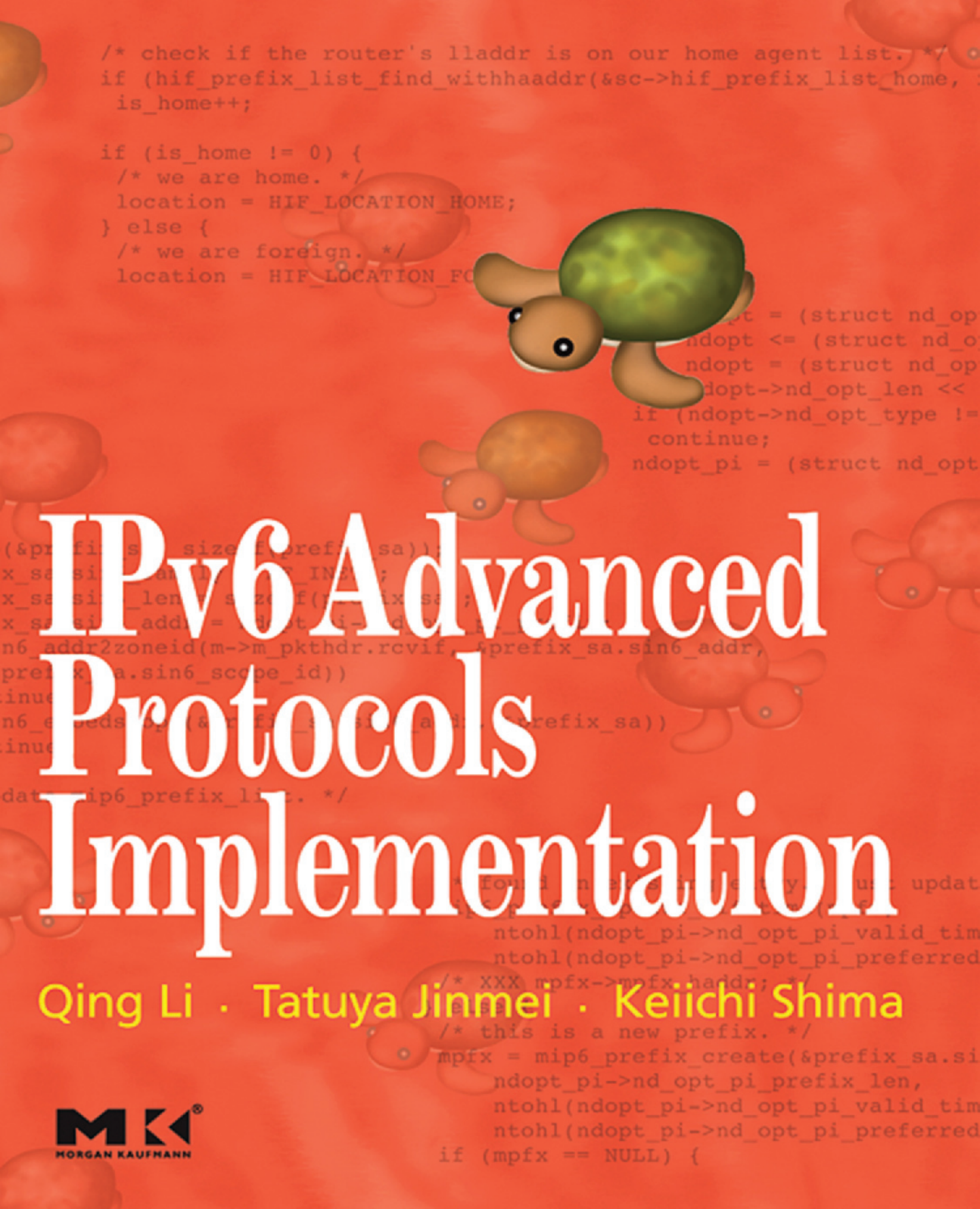

/* check if the router's lladdr is on our home agent list. */
if (hif_prefix_list_find_withhaaddr(&sc->hif_prefix_list_home,
is_home++;
if (is_home != 0) {
/* we are home. */
location = HIF_LOCATION_HOME;
} else {
/* we are foreign. */
location = HIF_LOCATION_FO
IPv6 Advanced Protocols Implementation
Qing Li · Tatuya Jinmei · Keiichi Shima
MORGAN KAUFMANN

IPv6 Advanced Protocols Implementation

The Morgan Kaufmann Series in Networking

Series Editor: David Clark, M.I.T.

IPv6 Advanced Protocols Implementation
Qing Li, Tatuya Jinmei, and Keiichi Shima

Computer Networks: A Systems Approach, 4e
Larry L. Peterson and Bruce S. Davie

Network Routing: Algorithms, Protocols, and Architectures
Deepankar Medhi and Karthikeyan Ramaswami

Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice
John Evans and Clarence Filsfils

Traffic Engineering and QoS Optimization of Integrated Voice and Data Networks
Gerald R. Ash

IPv6 Core Protocols Implementation
Qing Li, Tatuya Jinmei, and Keiichi Shima

Smart Phone and Next-Generation Mobile Computing
Pei Zheng and Lionel Ni

GMPLS: Architecture and Applications
Adrian Farrel and Igor Bryskin

Network Security: A Practical Approach
Jan L. Harrington

Content Networking: Architecture, Protocols, and Practice
Markus Hofmann and Leland R. Beaumont

Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices
George Varghese

Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS
Jean Philippe Vasseur, Mario Pickavet, and Piet Demeester

Routing, Flow, and Capacity Design in Communication and Computer Networks
Michal Pióro and Deepankar Medhi

Wireless Sensor Networks: An Information Processing Approach
Feng Zhao and Leonidas Guibas

Communication Networking: An Analytical Approach
Anurag Kumar, D. Manjunath, and Joy Kuri

The Internet and Its Protocols: A Comparative Approach
Adrian Farrel

Modern Cable Television Technology: Video, Voice, and Data Communications, 2e
Walter Ciciora, James Farmer, David Large, and Michael Adams

Bluetooth Application Programming with the Java APIs
C. Bala Kumar, Paul J. Kline, and Timothy J. Thompson

Policy-Based Network Management: Solutions for the Next Generation
John Strassner

Network Architecture, Analysis, and Design, 2e
James D. McCabe

MPLS Network Management: MIBs, Tools, and Techniques
Thomas D. Nadeau

Developing IP-Based Services: Solutions for Service Providers and Vendors
Monique Morrow and Kateel Vijayananda

Telecommunications Law in the Internet Age
Sharon K. Black

Optical Networks: A Practical Perspective, 2e
Rajiv Ramaswami and Kumar N. Sivarajan

Internet QoS: Architectures and Mechanisms
Zheng Wang

TCP/IP Sockets in Java: Practical Guide for Programmers
Michael J. Donahoo and Kenneth L. Calvert

TCP/IP Sockets in C: Practical Guide for Programmers
Kenneth L. Calvert and Michael J. Donahoo

Multicast Communication: Protocols, Programming, and Applications
Ralph Wittmann and Martina Zitterbart

MPLS: Technology and Applications
Bruce Davie and Yakov Rekhter

High-Performance Communication Networks, 2e
Jean Walrand and Pravin Varaiya

Internetworking Multimedia
Jon Crowcroft, Mark Handley, and Ian Wakeman

Understanding Networked Applications: A First Course
David G. Messerschmitt

Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application
Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair

Virtual Private Networks: Making the Right Connection
Dennis Fowler

Networked Applications: A Guide to the New Computing Infrastructure
David G. Messerschmitt

Wide Area Network Design: Concepts and Tools for Optimization
Robert S. Cahn

For further information on these books and for a list of forthcoming titles, please visit our Web site at http://www.mkp.com.

IPv6 Advanced Protocols Implementation

Qing Li
Blue Coat Systems, Inc.

Tatuya Jinmei
Toshiba Corporation

Keiichi Shima
Internet Initiative Japan, Inc.

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Morgan Kaufmann Publishers is an imprint of Elsevier

MORGAN KAUFMANN PUBLISHERS

Senior Acquisitions Editor Rick Adams
Publishing Services Manager George Morrison
Senior Production Editor Dawnmarie Simpson
Acquisitions Editor Rachel Roumeliotis
Production Assistant Lianne Hong
Cover Design Eric DeCicco
Cover Image Side-by-Side Design
Cover Illustration Side-by-Side Design
Composition diacriTech
Technical Illustration diacriTech
Copyeditor JC Publishing
Proofreader Janet Cocker
Indexer Joan Green
Interior printer The Maple-Vail Book Manufacturing Group
Cover printer Phoenix Color Corporation

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data
Li, Qing, 1971-
IPv6 advanced protocols implementation/Qing Li, Tatuya Jinmei, Keiichi Shima.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-12-370479-5 (hardcover: alk. paper)
ISBN-10: 0-12-370479-0 (hardcover: alk. paper) 1. TCP/IP (Computer network protocol)
I. Jinmei, Tatuya, 1971- II. Shima, Keiichi, 1970- III. Title.
TK5105.585.L536 2007
004.6′2–dc22

2006038489

ISBN: 978-0-12-370479-5

For information on all Morgan Kaufmann publications,
visit our Web site at *www.mkp.com* or *www.books.elsevier.com*

Printed in the United States of America
07 08 09 10 5 4 3 2 1

To Huaying, Jane and Adalia
in Him
—Qing Li

To my colleagues at KAME: working with you talented geeks was an exciting experience and has even made this derivative project possible.
—Tatuya Jinmei

To all KAME developers, all people who developed the Internet, and all people who will develop the future Internet.
—Keiichi Shima

This page intentionally left blank

Contents

This page intentionally left blank

Preface

This book is the second installment of our series detailing IPv6 and related protocols through the KAME implementation. KAME is a widely deployed de facto reference implementation for IPv6 and IP security protocols developed on multiple variants of the BSD operating systems.

The first installment of this series is titled *IPv6 Core Protocols Implementation*, which is referred to as the *Core Protocols* book below, and it focuses on the fundamentals of IPv6 and the essential protocols that are supported by most implementations. These essential protocols operate in IPv6-capable devices, large or small. Our *Core Protocols* book also describes IPv6 implication on higher layer protocols, such as TCP and UDP, and covers IPv6 related application programming interfaces.

This second book discusses those protocols that are found in more capable IPv6 devices, are commonly deployed in more complex IPv6 network environments, or are not specific to IPv6 but are extended to support IPv6. Specifically, this book engages the readers in more advanced topics, such as routing, multicasting, DNS, mobility, and security.

The general structure and style of this book is the same as that of the *Core Protocols* book; each chapter begins with a summary of the relevant specifications followed by line-by-line code description and analysis of the actual implementation.

We hope to help the readers establish a solid and empirical understanding of IPv6 with our book series. Our two books together cover a wide spectrum of the IPv6 technology and are paralleled by none.

