

Understanding Linux Networking

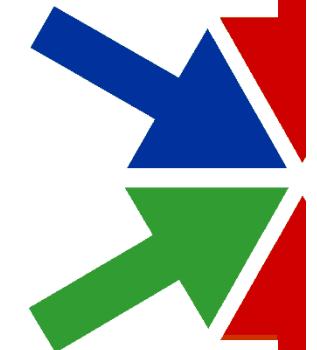
Lecture Book



Copyright © The PTR Group, Inc. 2007-2017
Linux is a registered trademark of Linus Torvalds

Understanding Linux Networking

Overview



Copyright 2007–2017,
The PTR Group, Inc.

Copyright Statement

Copyright © The PTR Group, Inc. 2007–2017

All trademarked product and company names are the property of their respective trademark holders

ALL RIGHTS RESERVED. No part of this publication may be reproduced in any form or by any means, electronic, mechanical, photocopy, recording, or otherwise, without the prior written permission of The PTR Group, Inc.

Welcome

- * The class is informal
- * Go ahead and ask questions at any time
- * Our goal is for you to walk out of here ready to apply your new-found knowledge
- * Remember this is your course and it will only be great if you help make it so

Course Structure

- ★ This course will consist of lecture periods followed by hands-on labs
- ★ The lecture notes are in your handbook
- ★ The labs are in your lab notebook
 - ▶ Some labs may be instructor driven
- ★ There are quizzes at the end of each chapter to make sure that you understand the material

“Administrivia”

- ★ General hours are ...
- ★ We will take periodic “bio” breaks
- ★ We will take one hour for lunch
 - ▶ Yes, the instructor will actually have to eat
- ★ Please, no audible cell phones/pagers
 - ▶ We would like your undivided attention
- ★ Please take a moment to fill out the registration sheet

Course Objectives

- ★ Learn about the networking features that Linux brings natively to the table
 - ▶ Addresses duplication of effort in creating a custom implementation of an existing protocol
- ★ Learn how to configure network security features such as firewalls
- ★ Learn to configure QoS, DNS and other network services
- ★ Understand the network programming APIs
- ★ Grasp the basics of dual-stack operation of IPv4 and IPv6
- ★ Understand the flow of packets through the Linux network stack

Course Overview – Day 1

- ★ Linux Networking Class Introduction
- ★ Overview of Linux networking and supported protocols
- ★ Networking device driver framework and packet management
- ★ Using dedicated networking hardware
- ★ NAPI Networking Driver and other network acceleration techniques
- ★ IPv4/IPv6 sockets APIs

Course Overview – Day 2

- ★ Implementing a new protocol and kernel socket backend
- ★ IP routing, quagga, netlink sockets
- ★ Linux Ethernet bonding
- ★ IP and Ethernet multicasting
- ★ Packet ingress and egress queuing model

Course Overview – Day 3

- Packet classification and Quality of Service (QoS)
- Firewalls, iptables and network filtering
- Ethernet bridging

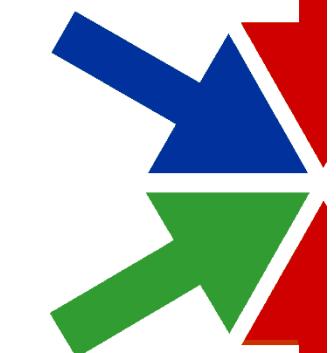
Course Overview – Day 4

- ★ Debugging network communications
- ★ Name resolution and service discovery
- ★ VLAN support
- ★ Network security

Chapter Break

Understanding Linux Networking

Overview of Linux Networking and Supported Protocols



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

★ Linux Network Highlights and Data Flow

★ Linux Network Protocols Overview

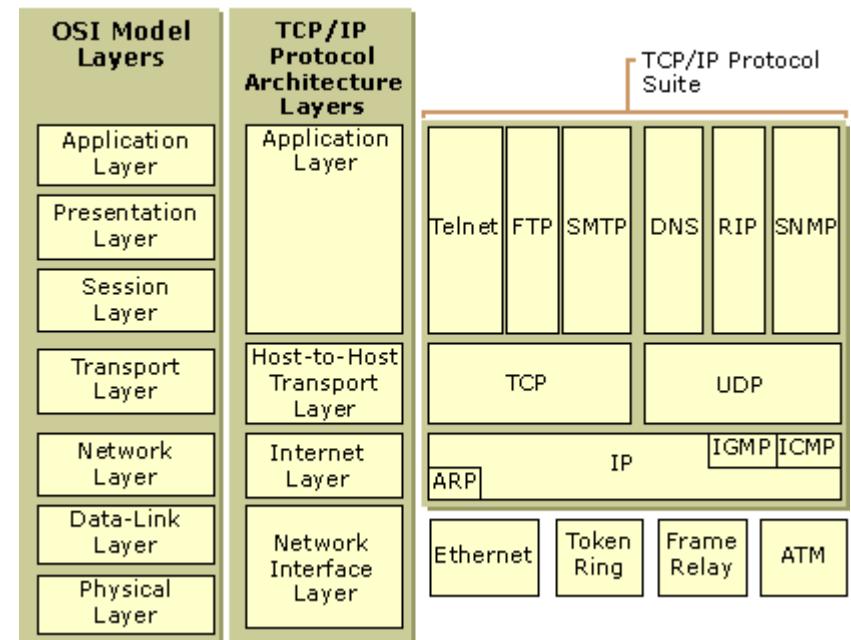
- ▶ Application Layer Protocols
 - Routing Protocols
- ▶ Transport Layer Protocols
- ▶ Internet Layer Protocols
- ▶ Link Layer Protocols

Linux Networking Overview

- ★ Linux is often the first target of a new protocol or of protocol research
 - ▶ Well understood and similar to the venerable BSD 4.4 stack -- available in source code
- ★ Modular network stack
 - ▶ Support for IPv4/IPv6 and several other protocols
- ★ Socket abstraction interface for applications
- ★ Supports most COTS networking hardware
 - ▶ Ethernet and wireless are both well-supported
- ★ Supports myriad protocols at all layers of the OSI model
- ★ Fully configurable firewall

Comparison of TCP/IP to OSI

- ★ The traditional ISO OSI protocol model doesn't map cleanly to TCP/IP
 - ▶ However, TCP/IP has won the battle for network supremacy
- ★ Still the OSI model is used to describe the various layers of a protocol solution
 - ▶ E.g., a layer-2 or layer-3 switch
- ★ We will use this same model to describe Linux's network facilities



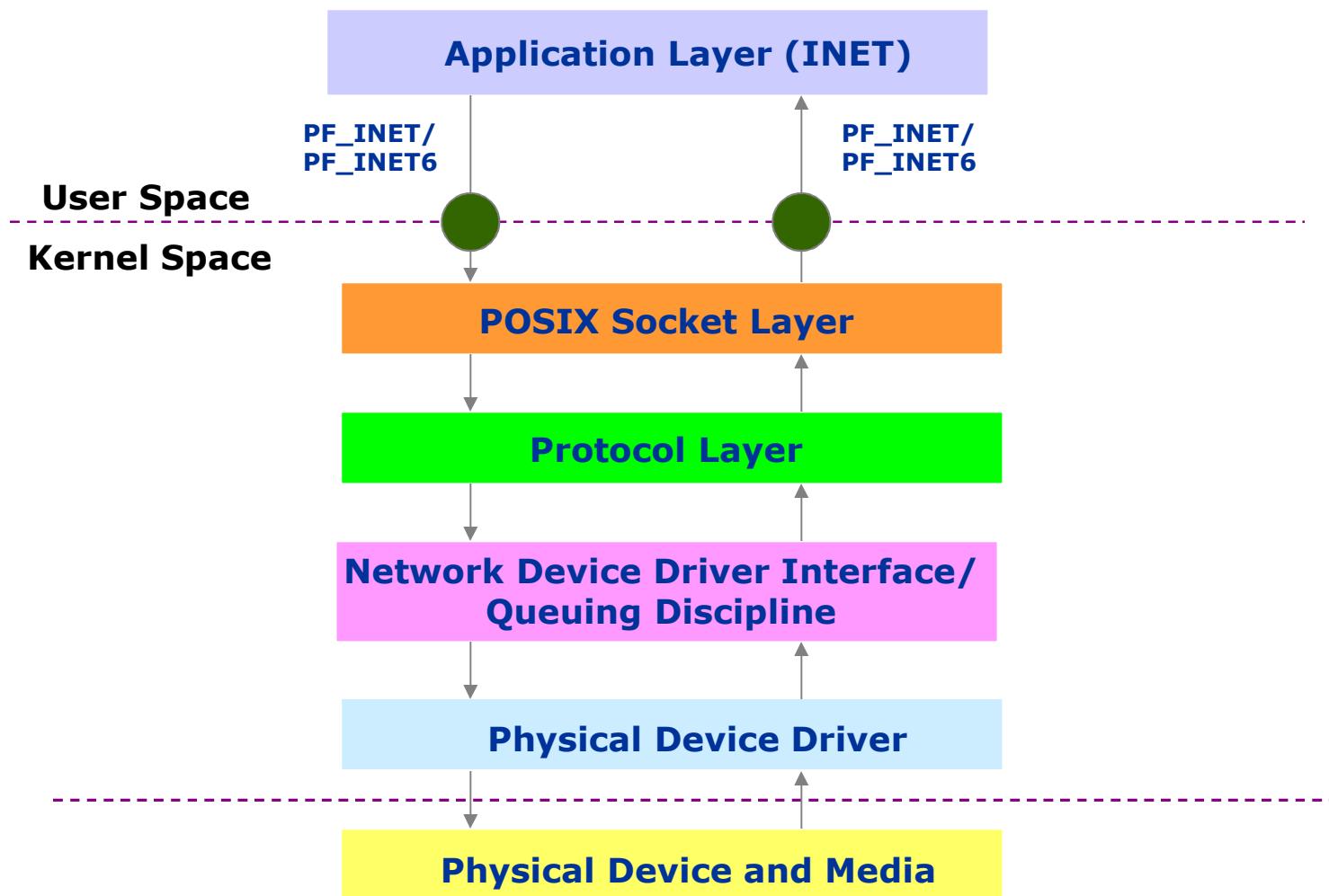
Source: proprofs.com

Terms used in this Material

★ To clarify terminology used in this class, we will use the following definitions:

- ▶ L2 – Data-link layer
 - E.g., Ethernet
- ▶ L3 – Network layer
 - IPv4/IPv6, TIPC, etc.
- ▶ L4 – Transport layer
 - TCP/UDP/ICMP/ICMPv6, etc.
- ▶ Ingress/input/RX/Rx
 - The receive side of the stack
- ▶ Egress/output/TX/Tx
 - The transmit side of the stack

Linux Network Data Flow



Linux-Supported Network Protocols

- ★ Linux supports a vast array of network protocols
- ★ We will briefly cover many of them here, grouped by “TCP/IP model” layers
 - ▶ Link Layer Protocols (L2)
 - ▶ Internet Layer Protocols (L3)
 - ▶ Transport Layer Protocols (L4)
 - ▶ Application Layer Protocols (L5+)
- ★ Note that some of the application layer protocols correspond to an application of the same name

L2–Related Protocols

★ Framing for physical interfaces such as:

- ▶ Ethernet, Infiniband, FDDI, USB, ISDN, IEEE 1394, IEEE 802.11abgn/ac/s, IEEE 802.15.4, CAN, IRDA, Bluetooth, WiMAX, NFC, X.25, ATM, AppleTalk, LAPB, and more

★ Additional L2–related protocols:

- ▶ MPLS, HSR, L2TP, PPP, ARP, RARP, IEEE 802.1d, IEEE 802.1Q/802.1ad (VLAN/VXLAN), IEEE 802.2 LLC, IEEE 802.3ad (LACP), B.A.T.M.A.N., Phonet, Ethernet multi-cast

★ In short, most of the L2 protocols available in the networking world

L3–Related Protocols

- ★ There is no shortage of L3–related protocols either
 - ▶ IPv4/IPv6, 6LoWPAN, DCCP, SCTP, TIPC, DECnet, IPX, ICMP, ICMPv6, IP multi-cast, MPLS
- ★ And helper protocols:
 - ▶ DCCP, RSVP/RSPV6, SNMP, NAT, IGMP, ECN, IPsec to name just a few
- ★ Essentially, if there is an RFC for the protocol and the protocol isn't proprietary, then there's likely a Linux version available

L4+-Related Protocols

★ At the application level, we have:

- ▶ SSH, POP, IMAP, SNMPv3, SMTP, TFTP, FTP, DNS/DNS6, HTTP/HTTPS, Telnet, NTPv4, NNTP, SIP, RDS, LDAP, MGCP, DHCP/DHCPv6, SLAAC, RTP, RTSP, XMPP, TLS/SSL, IKEv2, ISAKMP

★ Additionally, there is low-level socket support to allow you to create your own protocols

- ▶ Including raw socket support

Address Family Socket Support

- ★ AF_INET/AF_INET6
 - ▶ IPv4/IPv6
- ★ AF_TIPC
 - ▶ Ericsson's cluster protocol
- ★ AF_NETLINK
 - ▶ User-to-kernel sockets used to communicate from user space to kernel space and back as well as routing protocols
- ★ AF_PACKET
 - ▶ Low-level packet interface
- ★ AF_CAN
 - ▶ CAN bus protocol
- ★ AF_UNIX
 - ▶ Unix-domain sockets
- ★ AF_ALG
 - ▶ Interface to Kernel crypto API
- ★ Multiple others for older protocols like X.25, Appletalk, etc.

Routing Protocols

★ Distance–Vector oriented protocols

- ▶ RIPv1/RIPv2 (IPv4), RIPng (IPv6)

★ Link–state protocols

- ▶ OSPFv3, OSPF6, IS-IS, LSRPv2

★ BGPv4+

- ▶ Includes address family support for multicast and IPv6

★ LDP

- ▶ MPLS Label Distribution Protocol

★ BFD

- ▶ Bidirectional Forwarding Detection

NAT

- ✖ Linux supports both source and destination NAT
- ✖ L3 support for IP/IPv6 address rewriting
- ✖ L2 support for rewriting MAC addresses
- ✖ Can be problematic for some protocols like IPsec
 - ▶ Requires NAT traversal support

L2/L3/L4+ Firewalls

- ★ As we'll see later in the class, Linux supports packet filtering at L2, L3 and L4+ layers
- ★ The packet filter code provides visibility into the headers and potentially payload of the packet
 - ▶ You can create a L2/L3 “brouter”, stateless/stateful firewalls or pass the packet all the way to user space for proxy or application gateway usage

VPNs

- ★ Linux supports several different types of VPNs
- ★ TLS/SSL-based VPNs with interoperability with most major operating systems
- ★ IPsec-based VPNs including a Cisco-compatible VPN implementation
 - ▶ IPsec supporting both tunneling and transport mode
- ★ PPTP with MS-compatible encryption

Quality of Service (QoS)

- ★ Linux implements QoS support on both the ingress and egress sides of the stack
 - ▶ The kernel supports over 20 queuing disciplines with both classless and classful filtering options
- ★ Can be combined with the packet filter code to create sophisticated classification mechanisms
- ★ Allows for traffic policing, scheduling and traffic shaping to help reduce network congestion

Summary

- ★ Linux provides a full-featured, flexible, robust network environment
- ★ Tweaked over the years to provide optimal performance
 - ▶ Reviewed by several thousand developers worldwide including manufacturers like Cisco, Juniper and others
- ★ Broad protocol support throughout all layers
- ★ Standards-based API, allowing for the development of highly portable code

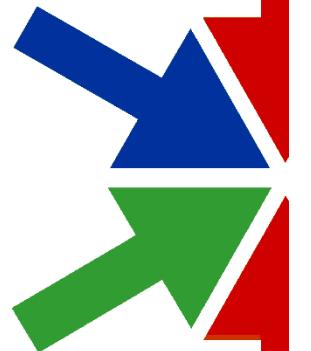
Questions

- * There is a direct, 1:1 correspondence between the layers in the TCP model and OSI network models
 - ▶ True or False?
- * User applications can use the socket abstraction for networking support
 - ▶ True or False?
- * Linux supports a Cisco-compatible VPN implementation
 - ▶ True or False?
- * Linux supports distance-vector routing protocols
 - ▶ True or False?
- * The AF_INET6 socket type can be used for low-level packet access in the kernel
 - ▶ True or False?

