XORP Forwarding Engine Abstraction

Version 0.2

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

March 10, 2003

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 three distinct roles: *interface management*, *forwarding table management*, and *interface specific packet I/O*. Each of these is briefly described here together with a description of the normal interaction with other processes. The presentation of the design and implementation follows in the next section and is followed by a status summary in the concluding section.

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 makes 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.

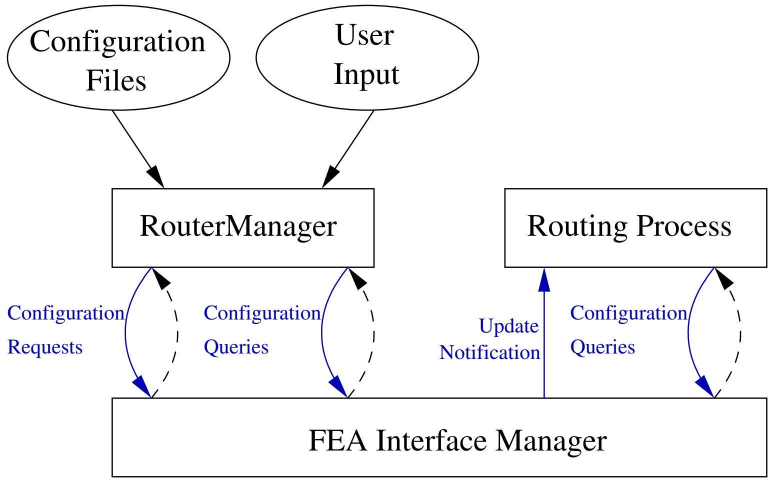

Figure 1: FEA Interface Management interaction with other XORP processes.

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.

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.

1.3 Interface Specific 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 and UDP[1] packets on specific interfaces. The transmission of packets through the FEA is straightforward, the routing process simply hands the FEA a packet and indicates which interface it should be sent on. The reception of packets is handled through a register-notify interface where the routing process registers which types of packets on which interfaces it is interested.

2 Design and Implementation

2.1 Overview

The FEA fulfills three discrete roles: Interface Management, Forwarding Table Management, and Interface Specific Packet 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 Interface Specific Packet I/O has 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 interprocess communication and each role of the FEA is represented by a distinct XRL interface. The Interface Management and Interface Specific Packet 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	fea_fti.xif	
Interface Specific Packet I/O	fea_rawpkt.xif	fea_rawpkt_client.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}`.

[1] At the time of writing UDP support is not present.

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.

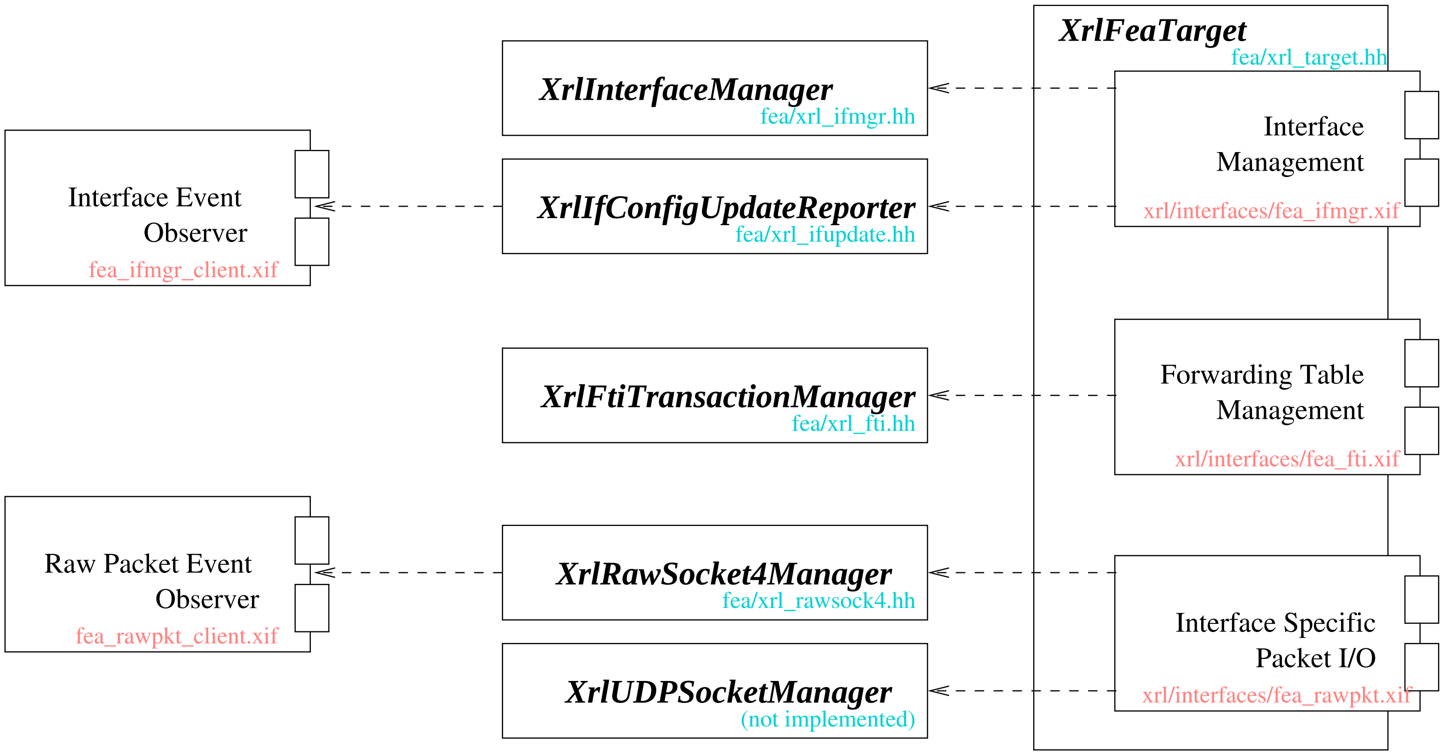

Figure 2: XRL Interfaces in relation to FEA classes.