This book consists of the following chapters:

- Chapter 1 ("IPv6 Unicast Routing Protocols") discusses general routing concepts and the fundamentals of various types of unicast routing protocols. This chapter details RIPng, a simple routing protocol for IPv6, and summarizes IPv6-specific extensions defined for the BGP4+ and OSPFv3 routing protocols. Comparisons are made among these protocols in regards to protocol complexity, stability, and the operational issues and solutions

offered by each. This chapter also provides the necessary background to implement IPv6 routing protocols on BSD variants through descriptions of the routing API for IPv6 and code narrations of KAME's RIPng implementation, the **route6d** daemon. This chapter concludes with configuration examples of **route6d** for some typical scenarios.

- Chapter 2 ("IPv6 Multicasting") discusses details about IPv6 multicasting, especially on multicast routing mechanisms. It first provides the basics of a host-to-router protocol and multicast routing protocols, specifically the Multicast Listener Discovery protocol version 1 (MLDv1) and Protocol Independent Multicast (PIM), focusing on IPv6 specific issues. The latter part of this chapter describes the KAME kernel implementation of MLDv1 and IPv6 multicast forwarding.
- Chapter 3 ("DNS for IPv6") describes IPv6 extensions to the DNS (Domain Name System) protocol specification and implementation. It begins with a general description of the DNS protocol and its extensions that support IPv6. It then describes KAME's DNS client (called a *resolver*) implementation, and highlights the support for IPv6. This section also gives a complete view of the `getaddrinfo()` library function, which was partially described in the *Core Protocols* book. The latter half of this chapter shows how to operate the BIND9 DNS server to support IPv6 with notes about common pitfalls and issues specific to IPv6-related operations.
- Chapter 4 ("DHCPv6") details DHCPv6 (Dynamic Host Configuration Protocol for IPv6) both on the protocol specification and on KAME's implementation. Although the basic concept of the protocol is largely derived from DHCP for IPv4 (DHCPv4), DHCPv6 has introduced various improvements in its design and the expected usage model differs from that of DHCPv4; this chapter clarifies such major differences. The implementation descriptions cover all protocol functionalities, that is, clients, servers, and relay agents, and will provide an in-depth understanding of how the protocol works. This chapter also provides how to operate DHCPv6 with the KAME implementation for some common usage scenarios.
- Chapter 5 ("Mobile IPv6") covers the IPv6 host mobility protocol known as Mobile IPv6. The chapter begins with a basic description of Mobile IPv6, and then details protocol specifications and data structures. The actual implementation is discussed in the middle of the chapter. The KAME Mobile IPv6 implementation supports both home agent and mobile node functions. The code description section will discuss all data structures and functions in detail. This chapter also provides a brief instruction of Mobile IPv6 operation with sample configuration files using the KAME Mobile IPv6 implementation at the end of the chapter.
- Chapter 6 ("IPv6 and IP Security") begins with an introduction of the IPsec protocols and the concept of keying in the context of the Internet Key Exchange (IKE) protocol. The remainder of this chapter then focuses on describing the popular **racoon** IKE daemon. Its configuration and operation are thoroughly explained. This chapter concludes with some practical examples of using **racoon**. Unlike other chapters, this chapter does not provide any code description because the basic mechanism of IP Security and most of its implementation are not specific to IPv6; including non-IPv6 specific code description would change the main objective of this book.

Intended Audience

In general, this book is intended for the same class of readers as was the *Core Protocols* book: developers implementing IPv6 and related protocols, and students who are going to start a project on these protocols, especially on top of or using the KAME/BSD implementation. Unlike the *Core Protocols* book, however, this book discusses more advanced topics, such as protocols that have been standardized relatively recently, so it can also be used as a reference to these protocols per se; DHCPv6 and Mobile IPv6 are two specific examples of this.

As in the *Core Protocols* book, it is assumed that readers are fluent in the C programming language. In addition, this book assumes knowledge of the basic notions of IPv6 and related protocols described in the *Core Protocols* book, though other references within this book will help those who cannot refer to the *Core Protocols* book to understand the contents. Chapters 2 and 5 also require general understanding of the BSD kernel implementation.

Unlike the *Core Protocols* book, each chapter of this book is quite independent; although there are several cross references among the chapters, readers can generally start from any chapter based on their interest.

Typographical Conventions

This book adopts the same typographical conventions as those for the *Core Protocols* book, which is summarized as follows:

Variable, function, or structure names, structure members, and programming language keywords are represented in a `constant-width` font when referred to in the code descriptions. Function names are in a `constant-width` font followed by parentheses, as in `ip6_mforward()`, and structure names are in a `constant-width` font followed by braces, as in `ip6_mh{}`.

Program names are displayed in bold fonts, as in **route6d**. The command line input and the output of a program are displayed in a `constant-width` font.

Accompanying CD-ROM

This book comes with two CD-ROMs. The first CD-ROM is an ISO image of FreeBSD4.8-RELEASE, which is the base operating system covered in Chapters 1, 2, 3, and 6. It is a bootable CD-ROM and includes installation files. The installation procedure is started by turning on the computer with the CD-ROM loaded. The detailed installation procedure can be found in the `INSTALL.TXT` file located in the root directory of the CD-ROM.

Similarly, the second CD-ROM is a bootable ISO image of FreeBSD4.9-RELEASE, which is the base operating system covered in Chapter 5.

Note: FreeBSD 4.8 and 4.9 RELEASEs are known to have several security flaws and are no longer supported by the FreeBSD project. Therefore, these systems should only be used for reference on learning the KAME implementation as part of reading this book. It is not advisable to use these versions of FreeBSD in a production environment connected to the Internet.

The first CD-ROM also contains the KAME source code discussed in this book. It is accessed via the `appendix` directory located at the root directory, which has two subdirectories, `kame-snap` and `rtadd6`.

The `kame-snap` subdirectory contains the following archive files:

- `kame-20030421-freebsd48-snap.tgz`
  A KAME snapshot for FreeBSD 4.8 taken on April 21, 2003.
- `kame-20040712-freebsd49-snap.tgz`
  A KAME snapshot for FreeBSD 4.9 taken on July 12, 2004. This is referred to in Chapter 5, and should be used with the FreeBSD 4.9 system contained in the second CD-ROM.
- `kame-dhcp6-20050509.tgz`
  KAME's DHCPv6 implementation included in a KAME snapshot taken on May 9, 2005, which is referred to in Chapter 4.

To install the KAME snapshot, unpack the archive, go down to the top level directory named `kame` (which is also referred to as `${KAME}` throughout this book), and see the `INSTALL` file located in the directory. For those who have the *Core Protocols* book, its Chapter 1 provides a more detailed description of the usage. Chapter 4 of this book explains how to install the DHCPv6 implementation.

The other subdirectory, `rtadd6`, contains the source code of the `rtadd6` program referred to in Chapter 1, which was newly written for this book.

Source Code Copyright

This book presents many parts of the source code developed by the KAME project and external contributors. It also refers to system header files that are part of the FreeBSD distributions. All of the source code has copyright notices, which are available in the copy of the code contained in the CD-ROM discs.

Reporting Errors and Errata Page

The authors are happy to receive error reports on the content of this book, and plan to provide an error correction page on the Internet. It will be available at the following web page: `http://books.elsevier.com/companions/9780123704795`.

Acknowledgments

The authors, first and foremost, thank all KAME developers. As in our first book, this book is half-filled with the KAME source code, which means they are the shadow authors of this book.

We are also deeply indebted to technical reviewers who read selected chapters of this book and provided many valuable comments and suggestions, as well as error corrections: Mark Andrews, David Borman, Francis Dupont, Daniel Hartmeier, Jeffrey Hsu, Akira Kato, T. J. Kniveton, Ted Lemon, Tsuyoshi Momose, George Neville-Neil, Yasuhiro Ohara, Shawn Routhier, Shoichi Sakane, Shigeya Suzuki, Shinsuke Suzuki, Christian Vogt, and Carl Williams. As with our first book, reviewing this book required thorough knowledge of the related protocol specifications, as well as high level programming skills. We knew very few people have such talents, and we were very lucky to have the world's best reviewer team.

The book cover is based on the well-known KAME turtle image, which was designated as a project mascot, and was designed by Manabu Higashida and Chizunu Higashida.

Next, we would like to thank our editors Rick Adams, Rachel Roumeliotis, Dawnmarie Simpson, and the editorial staff at Morgan Kaufmann/Elsevier for their continuing patience and encouragement over the three and a half years of this project.

Finally, we are grateful to Gary R. Wright and W. Richard Stevens. Their work inspired us to start our own project and kept us confident about the value of this work.

LI, Qing—I would like to thank Rick Adams for his keen understanding of the importance of this book, as it fulfills a market void. His prompt acceptance of my book proposal has been an invaluable motivation. I want to thank my wife Huaying Cheng for her understanding and support of me during this book project. I would like to thank VMware Inc. for its donation of a single license for the VMware Workstation 4 software. I would also like to thank MKS Software for its donation of a single license for the MKS Toolkit for Enterprise Developers version 8.6 software.

JINMEI, Tatuya—I would like to thank my current and former managers at Toshiba for their approval and support of this work: Yukio Kamatani, Toshio Murai, Yasuhiro Katsube, and Atsushi Inoue. My thanks also go to my "supervisors" at the WIDE project, Jun Murai and Hiroshi Esaki.

SHIMA, Keiichi—I thank all of the people who worked hard to publish this book and those who supported this work, especially my manager Eiiti Wada at Internet Initiative Japan, Inc. Also my thanks go to all operators, engineers, and researchers of the Internet.

This page intentionally left blank

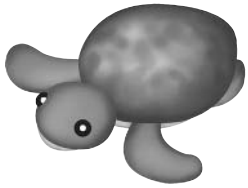
About the Authors

Li, Qing is a senior architect at Blue Coat Systems, Inc. leading the design and development efforts of the next-generation IPv6 enabled secure proxy appliances. Prior to joining Blue Coat Systems, Qing spent 8 years at Wind River Systems, Inc. as a senior architect in the Networks Business Unit, where he was the lead architect of Wind River's embedded IPv6 products since the IPv6 program inception at the beginning of 2000. Qing holds multiple U.S. patents. Qing is a contributing author of the book titled *Handbook of Networked and Embedded Control Systems* published in June of 2005 by Springer-Verlag. He is also the author of the embedded systems development book titled *Real-Time Concepts for Embedded Systems* published in April of 2003 by CMP Books. Qing participates in open source development projects and is an active FreeBSD src committer.

Jinmei, Tatuya, PhD, is a research scientist at Corporate Research & Development Center, Toshiba Corporation (Jinmei is his family name, which he prefers is presented first according to the Japanese convention). He had been a core developer of the KAME project since the launch of the project through its conclusion. In 2003, he received his PhD degree from Keio University, Japan, based on his work at KAME. He also coauthored three RFCs on IPv6 through his activity in KAME. His research interests spread over various fields of the Internet and IPv6, including routing, DNS, and multicasting.

Shima, Keiichi is a senior researcher at Internet Initiative Japan Inc. His research area is IPv6 and IPv6 mobility. He was a core developer of the KAME project from 2001 to the end of the project and developed Mobile IPv6/NEMO Basic Support protocol stack. He is now working on the new mobility stack (the SHISA stack) for BSD operating systems that is a completely restructured mobility stack.