Chapter Break

Network Device Drivers

NETR



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ✖ Overview of the Linux device driver framework
- ✖ Network device driver details
 - ▶ What makes them different
 - ▶ Configuration
 - ▶ Internal structures
 - ▶ Logic flow
 - ▶ Interrupt handling
 - ▶ Common options
 - ▶ PHY interface types

Defining a Device

★ Devices can be physical or virtual

- ▶ Physical devices like serial ports and disk controllers
- ▶ Virtual devices like RAM disks and loopback network interfaces



Source: gearfuse.com

★ Physical devices are accessed through memory/port mapping

- ▶ Port mapping special I/O cycles on the bus
 - This may require auxiliary hardware such as bridge chips

Defining a Device Driver

- ★ A device driver is software that enables the device
 - ▶ Can be library-oriented access from user space
 - This is typical for RTOSes
 - ▶ Or, a formal device driver that runs in kernel space and uses an exported API
- ★ We will be focusing on formal device drivers in this class

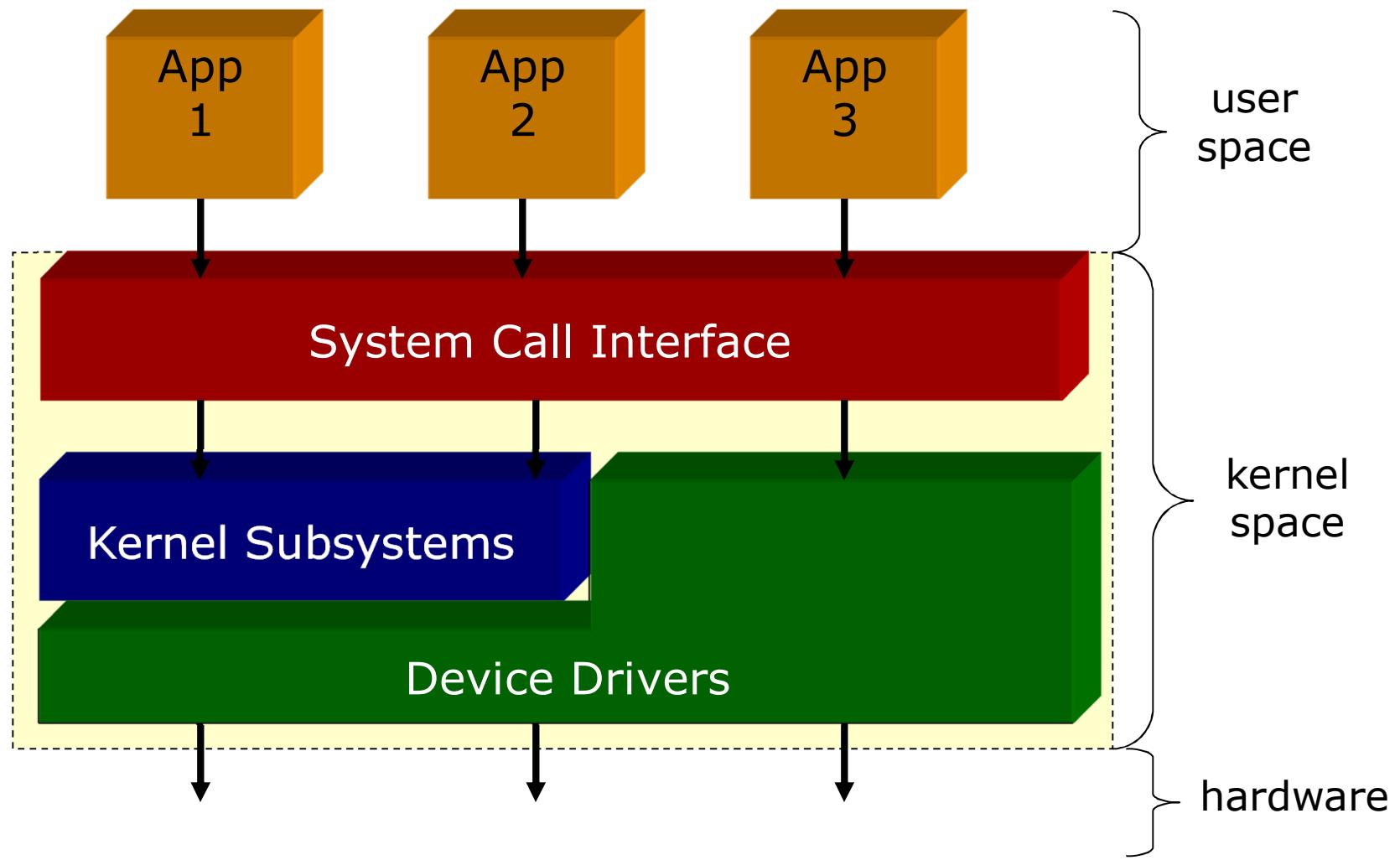


Source: <http://www.kadoee.com>

Static vs. Dynamically Loaded Driver

- ★ A driver can be statically linked into the kernel or dynamically loaded as a module
- ★ Statically-linked drivers are in <kernel>/drivers directory
 - ▶ They must be GPL code
- ★ Dynamically-loaded drivers
 - ▶ Loaded/unloaded with `insmod/rmmod` commands
 - ▶ Can have non-GPL licenses
- ★ No performance impact for dynamically-loaded drivers once they are loaded

How Things Fit Together



The `syscall()` Interface

- ★ `syscall()` is an assembly language interface between user and kernel space
 - ▶ Applications can pass 0–5 parameters to the kernel
- ★ In the 3.2.11 kernel, there are 272 system calls defined
 - ▶ Located in `include/asm-generic/unistd.h`
- ★ System call number identifies the service to be invoked

Data Flow

★ Data flow from user space to the hardware and back has consistent pattern:

- ▶ User application makes a system call
 - This causes a transition to supervisor state
- ▶ The kernel examines the call request
 - Invokes the appropriate service/driver
- ▶ The service/driver is responsible for data movement

Device Types and Linux

★ Linux knows about three primary device types

- ▶ Character-oriented devices
 - 80% of all devices in Linux
- ▶ File system or block-oriented devices
 - Fixed sized blocks of data
- ▶ Packet or network devices
 - Variable sized packets of data

★ Each type has unique API and requirements

★ We'll be discussing network device drivers in this chapter



Source: loc.gov

Network Device Drivers

- ★ Interface between hardware and network stack
- ★ Network devices are not directly accessed as files in the Linux I/O system, unlike Linux character and block devices

- ▶ There are no device nodes for network devices under /dev
 - E.g., no /dev/eth0



Source: microdesignrd.com

Network Driver Access

★ Network device drivers are accessed differently from other types of drivers:

- ▶ Configuration: configured using special network interface commands, such as `ip` and `ethtool`
- ▶ Output of data: data is output by the transmit logic of the network protocol stack
- ▶ Input of data: the reading of data is not initiated by commands from calling applications, as with some other types of drivers
 - I.e., network drivers tend to run asynchronously from user space
 - Received network data is “pushed” into the receive side of the network stack by the driver when it is received

Network Interface IP Address

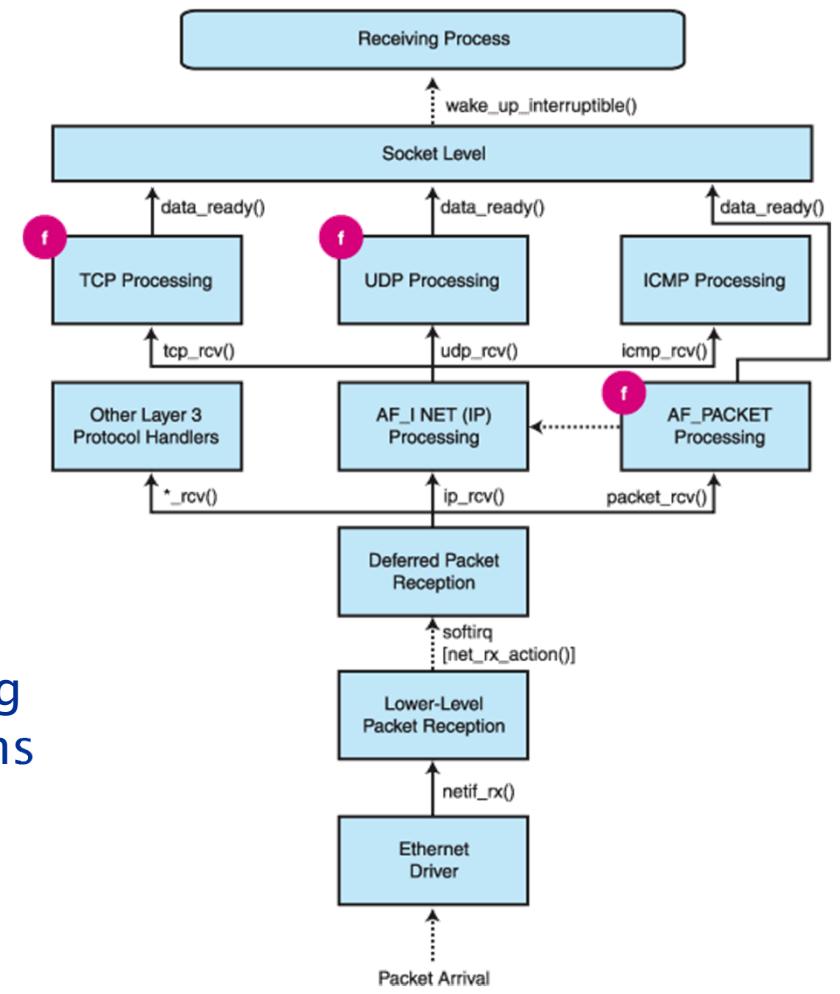
- ★ When a network driver initializes a network device, an interface is created for it
 - ▶ View current list of interfaces via “`ip link show`” or “`ifconfig -a`”
- ★ Set the interface IP address via `ip` or `ifconfig` commands
 - ▶ IP address only concerns the network stack logic, and how it handles packets for the interface
- ★ The driver doesn’t know its interface’s IP address, or even have a concept of an IP address



Source: ruddats.com

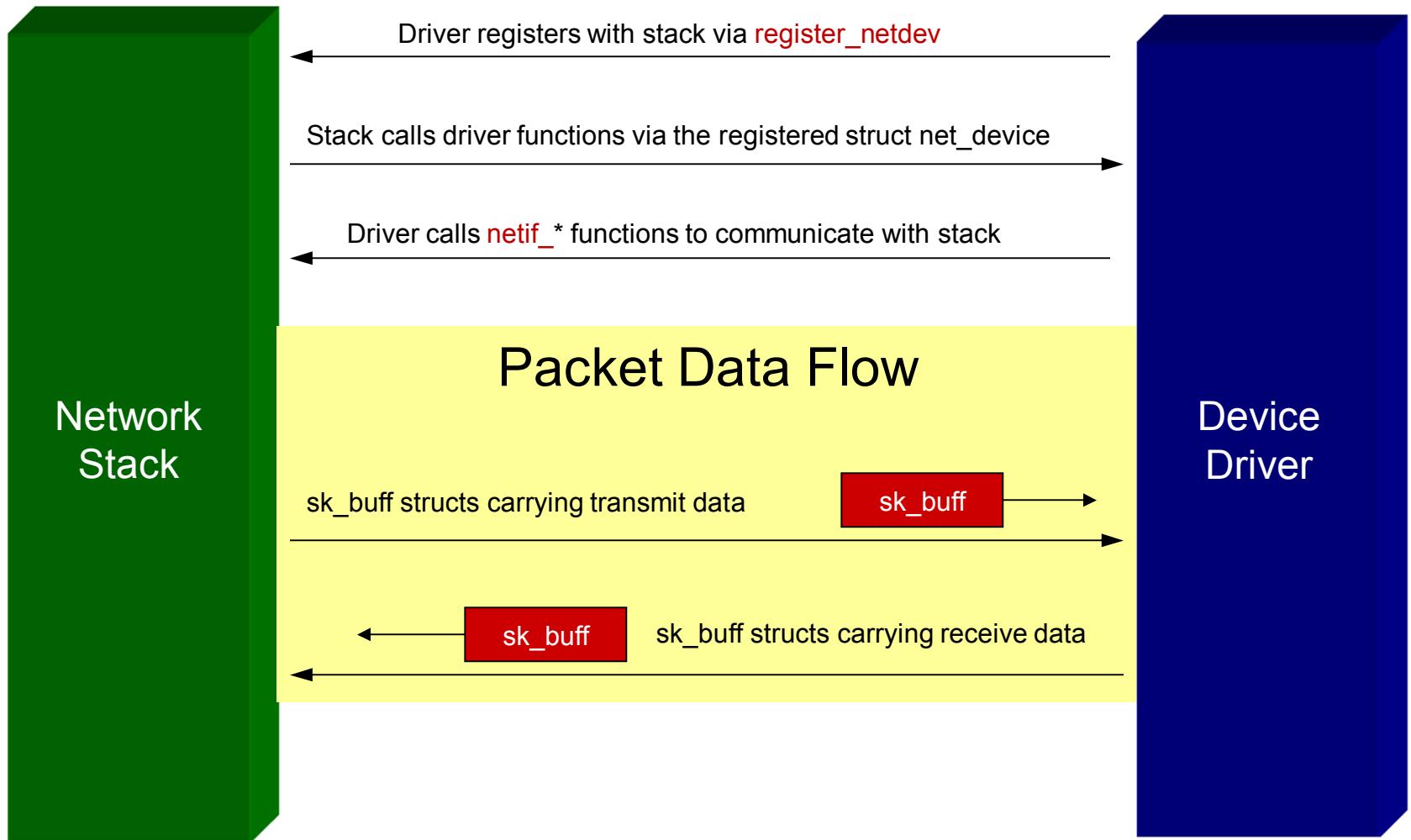
Linux Networking

- ✖ The Linux network stack is device-independent
- ✖ Linux provides a well-defined API to allow the various device drivers to register with (and communicate with) the network stack
- ✖ We will discuss:
 - ▶ Driver registration
 - ▶ Driver internal structures
 - ▶ Network stack functions provided for the network device driver to call
 - ▶ Socket buffers
 - ▶ Network driver logic flow highlights: Tx, Tx Timeout, Rx, interrupt handling
 - ▶ Common network device driver options
 - ▶ MII Interface and communicating with the transceiver hardware



Source: linuxjournal.com

Network Stack <-> Driver Interface



Driver Registration

- ★ Driver sets up its `struct net_device` then registers itself by calling `register_netdev()`
 - ▶ Must have `net_device_ops` structure configured
- ★ Network stack creates an interface for the driver upon registration
- ★ Call `unregister_netdev()` to remove the driver and registration

Network Device Structure

- * **struct net_device**
 - ▶ Used to describe the network device driver to the network stack
- * Call **alloc_etherdev()** to allocate space for the **net_device** struct, including any driver-specific private fields
 - ▶ This will silently call **ether_setup()** to establish Ethernet-like defaults
 - ▶ **alloc_netdev()** is the generic call
- * Use **free_netdev()** call to release allocated memory

```
void free_netdev(struct net_device *dev);
```
- * Driver fills in function pointers to driver routines in its **net_device** struct via the internal **net_device_ops** structure
- * In the **net_device** structure, set the **watchdog_timeo** field to the transmit timeout interval

struct net_device_ops Callbacks

* Common driver function pointers that are set in **net_device_ops** structure (not an exhaustive list):

▶ **ndo_init**

- Called once device is registered for late-stage initialization

▶ **ndo_uninit**

- Called when device is unregistered

▶ **ndo_open**

- Called when device transitions to the “up” state
- Allocates system resources for the driver and initializes network device hardware

▶ **ndo_stop**

- Called when device transitions to “down” state
- Reverses the actions of the “open” function

`struct net_device_ops` Callbacks (2)

▶ `ndo_start_xmit`

- Called when a packet needs to be transmitted
- Expected to start actual hardware data transmission

▶ `ndo_get_stats`

- Returns the device statistics to `ifconfig/ip/netstat`, etc.

▶ `ndo_do_ioctl`

- Perform device-specific ioctl requests

▶ `ndo_set_multicast_list`

- Configure MAC multicast list

▶ `ndo_tx_timeout`

- Callback when transmit times out

Example net_device_ops Init

```
static const struct net_device_ops tlan_netdev_ops = {
    .ndo_open                  = tlan_open,
    .ndo_stop                  = tlan_close,
    .ndo_start_xmit            = tlan_start_tx,
    .ndo_tx_timeout             = tlan_tx_timeout,
    .ndo_get_stats              = tlan_get_stats,
    .ndo_set_rx_mode            = tlan_set_multicast_list,
    .ndo_do_ioctl                = tlan_ioctl,
    .ndo_change_mtu              = eth_change_mtu,
    .ndo_set_mac_address          = eth_mac_addr,
    .ndo_validate_addr            = eth_validate_addr,
#endif CONFIG_NET_POLL_CONTROLLER
    .ndo_poll_controller         = tlan_poll,
#endif
};
```

- ★ This example shows the callback initialization for a TI Thunderlan Ethernet device
 - ▶ Includes some optional calls like changing the MTU and MAC address

`struct net_device_stats`

* Network driver uses this structure to collect statistics as packets are processed:

- ▶ `tx_packets`
- ▶ `tx_bytes`
- ▶ `tx_errors`
- ▶ `tx_fifo_errors`
- ▶ `tx_dropped`
- ▶ `tx_aborted_errors`
- ▶ `tx_carrier_errors`
- ▶ `tx_heartbeat_errors`
- ▶ `tx_window_errors`
- ▶ `collisions`
- ▶ `rx_packets`
- ▶ `rx_bytes`
- ▶ `rx_errors`
- ▶ `rx_fifo_errors`
- ▶ `rx_dropped`
- ▶ `rx_over_errors`
- ▶ `rx_length_errors`
- ▶ `rx_frame_errors`
- ▶ `rx_crc_errors`

`netif_*` routines

- ★ These are network stack functions provided for the network device driver to call
 - ▶ Used both for communications and for interface control
- ★ `netif_carrier_off`, `netif_carrier_on`
 - ▶ Informs the network stack about the link status
- ★ `netif_start_queue`
 - ▶ Informs the network stack that the driver is ready to begin transmission of data
 - ▶ Initializes the send/receive queues
- ★ `netif_rx`
 - ▶ Call this function to send an `sk_buff` containing received packet data up into the network stack

netif_* routines (2)

***netif_stop_queue**

- ▶ Tell network stack to stop the interface
- ▶ Do this when closing the interface, or when run out of room to add more transmit buffers and transmission of data must be stopped

***netif_wake_queue**

- ▶ Cause the network stack to recommence data transmission
- ▶ Does not reinitialize the queue

netif_* routines (3)

netif_device_present

- ▶ Use this function to query the network stack to find out if this device has been registered yet via `register_netdev`

netif_running

- ▶ Use this function to query the network stack to find out if the device has been “opened”

Socket Buffers – struct `sk_buff`

- ★ Descriptors that are used throughout the networking system to handle buffers of transmit and receive data
- ★ They are allocated from a slab cache to avoid spin locking
- ★ **`dev_alloc_skb`**
 - ▶ allocate an `sk_buff` to send received data up into network stack
- ★ **`dev_kfree_skb`**
 - ▶ free an `sk_buff` that is finished, e.g. when the data in a transmit `sk_buff` has been sent



Source: city-girl-electronics.com

struct sk_buff

★ include/linux/skbuff.h defines it:

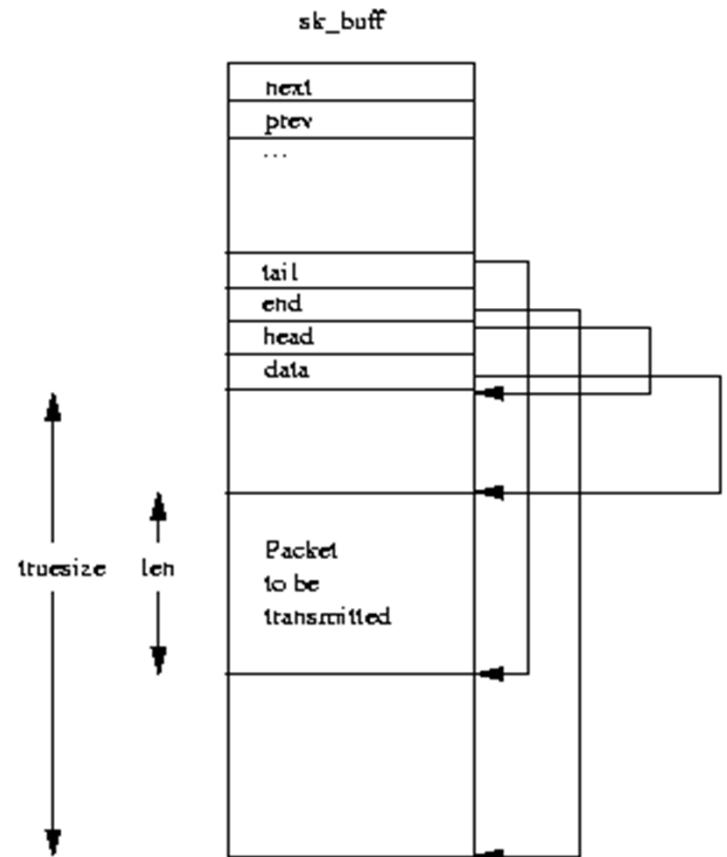
```
struct sk_buff {
    /* These two members must be first. */
    struct sk_buff *next;
    struct sk_buff *prev;

    ktime_t tstamp;

    struct sock *sk;
    struct net_device *dev;
    ...

    unsigned int len,
                data_len;
    __u16 mac_len,
          hdr_len;
    ...
    __be16 protocol;
    ...

    /* These elements must be at the end, see alloc_skb()
     * for details. */
    sk_buff_data_t transport_header;
    sk_buff_data_t network_header;
    sk_buff_data_t mac_header;
    sk_buff_data_t tail;
    sk_buff_data_t end;
    *head,
    *data;
    truesize;
    users;
};
```



