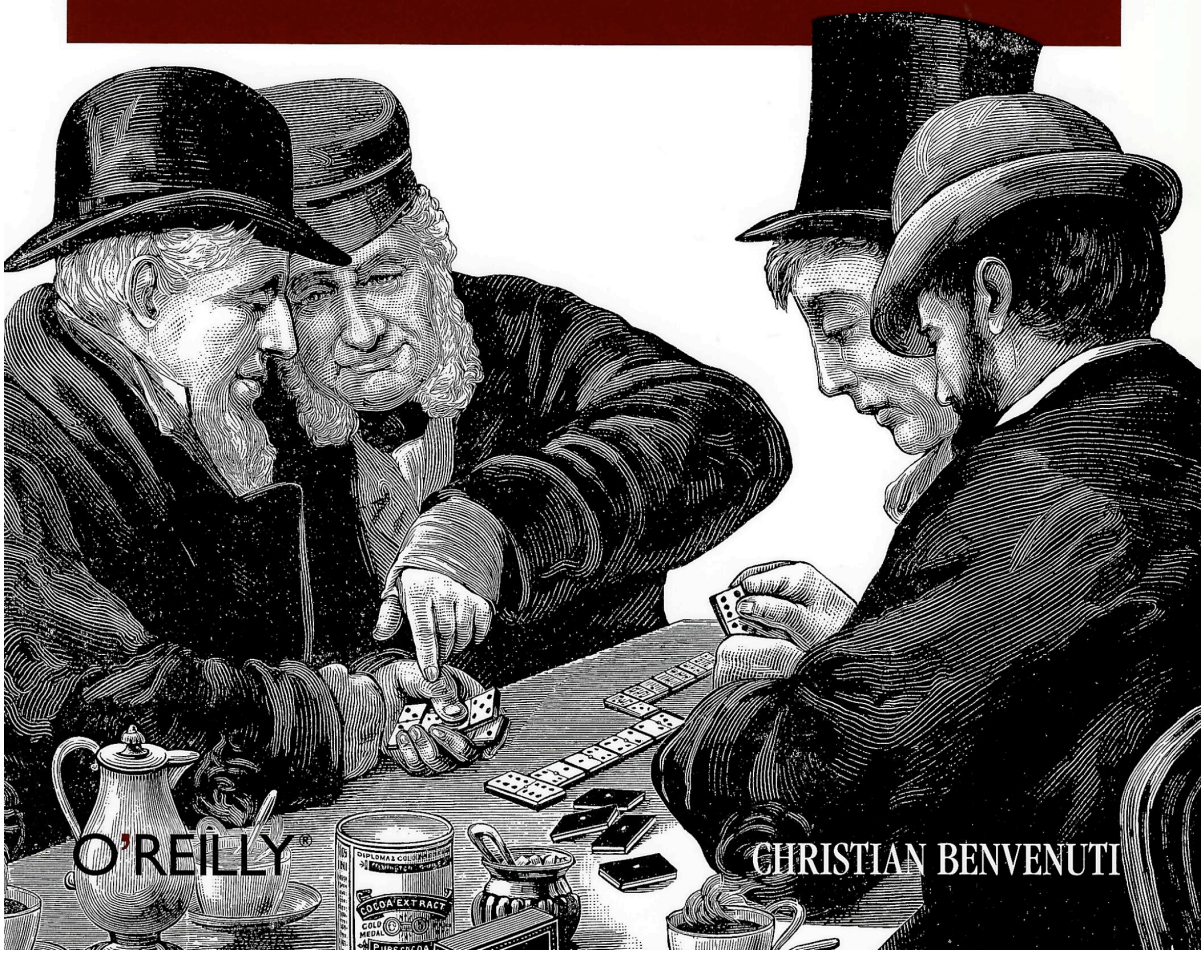
GUIDED TOUR TO NETWORKING ON LINUX

Understanding LINUX NETWORK INTERNALS



O'REILLY®

CHRISTIAN BENVENUTI

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

Understanding Linux Network Internals
by Christian Benvenuti

Copyright © 2006 O'Reilly Media, Inc. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Andy Oram

Production Editor: Philip Dangler

Cover Designer: Karen Montgomery

Interior Designer: David Futato

Printing History:

December 2005: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. The *Linux* series designations, *Understanding Linux Network Internals*, images of the American West, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

[M]
ISBN: 0-596-00255-6

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

net_device Structure

The net_device data structure stores all information specifically regarding a network device. There is one such structure for each device, both real ones (such as Ethernet NICs) and virtual ones (such as bonding* or VLAN†). In this section, I will use the words *interface* and *device* interchangeably, even though the difference between them is important in other contexts.