This page intentionally left blank



1 IPv6 Unicast Routing Protocols

1.1 Introduction

Any time when communication takes place between any pair of nodes, especially when that communication involves nodes that reside on different network segments, a decision must be made about where each packet should go. This decision is often known as a packet *routing* decision, or a packet *forwarding* decision. The intermediate network devices, commonly known as *routers*, perform the routing functions that involve making the routing decision normally based on each packet's final destination.

The routing decision could be made based on manually configured routing information at each router, but such practice is obviously impractical for a complex network of middle to large scale. *Routing protocols* provide the necessary information that enable the routers to make correct routing decisions automatically. Since a packet's destination may be a unicast destination or a multicast destination (treating broadcast destination as a special case of multicast), routing protocols are designed for either *unicast routing* or *multicast routing*. We will focus on the routing protocols in this chapter.

In the IPv4 world, RIPv2 [RFC2453], the integrated IS-IS [RFC1195], and OSPFv2 [RFC2328] are commonly deployed unicast routing protocols in networks of small to middle scale such as enterprise environments, while BGP-4 [RFC4271] is the common routing protocol deployed among large organizations such as Internet Service Providers (ISPs). In general, since the routing concept is identical between IPv4 and IPv6, these routing protocols have been naturally extended to support IPv6. Even though the packet formats may have changed, the principles remain largely the same.

Yet there are IPv6 specific issues. In particular, most IPv6 routing protocols rely heavily on link-local addresses since communication using these addresses is stable in terms of routing, thanks to their limited scope. On the other hand, the ambiguity of link-local addresses discussed

in Chapter 2 of *IPv6 Core Protocols Implementation*, "IPv6 Addressing Architecture", requires special care in implementing these protocols. It is therefore important to understand the details of the protocols and how they should be implemented even for those who are familiar with IPv4 routing protocols.

In this chapter we provide all the essential information to understand and implement IPv6 unicast routing protocols. We first describe the basic routing concepts followed by an introduction to IPv6 unicast routing protocols. These unicast routing protocols include RIPng [RFC2080], OSPFv3 [RFC2740] and BGP4+ [RFC2545]. We provide full coverage on the RIPng protocol. In addition, we summarize the general protocol operations of OSPFv3 and BGP4+ without diving into the protocol specifics, other than the IPv6-related protocol packets. Readers who do not require such advanced topics can safely skip these sections (1.5 and 1.6) as they are not needed in any other part of the book.

Sections that follow the protocol background focus on implementation, which will provide all of the essential information to develop IPv6 routing programs on BSD systems, covering the kernel architecture to routing application code. We first explain how to deal with IPv6 routing information on BSD systems, from the kernel internal data structures to application interfaces (APIs). We also note major pitfalls in handling link-local addresses with these APIs. We then describe the implementation of the **route6d** program, KAME's RIPng routing daemon, focusing on its RIPng protocol processing. The provenance of RIP is the **routed** program, a popular implementation that is widely available on various platforms. Its popularity is due to the simplicity in both its implementation and its operation. The **route6d** daemon is the IPv6 counterpart of **routed**.

Finally, we conclude this chapter by showing how to operate **route6d** for some typical scenarios.

1.2 Overview of Routing Concepts

Routing information enables a node to determine whether a given destination is reachable and where to send the packet en route to the destination. Routing information can be either configured statically or obtained dynamically. Routers exchange routing information with one another through one or more dynamic routing protocols. Each router builds a local database, called the *Routing Information Base* (RIB) to store the exchanged routing information. A subset of this RIB is then selected to build a *Forwarding Information Base* (FIB) for the purpose of forwarding packets.

The routing concepts are identical between IPv4 and IPv6. That is, the goal of routing is to find a loop-free path for the destination address between any pair of end systems, and the best path is chosen according to some defined criteria at the time of route selection. Many of the existing dynamic routing protocols have been updated to support IPv6. Three well-known routing protocols—RIP, OSPF and BGP—have been extended to support IPv6, resulting in RIPng, OSPFv3 (OSPF for IPv6) and BGP4+, respectively. Another deployed routing protocol, IS-IS, was also extended to support both IPv4 and IPv6 (see the note on page 4).

The choice of the routing protocol depends on many factors, such as the diameter of the routing domain, the size and complexity of the networks within the routing domain, the level of tolerance to changing network topology by applications, and the complexity and the ease of deployment of the routing protocol.

In general, routing protocols are classified as either *interior routing protocols* or *exterior routing protocols*, based on where the protocol is deployed. Interior routing protocol is also known as *interior gateway protocol* (IGP) while exterior routing protocol is also known as *exterior gateway protocol* (EGP).

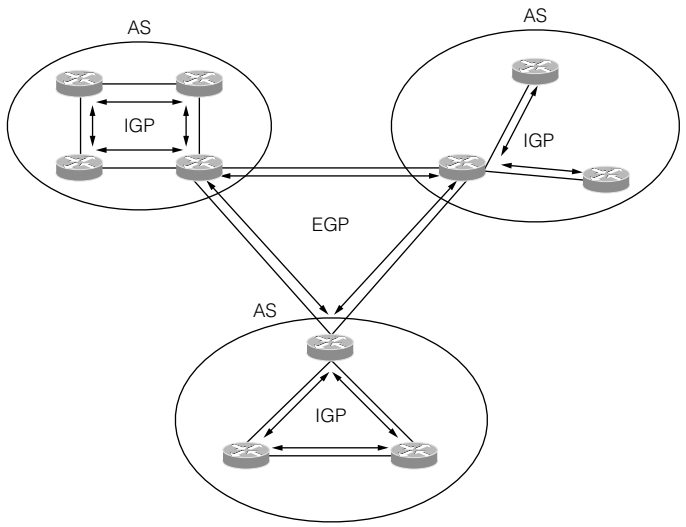
An interior routing protocol is deployed within a routing domain that is controlled by a single administrative entity. In this context, a routing domain is also known as an *autonomous system* (AS). Each autonomous system should have only one governing routing policy. For example, an interior routing protocol is deployed within the intranet of an organization, which may comprise multiple sub-networks. In other words, an interior routing protocol is deployed within a single routing domain to exchange routing information about these sub-networks among routers that belong to the same routing domain. Examples of interior routing protocols are RIPng and OSPFv3.

An exterior routing protocol is deployed among routing domains that are under the management of different administrative entities. For example, an exterior routing protocol is deployed between two different *Internet Service Providers* (ISPs). In other words, an exterior routing protocol is deployed to exchange routing information among routers that belong to different autonomous systems. BGP4+ is an example of an exterior routing protocol.

Within each AS, a small subset of the routers are situated at the boundary of the AS. These boundary routers, sometimes referred to as either *border gateways* or *edge routers*, exchange route information over EGP with other edge routers that belong to different ASs. An edge router also typically participates in IGP within its AS to advertise externally reachable networks, or it simply acts as the default router for the AS to reach the rest of the Internet. Figure 1-1 illustrates this relationship. In this example each AS has one edge router that participates in the EGP.

The purpose of running a dynamic routing protocol is to provide reachability information about networks and individual nodes to routers that participate in the routing domain. The

FIGURE 1-1



Relationship between IGP and EGP among different autonomous systems.

reachability information allows each router to compute the appropriate next hop or the paths to these networks and nodes using a specific routing algorithm. Whether the paths are loop-free depends on the routing protocol and the information distributed by the routing protocol. The way the routing algorithm works determines the type of information distributed in the routing protocol messages. Therefore routing protocols are also classified according to the routing algorithms by which the routing protocols are employed for route computation. The routing algorithms can be classified as *vector-based* algorithms or *link-state* algorithms. The vector-based algorithms can be further classified as either *distance vector* algorithms or *path-vector* algorithms. RIPng is a routing protocol representative of the distance vector algorithm; BGP4+ is a routing protocol representative of the path-vector algorithm; OSPFv3 is a routing protocol representative of the link-state algorithm.

Another link-state algorithm-based routing protocol is the *Intermediate System to Intermediate System* (IS-IS) routing protocol. IS-IS was originally designed for ISO's protocol stack known as the *Connectionless Network Protocol* (CLNP), which was meant to be the replacement of TCP/IP. The CLNP protocol stack was developed in anticipation of the greater adoption of the OSI's 7-layer communication model, but such migration has not taken place in reality. The IS-IS routing protocol is an IGP and is another link-state routing protocol. The *Integrated IS-IS* supports both CLNP and IP. In actual deployment, IS-IS is largely deployed for routing in the IP network. IS-IS is quite similar to OSPF. For this reason, we will focus on OSPF as the representative protocol for describing the link-state routing algorithm. IS-IS is defined by [ISO-10589]. The Integrated IS-IS is defined in [RFC1195]. The reader is encouraged to consult [ISIS-IPV6] for details on the IPv6 extension for IS-IS.

The routing protocols are designed to satisfy a different set of goals. A routing protocol, more precisely the algorithm used by the routing protocol, must be capable of selecting the optimal route according to predefined selection criteria. For example, a routing algorithm can select the best route according to the least number of hops traversed to reach the destination. A routing protocol must be robust to changing network topologies and network conditions. For example, the routing protocol must continue to function when an interface on the router fails, or when one or more routers fail. A routing protocol should have a good *convergence rate*. When network topologies or network conditions change, the routing protocol should have the ability to convey this information to all participating routers quickly to avoid routing problems. The convergence rate refers to the time taken for all routers in the domain to become aware of the changing condition. Routing protocols should be designed to have small operational overhead and should be relatively easy to deploy.

A predefined selection criteria determines what is considered the optimal route or the best route according to one or more metrics. The metrics can be either static or dynamic. Examples of static metrics are path length or monetary cost of using a particular path. Path length can be either simple hop counts, or the sum of the costs of all links in a given path. Typically a system administrator assigns the cost of each link. Examples of dynamic metrics are the measured network load, delay, available bandwidth, and reliability (such as error rate and drop rate).

1.3 Overview of Vector-based Algorithms and Link-State Algorithm

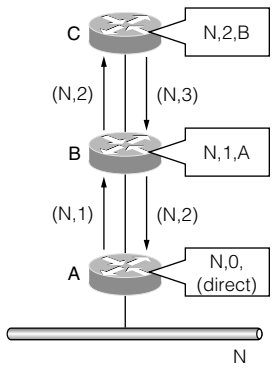
1.3.1 Distance-Vector Algorithm

A router running the distance-vector algorithm, as is the case with RIPng, initializes its local routing database with the addresses and costs of the directly attached networks and nodes. This information is exchanged with other directly connected routers through routing protocol messages. When a router receives routing messages from its neighboring routers, it adds the cost of the network on which the routing messages arrived to all of the destinations that are advertised in the routing messages. A destination can appear in multiple routing messages that were sent by different neighboring routers. The receiving router chooses the router that advertised the smallest metric to that destination as the preferred next hop. The smallest metric value is updated with the cost of the network. The receiving router then readvertises that destination with the updated metric.