Source: pku.edu.cn

Transmit Logic Flow

- ★ Network stack sends an `sk_buff` to driver using `ndo_start_xmit` function
- ★ Driver copies the packet data (or uses DMA) from the outgoing `sk_buff` to the device's descriptor ring (can use `sk_buff len` and `data` fields)
 - ▶ Device might be able to use the `sk_buff->data` pointer directly to avoid a copy
- ★ Driver frees the outgoing `sk_buff` using `dev_kfree_skb`
- ★ Driver updates the device's `net_device_stats` structure



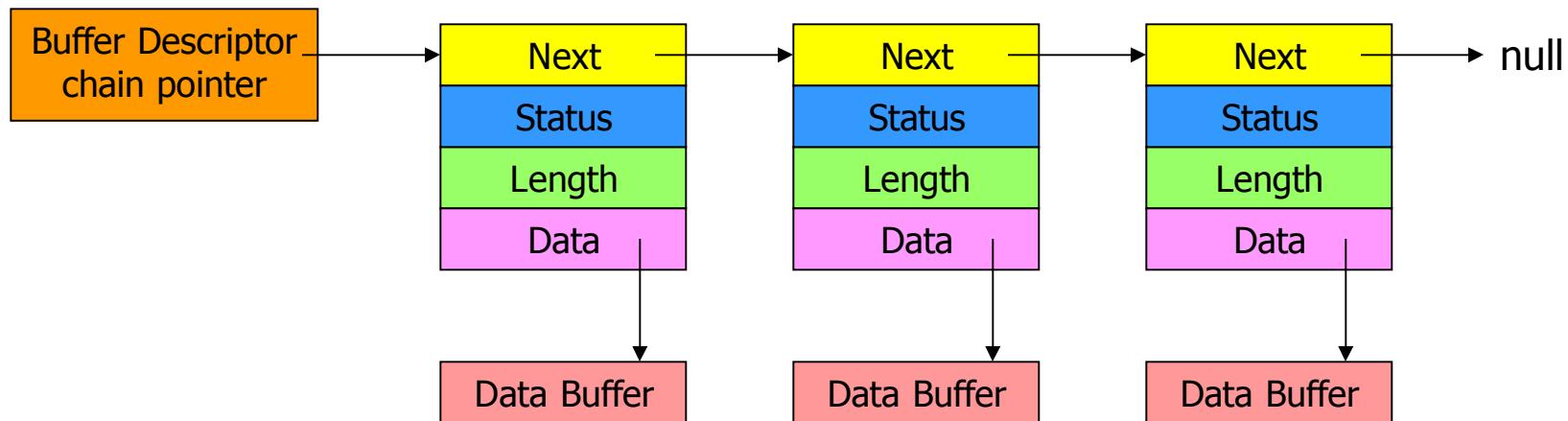
Source: flickr.com

Transmit Logic Flow (2)

- Driver issues hardware commands to the device to begin transmitting data
- If device's Tx descriptor ring is full, the driver calls `netif_stop_queue` to do flow control
- Driver sets the `trans_start` field in its `net_device` structure to the current “jiffies” timestamp
 - ▶ Supports the transmit timeout logic

Buffer Descriptor Diagram

- ★ Network drivers work with data as chains of buffer descriptors
 - ▶ Most network devices process input and output data using similar constructs
- ★ Another variant is for this to be a *ring* of descriptors



Transmit Timeout

* The driver's `ndo_tx_timeout` function is the transmit timeout callback

- ▶ The transmit timeout state indicates that “transmit complete” interrupts are not coming in
- ▶ This means we are not making room for further Tx data to be sent, so something is wrong

* The `ndo_tx_timeout` function resets hardware, updates statistics, and calls `netif_wake_queue` to restart data transmission



Source: zcache.com

Receive Logic Flow

- ★ Receive interrupt occurs and ISR is called
- ★ The ISR checks status and determines that data has been received
- ★ The ISR can either handle the reception of data directly, or schedule post-ISR work
 - ▶ In this case, the receive logic will run after interrupts are re-enabled



Source: rnnonline.org

Receive Logic Flow (2)

- ✖ Driver analyzes the received packet status, and statistics are updated based on packet “health”
- ✖ Driver allocates an `sk_buff` for the receive data using `dev_alloc_skb`
- ✖ Driver copies (or uses DMA) the received packet data into the `sk_buff`
 - ▶ In some cases, it is possible to use the data directly from its original receive buffer location (i.e., without a copy)

Receive Logic Flow (3)

- * It is common practice for the driver to call:

```
skb->protocol = eth_type_trans(skb, dev);
```

to set the **sk_buff** protocol field

- * `eth_type_trans()` also (internally) sets the `skb->pkt_type` field

Receive Logic Flow (4)

- * Driver sets the `skb->dev` field to the `net_device` pointer
- * The driver then sends the `sk_buff` up into the network stack using the `netif_rx()` function
- * The driver acknowledges the reception of data to the device hardware
- * Driver restores device descriptors and register settings to enable the reception of more data

Attaching IRQ

- * The network driver attaches its IRQ handler via `request_irq()`
- * Driver calls `free_irq()` when closing down the interface to release the IRQ

Interrupt Handling

★ High-level view of interrupt processing:

- ▶ Determine which interrupt(s) occurred
- ▶ Update driver statistics if error interrupts occurred
- ▶ “Tx complete” interrupt: Tx data space exists, so call `netif_wake_queue()` to tell the network stack to transmit more packets
- ▶ Rx interrupt: Handle received packets by calling Rx logic or scheduling the Rx bottom half

Network Driver Common Options

- Note that not all devices support all options
- Full duplex vs. half duplex
- Autonegotiated vs. hardwired speed and duplex
- 802.3x flow control/PAUSE frames
- Wake-on-LAN activity
- Software options for driver behavior (not hardware device settings per se):
 - ▶ Transmit timeout interval length
 - ▶ Max # events per interrupt
 - ▶ Promiscuous mode, multicast mode



Source: blogspot.com

MII Transceiver Interface

- ★ Many Ethernet devices use transceivers
 - ▶ Referred to as a PHY (physical network interface)
- ★ These may have a standard interface for communications
 - ▶ Examples:
 - MII (Media Independent Interface)
 - IEEE 802.3u
 - RMII (Reduced Media Independent Interface)
 - Half the pin count of MII PHYs
- ★ MII is a simple serial interface
 - ▶ Often implemented via “bit banging”
 - Sending/receiving a bit at a time on one GPIO line with another GPIO line serving as the MII data clock
- ★ This interface is used by the driver to negotiate link with the peer
 - ▶ Ensure that both the MAC hardware and transceiver agree on settings



Source: tradekorea.com

MII Transceiver Interface (2)

- ★ Common interface settings configured/ checked via the MII:
 - ▶ Link speed, 10/100/1000 Mbps
 - ▶ Link duplex, half/full
 - ▶ Link status
 - ▶ Link partner capabilities
 - ▶ Auto-negotiation advertising settings
 - ▶ Auto-negotiation completion status
- ★ The PHY driver is associated with the driver via the `phy_connect(...)` API
 - ▶ In most cases, you won't need to write any code if the PHY is already supported in Linux
- ★ User interface is via the `ethtool` command and the `phy_mii_ioctl(...)` call

Summary

- ★ Network device drivers interact with the Linux network stack, and are unlike other Linux drivers (char & block drivers)
- ★ There is a well-defined interface between the network stack and Linux network device drivers
- ★ Many Linux network device drivers exist, so when creating a new driver it is best to use an existing driver as a reference

Questions

- ★ Name three ways network drivers differ from other drivers
- ★ Many network devices are configured as Ethernets even if they aren't a physical Ethernet transport
 - ▶ True or False?
- ★ What is the “currency” of the network stack?
 - ▶ A. Packets
 - ▶ B. Network queues
 - ▶ C. sk_buffs
- ★ In Linux, the network protocol is separated from the transport mechanism
 - ▶ True or False?
- ★ Network device drivers are installed in the kernel using the same insmod/rmmod commands as character and block drivers
 - ▶ True or False?

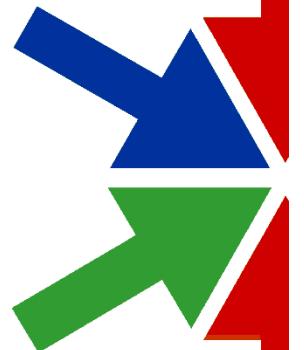
Chapter Break

Network Acceleration

Moving packets faster...



Source: gasgoo.com



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ❖ Protocol performance
- ❖ Freeing up CPU Cycles
- ❖ Types of TCP Offload
- ❖ TCP offload support in Linux
- ❖ NAPI
- ❖ Multi-core related acceleration techniques

Protocol Performance

- ★ TCP/IP was designed to perform well on the noisy, low-speed links found back in the 1970's
 - ▶ 1200 bps was *fast*
- ★ There are a number of checksums and other calculations that need to be done on a packet in IPv4
 - ▶ Some of these are eliminated in IPv6
- ★ General rule of thumb is that it takes 1Hz of CPU processing for every 1bit/s of TCP/IP traffic
 - ▶ E.g., 5 Gbits/sec would require 5 GHz of CPU processing
 - Full-duplex, 10 GigE would require eight 2.5 GHz cores!



Source: oftheising.com

Where are the Bottlenecks?

- ★ There are several bottlenecks in IPv4 packet processing
- ★ The 3-way connection establishment handshake
 - ▶ SYN – SYN/ACK – ACK
- ★ Checksums are required for the IP header, TCP and UDP checksums are across their respective headers and the payload
- ★ Sequence number calculations
- ★ Sliding window calculations and congestion control
- ★ Connection termination handshakes
 - ▶ FIN1 – ACK – FIN2 – ACK

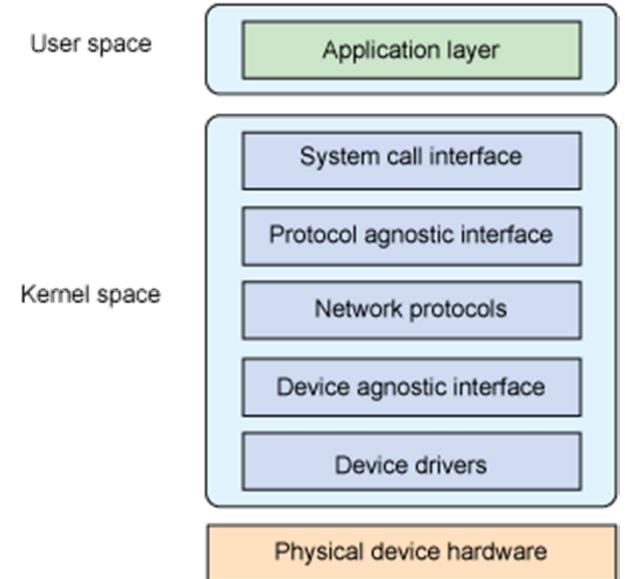
Improving Performance

- The checksums can easily be offloaded to the NIC
 - ▶ This is frequently done on 1GigE+ speed NICs
- The rest of the bottlenecks are a bit more problematic
 - ▶ The stack in Linux is implemented largely as a stateless engine
 - Offloading to hardware implies a coordination of the Linux stack to the tasks inside the hardware
 - Hence, a stateful implementation

TCP Offload Engines (TOE)

- ★ One approach to reducing the CPU load is to introduce a TCP offload engine to handle all of the TCP/IP processing overhead

- ★ An intelligent NIC with its own CPU/RAM and TCP processing
 - ▶ Linux supports the option of bringing in packets that bypass the normal protocol stacks



Source: ibm.com

TCP Offload Approaches

Parallel-stack, full offload

- ▶ There is a complete, parallel stack implemented in the hardware
- ▶ This approach uses a “vampire tap” between the application and transport layers to intercept packets and reroute them
- ▶ This is very controversial in the Linux stack



Source: dependable-mulch.com

HBA full offload

- ▶ Implemented for iSCSI processing
 - NIC looks like a SCSI disk interface instead of a network interface

TCP chimney partial offload

- ▶ The Linux stack maintains control of the connections, but the work is passed off to the NIC

TOE in the Kernel

* There are 3 primary examples of TOE in the kernel

- ▶ Chelsio, Qlogic and Broadcom NICs
 - Much of the code is supported as out-of-kernel patches

* There were attempts to make a generic TOE layer in 2007

- ▶ They were rejected as being too vendor specific

Fear and Loathing of TOE in Linux

- ★ Linux is arguably the most RFC-compliant protocol stack on the planet
 - ▶ TOE moves the implementation to the NIC
- ★ The arguments against TOE support contains various criticisms:
 - ▶ Security updates
 - The TOE engine firmware is proprietary
 - Therefore, Linux maintainers cannot address security issues
 - ▶ Point-in-time solution
 - Link speeds continue to increase
 - Today's great solution will be overcome by events within 6 months
 - ▶ Different network behavior
 - The TOE firmware will behave differently than the native stack
 - Can cause failures in network analysis tools that do not recognize the proprietary firmware's network timing and behaviors



Source: popartuk.com

Fear and Loathing of TOE in Linux (2)

▶ Performance

- Experience has shown that all of the additional programming required to support TOE handoffs with the kernel cause performance of protocols like HTTP to be below that of the Linux kernel itself

▶ Hardware-specific limits

- TOE NICs are more resource limited than the main Linux system
 - Very apparent under heavy load
- Subject to resource-based DOS attacks
 - Attacker simply exhausts the TOE card with SYN flooding attacks, etc.

▶ RFC compliance

- Linux kernel maintainers work diligently to keep Linux RFC compliant
 - Proprietary TOE firmware can go out of compliance and stay that way

Fear and Loathing of TOE in Linux (3)

▶ Linux features

- Linux has many QoS, netfilter and packet scheduling features that users depend on
 - The TOE firmware circumvents much of this control

▶ Poor user support

- The kernel maintainers answer to the users whereas many TOE NIC vendors have proven themselves less than responsive

▶ Long-term support

- One of the major plusses to Linux is its ability to run on old hardware
 - TOE vendors tend to end-of-life their support for older devices which would leave users hanging and force the kernel maintainers to try to support proprietary approaches

▶ Eliminates the global system view

- Linux has many tools that allow users to view, debug and implement networking policies
 - TOE NICs break the ability to understand the performance and policies from a system level

Upshot of TOE

- ★ TOE sounds like a good idea
 - ▶ But, its implementations have traditionally fallen short in terms of support and performance
- ★ Using the NIC's features for checksums is not viewed as a TOE implementation
 - ▶ Your driver can use these features as needed
- ★ If you feel that TOE is a potential solution, you are free to implement it with out-of-tree driver code
 - ▶ However, do not ever expect it to get accepted into the mainline kernel



Source: ironkatmoe.com

Improving Network Performance – NAPI

- ★ Beyond the normal TCP/IP bottlenecks, there is another issue with high packet rates in a Linux system
 - ▶ If each packet generates an interrupt, then higher packet rates are more disruptive to the system as a whole
- ★ The “New Applications Programming Interface” (NAPI) is an attempt to perform interrupt reduction and packet throttling at the driver level

Handling High-Speed Devices

- With 1GigE+ networking devices, more data must be processed asynchronously on the ingress path
- Traditionally, network drivers in Linux receive a hardware interrupt for notification of packet reception
- A burst of incoming network packets on a 1GigE+ interface can create large interrupt latencies and overhead in the system

Interrupts vs. Polling

- ★ There are different approaches the driver can use when receiving packets:
 - ▶ Interrupt Service Routine (ISR) called for each packet
 - Process packet, passing it up the stack
 - Best for tens to hundreds of packets per second
 - ▶ Driver periodically polls the device for any available packets
 - No interrupts needed
 - Best suited for thousands+ of packets per second

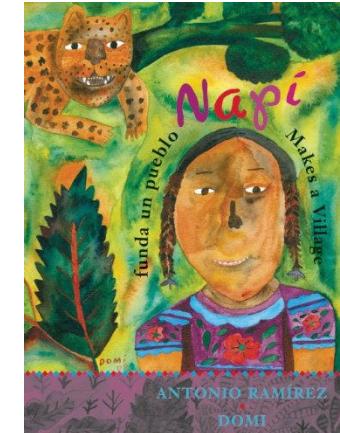


Source: theapplepeeled.com

- ★ NAPI is a blend of these two approaches

NAPI Driver Background

- ★ NAPI is a modification to the legacy networking device driver packet processing framework
- ★ Designed to improve the performance of higher speed networking devices using the following theory:
 - ▶ High packet–arrival rates can create thousands of interrupts per second
 - Given packet processing times, many of these notifications are redundant – the driver is currently processing the receive queue when successive interrupts arrive
 - ▶ NAPI allows drivers to run with interrupts disabled (in polling mode) during times of high traffic
 - Reduces overhead and latency
 - When all packets have been processed, driver returns back to interrupt mode



Source: latinbabybookclub.com

NAPI Driver Background (2)

- ★ NAPI drivers also support packet throttling
 - ▶ When system gets overloaded in extremely high traffic situations, NAPI driver will drop packets before they have a chance to traverse the stack
- ★ NAPI driver framework is entirely optional
 - ▶ A driver writer can still use the older Linux 2.4 framework for interfacing to the network stack
- ★ NAPI additions to the kernel do not break backwards compatibility
- ★ NAPI driver framework first appeared in Linux 2.5.x circa 2001–2002
 - ▶ Was also backported to Linux 2.4.20

NAPI Theory of Operation

★ A well-written, NAPI-compliant driver uses the following scheme:

- ▶ For the first RX packet, an interrupt is generated
- ▶ Per-packet RX interrupts from the device are then disabled
- ▶ The driver processes all available RX packets in polling mode
- ▶ When all packets have been processed, return device to interrupt mode



Source: white.se

NAPI Poll Function

*The poll function has the following prototype:

```
int (*poll) (struct napi_struct *napi, int budget);
```

- napi – a pointer to the NAPI context
- budget – number of packets the driver is allowed to pass into the network stack on this call
- return value – the number of processed packets

NAPI Poll Function (2)

★ The poll function should process all available incoming packets until the budget/quota has been reached

- ▶ For each packet:
 - Allocate a new `sk_buff`
 - Assign the packet's protocol type to `skb->protocol`
 - Send the packet up the stack with `napi_gro_receive()`
 - Lower performance interfaces (< 1G) may use `netif_receive_skb()`
- ▶ If all available packets have been processed (`pkt_processed_number < budget`)
 - Mark NAPI processing as complete by calling `napi_complete()`
 - Re-enable RX interrupts

Generic Receive Offload (GRO)

- Reduces the CPU overhead from delivering high volumes of incoming packets up the stack
 - ▶ Coalesces packets from the same flow into larger packets prior to sending them up the stack
 - ▶ Mimics receiving packets with a much larger MTU
 - ▶ Recommended for gigabit or faster interfaces
 - Especially for CPU constrained systems
- Use with NAPI via `napi_gro_receive()`
- Available since 2.6.29 kernel
 - ▶ 31 of 104 Ethernet drivers in 3.2 use GRO



Source: urbanorganicgardner.com

Leveraging the NIC and Multi-Core

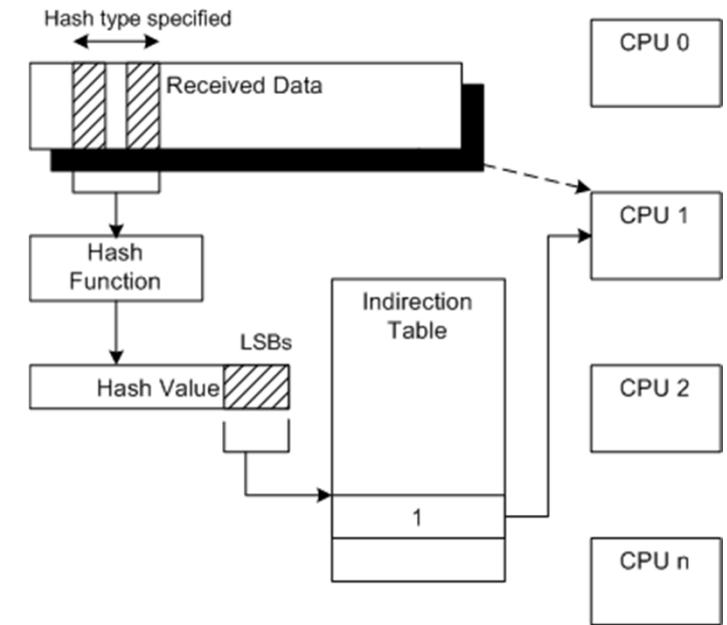
- ★ Many of today's modern NICs support additional hardware to speed packet processing
 - ▶ Focuses on moving packets rather than protocol processing
- ★ When running on a multi-core platform, there are some additional optimizations that we can apply to increase performance
 - ▶ Cache utilization, locality of reference, etc.