The net_device structures for all devices are put into a global list to which the global variable dev_base points. The data structure is defined in *include/linux/netdevice.h*. The registration of network devices is described in Chapter 8. In that chapter, you can find details on how and when most of the net_device fields are initialized.

Like sk_buff, this structure is quite big and includes many feature-specific parameters, along with parameters from many different layers. For this reason, the overall organization of the structure will probably see some changes soon for optimization reasons.

Network devices can be classified into *types* such as Ethernet cards and Token Ring cards. While certain fields of the net_device structure are set to the same value for all devices of the same type, some fields must be set differently by each model of device. Thus, for almost every type, Linux provides a general function that initializes the parameters whose values stay the same across all models. Each device driver invokes this function in addition to setting those fields that have unique values for its model. Drivers can also overwrite fields that were already initialized by the kernel (for instance, to improve performance). You can find more details in Chapter 8.

The fields of the net_device structure can be classified into the following categories:

- Configuration
- Statistics
- Device status
- List management
- Traffic management
- Feature specific
- Generic
- Function pointers (or VFT)

* Bonding, also called EtherChannel (Cisco terminology) and trunking (Sun terminology), allows a set of interfaces to be grouped together and be treated as a single interface. This feature is useful when a system needs to support point-to-point connections at a high bandwidth. A nearly linear speedup can be achieved, with the virtual interface having a throughput nearly equal to the sum of the throughputs of the individual interfaces.

† VLAN stands for Virtual LAN. The use of VLANs is a convenient way to isolate traffic using the same L2 switch in different broadcast domains by means of an additional tag, called the VLAN tag, that is added to the Ethernet frames. You can find an introduction to VLANs and their use with Linux at *http://www.linuxjournal.com/article/7268*.

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

the presence of multiple DHCP servers with overlapping address pools, and incorrect manual configurations.

To detect the presence of a duplicate address, a host can use gratuitous ARPs. If you send an ARP solicitation for your own address, you will receive a reply only when another host is configured with your IP address. If there is no duplicate address, no replies should be received.

Let's see an example using the topology in Figure 28-3. When Host A boots up, as soon as it configures its *eth0* interface with IP address 10.0.0.4, it sends a request asking who has IP address 10.0.0.4 (its own IP address). If none of the hosts in the subnet was misconfigured, Host A will not receive a reply. But since Host Bad_guy is configured with the same 10.0.0.4 IP address as Host A, it replies to the `ARPOP_REQUEST`, thus informing Host A of the presence of a duplicate address.

Of course, allowing hosts to send out ARP packets at random intervals on large networks is bad for performance. Instead, as shown in the section "Requests with zero addresses," a DHCP sever usually issues the request before granting an address to a client, which is a more scalable solution.

The Linux kernel does not generate any gratuitous ARP when you configure an IP address on the local interfaces. However, most Linux distributions come with the *iputils* package installed, which includes the *arping* command. *arping* can be used to generate `ARP_REQUEST` frames. When you enable a network interface with the */sbin/ifup* command (part of the *initscripts* package), it uses *arping* to check for duplicate addresses.

Virtual IP

Another common use for gratuitous ARP is to allow failover in a pool of servers. Commonly, to provide redundancy, a site provides one active server along with a number of similarly configured hosts in standby mode. When the active server fails for some reason, a mechanism often referred to as a *heartbeat timer* (implemented through some protocol on the pool of servers) detects the failure and triggers the election of a new active server. This new server generates a gratuitous ARP to update the ARP cache of all the other hosts in the network. Because the new server has taken the IP address of the old server, the `ARPOP_REQUEST` is not answered, but all the recipients update their caches accordingly.