Figure 1-2 illustrates how the distance-vector algorithm works for a very simple network topology (a more interesting example will be shown in Section 1.4). There are three routers (A, B and C) connected in series, and router A is attached to a leaf network N. For simplicity, let us just concentrate on the routing information about network N, and assume that the cost of any link is 1.

The arrows shown in Figure 1-2 are labeled with the routing information distributed among the routers, which highlights the destination information (N) and the total cost to reach the destination. The box drawn next to each router represents its routing table, whose entry is a combination of <destination, metric, next hop router>. For example, router B accepts the information advertised by router A (which by default has the smallest metric because that route is the only route about network N) and installs the route to its routing table. Router B then readvertises that route toward router C with the updated cost. Eventually all of the routers will converge to a stable state in which each router knows the path to leaf network N. Router C forwards any packet destined to network N toward router B, which then forwards the packet to router A. Router A will then deliver the packet to the final destination on N.

FIGURE 1-2



Routing with the distance-vector algorithm.

As seen in this example, the major advantage of the distance-vector algorithm is its simplicity. The algorithm is easy to understand and implement. In fact, KAME's RIPng implementation consists of only about 3500 lines of source code written in C, including all optional features.

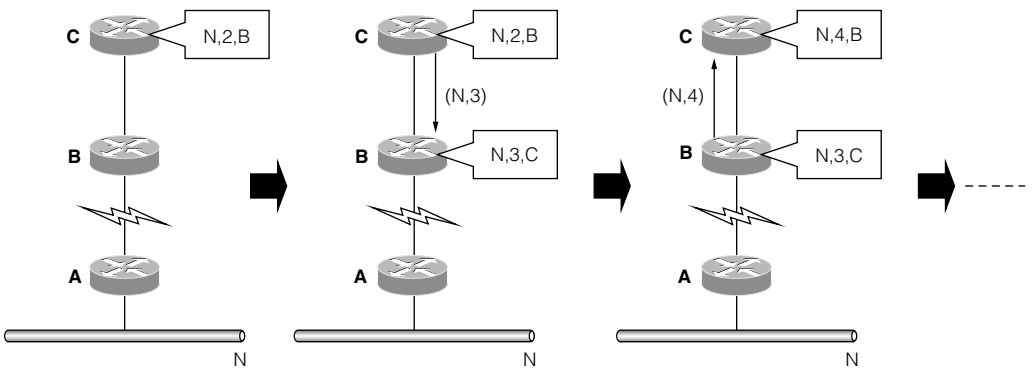
The simplicity comes with a different cost. One major disadvantage of the distance-vector algorithm is that it is vulnerable to changes in topology.

Consider the scenario shown in Figure 1-3, where the link between router A and router B is down. Router B detects the link failure, removes the route information about network N because B knows N is now unreachable. In the pure distance-vector algorithm, router B does nothing further. Since the information is still in router C, router C advertises that stale route back to B, which is then installed in router B with a higher cost. B accepts C's advertisement because B is aware that router A is no longer reachable due to the dead link, and B has not accepted any route about network N from C previously. At this point a routing loop between routers B and C is created. Router B will subsequently advertise that same route back to router C with a higher cost. This higher cost route will override C's entry because router C knows its route about N came from B's original advertisement. This iterative process continues until the advertised cost reaches some upper limit set by the protocol, allowing these routers to finally detect the failure. This symptom is called the *counting to infinity* problem.

Although several techniques are available to mitigate this problem (see Section 1.4.3), none of them can completely eliminate the algorithm flaw. Even when such remedies do solve the problem in some types of deployment, route convergence generally takes a longer time with the distance-vector algorithm than other algorithms on a topology change, especially when the network contains slower links.

In the distance-vector algorithm, the router stores and distributes only the current best route to any known destination. Therefore, the computation of a route at each router depends largely on the previous computation results made by other routers. Additionally, since only the distance to any destination is given, it is impossible for the distance vector algorithm to identify the origin of a route and to guarantee loop-free routes.

FIGURE 1-3



Counting to infinity problem with the distance vector algorithm.

1.3.2 Path-Vector Algorithm

With the path-vector algorithm, the reachability information does not include the distance to the destination. Instead, the reachability information includes the entire path to the destination, not just the first hop as is the case for the distance-vector algorithm. A router running the path-vector algorithm includes itself in the path when redistributing a route. The path-vector algorithm allows a router to detect routing loops. Consider the path-vector example depicted in Figure 1-4.

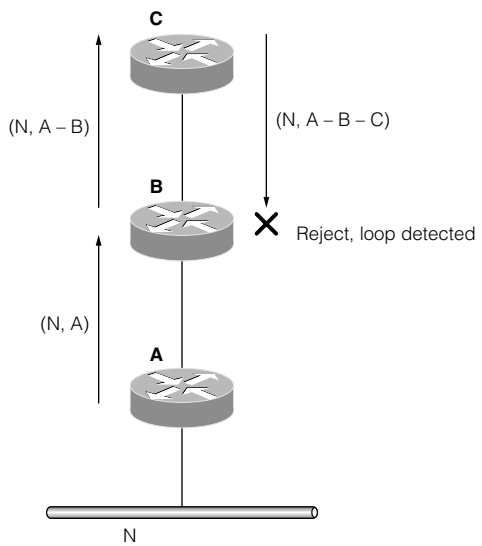
Each router advertises the destination network N along with the full path to reach N. Router A advertises the route about N to B. The only router on this path is A because N is a directly attached network. Router B adds itself into the path when it redistributes that route to router C. At router C the path to N contains A and B. If router C were to advertise this route back to B, B would find itself in that path and immediately detect the routing loop. In this case B will reject this route advertisement from C, thus breaking the loop.

Since the complete path information is available for each advertised route, the path-vector algorithm allows better policy control in terms of what advertised routes to accept, and allows policy to influence route computation and selection.

1.3.3 Link-State Algorithm

A router running the link-state algorithm advertises the state of each of its attached links, called its *link-state*, to its adjacent routers. A receiving router of the advertised state stores this information in its local database. This receiver then redistributes the received link-state information

FIGURE 1-4



Loop detection in path-vector algorithm.

unmodified to all of its adjacent routers, resulting in every router in that AS which participates in the routing protocol to receive the same link-state information. Each router then computes the paths to all possible destinations based on this link-state information independently.

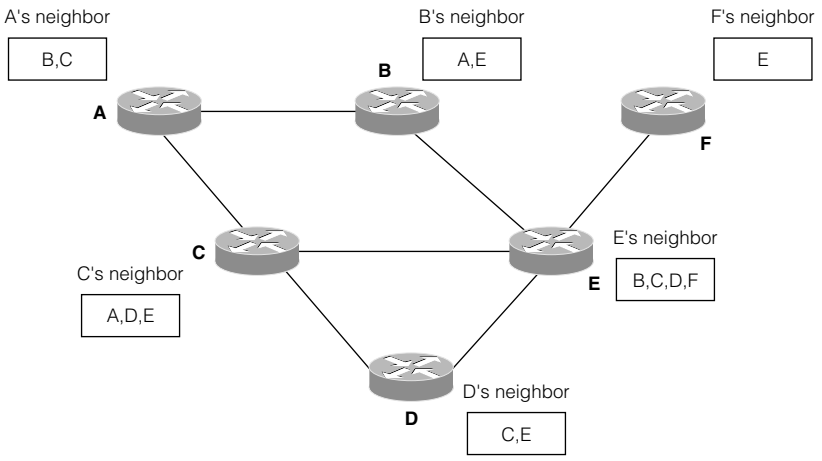
Figure 1-5 illustrates a state where routers in a routing domain have collected all link-states. For simplicity, this example assumes that a link-state is a set of neighbor routers. In actual routing protocol such as OSPFv3, a link-state contains many more parameters, such as per link cost and leaf network information. This example also assumes that some *flooding* mechanism is provided to advertise the link-states throughout the routing domain.

Once a router collects the link-states from all other routers, it can construct a tree-based map (also known as a *shortest path tree*) of the entire network that gives the shortest path to every part of the network as shown in Figure 1-6. The procedure used for the tree construction is called the *Dijkstra* algorithm, which is explained in Section 1.6.5.

Once every router computes the map, packet forwarding can be done based on the map. Figure 1-7 illustrates the forwarding path from router A to router F. Each router forwards the packet to the appropriate next hop following the path in the tree, and the result is a loop-free, shortest route to the destination.

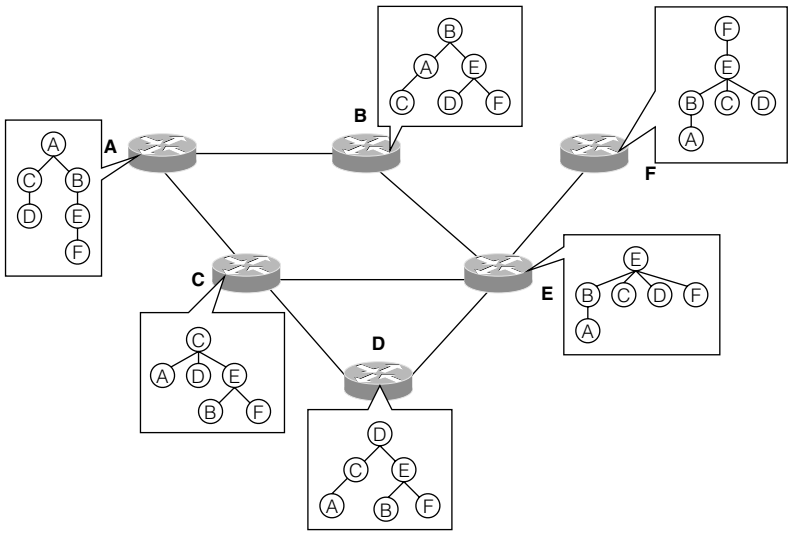
The flooding of the link-state information which originated from all of the routers enables each router to gain a complete view of the topology of the routing domain. Each router can build a routing table independently. As can be seen from the comparisons made between distance-vector algorithm and link-state algorithm, in the distance-vector algorithm, each router sends its entire routing database to neighboring routers because vector-based algorithms deploy a distributed route computation scheme. In contrast, since only per router link-state information is distributed, the amount of information that is exchanged among routers that run the link-state algorithm is considerably smaller. In response to changing network conditions, information exchanged among routers running the distance-vector may contain stale information,

FIGURE 1-5



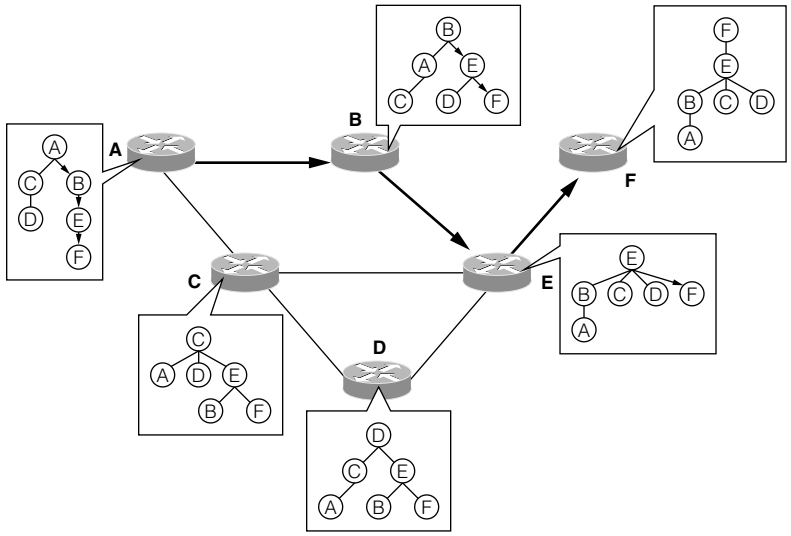
First stage of the link-state algorithm: flooding link-states (each router advertises its local state throughout the domain).