Scaling in the Linux Network Stack

- ★ We've just examined NAPI as a technique to improve frame performance
 - ▶ However, NAPI needs some help in a multi-core environment
- ★ Later kernels (2.6.36+) implement a set of complementary techniques to increase performance for multi-processor systems
 - ▶ RSS: Receive Side Scaling
 - ▶ RPS: Receive Packet Steering
 - ▶ RFS: Receive Flow Steering
 - ▶ Accelerated RFS
 - ▶ XPS: Transmit Packet Steering

Receive Side Scaling

- ★ Modern NICs support multiple RX and TX descriptor queues
- ★ On RX, a NIC can send different packets to different queues to distribute the load across CPUs
 - ▶ The NIC distributes packets by applying a filter to each packet that assigned the packet to a logical flow
 - ▶ Packets for each flow are steered to a separate RX queue processed by separate CPUs
- ★ Multi-queue can also be used by traffic prioritization, but that is not the focus of RSS



Source: microsoft.com

Receive Side Scaling (2)

- ★ The filter is typically a hash function over the network and/or transport layer header
 - ▶ Example: a 4-tuple hash over IP address and TCP port of a packet
- ★ Common hardware implementation of RSS uses a 128-entry indirection table where each entry stores a queue number
 - ▶ The receive queue is determined by masking the low-order 7 bits of a Toeplitz hash and using this number as the key in the indirection table
- ★ Some advanced NICs allow steering packets to queues based on programmable filters
 - ▶ E.g., webserver-bound TCP port 80 packets are directed to their own queue
- ★ Filters can be configured from **ethtool --config-ntuple**

Receive Packet Steering

- ★ RPS is a software implementation of RSS
 - ▶ Whereas RSS picks the queue (and hence the CPU) for the hardware ISR, RPS selects the CPU to perform the processing after the ISR completes
- ★ Accomplished by packing the packet on a “backlog queue” and waking up the CPU for processing
 - ▶ RPS is called during the bottom half of the RX ISR when the driver sends the packet up the stack with `netif_rx()`/`netif_receive_skb()`
- ★ RX queue is determined via a 2- (port) or 4-tuple (IP address) Toeplitz hash
 - ▶ Each RX queue has a list of associated CPUs which can handle the processing



Source: blogspot.com

Receive Packet Steering (2)

- ★ Kernel must be compiled with CONFIG_RPS config symbol (default for SMP)
 - ▶ RPS is disabled by default
- ★ CPUs used in RPS can be configured via
`/sys/class/net/<dev>/queues/rx-<n>/rps_cpus`
 - ▶ This is a bitmap of the CPUs for this RX queue
- ★ For high interrupt devices, the CPU that handles the ISR should be excluded from processing the queue
 - ▶ Use ISR affinity to lock the ISR to a CPU

Receive Flow Steering

- ★ RPS steers packets based solely on the hash and results in generally good load distribution
 - ▶ However, it does not take application locality into account
- ★ RFS seeks to increase data cache hit rate by steering packet processing to the CPU that is running the thread that is consuming the packets
- ★ The computed hash is used as an index into a table that indicates the CPU index that last processed the flow

Receive Flow Steering (2)

- ★ Requires the CONFIG_RFS kernel option
- ★ Remains disabled until configured via
`/proc/sys/net/core/rps_sock_flow_entries`
- ★ The number of entries in the per-queue flow
are set through:
`/sys/class/net/<dev>/queues/rx-<n>/rps_flow_cnt`
- ★ The suggested flow count depends on the
expected number of active connections at
any given time
 - ▶ A value of 32768 for `rps_sock_flow_entries`
works well for a moderately-loaded server

Accelerated RFS

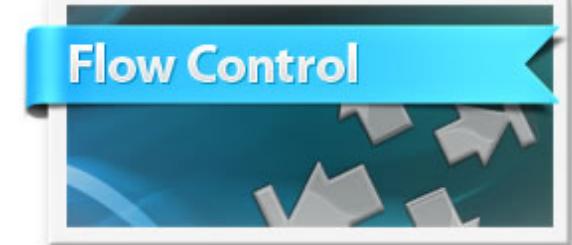
* ARFS is to RFS what RSS is to RPS

- ▶ A hardware-accelerated load balancing mechanism that uses a soft state to steer flows to the CPU where the application is running

* The networking stack calls

`ndo_rx_flow_steer()` to communicate the desired hardware queue for packets that match a particular flow

- ▶ The hardware queue for a flow is derived from the CPU recorded in the `rps_dev_flow_table`



Source: mydevelopersgames.com

Accelerated RFS (2)

- ★ This option requires the CONFIG_RFS_ACCEL kernel option and hardware that supports hardware queueing acceleration
- ★ IRQ affinity is used to create reverse maps of IRQs to CPUs to handle the automatic queuing of packets to the correct CPU
- ★ Requires n-tuple filtering be enabled via ethtool

Transmit Packet Steering (XPS)

- ★ XPS is mechanism for selecting which TX queue to use when transmitting a packet on a multi-queue device
 - ▶ Queues are assigned to a subset of CPUs involved in the flow
 - Reduces TX lock contention and increases cache hit rate for data
- ★ Configured via:
`/sys/class/net/<dev>/queues/tx-<n>/xps-cpus`
- ★ For a NIC with a single TX queue, XPS has no effect
 - ▶ For a multi-queue device, each CPU should map to its own queue
 - ▶ A 1-1 mapping results in no TX contention

Summary

- ★ It is possible to use the NIC services to speed basic protocol processing such as checksum calculations
 - ▶ However, the actual use of TCP Offload Engines, while possible, is discouraged
- ★ NAPI is the New API for high-speed Linux network drivers
- ★ NAPI reduces the number of interrupts that must be processed for a device receiving packets at a high rate
 - ▶ NAPI can discard packets at the hardware to reduce chances of over-running the kernel
- ★ The use of the GRO feature in the kernel allows multiple packets from the same stream to be combined into a larger packet to reduce protocol processing overhead
- ★ NIC Tx/Rx acceleration can also be leveraged in conjunction with cache utilization and locality of reference assignments

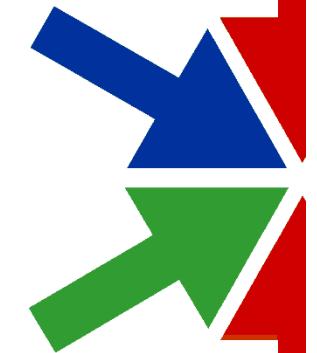
Questions

- ★ Checksum calculations are a significant bottleneck in TCP/IP protocol processing
 - ▶ True or False?
- ★ TCP offload engines are a commonly used feature of the Linux kernel
 - ▶ True or False?
- ★ NAPI addresses frequent RX interrupts by turning them off after the first interrupt of a burst
 - ▶ True or False?
- ★ What is the name of the hashing algorithm used in Receive Packet Steering mode?
 - A. Toeplitz Hash
 - B. Tirpitz Hash
 - C. Toejam Hash
- ★ XPS has no effect for a NIC with a single TX queue
 - ▶ True or False?

Chapter Break

The Socket API

Getting your data on the ‘net



What We Will Cover

- ❖ Network administration
- ❖ Sockets API
- ❖ Movement to IPv6
- ❖ IPv6 addressing and address assignment
- ❖ Dual-stack operation

Big Picture

- ★ The Linux network stack is very robust and reliable thanks to years of testing in the open-source community
- ★ Users see a socket interface for network communications
 - ▶ Multiple address families
- ★ With the exhaustion of the IPv4 address space, IPv6 and its variants like 6LoWPAN are becoming increasingly important
- ★ However, making applications support both IPv4 and IPv6 simultaneously requires a bit of work



Source: yves-in-a-box.deviantart.com

Network Administration

- ★ Network interfaces in Linux have administrative interfaces in the form of the **ethtool**, **ifconfig**, **route**, **arp**, **netstat**, **ss** and **ip** commands
 - ▶ These address features at the data link, network and transport layers
- ★ **Ifconfig**, **route**, **arp** and **netstat** are now deprecated
 - ▶ The iproute2 commands (**ip**) are now the preferred syntax
 - ▶ However, the older applications and syntax are still in wide use
- ★ Most of the network administration tasks require root privileges
 - ▶ Often, network interfaces are configured during the boot sequence
- ★ Support for automatic address assignment is an important feature

Interface IP Addressing

* The IP address identifies a machine on a network

- ▶ Only the network stack logic is concerned with the IP address
- ▶ Remember, the driver *never knows* an interface's IP address
 - It has no concept of an IP address

* Examples

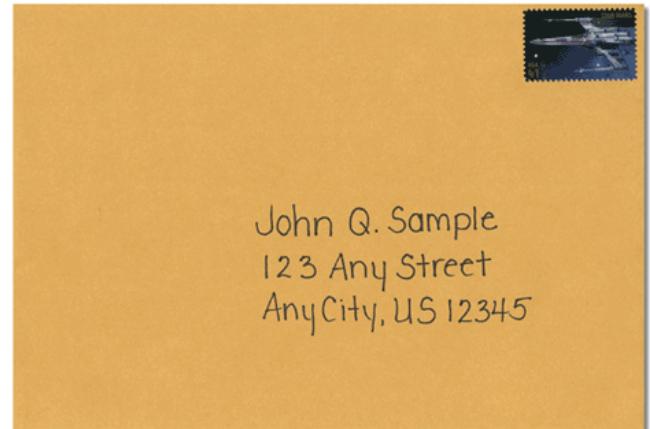
▶ Configure and activate eth0

```
# ip addr add 192.168.1.1/24 broadcast 192.168.1.255 dev eth0
      or
# ifconfig eth0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255 up

# ip -6 addr add 2001:0db8:0:f101::1/64 dev eth0
      or
# ifconfig eth0 inet6 add 2001:0db8:0:f101::1/64
```

▶ Shutdown the eth0 interface

```
# ip link set eth0 down
      or
# ifconfig eth0 down
```



Source: writeonresults.com

Multi-homing

★ Single link with multiple IP addresses

```
# ip addr add 192.168.1.1/24 broadcast 192.168.1.255 dev eth0
# ip -6 addr add fec0::1/64 dev eth0
# ip -6 addr add fe80::1/64 dev eth0
# ip -6 addr add 1:1:1:1:1:1:1:1/64 dev eth0
# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether d4:be:d9:6c:9e:b0 brd ff:ff:ff:ff:ff:ff
        inet 192.168.1.1/24 brd 192.168.1.255 scope global eth0
            inet6 1:1:1:1:1:1:1:1/64 scope global
                valid_lft forever preferred_lft forever
            inet6 fec0::1/64 scope site
                valid_lft forever preferred_lft forever
            inet6 fe80::1/64 scope link
                valid_lft forever preferred_lft forever
            inet6 fe80::d6be:d9ff:fe6c:9eb0/64 scope link
                valid_lft forever preferred_lft forever
```

Deprecated vs. New Commands

* Now that iproute2 commands are preferred, here is a quick summary of the old command and its modern equivalent:

Old Command	New Command
• arp	ip n, ip neighbor
• ifconfig	ip a, ip link, ip -s link
• iptunnel	ip tunnel
• iwconfig	iw
• netstat	ss, ip route, ip maddr
• route	ip r, ip route
• nameif	ip link set dev <if> name <newname>

Viewing Current Network Status

★ The **ss** command can be used to display network connections, open ports, and extended socket statistics

► E.g., list all open TCP sockets :

```
# > ss -t -a
```

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
LISTEN	0	128	127.0.0.1:17600	*:*
LISTEN	0	128	*:39649	*:*
LISTEN	0	64	*:nfs	*:*
LISTEN	0	1	127.0.0.1:openvpn	*:*
LISTEN	0	50	127.0.0.1:mysql	*:*
...				
ESTAB	0	0	192.168.101.8:46478	192.155.48.48:http
CLOSE-WAIT	38	0	192.168.101.8:34333	54.192.54.115:https
ESTAB	0	0	192.168.101.8:58373	199.16.156.120:https
ESTAB	0	0	127.0.0.1:53866	127.0.0.1:openvpn
...				

Sockets

- ★ The term “socket” came from BSD Unix when they put forth the idea that network communications should look like file I/O
- ★ A network endpoint associated with a file descriptor
 - ▶ Provides a bidirectional communications path
- ★ Read/write operations are performed on a socket as though it were a file
 - ▶ Files have names so you can open them... network connections do not
 - You may have to “bind a name” to a socket so it can be opened

Ports

- ★ Abstract destination point for network traffic
- ★ Identified by a short (16 bit) integer
 - ▶ As such, port numbers range from 1 to 65,535
 - Port numbers below 1024 are reserved for system services
 - Known as *reserved ports*
- ★ Many port numbers are associated with familiar services like HTTP or Telnet
 - ▶ See `/etc/services` for a (nearly) complete list of standard port numbers

IPv4 Socket Address

- ★ 6-byte address composed of an IP address and port number
 - ▶ This combination forms an address that is unique for a given socket
 - Also called the socket's "name"
- ★ These are seen as `sin_port` and `sin_addr` in the address structure below, from `/usr/include/netinet/in.h`

```
/* Socket address, internet style. */
struct sockaddr_in {
    short          sin_family;
    u_short        sin_port;
    struct in_addr sin_addr;
    char           sin_zero[8];
};
```

Types of Socket Communications

★ Typically, sockets are treated as either stream or datagram-oriented endpoints

▶ Streams are like virtual circuits

- E.g., a telephone call
 - There are set-up, communication, and tear-down phases

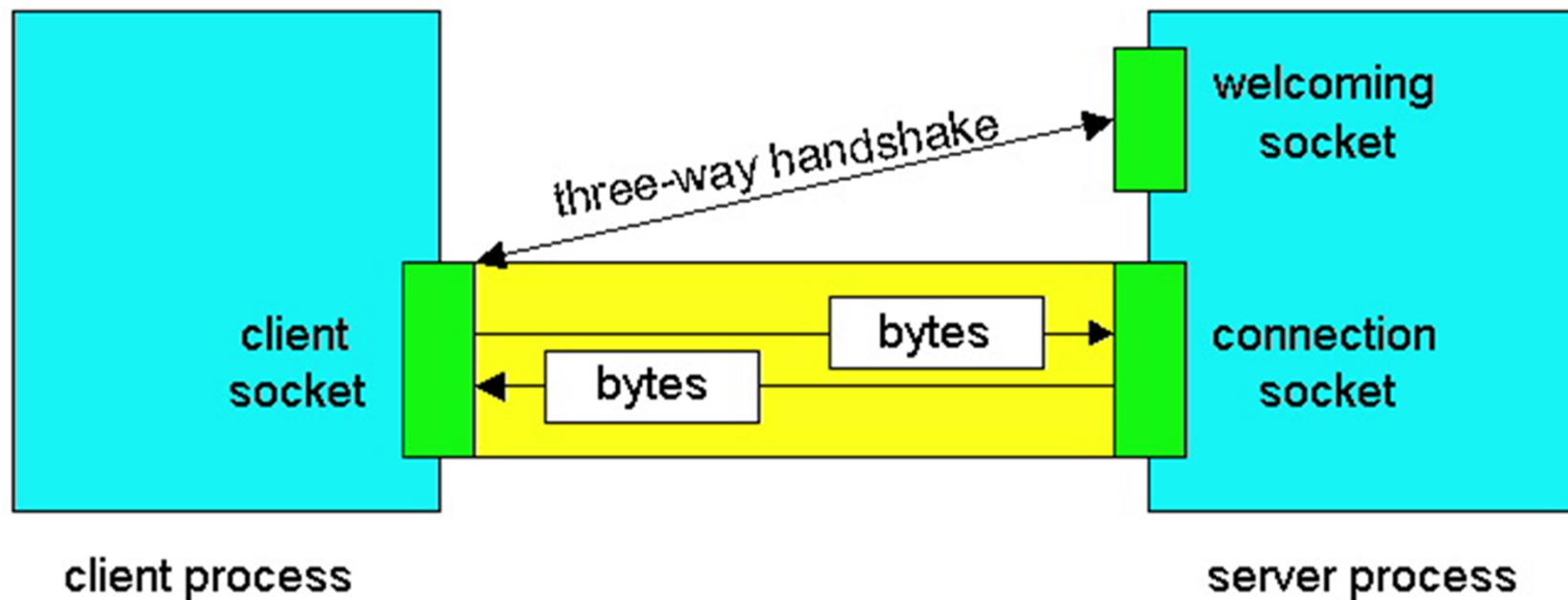
▶ Datagrams are like snail-mail letters

- No guarantee that they will arrive, or in the order they were sent
- Datagrams are a *best effort* delivery mechanism