Note that in this way, the IP layer and higher layers can keep communicating without even noticing the change. Of course, because heartbeats are sent out at regular intervals, a small window of time exists after the old server fails and the new one takes over, during which traffic is not delivered. So some nodes may discover the failure and mark their neighbor entries as failed until the new `ARPOP_REQUEST` arrives.

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

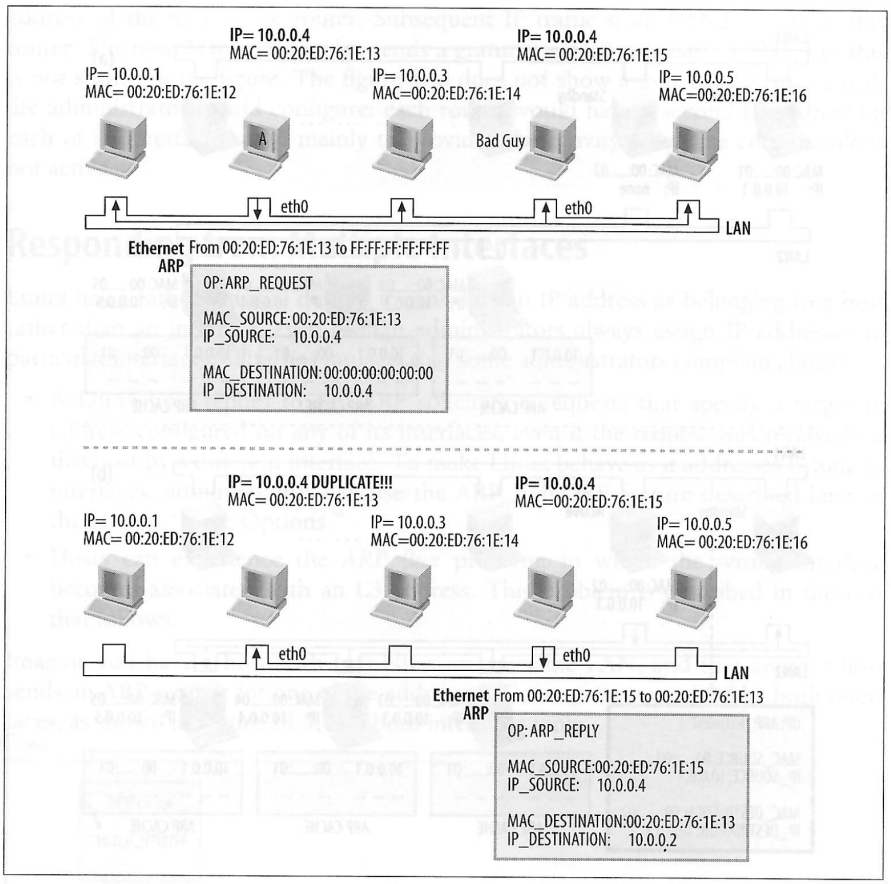


Figure 28-3. Example of duplicate address detection

The example in Figure 28-4* shows two routers, one taking the active role and the other taking the standby role (a). The server labeled Active has the IP address 10.0.0.1. The hosts of LAN2 use this router to communicate with the hosts of LAN1, and vice versa.

A failover system is in place so that when the Active router fails, the Standby router takes over the IP address 10.0.0.1 and becomes the Active router (b). When the Standby router becomes the new Active router, sends out a gratuitous ARP request that changes the entries of all local hosts (c) so that 10.0.0.1 is associated with the L2

* The MAC addresses in the figure are truncated for convenience. For example, 00:...:03 stands for 00:00:00:00:00:03. I used simple MAC addresses like that one to simplify the figure.