FIGURE 1-6



Second stage of the link-state algorithm: building the shortest path tree.

FIGURE 1-7



Routing based on the shortest path tree.

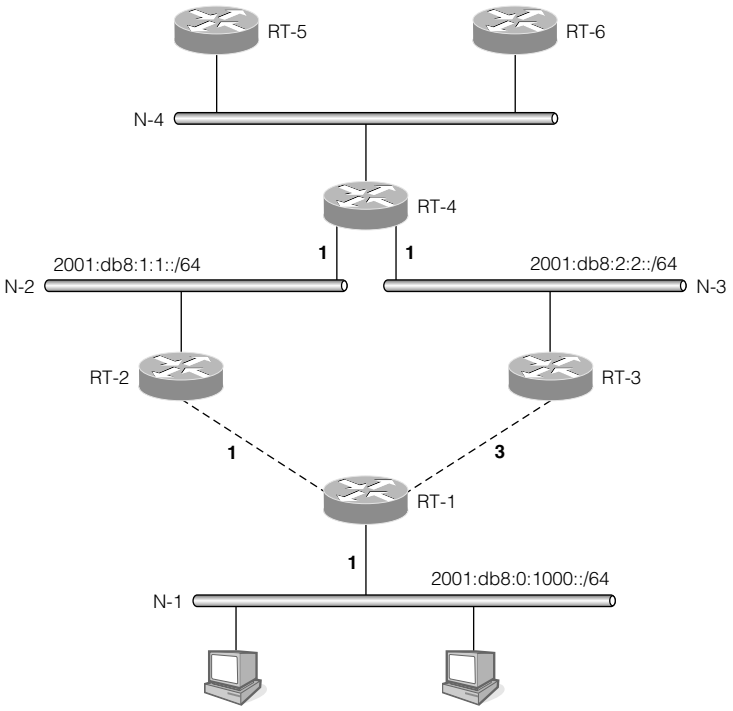
that is, an unreachable network may still be advertised as reachable by some routers. Such stale information will result in longer convergence time. Since the link-state algorithm exchanges information that pertains to a specific router, each router is more independent in calculating its routing database. This is one reason that the link-state algorithm has a fast convergence rate.

In conclusion, the distance-vector algorithm sends global routing information (a router's entire routing table) locally, while the link-state algorithm floods local information (attached interfaces and links) globally.

1.4 Introduction to RIPng

The RIPng protocol is based on the distance-vector algorithm commonly known as the Bellman-Ford algorithm. Consider the example in Figure 1-8. Router RT-1 advertises its directly connected network N-1 of prefix `2001:db8:0:1000::/64` with a metric of 1 on the point-to-point links to RT-2 and RT-3. The costs of the links from RT-1 to RT-2 and RT-3 are 1 and 3 respectively, which have RT-2 and RT-3 advertise the same prefix on networks N-2 and N-3 (where router RT-4 resides) but with different metrics. RT-2 advertises a metric value of 2 while RT-3 advertises a metric value of 4. After processing the routing messages from RT-2 and RT-3, RT-4

FIGURE 1-8



Example of RIPng route propagation.

selects the route with the smaller metric and chooses RT-2 as the next hop to reach network `2001:db8:0:1000::/64` with a metric of 2. RT-4 adds the cost of its network to the received metric and advertises that prefix with a metric of 3 on network N-4. RT-4 also advertises prefixes `2001:db8:1:1::/64` and `2001:db8:2:2::/64` for networks N-2 and N-3 on N-4. RT-5 and RT-6 receive this routing information and build their routing tables.

We can see from this example that RT-4 is able to compute the optimal route to prefix `2001:db8:0:1000::/64` from just the information exchanged with RT-2 and RT-3. The direction going toward this network is through RT-2.

1.4.1 RIPng Message Formats

Figure 1-9 depicts the RIPng protocol message format. Each decimal value in parentheses refers to the field size in bytes.

command This field specifies whether the RIPng message is a request message or a response message. A value of 1 indicates a request message, a value of 2 indicates a response message.

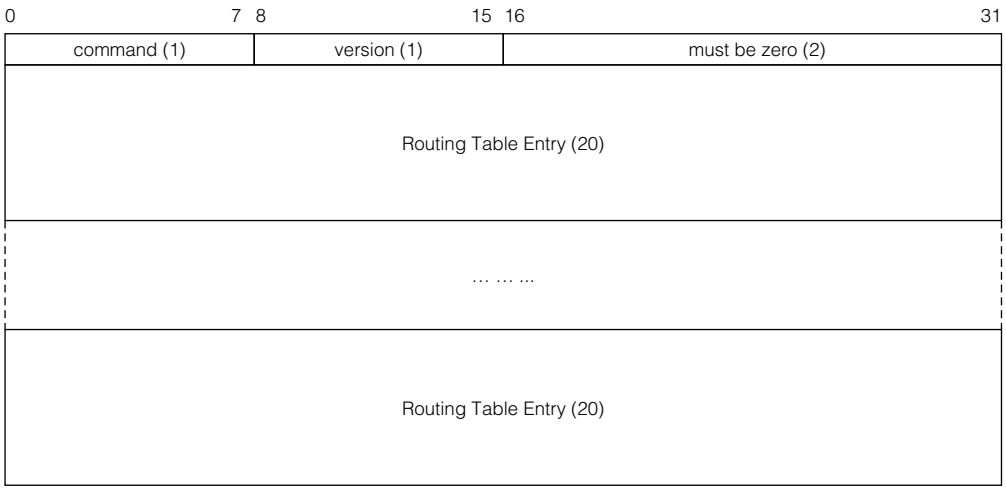
version This field specifies the version of the RIPng protocol in operation. [RFC2080] defines version 1 of RIPng.

The next two bytes must be set to zero by the sender.

Each Routing Table Entry (RTE) field is 20 bytes in size and specifies a reachable IPv6 destination. The format of the RTE field is shown in Figure 1-10.

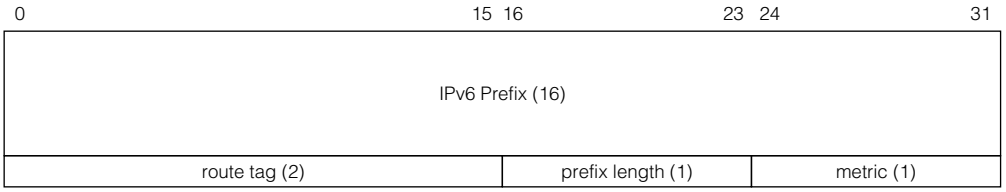
IPv6 prefix The IPv6 prefix is 16 bytes in size and specifies either an IPv6 network or an IPv6 end node depending on the prefix length. The prefix is stored in network byte order.

FIGURE 1-9



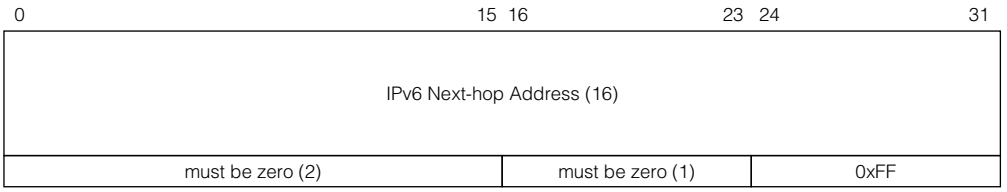
RIPng protocol message format.

FIGURE 1-10



Routing table entry.

FIGURE 1-11



Specifying next hop address.

route tag The route tag is considered an attribute of the advertised destination. The route tag was designed for distinguishing the origin of the route, that is, whether the advertised prefix was imported from another routing protocol. In practice the route tag is used by operators to define sets of routes for custom handling in RIPng. The receiving router must preserve this field and readvertise it with the prefix.

prefix length The prefix length specifies the number of significant bits of the prefix field, counting from left to right. The valid prefix length is 0 to 128 inclusive. The prefix field is ignored if the prefix length is 0, which indicates that the advertised route is a default route. In this case it is good practice to set the prefix field to `0:0:0:0:0:0:0:0`, or `::` in compressed form.

metric Even though the metric field is 1 byte in size, the valid values are 0 to 15, which specifies the cost of reaching the advertised destination. Value 16, known as infinity, indicates the advertised destination is not reachable. We will revisit the infinity value in Section 1.4.2.