Inter-Process Communication

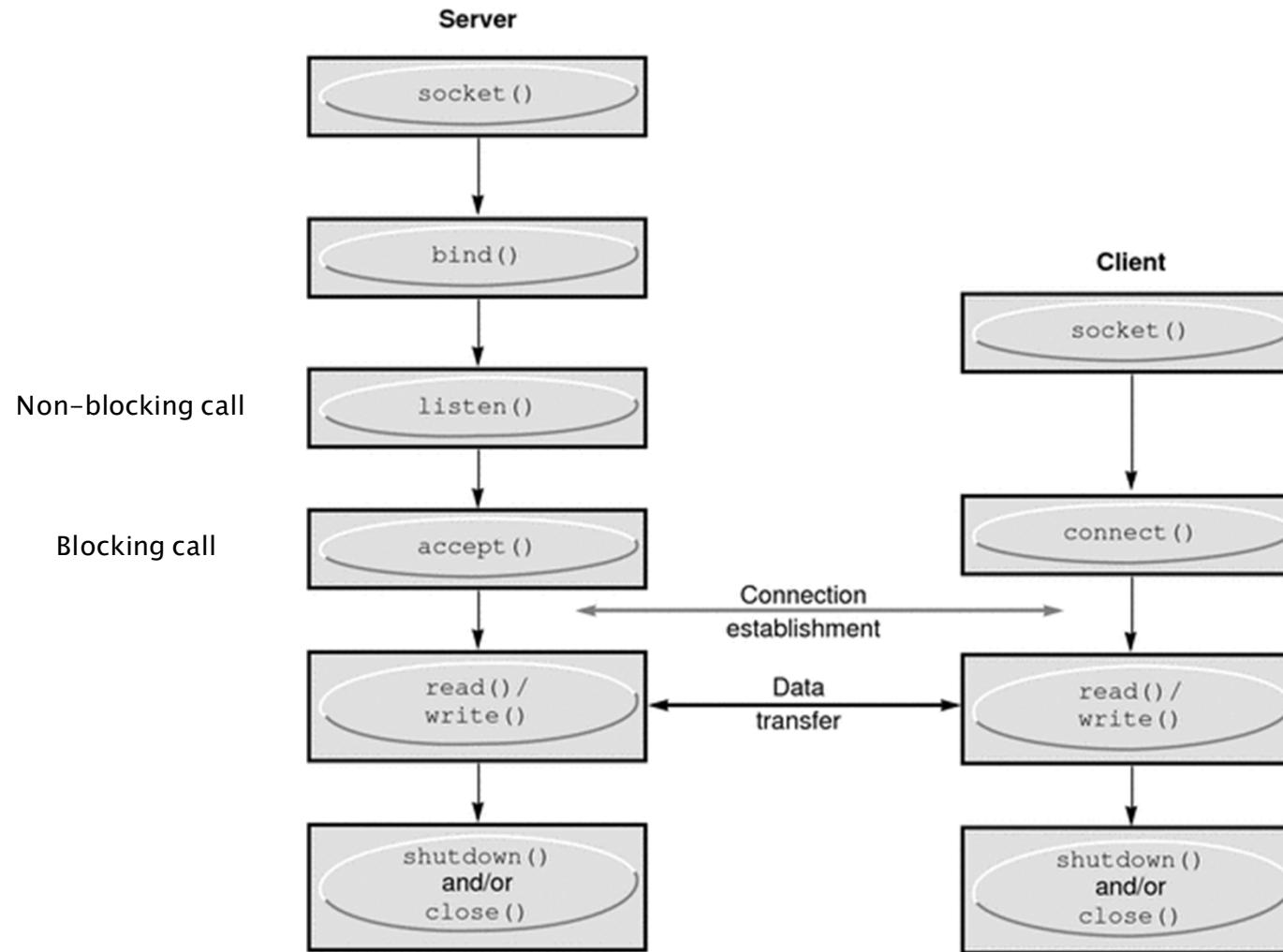
- ❖ Sockets can also be used as an inter-process communication (IPC) mechanism
- ❖ Linux supports Unix-domain, TIPC, netlink, CAN and Internet sockets among others
 - ▶ The type of socket you choose depends on the connectivity that you are trying to achieve
 - Unix Domain sockets work within a single machine
 - TIPC sockets are LAN only
 - Internet sockets work on both LAN and WAN

Typical Client-Server Application



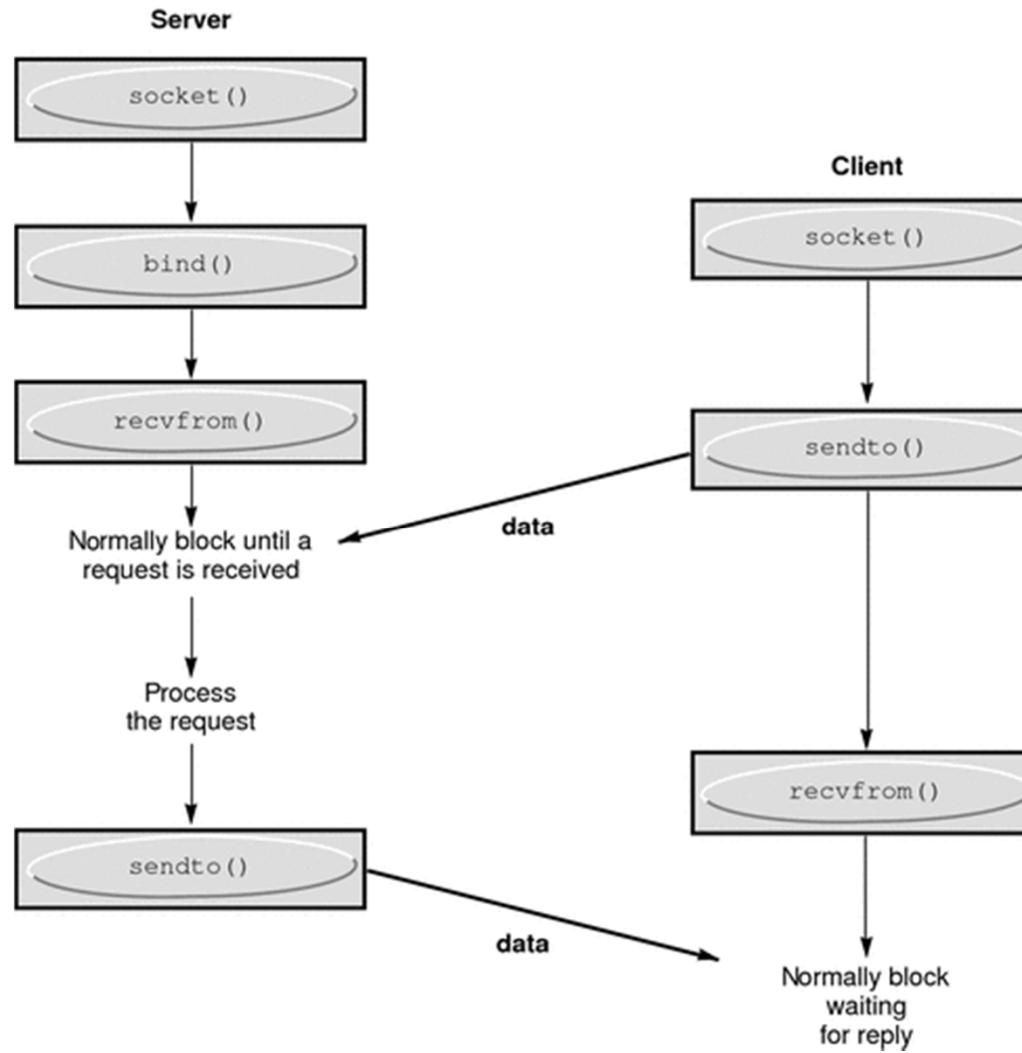
Source: Joobak Lee

Stream Communication Flow



Source: docs.oracle.com

Datagram Communication Flow



Source: docs.oracle.com

From the Server's Perspective

* We generally have two endpoints for stream sockets

▶ The server and the client

- The server sits and waits for a connection
- The client connects whenever it needs to communicate with the server

* The server needs to have a known endpoint name for the connection

▶ I.e., there needs to be a telephone number to dial and someone waiting for the call

- This is the socket address (IP and port number)

▶ The socket address is “bound” to a socket

- Once bound, no other socket can use that socket address on the server



Source: random-spots.blogspot.com

Example TCP Server

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[]) {
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;
    char sendBuff[1025];
    time_t ticks;
```



Source: hp.com

Example TCP Server 2

```
listenfd = socket(AF_INET, SOCK_STREAM, 0);
memset(&serv_addr, '0', sizeof(serv_addr));
memset(sendBuff, '0', sizeof(sendBuff));

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(5000);

bind(listenfd, (struct sockaddr*)&serv_addr,
      sizeof(serv_addr));
listen(listenfd, 10);
```

Example TCP Server 3

```
while(1) {
    connfd = accept(listenfd,
                    (struct sockaddr*)NULL, NULL);

    ticks = time(NULL);
    sprintf(sendBuff, sizeof(sendBuff),
            "%.24s\r\n", ctime(&ticks));
    write(connfd, sendBuff, strlen(sendBuff));

    shutdown(connfd, SHUT_RDWR);
    close(connfd);
    sleep(1);
}
```

Example TCP Client

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]) {
    int sockfd = 0, n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;
```



Source: lyricspond.com

Example TCP Client 2

```
if(argc != 2) {  
    printf("\n Usage: %s <ip of server> \n", argv[0]);  
    return 1;  
}  
  
memset(recvBuff, '0', sizeof(recvBuff));  
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {  
    printf("\n Error : Could not create socket \n");  
    return 1;  
}  
  
memset(&serv_addr, '0', sizeof(serv_addr));  
serv_addr.sin_family = AF_INET;  
serv_addr.sin_port = htons(5000);
```

Example TCP Client 3

```
if/inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0) {  
    printf("\n inet_pton error occurred\n");  
    return 1;  
}  
  
if( connect(sockfd, (struct sockaddr *)&serv_addr,  
            sizeof(serv_addr)) < 0) {  
    printf("\n Error : Connect Failed \n");  
    return 1;  
}  
  
while ((n = read(sockfd, recvBuff, sizeof(recvBuff)-1))>0) {  
    recvBuff[n] = 0;  
    if(fputs(recvBuff, stdout) == EOF) {  
        printf("\n Error : Fputs error\n");  
    }  
}
```

Example TCP Client 4

```
if(n < 0) {  
    printf("\n Read error \n");  
}  
  
shutdown(sockfd, SHUT_RDWR);  
close(sockfd);  
return 0;  
}
```

What About Datagrams?

★ Datagrams are fast and don't require polling to get the whole message like stream sockets do

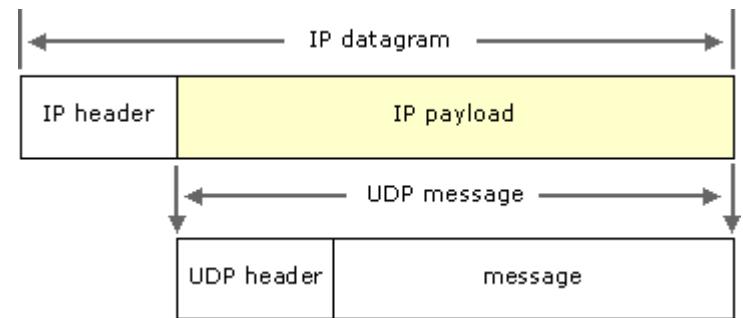
- ▶ But, datagrams are unreliable
 - No guaranteed delivery

★ There's no connection with datagrams

- ▶ No `listen(...)`, `accept(...)`, `connect(...)`

★ `send(...)`/`write()` and `recv(...)`/`read()` are replaced by `sendto(...)`/`recvfrom(...)`

- ▶ These functions take a `sockaddr` structure containing the recipient's or sender's address



Source: microsoft.com

Socket Options

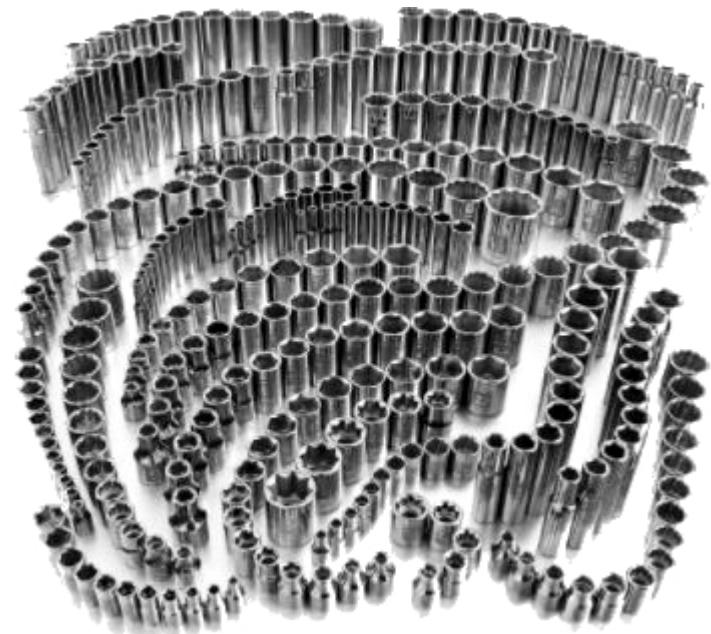
- ★ Sockets have many options that can be set to alter performance/behavior

- ▶ E.g., the send buffer and receive buffer sizes are commonly adjusted

- ★ This is done using the `setsockopt(...)` call

```
int bufSize = 49152;  
setsockopt(fd, SOL_SOCKET, SO_SNDBUF,  
          &bufSize, sizeof(bufSize));
```

- ★ For example, max `SO_SNDBUF`/`SO_RCVBUF` buffer sizes are visible in:
 - ▶ `/proc/sys/net/core/rmem_max`
 - ▶ `/proc/sys/net/core/wmem_max`



Source: kmart.com

Additional Socket Options

★ TCP level

- ▶ **TCP_NODELAY**
 - Disable the Nagle algorithm (packet coalescing)

★ Socket level

- ▶ **SO_REUSEADDR**
 - Allow reuse of local address
- ▶ **SO_KEEPALIVE**
 - Keep connection alive by periodic transmission
- ▶ **SO_LINGER**
 - Block close until outgoing data has been sent
- ▶ **SO_SNDBUF**
 - Set send buffer size
- ▶ **SO_RCVBUF**
 - Set receive buffer size
- ▶ **SO_BROADCAST**
 - Permit sending of broadcast messages

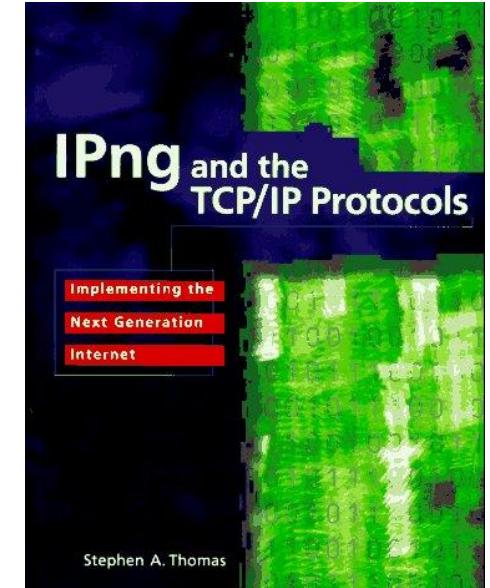
★ There are several others, including multicast-related join/leave options

Transitioning to IPv6

- ❖ When IPv4 was first developed, the Internet was a much smaller place
- ❖ ARPANET was funded by DARPA and primarily targeted at military communications
 - ▶ Replacement for the aging Telex system
- ❖ When ARPANET became the WWW, no one anticipated the explosion of growth
- ❖ As the Internet continues to expand, we really not have much choice but to transition to IPv6

IPv6 History

- ★ Back in the early 1990s, the IETF foresaw the exhaustion of the 32-bit IPv4 address space
 - ▶ IPng project was born in 1994
- ★ IPv6 was finalized in December of 1998
 - ▶ RFC 2460
- ★ There actually was a test framework known as IPv5
 - ▶ But, it was never deployed



Source: openlibrary.org

IPv4 Address Issues

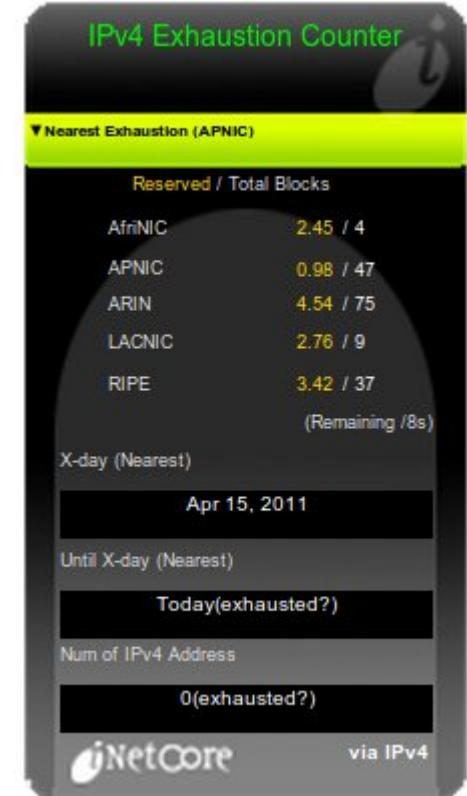
- ✖ IPv4 (RFC 791) uses a 32-bit address space
 - ▶ Seemed like enough in 1981
- ✖ Originally split into different address classes
 - ▶ Class A (7/24), B (14/16), C (21/8)
 - ▶ Class D is reserved for multicasting and Class E is used for research
- ✖ As we started to run out of addresses, the IETF introduced Classless Inter-Domain Routing (CIDR)
 - ▶ Addresses were expressed in addr/X format
 - E.g., 192.168.101.130/25 (255.255.255.128)
 - ▶ NAT became the rule of the day

Reasons for Switching to IPv6

- ★ We've run out of IPv4 addresses
 - ▶ ARIN is no longer able to fulfill requests for IPv4 addresses
- ★ IPv6 is being mandated by most governments
 - ▶ We probably can't ignore this one forever ☺
- ★ We want to regain end-to-end transparency
 - ▶ Reduction of latency is important for streaming media applications
 - ▶ Middle devices like NAT boxes add considerable latency
- ★ Core gateways are being overburdened by address bloat
 - ▶ We experienced a major blackout of the Internet in 2014 thanks to exceeding the 512K TCAM threshold
- ★ IPv6 has support for IPsec encryption

Whoops!