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

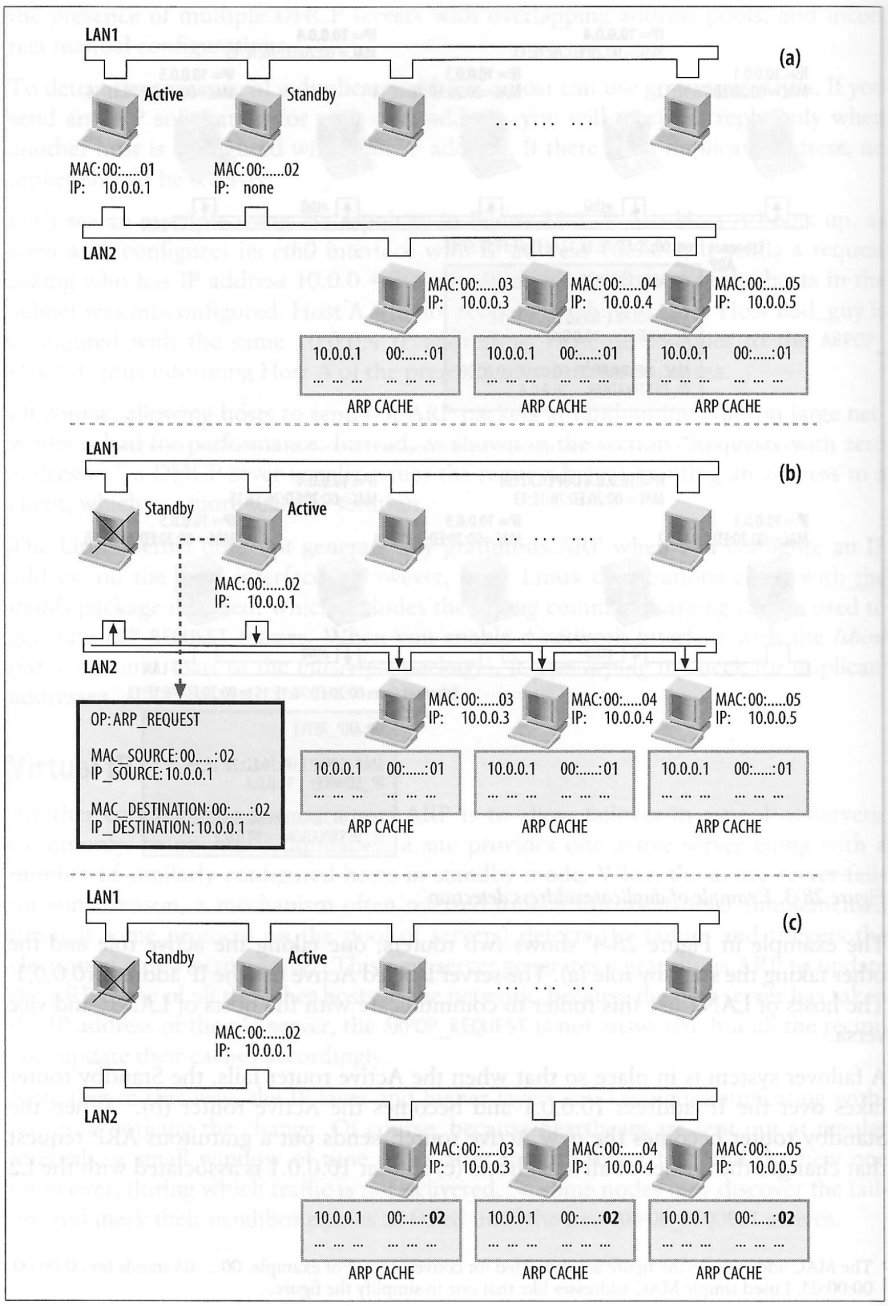


Figure 28-4. Example of gratuitous ARP

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

address of the new active router. Subsequent IP traffic from LAN2 comes to this router. The new Active router also sends a gratuitous ARP request to LAN1, but this is not shown in the figure. The figure also does not show another detail that a real-life administrator would configure: each router would have a second IP address on each of its interfaces, used mainly to provide connectivity when the current role is not active.

