Managing Complexity of Designing Routing Protocols Using a Middleware Approach

Cosmina Ivan[1], Vasile Dadarlat[2], and Kalman Pusztai[3]

[1] Department of Computer Science & Information Systems
University of Limerick, Ireland
cosmina.ivan@ul.ie
[2] Department of Electronics & Computer Engineering
University of Limerick, Ireland
vasile.dadarlat@ul.ie
[3]Department of Computer Science
Technical University of Cluj, Romania
kalman.pusztai@cs.utcluj.ro

Abstract. Designing and architecting new routing protocols is an expensive task, because they are complex systems managing distributed network state, in order to create and maintain the routing databases. Existing routing protocol implementations are compact, bundling together a database, an optimal path calculation algorithm and a network state distribution mechanism. The aim of this paper is to present a middleware-based approach for designing and managing routing protocols based on the idea of decomposing routing protocols into fundamental building blocks and identifying the role of each component, and also to propose a framework for composing dynamically new routing protocols making use of a distributed object platform.

1 A New Approach for Designing Protocols

A routing protocol allows the communication nodes of different networks to exchange information in order to update and maintain routing tables. Well known examples of routing protocols are: RIP, IGRP, EIGRP, and OSPF. The routers are capable to support multiple independent routing protocols and maintain forwarding tables for several routed protocols [3].

Traditionally, the term middleware was applied to software that facilitates remote database access and systems transactions, some common middleware categories may include: TPs, DCE or RPC systems, various ORBs. In newer accepts, the role of middleware would be to manage the complexity and heterogeneity of distributed infrastructures, providing simpler programming environments for distributed-application developers and new communication services. Well-known platforms

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 737-741, 2002.

offering middleware solutions for programming various distributed objects systems are OMG's CORBA and Microsoft's DCOM [9], [10].

As a general observation, the existing routing protocols are developed and implemented in a compact, monolithic manner, and they are highly integrated into each vendor's equipment, or even specific to each vendor. This approach obstructs the dynamic introduction of new routing protocols and services into the Internet. Making use of distributed objects technologies, it is possible to design a *binding model*, used by the programmable objects to construct the traditional services, as well as a design solution for new routing services. [2], [6].

On making use of OO-distributed technologies one must design the routing protocol implementations based on separating the routing database, the routing information announcement and the optimal path calculation. This technique would allow protocol developers to create various routing architectures in a modular fashion and to be able to introduce new routing services in a dynamic way, as a basis for the programmability of routing protocols [2], [10].

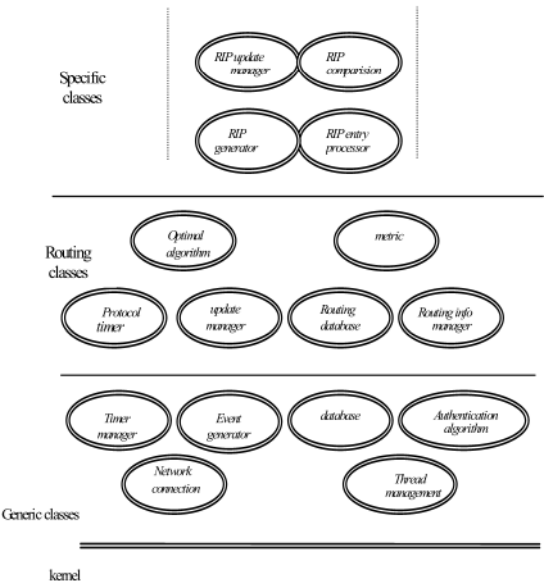

Fig. 1. RIP class hierarchy

2 Design and Implementation Considerations

We propose a binding model for characterizing each routing protocol used for decomposing the protocol into fundamental blocks and a hierarchy of classes, which reflects the general structure of any routing protocol implementations. The most important components of *the binding model* are:

- *Access interfaces*, acting above the operating system's kernel
- *RI manager* managing the routing information in the network
- *Event generator*, controls internally the transmission of routing information for the database updating, or the calculus of the optimal paths
- *Database object*, used for maintaining the distributed routing state of nodes, and *update manager object*, which updates the database when new routing information is received
- *Optimal path algorithms*, are components which operate on the contents of the database to calculate forwarding tables

By introducing standard interfaces between these routing components, code reuse could be enabled assuring the independency of the specific routing protocols and being fully programmable. By allowing components to create bindings at run-time, we enable the dynamic introduction of new routing services into networks.

To realize the binding model was designed a routing SDK named RP-ware, consisting of a hierarchy of base class's implemented using Java and making use of CORBA services. The prototype implements a set of three groups of classes: generic classes which offers basic functionality, a middleware layer containing routing component classes and the lower layer which contains the protocol specific classes. (Figure 1).