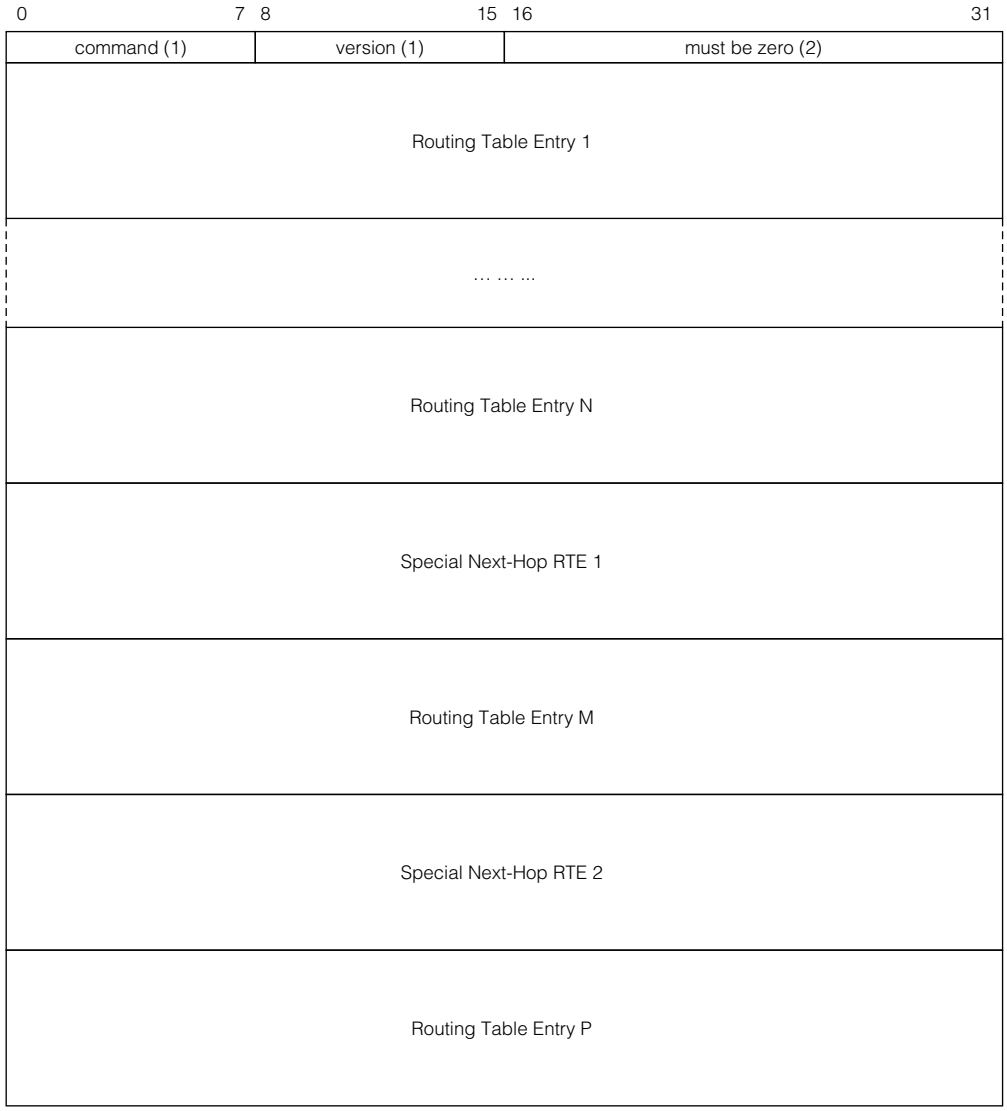
In general the receiving router sets the advertising router as the next hop for the advertised destination. The advertising router may optionally specify a next hop router for one or more destinations by a special RTE. Figure 1-11 illustrates the special RTE that is used for specifying a next hop address.

The IPv6 next hop address field contains a link-local address of the next hop router, which belongs to the interface that shares the same network segment as the advertising router. The metric field is set to 0xFF. If the next hop address is `::`, then the originating router of the message is used as the next hop router. If the next hop address is not a link-local address, then again the originating router is used as the next hop router. All of the RTEs that follow this

special RTE will use the given address as the next hop router until either another special RTE is encountered, or the end of the message is reached. For example, consider Figure 1-12.

In this example, the advertising router is treated as the next hop router for RTE 1 to N. The next hop router specified in the special RTE 1 applies to destination M, and the next hop router specified in the second special RTE applies to destination P.

FIGURE 1-12



RIPng Route Advertisement containing next hop information.

The number of RTEs that can be carried in a single RIPng message is limited by the link MTU. The formula for calculating the number of RTEs is given as:

$$\text{number of RTEs} = \frac{\text{MTU} - (\text{length of IPv6 headers}) - (\text{UDP header length}) - (\text{RIPng header length})}{\text{size of RTE}} \quad (1.1)$$

1.4.2 RIPng Operation

The RIPng protocol operates over UDP. The IANA assigned port number for the RIPng process is 521.

A router sends its entire routing table to all its directly connected neighboring routers every 30 seconds—called a *regular update*. This unsolicited transmission has a UDP source port 521 and a destination port 521. The source address must be a link-local address of the transmitting interface of the originating router. The destination address is the *all-rip-routers* multicast address `ff02::9`.

When a router first comes up and is in the initialization phase, it may request other routers to send their routing tables in order for it to populate its routing table. This request may be sent to the all-rip-routers multicast address on each attached interface. If there exists only a single RTE in the RIPng request message, and the prefix in this RTE is `::`, the prefix length is 0, and the metric value is 16, then the requesting router is asking the receiving router to send its entire routing table.

A router may send a request to a specific peer soliciting a specific list of destinations. In this case, the receiving router processes each RTE by performing a search of the given prefix in its routing table. If an entry is found, then the metric value is retrieved and is set in the RTE. If an entry is not found, then the metric value is set to 16 indicating the destination is not reachable from this router's perspective. Once all of the RTEs have been processed, the command field is changed from request to response and is sent to the requesting router.

When a router receives a message from a neighbor, if it contains a destination that is not already in its routing table, or if either the metric or the next hop address of an existing route entry is updated by the newly received RTE, then the corresponding route entry in the routing table is created or updated with new information. The next hop address of the entry is set to the source address of the received message or the next hop address specified by a next hop RTE (shown in Figure 1-11); note that in either case the address is a link-local address. The receiving router will then send an update message on all other interfaces. This process is called a *triggered update*, and is limited to one transmission per 1 to 5 seconds, depending on the timer expiration.

There are two timers associated with each route entry: the *timeout* timer and the *garbage collection* timer. The timeout timer is initialized to 180 seconds when the route entry is first created. Each time a response message which contains the destination of this route entry is received, the timeout timer is reset to 180 seconds. The route entry is considered expired if a message has not been received which covers that destination. In this case, once the route entry expires, a garbage collection timer is initialized to 120 seconds for the expired entry. The route entry is removed from the routing table when the garbage collection timer expires. The reason for setting the garbage collection timer is to aid convergence, as explained in Section 1.4.3.

The following points summarize the key characteristics of the RIPng protocol:

- The routing algorithm selects a best route for each possible destination using distance as the main selection criteria.
- Each piece of routing information consists of a destination, a gateway and the distance to the destination.
- A router exchanges routing information with only directly connected routers. Route information from a new neighboring router can be dynamically reflected, but the state of each router is not maintained.
- Distribution of routing information is unreliable because the routing information is exchanged over UDP without any application-level acknowledgments.
- Origin of a route cannot be identified.
- Route computation is distributed in that selection decision made at one router depends largely on the route selection decisions made by other routers.
- Routing loop cannot always be detected or avoided.
- The algorithm can be vulnerable to topological changes and can converge slowly.

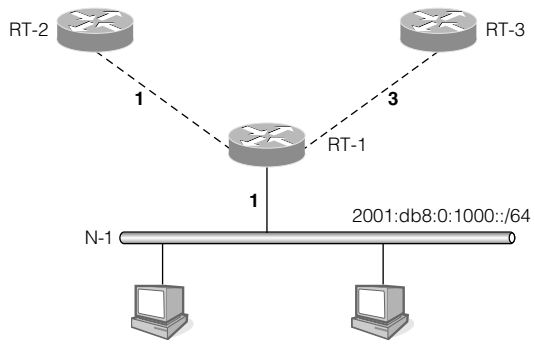
1.4.3 Problems with RIPng

The main advantage of RIPng is that RIPng is a simple routing protocol and its implementation is fairly straightforward. This simplicity, however, causes a number of operational problems. The most visible problems are its inability to detect routing loops in more complex network topologies and that it may converge slowly in some situations. This was briefly discussed in Section 1.3.1; this section revisits and details the problem in the context of the RIPng protocol. Consider the example given in Figure 1-13.

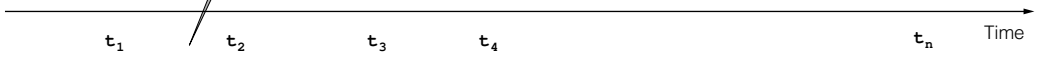
In this figure, the horizontal axis represents time. The first vertical column lists the available routers RT-1, RT-2 and RT-3. Starting from the second column, each pair of values represents each router's perspective on the reachability of the IPv6 prefix `2001:db8:0:1000/64`, that is, the gateway to the prefix and the cost of that path. For example, at time t_1, RT-1 sees the prefix `2001:db8:0:1000::/64` as directly reachable, and the cost of that route is 1. At the same time, router RT-2 and RT-3 view the prefix as reachable through router RT-1, and the costs of the route are 2 and 4 respectively.

Router RT-1 advertises network `2001:db8:0:1000::/64` to both RT-2 and RT-3 with metric 1. At time t_2, the link between RT-1 and N-1 is broken. This link is the only one to reach N-1. Now RT-1 correctly marks `2001:db8:0:1000::/64` as unreachable. However, at time t_3 both RT-2 and RT-3 still advertise a route to `2001:db8:0:1000::/64` with metric 2 and 4 respectively. At time t_4, upon receiving these routes from RT-2 and RT-3, RT-1 incorrectly thinks that N-1 is reachable through either RT-2 or RT-3. Since RT-2 advertises a smaller metric, RT-1 treats RT-2 as the next hop router and inserts the route with the updated metric 3 into its routing table. This route entry is then advertised to RT-2 and RT-3 causing both RT-2 and RT-3 to think N-1 is now reachable via RT-1 but with a new metric value. RT-2 and RT-3 update their metric values accordingly and readvertise the route to RT-1. RT-1 again updates its metric value and then advertises the update route back to RT-2 and RT-3. This process continues until eventually

FIGURE 1-13



RT-1	directly, 1	Link between RT-1 and N-1 breaks	unreachable	RT-2, 3	RT-2, 5	RT-2, 6	RT-2, 7	···	RT-2, 15
RT-2	RT-1, 2		RT-1, 2	RT-1, 4	RT-1, 6	RT-1, 7	RT-1, 8	···	RT-1, 16
RT-3	RT-1, 4		RT-1, 4	RT-1, 7	RT-1, 9	RT-1, 10	RT-1, 11	···	RT-1, 16



t_1 t_2 t_3 t_4 t_n

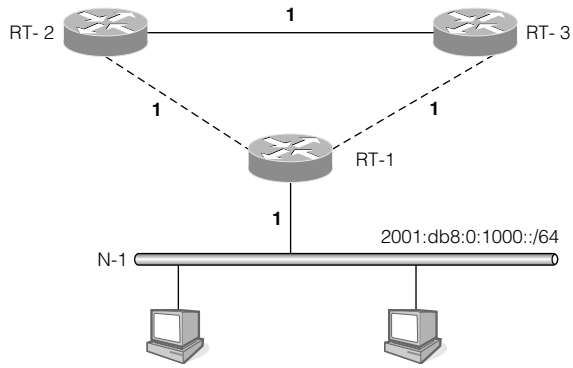
Problem of counting to infinity.

all three routers have a metric value of 16 to `2001:db8:0:1000::/64`, which indicates this network is no longer reachable, that is, this is a symptom of the counting to infinity problem described in Section 1.3.1.

With the counting to infinity problem, none of the routers would know that N-1 is no longer reachable until the metric value reaches 16 at a time long passed t_2, (i.e., at time t_n). One reason that the metric for RIPng has a maximum allowed value of 16 is that the larger the allowable metric, the longer RIPng takes to reach the convergence state. The maximum value of 16 also implies that RIPng is limited to networks that have diameters of at most 15 hops, assuming the cost of each hop is 1. Note in this example, RT-3 converges faster than both RT-1 and RT-2 due to its larger metric value.