Responding from Multiple Interfaces

Linux has a rather unusual design: it considers an IP address as belonging to a host rather than an interface, even though administrators always assign IP addresses to particular interfaces.* This has impacts that some administrators complain about:

- A Linux host replies to any ARP solicitation requests that specify a target IP address configured on any of its interfaces, even if the request was received on this host by a different interface. To make Linux behave as if addresses belong to interfaces, administrators can use the ARP_IGNORE feature described later in the section "/proc Options."
- Hosts can experience the *ARP flux* problem, in which the wrong interface becomes associated with an L3 address. This problem is described in the text that follows.

Imagine you have a host with two NICs on the same LAN, and that another host sends an ARP request for one of the addresses. The request is received by both interfaces, as shown in Figure 28-5, and both interfaces reply.

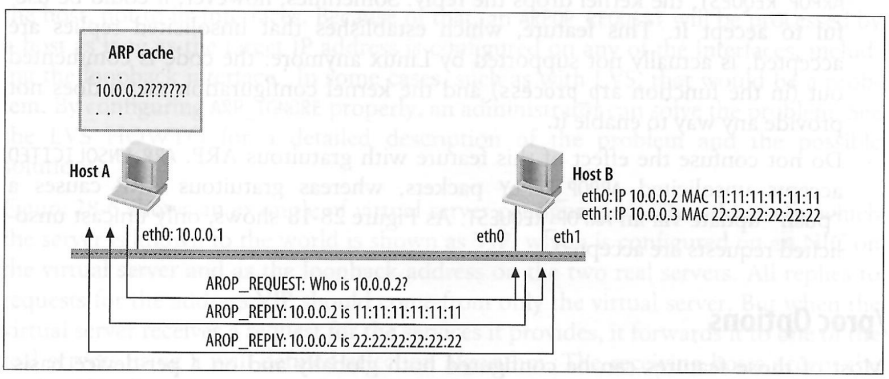


Figure 28-5. The ARP flux problem

* Using the options described in the section "Tunable ARP Options," you can make Linux behave as if IP addresses belonged to the interfaces. For an interesting discussion of this design, including its advantages and disadvantages, you can refer to the (pretty long) thread "ARP responds on all devices" on the *netdev* mailing list, which is archived at *http://oss.sgi.com/archives/netdev*.

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC

Linux

GUIDED TOUR TO NETWORKING ON LINUX

UNDERSTANDING LINUX NETWORK INTERNALS

Linux is popular partly because of its efficient and feature-rich network stack. If you've ever wondered how Linux carries out the complicated tasks assigned to it by the IP protocols—or just want to learn about modern networking through real-life examples—*Understanding Linux Network Internals* is your guide.

Like the popular O'Reilly book *Understanding the Linux Kernel*, this volume clearly explains basic network concepts and teaches you how to follow the actual C implementation code. Although some previous experience with TCP/IP protocols is helpful, you can learn a great deal from this text about the protocols and their many uses. Once you thoroughly understand these networking tools, you can then use the book's code walk-throughs to figure out exactly how the most sophisticated parts of the Linux kernel work.

Part of the difficulty in understanding networks—and implementing them—is that networking tasks are broken up and performed at many different times by different pieces of code. One of the strengths of this book is that it integrates the pieces and show you the relationships between far-flung functions and data structures. *Understanding Linux Network Internals* is both a big-picture discussion and a no-nonsense guide to the details of Linux networking.

Author **Christian Benvenuti**, an operating system designer who specializes in networking, explains much more than how Linux code works. He shows the purposes of major networking features, discusses trade-offs involved when choosing one solution over another, and includes numerous flowcharts and other diagrams to help bring the material into focus.

Topics in this book include:

- Key problems with networking
- Network interface card (NIC) device drivers
- System initialization
- Layer 2 (link-layer) tasks and implementation
- Layer 3 (IPv4) tasks and implementation
- Neighbor infrastructure and protocols (ARP)
- Bridging
- Routing
- ICMP

O'REILLY®
www.oreilly.com

ISBN 0-596-00255-6 US $49.95 CAN $69.95

9 780596 002558 6 36920 00255 0

Safari® BOOKS ONLINE ENABLED
Includes FREE 45-Day Online Edition

Exhibit 1062
Microsoft Corporation v. Edge Networking Systems, LLC