netlink mechanism (*e.g.,* in case of Linux). There are plug-ins to read, set or monitor the forwarding table information at the granularity of one entry, or the whole table.

The `FtiTransactionManager` presents a transactional interface for configuring the `Fti` instance. Command classes exist for each possible modifier operation on the `Fti` instance. The `Fti` methods `start_configuration` and `end_configuration` are called at the start and end of the transaction.

Note that the XRL interface for adding/deleting routes is `redist_transaction{4,6}` which is a generic XRL interface used by the RIB to redistribute routes to interested parties.

The Forwarding Table Management also provides interface for processes to register interest in receiving updates whenever the Forwarding Information Base changes. The FEA is observing all FIB changes within the underlying system (including those triggered by the FIB manipulation by the FEA itself). Those changes are propagated to all instances of the `FibClient` class (implemented within the `XrlFtiTransactionManager` class).

2.4 Raw Packet I/O

The Raw Packet I/O role of the FEA provides a means for XORP processes to send and receive raw packets on particular interfaces. This is an essential function since in a XORP router the forwarding plane may reside on a different machine to the routing processes, it may be distributed across several machines, or may have custom network interfaces that require special programming.

The FEA supports both IPv4 and IPv6 raw packets. The text below describes the IPv4 implementation and the IPv4-specific classes; the IPv6 implementation is similar except the class names contain `6` instead of `4`.

The raw packet interface is managed by the `XrlRawSocket4Manager` class. This manages a single instance of a `FilterRawSocket4`[1]. The `FilterRawSocket4` encapsulates a raw socket and allows raw IPv4 packets to be written and filters attached to parse raw packets as they are received. The `XrlRawSocket4Manager` allows an arbitrary number of filters to be associated with the active raw socket. The filters are each notified when a raw packet is received on the raw socket. The XrlRawSocket4Manager allows other XORP processes to receive packets via XRL on the basis on filter conditions. Currently (March 2006), the only implemented filter is the `XrlVifInputFilter` which allows processes

[1]The current implementation only works on single machine XORP forwarding planes

to receive raw packets on the basis of the receiving VIF. In principle, filters could be written to match on any field within a packet and perform an action.

2.5 TCP/UDP Socket I/O

Similar to the Raw Packet I/O (see Section 2.4), the FEA provides a means for XORP processes to perform TCP or UDP socket operations and to send and receive TCP/UDP packets. This is an essential function since in a XORP router the forwarding plane may reside on a different machine to the routing processes, it may be distributed across several machines, or may have custom network interfaces that require special programming.

The TCP/UDP socket interface is managed by the `XrlSocketServer` class. This manages TCP and UDP IPv4 and IPv6 sockets[2]. The `XrlSocketServer` performs the particular TCP/UDP socket operations on the underlying system (opening and closing a socket, bind, send and receive, etc), and provides the XRL front-end interface. Note that for simplicity some of the socket XRL interface combines several system socket operations in one atomic FEA operation. For example, the `socket4/0.0/tcp_open_bind` XRL interface creates a TCP socket that binds it to a specific local address.

3 Status

There are two versions of the FEA: `fea` and `fea_dummy`. The `fea` is a version of the FEA that contains plug-ins to access the forwarding plane by using the following mechanisms:

- *getifaddrs(3)*, *sysctl(3)*, *ioctl(3)*, *Linux netlink(7) sockets*, *Linux /proc* and *Windows IP Helper API* to obtain interface-specific information.
- *ioctl(3)*, *Linux netlink(7) sockets* and *Windows IP Helper API* to set interface-specific information.
- *BSD routing sockets* and *Linux netlink(7) sockets* for observing changes in the interface-specific information.
- *BSD routing sockets*, *Linux netlink(7) sockets* and *Windows IP Helper API* to lookup a single forwarding entry in the forwarding plane.
- *sysctl(3)*, *Linux netlink(7) sockets* and *Windows IP Helper API* to obtain the whole forwarding table from the forwarding plane.
- *BSD routing sockets*, *Linux netlink(7) sockets* and *Windows IP Helper API* to set a single forwarding entry or the whole table in the forwarding plane.
- *BSD routing sockets* and *Linux netlink(7) sockets* to observe changes in the forwarding table.

In other words, currently (March 2006) the `fea` supports FreeBSD, NetBSD, OpenBSD, MacOS X, Linux, and Windows Server 2003 (see file `$XORP/BUILD_NOTES` about the specific OS versions the FEA has been tested on). In addition, there is also support for the Click forwarding plane (both kernel-space and user-space). The Click support utilizes the above plug-in based architecture by providing the appropriate support to add or delete routes for example to kernel or user-level Click.

[2]The current implementation only works on single machine XORP forwarding planes

The `fea_dummy` is a substitute FEA and may be used for development testing purposes. The `fea_dummy` represents an idealized form of FEA, other FEA's may differ in their responses due to architectural differences. Therefore processes that interact with the FEA should regard `fea_dummy` interactions as indicative, but not definitive.

The FEA's are still a work in progress and no doubt have some bugs. Any contributions or bug fixes are welcome.

References

[1] XORP Multicast Forwarding Engine Abstraction. XORP technical document. http://www.xorp.org/.