As illustrated by this example, due to the counting to infinity problem, RIPng has a large convergence time when it is deployed in more complex networks. As can be seen from this example, the source of the problem is that RT-2 is advertising to RT-1 a route which RT-2 had learned from RT-1. The problem disappears if RT-2 never advertises any route that was learned from RT-1 back to RT-1. This solution is called the *Split Horizon* algorithm. Alternatively, RT-2 may advertise the route that it learned from RT-1 back to RT-1, but RT-2 sets the metric to 16 which indicates that destination is not reachable through RT-2, which is known as *route poisoning*. The garbage collection timer mentioned in the pevious section, also known as the *hold-down* timer, is used for route poisoning. During the hold-down time, an expired route is

FIGURE 1-14



Network configuration causing RIPng routing loop.

advertised to the neighbors with a metric of 16. Split horizon combined with route poisoning is called *Split Horizon with Poisoned Reverse*.

The Split Horizon algorithm solves the problem depicted in Figure 1-13, but this algorithm still cannot detect the routing loop if the network has a configuration as shown in Figure 1-14.

In this figure, because RT-2 and RT-3 share a common link, the two routers will advertise `2001:db8:0:1000::/64` with a metric of 2 to each other. Due to Split Horizon with Poisoned Reverse, RT-2 and RT-3 both advertise to RT-1 that `2001:db8:0:1000::/64` is unreachable. When the link between RT-1 and N-1 is broken, RT-1 will mark N-1 as unreachable. At this point RT-2 considers the new route to `2001:db8:0:1000::/64` is through RT-3 with a metric of 3. RT-2 will also advertise this route to RT-1. RT-1 is led to believe that `2001:db8:0:1000::/64` is now reachable through RT-2. Again the counting to infinity problem occurs and the Split Horizon algorithm could not detect the routing loop in this configuration.

1.5 Introduction to BGP4+

The BGP-4 protocol as defined in [RFC4271] is an exterior routing protocol that is mainly deployed between different autonomous systems (ASs). Since the original BGP-4 specification assumes the routing protocol operates over the IPv4 network, the routing messages carry only IPv4 routes. [RFC2858] updates the BGP-4 specification to support additional protocols such as IPv6. The extended BGP is commonly known as BGP4+. The specific use of BGP4+ by IPv6 is documented in [RFC2545]. We will use BGP4+ to refer to BGP4+ as deployed in the IPv6 network. We will also use the terms BGP4+ and BGP interchangeably for the remainder of this chapter.

In the context of BGP4+, each AS has an *Autonomous System Number* or ASN. The ASN can be either a public ASN or a private ASN. A public ASN is a globally unique identifier and is assigned by an organization such as RIR or NIR. The IANA has reserved AS64512 to AS65535 as private ASNs. [RFC1930] discusses ASN in detail.

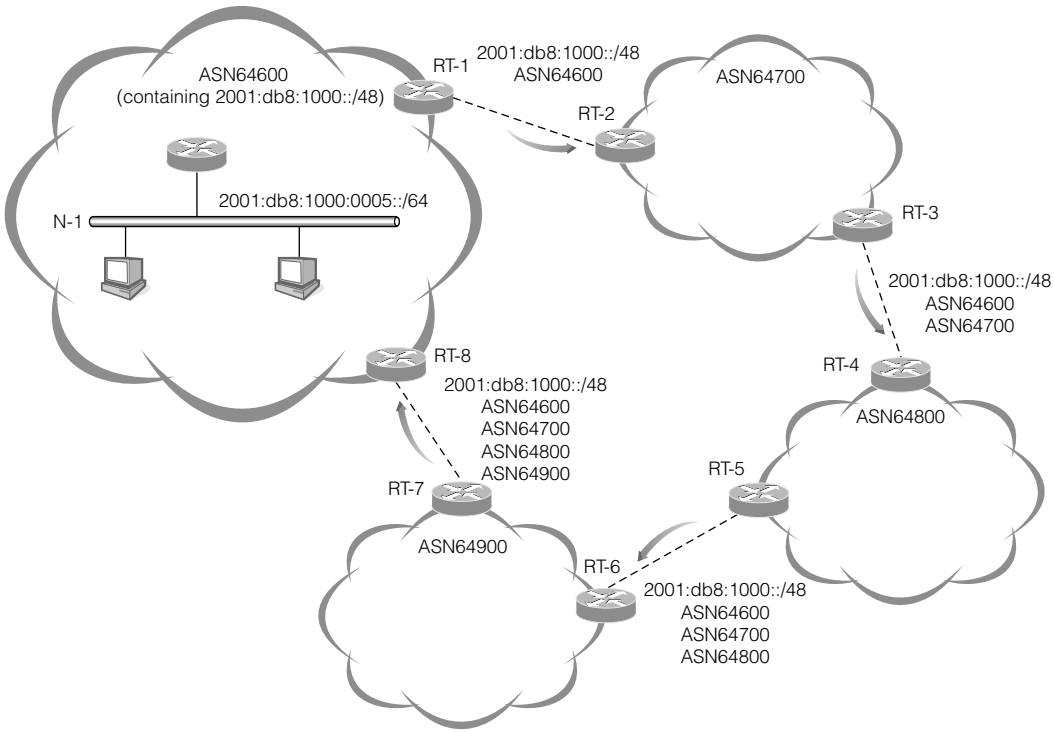
RIR stands for the *Regional Internet Registry*, which is responsible for the allocation and management of IP adresses and ASNs for a specific region of the world. Today there exist five RIRs: AfriNIC (Africa), APNIC (Asia Pacific), ARIN (North America), LACNIC (Central and South America), and RIPE-NCC (Europe).

NIR stands for the *National Internet Registry*, which is responsible for IP address allocation and management for a specific country.

BGP4+ uses the path-vector algorithm and solves the routing loop detection problem by including the path to the destination in the route message. When a BGP4+ router receives a route update, a router will update the path information to include its ASN before redistributing that route to other ASs. Since BGP4+ is an exterior routing protocol, routing information is exchanged among ASs, each with a different routing policy that governs what information could be made externally visible. For this reason, the path information carried in the route message is a list of ASNs instead of a list of specific routers in order to hide the internal topology of each AS on that path. Consider the example given in Figure 1-15.

As illustrated in the figure, each router records its ASN in the route message before distributing that route to another router. But first, each router must validate the received route message

FIGURE 1-15



BGP4+ routing loop detection.

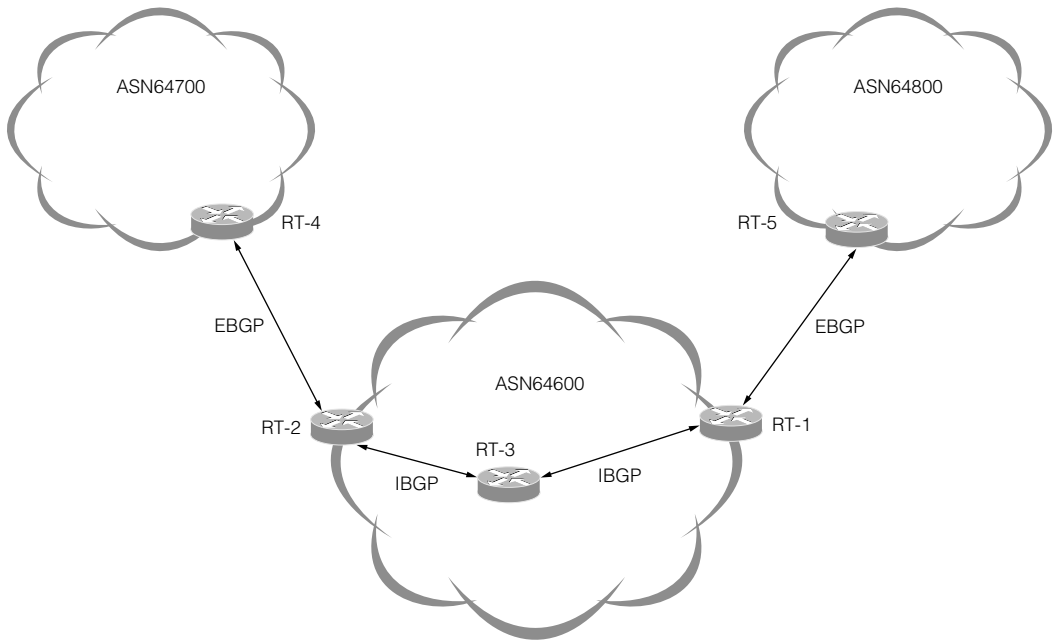
by examining the path information and verify that its ASN is not present in the path. Routers RT-2 to RT-7 accept the received route message according to this rule. RT-8 finds its ASN in the route message originated from RT-7, therefore detecting the routing loop. In this case RT-8 rejects the advertised route from RT-7, thus breaking the loop.

1.5.1 BGP4+ Operation

BGP4+ operates over TCP. A BGP4+ router establishes a *peering* relationship with another BGP4+ router by establishing a TCP connection to port 179 of that other router. The two BGP4+ routers are called *BGP peers*, also known as *BGP speakers*. Typically BGP4+ is deployed for inter-AS routing, but large organizations and enterprises that have hundreds of branch offices also deploy BGP4+ within an AS. When a BGP4+ router peers with another BGP4+ router of the same AS, these routers are called *internal BGP* (IBGP) peers. When a BGP4+ router peers with a BGP4+ router of another AS, these routers are called *external BGP* (EBGP) peers. Figure 1-16 illustrates the concept of EBGP and IBGP peers.

As shown in the figure, routers RT1, RT2 and RT3 are IBGP peers because these routers belong to the same AS(64600). RT1 and RT5 are EBGP peers. Similarly RT2 and RT4 are EBGP peers. In this figure, router RT3 acts as a *route reflector* that redistributes routing information learned from one IBGP peer to another IBGP peer. BGP route reflector is fully described in [RFC2796].

FIGURE 1-16

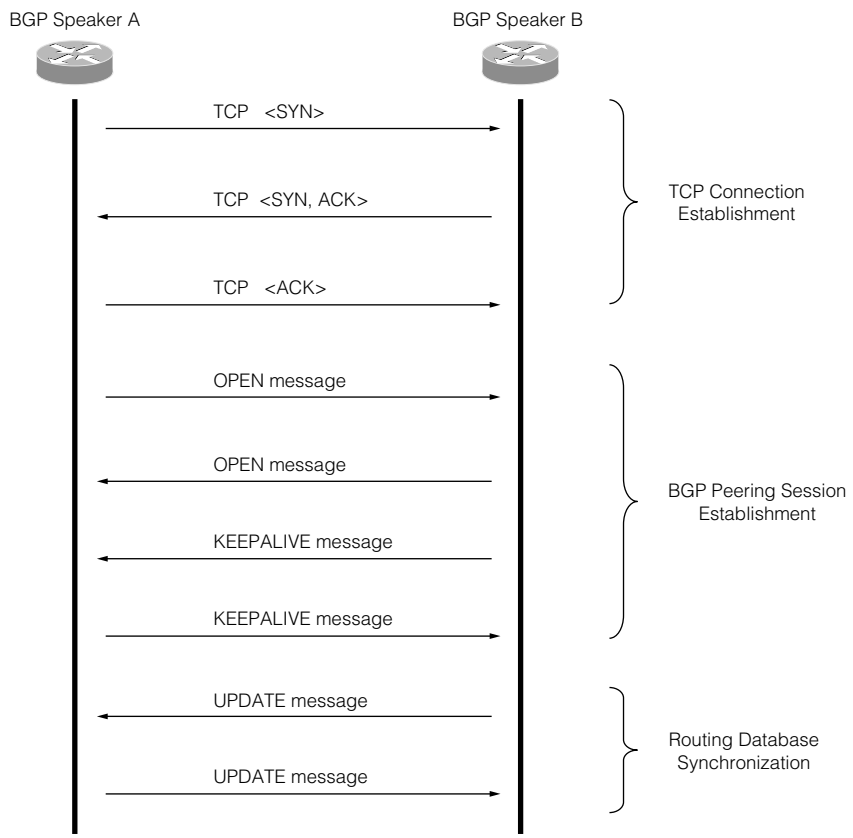


BGP4+ IBGP and EBGP peers.