- ✖ After forecasting that we'd run out of addresses for the past decade, we finally did it
- ✖ Did the Internet stop?
 - ▶ Nope
- ✖ However, the RIRs are getting aggressive about reclaiming unused address space
 - ▶ Not an issue if you're hiding behind a NAT frontend

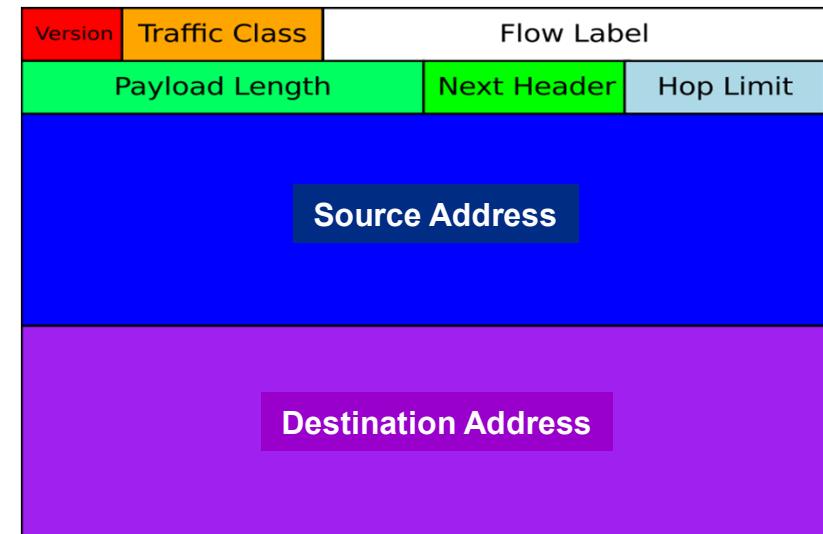


Source: inetcore.com

IPv6 is a Simpler Protocol (for Routers)

- ★ IPv4 is a complex protocol
 - ▶ Many fields that need to be interrogated by routers
- ★ IPv6 header has a fixed 40-octet length
 - ▶ IPv4 header length ranges from 20-60 octets
- ★ IPv6 moves the IPv4 options to additional headers

bit offset	0–3	4–7	8–13	14–15	16–18	19–31				
0	Version	Header Length	Differentiated Services Code Point	Explicit Congestion Notification	Total Length					
32	Identification			Flags	Fragment Offset					
64	Time to Live		Protocol		Header Checksum					
96	Source IP Address									
128	Destination IP Address									
160	Options (if Header Length > 5)									
160 or 192+	Data									



Source: wikipedia.org

IPv6 Addresses

- ★ IPv6 addresses are certainly more complex
 - ▶ 128-bit IPv6 vs. 32-bit IPv4
- ★ Special addresses include:
 - ▶ ::1 – Loopback (equivalent to 127.0.0.1 in IPv4)
 - ▶ :: – Unspecified (equivalent to 0.0.0.0/INADDR_ANY in IPv4)
- ★ IPv6 does not support broadcast
 - ▶ Only multicast addresses with FF02::1 as all-nodes multicast
- ★ IPv6 link-local addresses can be based on your hardware MAC address
 - ▶ MAC: 5c:26:0a:26:76:dc
 - ▶ IPv6: fe80::5e26:aff:fe26:76dc/64
 - Known as an EUI-64 address
- ★ Addresses can also be auto-assigned via SLAAC or DHCPv6

IPv6 Address Notation

* Example (these are all equivalent):

- ▶ 2008:0db8:0000:0000:0000:0000:1978:57ac
- ▶ 2008:0db8:0000:0000:0000::1978:57ac
- ▶ 2008:0db8:0:0:0:1978:57ac
- ▶ 2008:0db8::1978:57ac
- ▶ 2008:db8::1978:57ac

* An IPv6 URL is enclosed in brackets

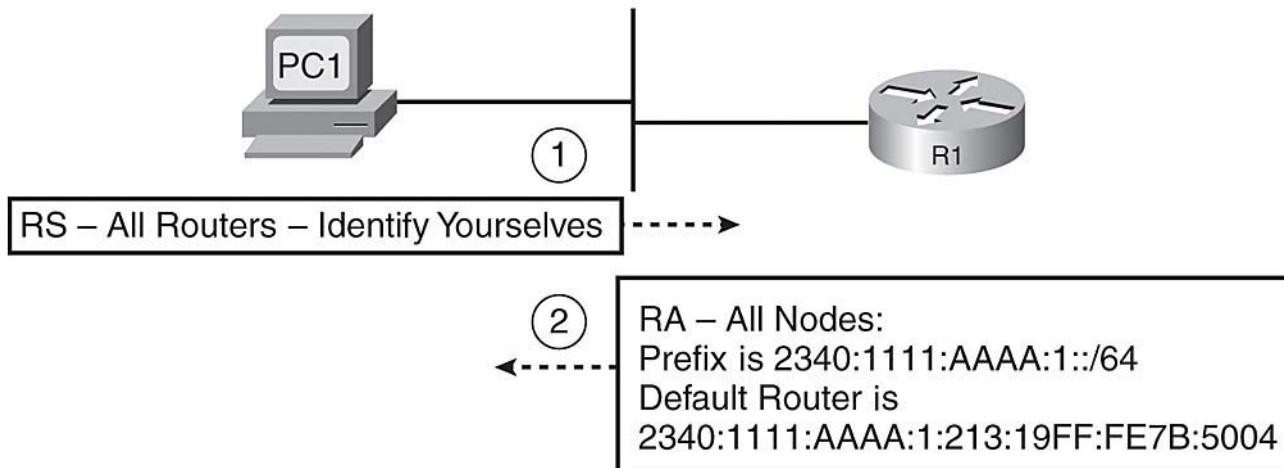
- ▶ [http://\[2008:0db8::1978:57ac\]/](http://[2008:0db8::1978:57ac]/)
- ▶ [https://\[2008:0db8::1978:57ac\]:443/](https://[2008:0db8::1978:57ac]:443/)
- ▶ These addresses cry out for DNS

* IPv6 addresses use CIDR notation

- ▶ 2008:0db8:1234::/48
 - 2008:0db8:1234:0000:0000:0000:0000 through 2008:0db8:1234:ffff:ffff:ffff:ffff

SLAAC

- ★ Stateless Address Autoconfiguration
- ★ When an IPv6 host initially connects to a network it issues a link-local RS multicast message
- ★ Routers respond by multicasting an RA message
 - ▶ Contains network-layer configuration parameters
- ★ SLAAC automatically derives a routable IPv6 address from the RA message and configures the host



Source: networkworld.com

SLAAC (2)

- ★ SLAAC is a Zero Configuration Networking (zeroconf) tool that requires no manual intervention
 - ▶ A previously unconfigured host will “just work” when connected to the network
- ★ However, the stateless nature of SLAAC does not allow control over how addresses are allocated
 - ▶ No address pool like DHCP
- ★ Unfortunately, SLAAC does not provide for configuring other protocols such as NTP
 - ▶ Although wide deployment of RFC 6106 Recursive DNS and DNS search list support does supply DNS resolver addresses



Source: staples.com

DHCPv6

- ★ Dynamic Host Configuration Protocol for IPv6
- ★ For situations where SLAAC is insufficient, DHCPv6 offers stateful autoconfiguration
 - ▶ Supplies IPv6 address assignment and DNS resolver configurations to clients
 - ▶ Very similar to DHCPv4
- ★ DHCPv6 has a number of advantages over SLAAC
 - ▶ Address allocation control
 - ▶ Access control
 - ▶ Can configure other services (e.g., NTP)
 - ▶ Similar to DHCPv4 in concept to help existing system administrators
- ★ Supports a stateless mode for configuring “other” protocols such as NTP, DNS but not tracking IP address allocation

IPv4/IPv6 Coexistence

- ★ For those O/Ses that support IPv6, most support “dual stack”
 - ▶ Both IPv4 and IPv6 are resident and can route packets
- ★ If you have an IPv6 device and must route across IPv4, there are also tunneling approaches
 - ▶ 6to4, Teredo, 6in4, and more
- ★ There are also tunnel brokers
 - ▶ Tunnel endpoints to bypass IPv6-ignorant ISPs

IPv6 Commands

- ★ Most of your favorite networking commands exist with a “6” appended
 - ▶ **ping6, traceroute6, iptables6**, etc.
- ★ Most O/S variants already have IPv6 support
 - ▶ Linux, OS/X, Windows
- ★ Many RTOSes support IPv6
 - ▶ VxWorks, ThreadX, QNX, OSE, LynxOS
 - ▶ However, many others still do not have IPv6 support

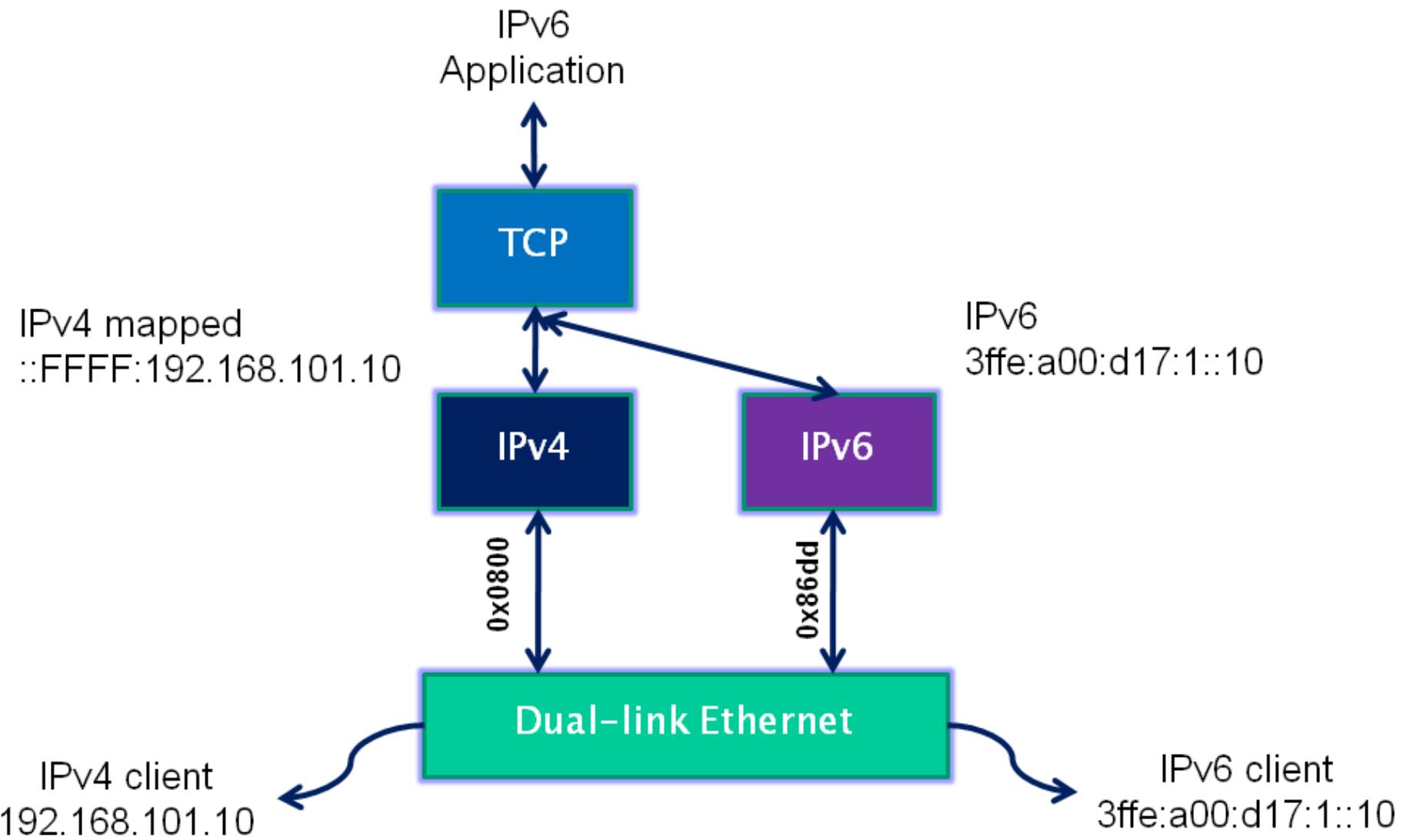
The Good News...

- ★ The code flow for IPv6 is identical to that of IPv4
 - ▶ `socket()`, `bind()`, `listen()`, `accept()`, etc.
- ★ The address structures in the API calls need to change to handle the 128-bit addresses
- ★ The changes are related to those APIs that expose the size of the IP address or manipulate the address in some way
 - ▶ Especially those that handle name-to-address resolution

Strategies

- ❖ Since many O/Ses support dual stack, IPv4 code will continue to run for the foreseeable future
 - ▶ For now, you might get away with doing nothing
- ❖ We could start developing IPv6-only code
 - ▶ The simplest conversion approach
- ❖ However, IPv4 is expected to still be with us for the next 10–15 years
 - ▶ So we probably want to create code that supports both IPv4 and IPv6 stacks

Dual Stack Operation IPv6-Only Application



Porting Applications to IPv6-Only

★ Brute-force conversion is straightforward

- ▶ The `sockaddr_in` structure becomes `sockaddr_in6`
- ▶ Address family becomes `AF_INET6/PF_INET6`
- ▶ Most of the socket API calls stay the same

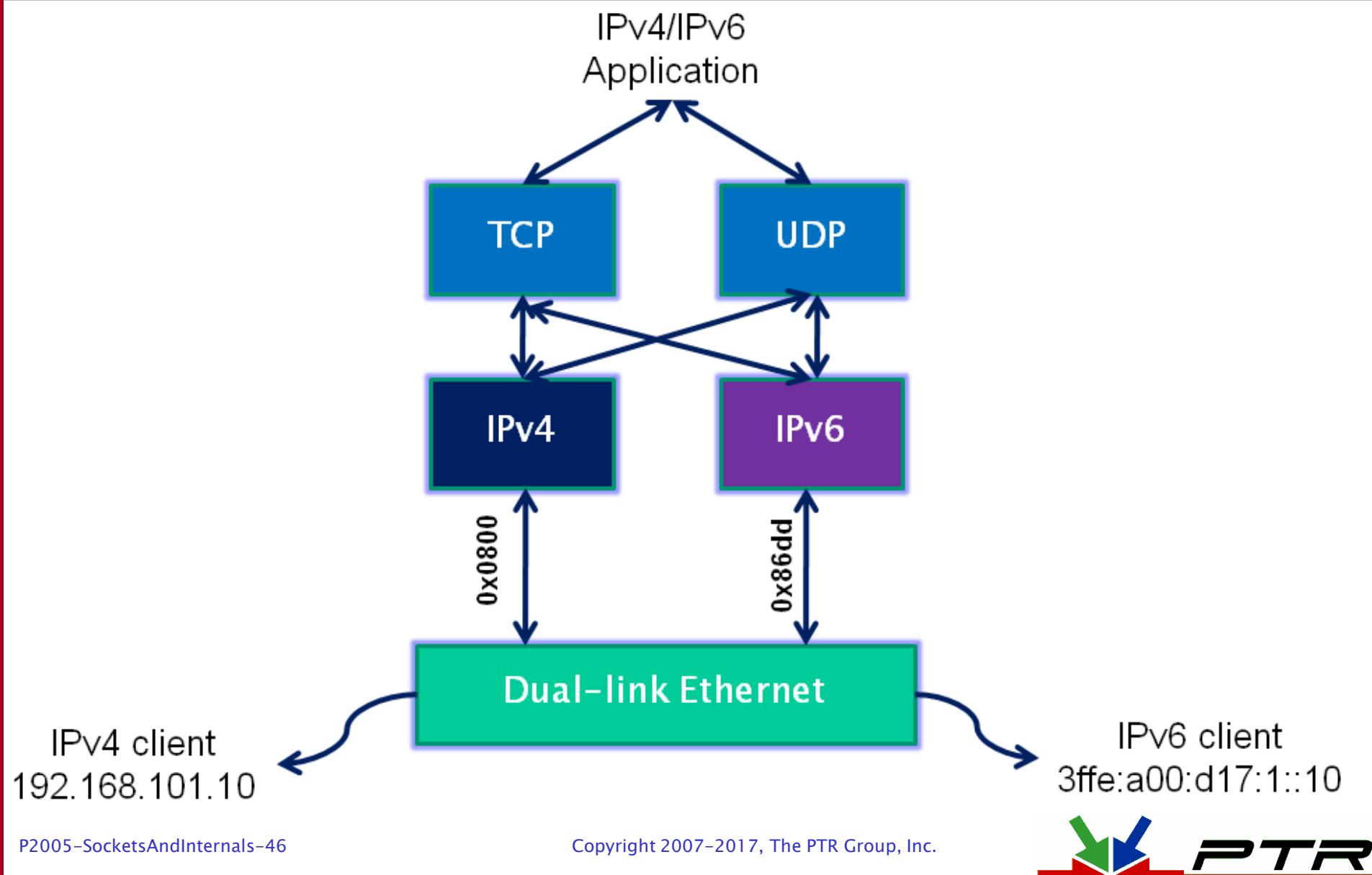
★ If an application embeds the address in the protocol (e.g., FTP and NTPv3), then they need more rework

API Comparison

	IPv4 (AF_INET)	IPv6 (AF_INET6)/POSIX
Data Structures	<code>PF_INET</code> <code>in_addr</code> <code>sockaddr_in</code> <code>sockaddr</code>	<code>PF_INET6</code> <code>in6_addr</code> <code>sockaddr_in6</code> <code>sockaddr_storage</code>
Address Conversion Functions	<code>gethostbyname()</code> <code>gethostbyaddr()</code>	<code>getnameinfo()</code> <code>getaddrinfo()</code>
Name/Address Functions	<code>inet_aton()</code> <code>inet_addr()</code> <code>inet_ntoa()</code>	<code>inet_pton()</code> <code>inet_ntop()</code>

Red functions work with both IPv4 and IPv6

Dual Stack-Aware Application



IPv4 Structures

```
#include <netinet/in.h>

// IPv4 AF_INET sockets:

struct sockaddr_in {
    short          sin_family;    // e.g. AF_INET, AF_INET6
    unsigned short  sin_port;     // e.g. htons(3490)
    struct in_addr  sin_addr;     // see struct in_addr, below
    char           sin_zero[8];   // zero this
};

struct in_addr {
    unsigned long   s_addr;       // load with inet_pton()
};

// Pointers to socket address structures are often cast to pointers
// to this type before use in various functions and system calls:

struct sockaddr {
    unsigned short  sa_family;    // address family, AF_xxx
    char           sa_data[14];   // 14 bytes of protocol address
};
```

IPv6 Structures

```
#include <netinet/in.h>

// IPv6 AF_INET6 sockets:

struct sockaddr_in6 {
    u_int16_t      sin6_family;    // address family, AF_INET6
    u_int16_t      sin6_port;     // port number, Network Byte Order
    u_int32_t      sin6_flowinfo; // IPv6 flow information
    struct in6_addr sin6_addr;    // IPv6 address
    u_int32_t      sin6_scope_id; // Scope ID
};

struct in6_addr {
    unsigned char   s6_addr[16];   // load with inet_pton()
};

// General socket address holding structure, big enough to hold either
// struct sockaddr_in or struct sockaddr_in6 data:

struct sockaddr_storage {
    sa_family_t      ss_family;    // address family

    // all this is padding, implementation specific, ignore it:
    char            __ss_pad1[_SS_PAD1SIZE];
    int64_t         __ss_align;
    char            __ss_pad2[_SS_PAD2SIZE];
};
```

Example IPv4 Server Setup

```
struct sockaddr addr;
int newFd;
int s = socket(PF_INET, SOCK_STREAM, 0);

memset(&addr, 0, sizeof(addr));
struct sockaddr_in * ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET;
ia->sin_port = htons(5002);

bind(s, &addr, sizeof(struct sockaddr_in));
listen(s, 5);

while (1) {
    memset(&addr, 0, sizeof(addr));
    socklen_t alen = sizeof(struct sockaddr);
    newFd = accept(s, &addr, &alen);
    pthread_create(&pt, NULL, &process, (void *) &newFd);
}
```

Example IPv6 Server Setup

```
struct sockaddr_in6 addr;
int newFd;
int s = socket(PF_INET6, SOCK_STREAM, 0);

memset(&addr, 0, sizeof(addr));
struct sockaddr_in6 * ia = (struct sockaddr_in6*) &addr;
ia->sin6_family = AF_INET6;
ia->sin6_port = htons(5002);

bind(s, &addr, sizeof(struct sockaddr_in6));
listen(s, 5);

while (1) {
    memset(&addr, 0, sizeof(addr));
    socklen_t alen = sizeof(struct sockaddr);
    newFd = accept(s, (struct sockaddr *) &addr, &alen);
    pthread_create(&pt, NULL, &process, (void *) &newFd);
}
```

IPv4 Client Setup

```
struct sockaddr addr;
struct sockaddr_in *ia;
int s = socket(PF_INET, SOCK_STREAM, 0);

memset(&addr, 0, sizeof(addr));
ia = (struct sockaddr_in*) &addr;
ia->sin_family = AF_INET;
ia->sin_port = htons(5002);
ia->sin_addr.s_addr = htonl(INADDR_LOOPBACK);

connect(s, &addr,
        sizeof(struct sockaddr_in));
process(s);
close(s);
```

IPv6 Client Setup

```
struct sockaddr_in6 addr;
struct sockaddr_in6 *ia;
int s = socket(PF_INET6, SOCK_STREAM, 0);

memset(&addr, 0, sizeof(addr));
ia = (struct sockaddr_in6*) &addr;
ia->sin6_family = AF_INET6;
ia->sin6_port = htons(5002);
ia->sin6_addr.s6_addr = in6addr_loopback;

connect(s, (struct sockaddr *) &addr,
        sizeof(struct sockaddr_in6));
process(s);
close(s);
```

Testing Your Site's IPv6 Readiness

* There is a test site: <http://test-ipv6.com>

Test your IPv6 connectivity.