A short presentation of the functionality for some of these classes is made below a *database class* supports generic methods for creating new databases. The *network connection class* provides the mechanisms used at the interface with the system level, for transmitting routing messages to the network. A range of communication mechanisms will be supported, including TCP and UDP socket management or remote method invocations (RMI)[8]. Because many routing protocols are multithreaded, a *thread management class* will wrap operating system specific methods. Routing component classes are defined or derived to allow the construction of different routing architectures. A *routing database* extends the generic *database* class, encapsulating all the route entries available at a network node and will provide information about the network nodes and their associated paths. Protocol specific classes are used to describe the objects that participate in a specific routing protocol implementation.

The components of a programmable RIP implementation are shown in Figure 2. The component that directly communicates with the network is the *network connection* object (based on UDP sockets, or remote method invocations). In the case of sockets, a *RIP packet processor* object is involved, reading routing protocol specific headers from the received packets. Once a packet has been examined and verified, a route entry is extracted and sent to the *RIP entry processor* object. The *RIP update manager* object is then invoked, which exchanges information with the *metric* object, the *RIP route comparison* object, and the *routing database*. Following this, the *RIP route comparison* object inputs the two entries and produces the comparison result based on a specified set of metrics. If the received entry is better than the existing one, the *routing database* object is used to replace the existing entry. For this purpose the *RIP event generator* initiates a RIP triggered update, issued by the *RIP update manager*. Updated entries are finally transmitted to the neighboring routers via

the *routing info manager* and *network connection* objects. Regular RIP updates are periodically initiated, also using a *routing info manager* object.

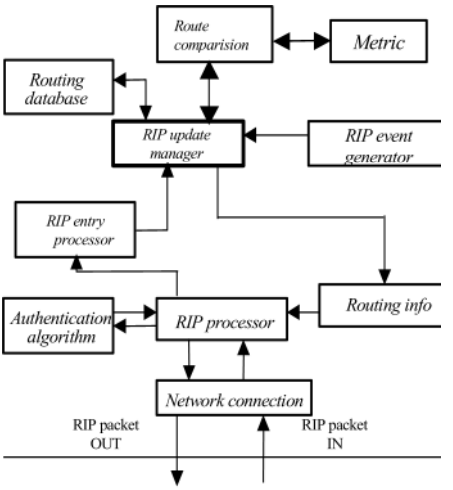

Fig. 2. The programmable RIP

The interfaces were written using Interface Definition Language (IDL)[8] and compiled using Orbacus 4.1.1 over NT platform. Since the functions of a router are over a wide range, we have provided interfaces for the following modules: the retrieval of information about host and interfaces, and IP route additions and deletions which result in kernel routing table updates. We also intend to extend our project to offer support for other routing protocols covering BGP, OSPF. An extract from the idle definition file for the RIP update manager:

```
struct rip_interface
{
/* RIP is enabled on this interface. */
  int enable_network;
  int enable_interface;
  /* RIP is running on this interface. */
  int running;
  /* RIP version control. */
  int ri_send;
  int ri_receive;
};
interface rip_interface
{
void rip_interface_clean ();
void rip_interface_reset ();
}
```

3 Conclusions

In our work, we have identified similarities between a numbers of routing protocols and designed a routing development kit and middleware for programming routing protocols. We have argued for the construction of routing protocols by extending and combining a set of components, hierarchically organized. This leads to an open programmable design that can greatly facilitate the design and deployment of new routing architectures. We have proposed a set of components and have outlined their interaction in order to construct a routing protocol. By being able to dynamically program routing protocols we can evaluate and change their structure in a dynamic manner, as is needed.

References

1. Lazar, A.A., : Programming Telecommunication Networks, *IEEE Network,* vol.11, no.5, September/October 1997.
2. Yemini, Y., and Da Silva, S, "Towards Programmable Networks", *IFIP/IEEE International Workshop on Distributed Systems: Operations and Managemen*t, L'Aquila, Italy, October,(1996)
3. Malkin, M. ,:IP version 2- IETF Network Working Group RFC 1723, (1994.)
4. Merwe J. et al., "The Tempest: A Practical Framework for Network Programmability", *IEEE Network Magazine, 12(3)*, (1998)
5. Merit GateD Consortium, http://www.gated.net
6. Moy, J.,:OSPF version 2- IETF Network Working Group RFC 2328, April 1998
7. Staccatos, V., Kounavis, E., Campbell, A.,: Sphere- A binding model and middleware for routing protocols, *OPENARCH' 01*, Alaska, April 27-28, 2001.
8. www.omg.org
9. Vinoski, S. "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol. 14, February 1997.