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 a derived version is required for each supported forwarding plane architecture. 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 `IfConfigErrorReporterBase`.

The interface management role of the FEA is expected to report configuration changes to other XORP processes. The `IfConfig` classes are therefore expected to use the `XrlIfConfigUpdateReporter` class to report configuration changes.

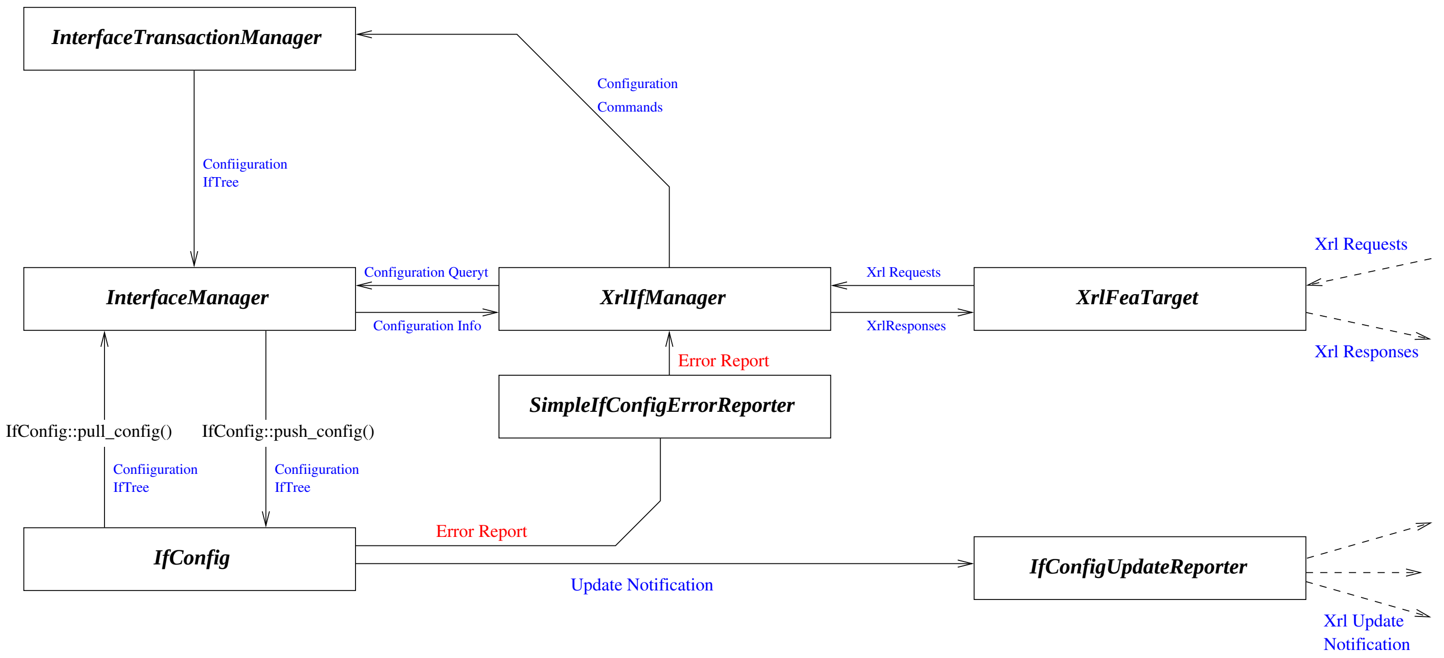

Figure 3: FEA Interface Management classes and their interactions

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, 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 a class derived from the `Fti` that understands how to program the forwarding plane.

The particular `Fti` classes implement the interface described by the abstract `Fti` class. These include 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`. Child classes override these methods to perform any additional operations they need to conduct in order to change the forwarding table state.

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.

2.4 Interface Specific Packet I/O

The Interface Specific Packet I/O role of the FEA provides a means for XORP processes to send and receive 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. At the time of writing, only the sending and receiving of raw IPv4 packets is implemented. Support for UDP and IPv6 will follow in future and should follow a similar design pattern to the raw IPv4 packet handling.

The raw packet interface is managed by the `XrlRawSocket4Manager` class. This manages a single instance of a `FilterRawSocket4`[2]. 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

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

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. The only implemented filter at the time of writing is the `XrlVifInputFilter` which allows processes 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.

3 Status

At the time of writing two versions of the FEA are supported: `fea_rtsock` and `fea_dummy`. The `fea_rtsock` is a version of the FEA that relies on the host operating system supporting BSD style routing sockets. This has been developed on FreeBSD 4.x, but should port with minor modifications to other BSD's with relatively minor changes. 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. FEA's for Click and Linux netlink need to be written, and FEA's for any other architecture would be welcomed. There is a now defunct Click FEA in the `$XORP/fea` directory that should be possible to resurrect.