[Summary](#) [Tests Run](#) [Technical Info](#) [Share Results / Contact](#)

 Your IPv4 address on the public internet appears to be 68.100.143.100

 Your IPv6 address on the public internet appears to be 2001:0:53aa:64c:1835:61f6:bb9b:709b
Your IPv6 service appears to be: Teredo
(unknown result code: teredo-ipv4pref)

 [World IPv6 day](#) is June 8th, 2011. No problems are anticipated for you with this browser, at this location. [\[more info\]](#)

 Congratulations! You appear to have both IPv4 and IPv6 internet working. If a publisher publishes to IPv6, your browser will connect using IPv6. Note: Your browser appears to prefer IPv4 over IPv6 when given the choice. This may in the future affect the accuracy of sites who guess at your location.

 Your DNS server (possibly run by your ISP) appears to have no access to the IPv6 internet, or is not configured to use it. This may in the future restrict your ability to reach IPv6-only sites. [\[more info\]](#)

Your readiness scores

10/10 for your IPv4 stability and readiness, when publishers offer both IPv4 and IPv6

9/10 for your IPv6 stability and readiness, when publishers are forced to go IPv6 only

Click to see [test data](#)

(Updated server side IPv6 readiness stats)

Summary

- ★ Linux has rich networking support
 - ▶ You may have to reconfigure the kernel to enable some of the features
- ★ Network interfaces are administered via special applications like `ip` and `ss`
- ★ Linux networking presents the POSIX socket interface to application developers
- ★ It is possible to write applications that support both IPv4 and IPv6 simultaneously

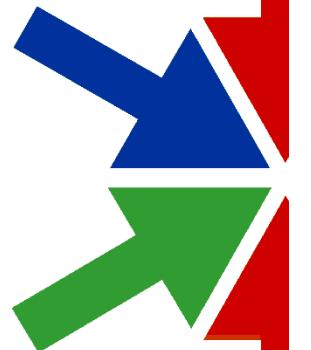
Questions

- * ip is the Linux command for manipulating the protocol addresses of network interfaces
 - ▶ True or False?
- * listen(...) is a blocking call
 - ▶ True or False?
- * IPv6 is a brand new protocol
 - ▶ True or False?
- * IPv6 uses
 - A) 64-bit addresses
 - B) 32-bit addresses
 - C) 128-bit addresses
 - D) 256-bit addresses
- * It is not possible to have one piece of code support both IPv4 and IPv6 simultaneously
 - ▶ True or False?

Chapter Break

Socket Kernel Internals

PTR



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

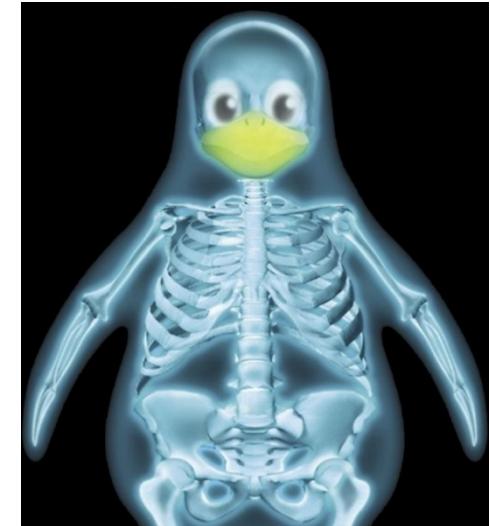
- * The BSD/POSIX socket backend
- * The `sys_socketcall()` entry point
- * Adding a new protocol to the kernel

BSD vs. POSIX Sockets

- ★ The Berkeley Software Design (BSD) model for the socket abstraction has become a defacto standard
 - ▶ Even Microsoft WinSock is based on BSD sockets
- ★ BSD sockets eventually evolved into the POSIX socket API
 - ▶ This is where functions like `gethostbyname()` got deprecated in favor of `getaddrinfo()`, etc.
- ★ This chapter will focus on the kernel implementation of the socket API

Socket Internals

- ★ Next, we'll take a look at the internals of `sys_socketcall(...)`
- ★ We will look at the `SYS_SOCKET` operation to illustrate the data structures and other internal APIs involved in creating a socket
- ★ These are internal calls used by code libraries
 - ▶ Not called directly by user code



Source: desktopnexus.com

Backend Socket Interface

- * The socket interface in kernel space is made up of just one system call:

```
asmlinkage long sys_socketcall(int call,  
                                unsigned long __user *args);
```

- * All socket-related calls from user space go through this function

- ▶ Defined in net/socket.c



Source: gutenberg.org

sys_socketcall()

- ★ Linux defines **sys_socketcall(...)** as an entry in the system call jump table, defined in **include/asm/unistd.h**:

```
#define __NR_socketcall 102
```

- ★ The socket function to be invoked in the kernel is passed from user space as a parameter to **sys_socketcall(...)**

- ▶ E.g., the user-space library will map a user call to **socket(...)** by generating a system call **sys_socketcall(SYS_SOCKET, ...)**

sys_socketcall() – snippet

- ★ The following is part of the definition of `sys_socketcall(...)`
 - ▶ It copies the user-provided arguments to kernel space and calls `sys_socket(...)`

```
SYSCALL_DEFINE2(socketcall, int, call, unsigned long __user *, args)
{
    unsigned long a[6];
    unsigned long a0, a1;
    int err;
    unsigned int len;

    if (call < 1 || call > SYS_SENDDMSG)
        return -EINVAL;

    /* copy_from_user should be SMP safe. */
    if (copy_from_user(a, args, nargs[call]))
        return -EFAULT;

    audit_socketcall(nargs[call] / sizeof(unsigned long), a);

    a0 = a[0];
    a1 = a[1];
```

sys_socketcall() – snippet (2)

```
switch (call) {
case SYS_SOCKET:
    err = sys_socket(a0, a1, a[2]);
    break;
case SYS_BIND:
    err = sys_bind(a0, (struct sockaddr __user *)a1, a[2]);
    break;
case SYS_CONNECT:
    err = sys_connect(a0, (struct sockaddr __user *)a1, a[2]);
    break;
case SYS_LISTEN:
    err = sys_listen(a0, a1);
    break;
case SYS_ACCEPT:
    err = sys_accept4(a0, (struct sockaddr __user *)a1,
                      (int __user *)a[2], 0);
    break;
[...]
```

sys_socketcall() – parameters

- ★ Supported values for the "call" parameter to `sys_socketcall(...)` are:

<code>#define SYS_SOCKET</code>	1	<code>/* sys_socket(2) */</code>
<code>#define SYS_BIND</code>	2	<code>/* sys_bind(2) */</code>
<code>#define SYS_CONNECT</code>	3	<code>/* sys_connect(2) */</code>
<code>#define SYS_LISTEN</code>	4	<code>/* sys_listen(2) */</code>
<code>#define SYS_ACCEPT</code>	5	<code>/* sys_accept(2) */</code>
<code>#define SYS_GETSOCKNAME</code>	6	<code>/* sys_getsockname(2) */</code>
<code>#define SYS_GETPEERNAME</code>	7	<code>/* sys_getpeername(2) */</code>
<code>#define SYS_SOCKETPAIR</code>	8	<code>/* sys_socketpair(2) */</code>
<code>#define SYS_SEND</code>	9	<code>/* sys_send(2) */</code>
<code>#define SYS_RECV</code>	10	<code>/* sys_recv(2) */</code>

sys_socketcall() – parameters (2)

★ And more:

#define SYS_SENDTO	11	/* sys_sendto(2) */
#define SYS_RECVFROM	12	/* sys_recvfrom(2) */
#define SYS_SHUTDOWN	13	/* sys_shutdown(2) */
#define SYS_SETSOCKOPT	14	/* sys_setsockopt(2) */
#define SYS_GETSOCKOPT	15	/* sys_getsockopt(2) */
#define SYS_SENDMSG	16	/* sys_sendmsg(2) */
#define SYS_RECVMSG	17	/* sys_recvmsg(2) */
#define SYS_ACCEPT4	18	/* sys_accept4(2) */
#define SYS_RECVMMSG	19	/* sys_recvmmmsg(2) */
#define SYS_SENDMMSG	20	/* sys_sendmmmsg(2) */

sys_socket()

- ★ This function allocates the socket structure and calls `sock_create(...)` and `sock_map_fd(...)`
 - ▶ All located in `inet/socket.c`

```
SYSCALL_DEFINE3(socket, int, family, int, type, int, protocol)
{
    int retval;
    struct socket *sock;
    int flags;
[...]

    flags = type & ~SOCK_TYPE_MASK;
[...]

    if (SOCK_NONBLOCK != O_NONBLOCK && (flags & SOCK_NONBLOCK))
        flags = (flags & ~SOCK_NONBLOCK) | O_NONBLOCK;

    retval = sock_create(family, type, protocol, &sock);
    if (retval < 0)
        goto out;

    retval = sock_map_fd(sock, flags & (O_CLOEXEC | O_NONBLOCK));
[...]
```

`sock_create()`

* This function performs the following tasks:

- ▶ Checks the protocol family parameter to see if the protocol is available
 - If not, it may try to load the appropriate module
 - E.g., for IPv6, it may try to load ipv6.ko
- ▶ Invokes `sock_alloc(...)`
- ▶ Calls the underlying protocol-specific socket create function
- ▶ If all goes well, `inet_create(...)` is invoked

sock_alloc()

- ★ This function initializes an inode, allocates the socket structure, then binds the two
- ★ A socket structure pointer is returned to the caller

```
static struct socket *sock_alloc(void)
{
    struct inode *inode;
    struct socket *sock;

    inode = new_inode(sock_mnt->mnt_sb);
    if (!inode)
        return NULL;

    sock = SOCKET_I(inode);

    inode->i_ino = get_next_ino();
    inode->i_mode = S_IFSOCK | S_IRWXUGO;
    inode->i_uid = current_fsuid();
    inode->i_gid = current_fsgid();

    percpu_add(sockets_in_use, 1);
    return sock;
}
```

sock_map_fd()

- ★ This function creates a file descriptor and installs it into the thread's file descriptor table
 - ▶ A new file descriptor is returned

```
int sock_map_fd(struct socket *sock, int flags)
{
    struct file *newfile;
    int fd = sock_alloc_file(sock, &newfile, flags);

    if (likely(fd >= 0))
        fd_install(fd, newfile);

    return fd;
}
```

struct socket

* General POSIX socket structure

* Defined in: include/linux/net.h

```
struct socket {  
    socket_state           state;  
    short                 type;  
    unsigned long          flags;  
    struct socket_wq __rcu *wq;  
    struct file            *file;  
    struct sock             sk;  
    const struct proto_ops *ops;  
};
```

Structure Breakdown

>Description of socket structure members:

- ▶ **state**
 - Socket state (`SS_CONNECTED`, etc.)
- ▶ **type**
 - Socket type (`SOCK_STREAM`, etc.)
- ▶ **flags**
 - Socket flags (`SOCK_ASYNC_NOSPACE`, etc.)
- ▶ **wq**
 - Wait queue for several uses
- ▶ **file**
 - File pointer for garbage collection and other file-related operations
- ▶ **sk**
 - Internal networking protocol-agnostic socket representation
 - Points to a `sock` structure for binding with lower protocol-specific sockets
- ▶ **ops**
 - Protocol-specific socket operations/functions

struct proto_ops

* Contains protocol-specific callbacks

* Defined in: linux/net.h

```
struct proto_ops {  
    int          family;  
    struct module *owner;  
    int          (*release)(struct socket *sock);  
    int          (*bind)(struct socket *sock,  
                        struct sockaddr *myaddr, int sockaddr_len);  
    int          (*connect)(struct socket *sock, struct sockaddr *vaddr,  
                           int sockaddr_len, int flags);  
    int          (*socketpair)(struct socket *sock1, struct socket *sock2);  
    int          (*accept)(struct socket *sock,  
                        struct socket *newsock, int flags);  
    int          (*getname)(struct socket *sock, struct sockaddr *addr,  
                           int *sockaddr_len, int peer);  
    unsigned int  (*poll)(struct file *file, struct socket *sock,  
                         struct poll_table_struct *wait);  
    [...]
```

Installing a New Protocol

- * In order to install a new protocol/protocol family, we need to develop the code for the `proto_ops` structure and create a number of structures such as the `net_proto_family` structure
- * This would then be installed in the initialization function of a loadable module for the protocol
- * We will take a look at the IPv6 module as an example

The Basic Flow...

1. We need to write the protocol operations callbacks and create the `proto_ops` structure:
2. Register the base L4 protocol with the kernel
3. Register the protocol type with the kernel
4. Register the protocol family with the kernel
5. Register the `proto_ops` for each socket type and define the socket creation method
6. Finally, create the unregister that undoes all of this and wrap it in a KLM with `module_init()` and `module_exit()` calls

The proto_ops Structure

```
const struct proto_ops inet6_stream_ops = {
    .family      = PF_INET6,
    .owner       = THIS_MODULE,
    .release     = inet6_release,
    .bind        = inet6_bind,
    .connect     = inet_stream_connect, /* ok */
    .socketpair  = sock_no_socketpair, /* a do nothing */
    .accept      = inet_accept,      /* ok */
    .getname     = inet6_getname,
    .poll         = tcp_poll,        /* ok */
    .ioctl        = inet6_ioctl,      /* must change */
    .listen       = inet_listen,      /* ok */
    .shutdown     = inet_shutdown,    /* ok */
    .setsockopt   = sock_common_setsockopt, /* ok */
    .getsockopt   = sock_common_getsockopt, /* ok */
    .sendmsg      = inet_sendmsg,     /* ok */
    .recvmsg      = inet_recvmsg,     /* ok */
    .mmap         = sock_no_mmap,
    .sendpage     = inet_sendpage,
    .splice_read  = tcp_splice_read,
#endif CONFIG_COMPAT
    .compat_setsockopt = compat_sock_common_setsockopt,
    .compat_getsockopt = compat_sock_common_getsockopt,
#endif
};
```

Registering the Protocol

- Continuing with the IPv6 example, we will need to create and register an L4 protocol for each socket type

```
struct proto tcpv6_prot = {
    .name          = "TCPv6",
    .owner         = THIS_MODULE,
    .close         = tcp_close,
    .connect       = tcp_v6_connect,
    .disconnect    = tcp_disconnect,
    .accept        = inet_csk_accept,
    .ioctl         = tcp_ioctl,
    .init          = tcp_v6_init_sock,
    .destroy       = tcp_v6_destroy_sock,
    .shutdown      = tcp_shutdown,
    .setsockopt   = tcp_setsockopt,
    .getsockopt   = tcp_getsockopt,
    .recvmsg       = tcp_recvmsg,
    .sendmsg       = tcp_sendmsg,
    .sendpage      = tcp_sendpage,
    .backlog_rcv  = tcp_v6_do_rcv,
    .release_cb   = tcp_release_cb,
    .mtu_reduced  = tcp_v6_mtu_reduced,
    .hash          = tcp_v6_hash,
    .unhash        = inet_unhash,
    .get_port      = inet_csk_get_port,
```

Registering the Protocol #2

```
.enter_memory_pressure = tcp_enter_memory_pressure,
.sockets_allocated     = &tcp_sockets_allocated,
.memory_allocated      = &tcp_memory_allocated,
.memory_pressure       = &tcp_memory_pressure,
.orphan_count          = &tcp_orphan_count,
.sysctl_wmem           = sysctl_tcp_wmem,
.sysctl_rmem           = sysctl_tcp_rmem,
.max_header             = MAX_TCP_HEADER,
.obj_size               = sizeof(struct tcp6_sock),
.slab_flags             = SLAB_DESTROY_BY_RCU,
.twsk_prot              = &tcp6_timewait_sock_ops,
.rsk_prot               = &tcp6_request_sock_ops,
.h.hashinfo              = &tcp_hashinfo,
.no_autobind            = true,

#endif CONFIG_COMPAT
.compat_setsockopt      = compat_tcp_setsockopt,
.compat_getsockopt      = compat_tcp_getsockopt,
#endif
#endif CONFIG_MEMCG_KMEM
.proto_cgroup            = tcp_proto_cgroup,
#endif
};
```

Registering the Protocol #3

- We will then reference the previous structure when we register the specific protocol variant:

```
static struct inet_protosw tcpv6_protosw = {  
    .type        = SOCK_STREAM,  
    .protocol    = IPPROTO_TCP,  
    .prot        = &tcpv6_prot,  
    .ops         = &inet6_stream_ops,  
    .no_check    = 0,  
    .flags       = INET_PROTOSW_PERMANENT |  
                  INET_PROTOSW_ICSK,  
};
```