The BGP4+ routers cannot exchange any routing information until the peering process completes successfully. BGP defines a *Finite State Machine* (FSM) to represent its operation. Associated with the FSM is a set of events and timers that trigger state transition. The BGP peering process is part of the FSM but we omit the discussion of the BGP FSM in detail. Instead, we will describe the peering process through one possible scenario as illustrated in Figure 1-17.

The peering process begins by first establishing a TCP connection. One BGP4+ router initiates the TCP connection to another BGP4+ router. It is possible that both routers try to initiate the TCP connection to each other at the same time. In order to avoid establishing two TCP connections, the BGP router with the smaller BGP Identifier (see Figure 1-19) will cancel its TCP connection request. The OPEN message is the first BGP message sent once the underlying TCP connection has been established successfully. A subsequent KEEPALIVE message confirms the OPEN message. Notice two BGP speakers send the OPEN and the KEEPALIVE messages. The BGP speakers then exchange their routing databases through the UPDATE message once the BGP peering session is established.

FIGURE 1-17



BGP4+ peering process.

For each of its peers, a BGP router maintains a database that stores the routes advertised by its peer, and a separate database that stores the routes that it advertised to the peer. By maintaining separate databases on incoming and outgoing route exchanges, each BGP router is able to determine which updated routes affect its peer, and distributes to its peer only those updates to reduce routing traffic.

Since BGP is mainly deployed between different ISPs or between different companies, routing policies and enforcement of those policies play a significant role in BGP. It is important for an ISP or a company to define what types of routes can be accepted from a peer, what types of routes can be distributed to a peer, what external routes can be redistributed internally and externally, which entry points inbound traffic should take, which exit points outbound traffic should take, and much more.

The following points summarize the key characteristics of the BGP4+ routing protocol:

- The routing algorithm selects a best route for each possible destination by examining the path information in conjunction with the locally administered routing policies.
- Routes are re-advertised to other routers that are not directly connected. There is no dynamic discovery of neighboring routers. Routes are exchanged with configured peer routers.
- Each piece of routing information consists of the destination, a gateway and the entire path to the destination.
- Distribution of routing information is reliable because route exchanges are carried over TCP.
- Origin of each route can be identified.
- Routing loops can be easily detected and avoided.
- Route computation is distributed in that selection decision made at one router depends largely on the route selection decisions made by other routers; however, decision of route selection can be made by local policy with a good degree of flexibility due to the explicit loop avoidance mechanism.

1.5.2 BGP4+ Messages

There are four message types in BGP, which are shown in Table 1-1.

Message Header

The BGP message header format is shown in Figure 1-18.

Marker This 4-byte field must be filled with all ones.

Length This 2-byte field specifies the size of the BGP message. The message header is 19 bytes, so Length can have the minimum value of 19. The maximum value that Length can have is 4096 including the header size.

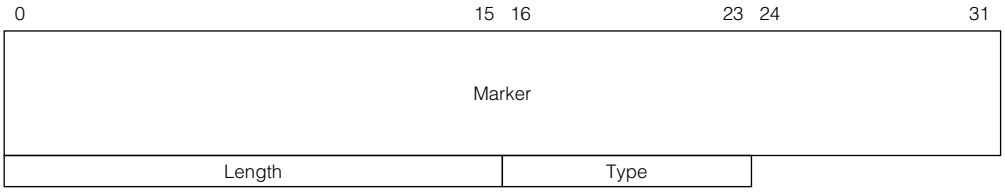
Type This 1-byte field specifies the BGP message type as described in Table 1-1.

TABLE 1-1

Type	*Name*	*Description*
1	**OPEN**	The OPEN message is the first message that is sent over the TCP connection to initiate the peering exchange.
2	**UPDATE**	The UPDATE message carries routing information and is exchanged between the peers. A router also sends the UPDATE message to withdraw a previously advertised route.
3	**NOTIFICATION**	A router sends the NOTIFICATION message when it detects an error condition and closes the connection.
4	**KEEPALIVE**	Instead of relying on the TCP keep-alive mechanism, a BGP4+ router sends the KEEPALIVE message to detect the liveliness of its peer. The KEEPALIVE message is also sent in response to an OPEN message to complete the initial peering handshake.

BGP message types.

FIGURE 1-18



BGP4+ message header.

OPEN message

The OPEN message is the first message that is exchanged between two BGP speakers once the TCP connection is established between them. The OPEN message serves as the request to establish a peering relation. The OPEN messages also allow the BGP speakers to identify each other's capabilities. The BGP speakers may fail to establish the peering relationship if incompatibilities are found.

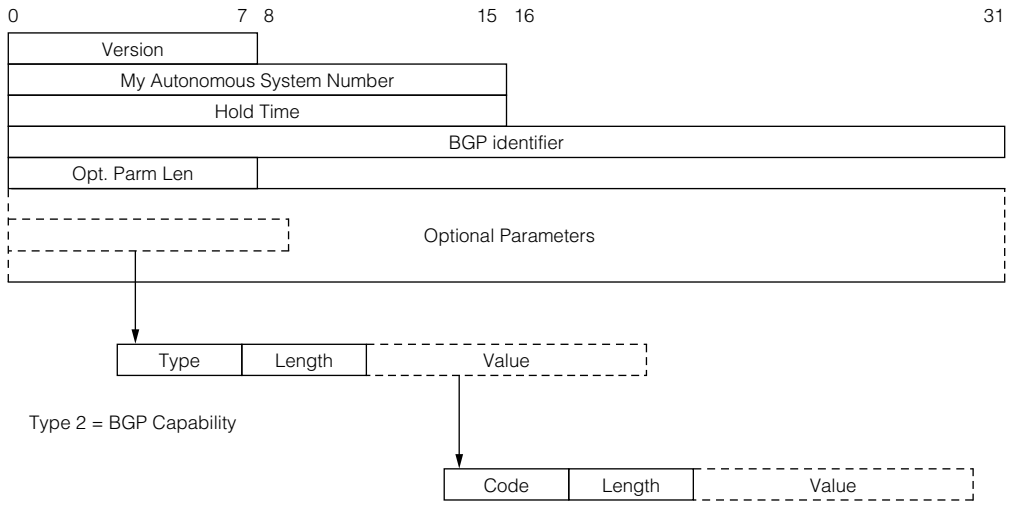
The OPEN message contains the message header and the additional fields that are shown in Figure 1-19.

Version This 1-byte field specifies the BGP protocol version number. The current BGP version is 4.

My Autonomous System Number This 2-byte field contains the ASN of the sender of the OPEN message.

Hold Time This 2-byte field specifies the maximum duration between successive KEEPALIVE or UPDATE messages. This value is proposed from the sender. The receiver sets its Hold

FIGURE 1-19



BGP4+ OPEN message.

Time to be the smaller of its configured value and the proposed Hold Time from the received OPEN message. A zero Hold Time implies the sender does not need to send any message. The Hold Time must be at least 3 seconds if its value is not zero. In this case the TCP connection is closed if the receiver does not receive a KEEPALIVE, UPDATE or NOTIFICATION message when the Hold Time expires. The value received in the Hold Time field may cause a BGP speaker to reject the peering request.

BGP Identifier The 4-byte field contains a valid IPv4 unicast address of the sender.

Optional Parameters Length This 1-byte field specifies size of the optional parameters that are present in the message. No options are present if the length is zero. Otherwise these optional parameters will be negotiated with the receiver.

Optional Parameters The variable length Optional Parameters field contains the parameters to be negotiated with the receiver. Each optional parameter has the <type, length, value> format. The Type 2 parameter represents the BGP capabilities. The value field of a Type 2 parameter is encoded as <code, length, value>. Table 1-2 lists the currently defined capability codes, their descriptions and the documents in which they are defined.

KEEPALIVE message

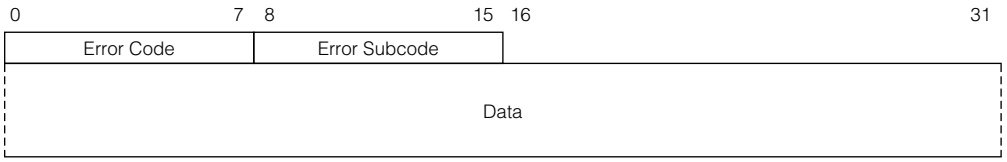
The KEEPALIVE message contains only the message header and is therefore 19 bytes in size. The KEEPALIVE message is sent to avoid the expiration of the Hold Time and serves the same purpose as the TCP keepalive packets, that is, to verify the connection state. The KEEPALIVE message is rate limited and more than one message per second must not be sent. The KEEPALIVE message must not be sent if the negotiated Hold Time is zero.

TABLE 1-2

Value	*Description*	*Reference*
0	Reserved	[RFC3392]
1	Multiprotocol Extensions for BGP-4	[RFC2858]
2	Route Refresh Capability for BGP-4	[RFC2918]
3	Cooperative Route Filtering Capability	[ROUTE-FILTER]
4	Multiple routes to a destination capability	[RFC3107]
5–63	Unassigned	
64	Graceful Restart Capability	[IDR-RESTART]
65	Support for 4-byte AS number capability	[AS4BYTES]
66	Deprecated (2003-03-06)	
67	Support for Dynamic Capability (capability specific)	
68–127	Unassigned	
128–255	Vendor Specific	

BGP capability codes.

FIGURE 1-20



BGP4+ NOTIFICATION message.

NOTIFICATION message

The NOTIFICATION message is sent when an error condition is detected by a BGP speaker. The BGP speaker terminates the connection immediately after sending the message. The NOTIFICATION message contains the message header and the additional fields that are shown in Figure 1-20.

Error Code This 1-byte field indicates the type of error that has occurred either during the peering process or during an established BGP session.

Error Subcode The value of this 1-byte field depends on the value of the Error Code field.

Data The Data field is variable in length and its content depends on both the Error Code and the Error Subcode. At a minimum the NOTIFICATION message is 21 bytes in size if Data is not present.

The various error codes are listed in Table 1-3.