- Here is the call to register the protocol callbacks:

```
/* register inet6 protocol */  
ret = inet6_register_protosw(&tcpv6_protosw);
```

Adding the Protocol Type to the Kernel

Next, we add the protocol *type* to the kernel:

```
static const struct inet6_protocol tcpv6_protocol = {  
    .early_demux =      tcp_v6_early_demux,  
    .handler      =      tcp_v6_rcv,  
    .err_handler =      tcp_v6_err,  
    .flags        =  
INET6_PROTO_NOPOLICY|INET6_PROTO_FINAL,  
};  
  
ret = inet6_add_protocol(&tcpv6_protocol, IPPROTO_TCP);
```

Different `proto_ops` per Socket Type

- ★ Each socket type will need their own `proto_ops` calls
 - ▶ E.g., stream sockets vs. datagram sockets
- ★ The `proto_ops` structure is then referenced when you register the socket type using the `sock_register()` command

The `net_proto_family` Structure

- * The `net_proto_family` structure looks like this:

```
static const struct net_proto_family inet6_family_ops = {  
    .family = PF_INET6,  
    .create = inet6_create,  
    .owner   = THIS_MODULE,  
};
```

- * Then we need to register this structure in order to be able to create sockets ala:

```
/* Register the family here so that the init calls below will  
 * be able to create sockets. (?? is this dangerous ??)  
 */  
err = sock_register(&inet6_family_ops);
```

The Protocol Init and Exit Routines

- * There is yet another structure related to the initialization and removal of L4 protocol handlers that needs to be dealt with:

```
static struct pernet_operations tcpv6_net_ops = {  
    .init          = tcpv6_net_init,  
    .exit          = tcpv6_net_exit,  
    .exit_batch   = tcpv6_net_exit_batch,  
};
```

- * This will be registered in the module's init function which we will examine in a couple of charts
- * The network init command is then called any time a socket of this type is created

```
static int __net_init tcpv6_net_init(struct net *net)  
{  
    return inet_ctl_sock_create(&net->ipv6.tcp_sk, PF_INET6,  
                               SOCK_RAW, IPPROTO_TCP, net);  
}
```

Installing and Removing the Module

- ★ In using kernel loadable modules (KLMs), we need code to be called during the installation of the module as well as exit code for module removal
- ★ These will be referenced in the `module_init()` and `module_exit()` sections of the KLM source
 - ▶ Our Linux device driver class goes into the requirements for this in detail

The Module Installation Code

* Here is the actual TCPv6 installation code:

```
int __init tcpv6_init(void) {
    int ret;

    ret = inet6_add_protocol(&tcpv6_protocol, IPPROTO_TCP);
    if (ret) goto out;

    /* register inet6 protocol */
    ret = inet6_register_protosw(&tcpv6_protosw);
    if (ret) goto out_tcpv6_protocol;

    ret = register_pernet_subsys(&tcpv6_net_ops);
    if (ret) goto out_tcpv6_protosw;

out:
    return ret;
out_tcpv6_protosw:
    inet6_unregister_protosw(&tcpv6_protosw);
out_tcpv6_protocol:
    inet6_del_protocol(&tcpv6_protocol, IPPROTO_TCP);
    goto out;
}
```

The Module Exit Code

* Naturally, if we want to remove the protocol from the kernel, we need to undo all of the registrations:

```
void tcpv6_exit(void)
{
    unregister_pernet_subsys(&tcpv6_net_ops);
    inet6_unregister_protosw(&tcpv6_protosw);
    inet6_del_protocol(&tcpv6_protocol, IPPROTO_TCP);
}
```

Summary

- ★ System calls like `sys_socket(...)` are used to call into the kernel for socket-related services
- ★ In order to install a new protocol into the kernel, we will need to create the `net_proto_ops` structure, write the code to support these functions and register them with the kernel
- ★ We then need to register each new socket type and install the initialization and exit code for the new socket type
- ★ This process is not documented anywhere other than the kernel itself
 - ▶ Use the source, Luke



Source:wikipedia.org

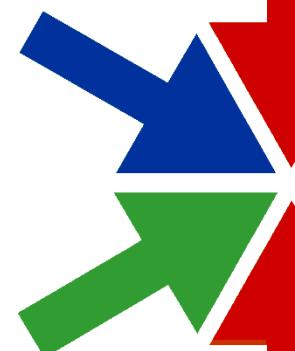
Questions

- * The function `sys_socketcall(...)` is the entry point into the kernel's jump table for socket operations
 - ▶ True or False?
- * The function call `sock_map_fd(...)` is responsible for creating the file descriptor that is used in socket operations
 - ▶ True or False?
- * Protocols cannot be loaded dynamically into the kernel
 - ▶ True or False?
- * The TCPv6 protocol type is registered with the kernel via the `inet6_add_protocol()` call
 - ▶ True or False?
- * The `sock_register()` call is used to register the socket creation code for a given socket family
 - ▶ True or False?

Chapter Break

IP Routing and Forwarding

IP Routing and Forwarding



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ★ IP Routing/Forwarding Basics
- ★ Routers and Routing Tables
- ★ IP Routing Types
- ★ Static Routing
- ★ Dynamic Routing
 - ▶ Routing Information Protocol (RIP)
 - ▶ Open Shortest Path First (OSPF)
- ★ Quagga and routing protocols
- ★ Quagga example
 - ▶ Configuring zebra and ripd
 - ▶ Debugging RIP
 - ▶ Results
- ★ Netlink sockets

Routers

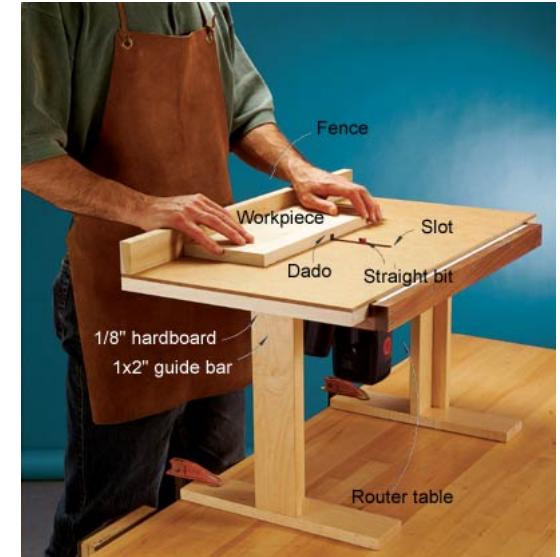
- ★ A router is a host that is connected to 2 or more independent networks
- ★ IP packets may be forwarded through a series of routers until reaching a destination
 - ▶ The IP layer at each router must decide where to forward each IP packet
- ★ The forwarding an IP packet to the next node is known as taking a “hop”
 - ▶ These days, the header’s TTL field is interpreted as a hop count
- ★ In order to enable routing across interfaces in Linux, we need to echo a “1” to `/proc/sys/net/ipv4/ip_forward`
`# echo 1 > /proc/sys/net/ipv4/ip_forward`



Source: ciscorouting.com

Routing Table

- ★ A routing table stores information on routes to specific IP hosts and networks
 - ▶ All IP nodes (hosts and routers) have routing tables
- ★ An IP node uses the routing table to determine the location of the next hop for an IP packet
 - ▶ This may be the final destination address or another router for the next hop



Source: woodmagazine.com

Routing Table (2)

- ★ A typical routing table entry contains a destination address/netmask, next-hop gateway address, egress interface, and cost metric
 - ▶ Other values and flags may also exist, depending on the implementation
- ★ When a packet arrives at a router, the routing table is searched to find the best route that matches the destination IP
 - ▶ Done by taking the longest prefix match destination

Accessing the Routing Table

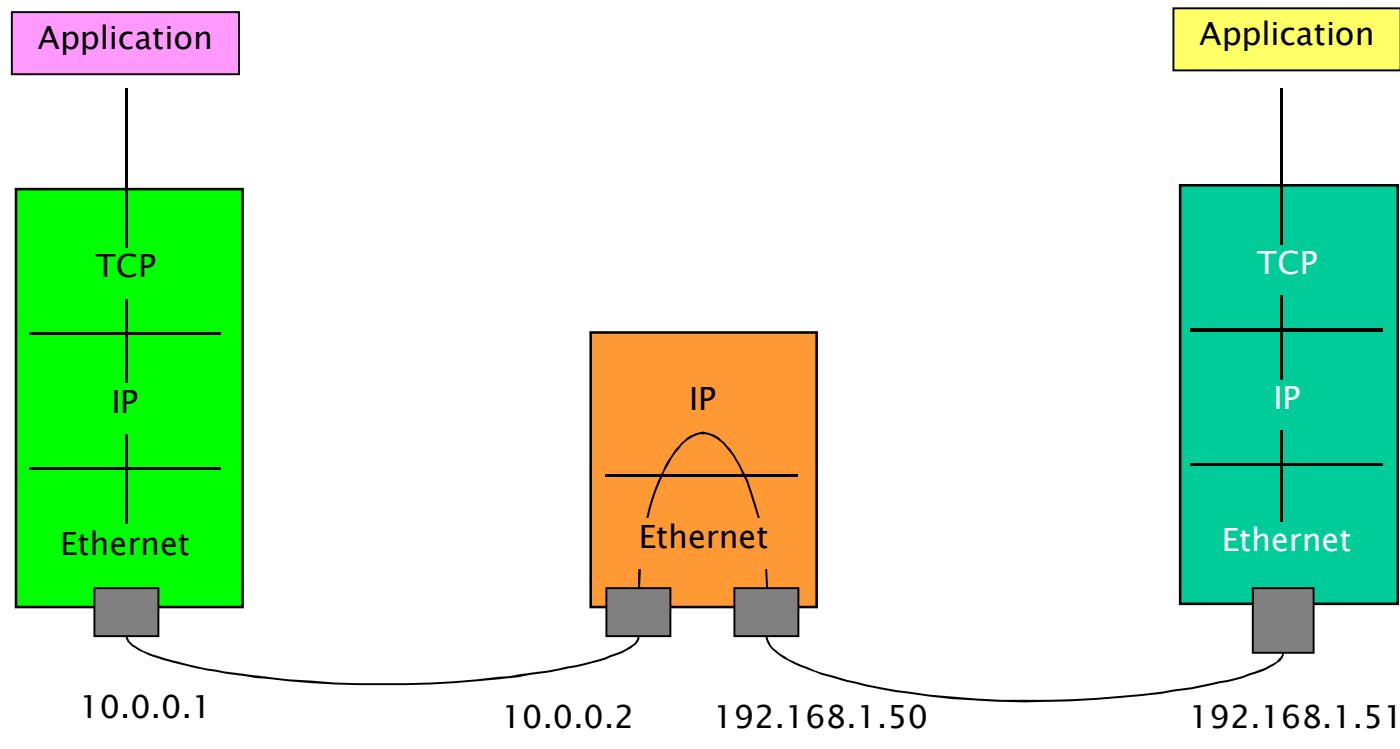
- * In Linux, the routing table can be displayed with `ip route` or `route`

```
$ ip route
default via 192.168.101.1 dev eth0 proto static
10.11.12.0/24 via 10.11.21.73 dev tun0 proto static
10.11.21.1 via 10.11.21.73 dev tun0 proto static
10.11.21.73 dev tun0 proto kernel scope link src 10.11.21.74
192.168.101.0/24 dev eth0 proto kernel scope link src 192.168.101.8 metric 1
```

```
$ route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
0.0.0.0         192.168.101.1   0.0.0.0        UG    0      0      0 eth0
10.11.12.0      10.11.21.73    255.255.255.0  UG    0      0      0 tun0
10.11.21.1      10.11.21.73    255.255.255.255 UGH   0      0      0 tun0
10.11.21.73     0.0.0.0        255.255.255.255 UH    0      0      0 tun0
192.168.101.0   0.0.0.0        255.255.255.0   U     1      0      0 eth0
```

- * A default route has a destination address/netmask of 0.0.0.0/0
 - ▶ Default routes are used when no other routes match an IP packet's destination
- * If no default route is specified and no matching route exists for an IP packet, it is dropped
 - ▶ An ICMP Destination Unreachable message is sent to the source

IP Routing Example



Routing Table

Network	Gateway
192.168.1.0	10.0.0.2

ARPs will be performed on first contact

Routing Table

Network	Gateway
10.0.0.0	192.168.1.50

Static vs. Dynamic Routing

★ Static Routing

- ▶ Routes are added, removed, and changed manually by a system administrator
- ▶ No remote router discovery
- ▶ Not fault tolerant
 - Static router failures are not detected by neighboring routers

★ Dynamic Routing

- ▶ Routes are updated automatically through dynamic routing protocols
 - Routing Information Protocol (RIP)
 - Open Shortest Path First (OSPF)
 - And several more...
- ▶ Supports remote router discovery
- ▶ Fault tolerant
 - Dynamic router failures are automatically detected and routing entries are updated



Source: vectorcast.com

Adding Routes in Linux

- * Routes can be added statically or dynamically
- * Static routes are added with the `ip route` command

```
# ip route add 192.168.1.0/24 via 10.0.0.2  
# ip route add default via 10.0.0.2
```

- ▶ Must have admin privileges (superuser)

- * Dynamic routes are added by a routing protocol, and may change over time
- * The older `route add` command also still works although it is now deprecated



Non-routable and Loopback Addresses

* IPv4 has a number of address ranges that are for use on the LAN only and do not traverse routers

- ▶ 10.0.0.0/8
- ▶ 172.16.0.0/16 – 172.31.0.0/16
- ▶ 192.168.0.0/16
- ▶ 127.0.0.0/8

* IPv6 also has a block of Unique Local Addresses from RFC 4193

- ▶ fc00::/7

* The IPv6 loopback address is ::1

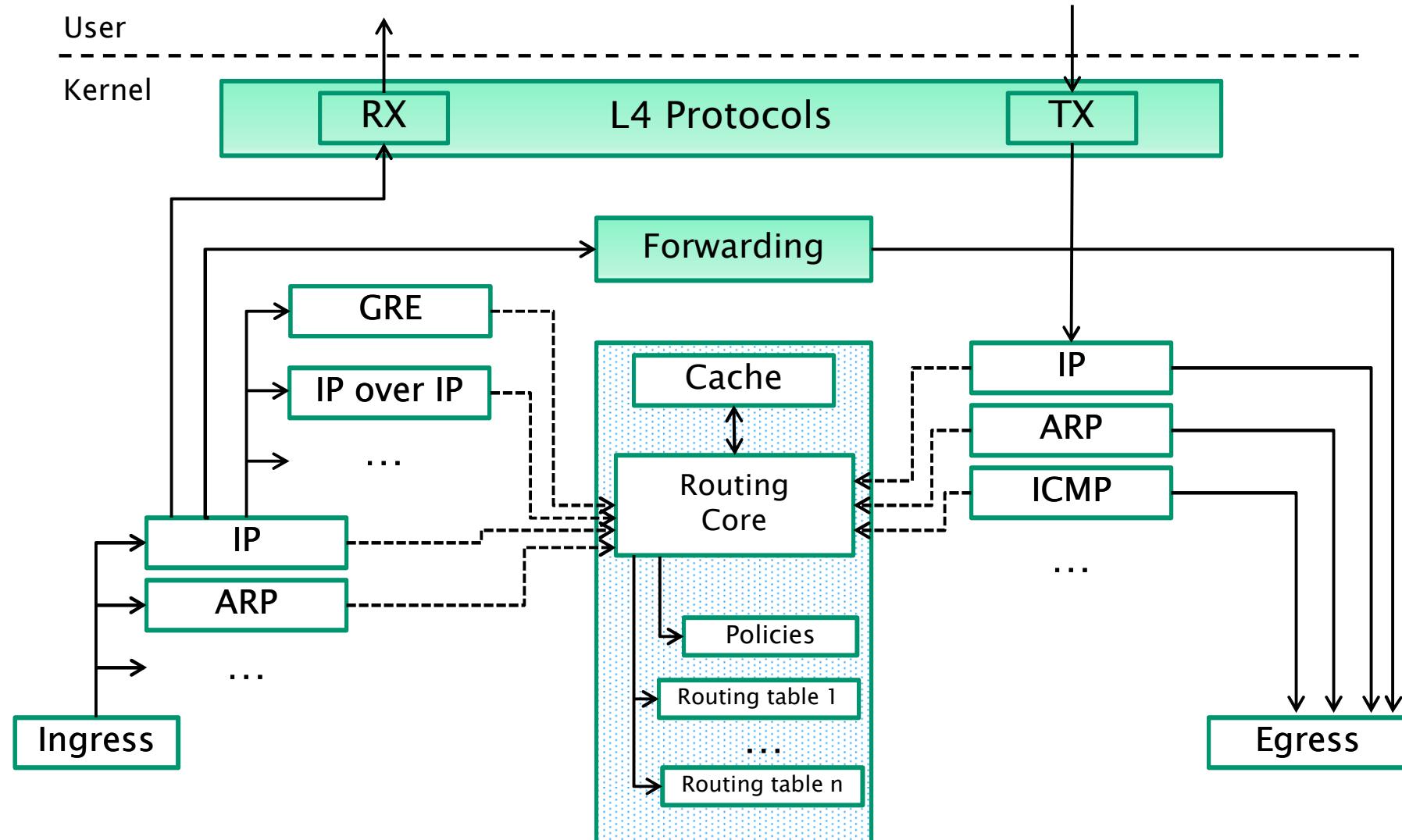
The Routing Table

- ★ The routing table is the core of the routing subsystem
 - ▶ Also known as the Forwarding Information Base (FIB)
- ★ In order to help traverse a routing table with many entries, Linux supports a routing cache
 - ▶ A quick hash of previously searched routes
 - ▶ Created via the `ip_rt_init(...)` call when the routing system is initialized
- ★ Linux splits the routing cache into two pieces
 - ▶ A protocol-dependent cache (IPv4 / IPv6, etc.) and a protocol independent destination cache known as DST
 - DST stores characteristics such as whether IPsec is needed

Which Route?

- ★ Matching a route is actually a mathematical function
 - ▶ A route matches a destination if the network address logically ANDed with the netmask precisely equals the destination address logically ANDed with the netmask
- ★ Or, more simply, a route matches if the number of bits of the network address specified by the netmask match that same number of bits in the destination address
 - ▶ Longer netmasks mean a better match
 - ▶ Also known as the “longest prefix match” lookup
- ★ The kernel will always check the routing cache first and then fall back to the routing table
 - ▶ The cache might contain an entry for a recently sent packet that’s already hashed

Routing Big Picture



Adding More Control to Routing

- ★ Early routers simply routed packets without looking at the kind of traffic
- ★ Modern requirements need to take into account:
 - ▶ Separating streams based on security labels or for accounting purposes
 - ▶ Sending real-time or streaming traffic over different interfaces
 - E.g., implementing QoS
- ★ This ability is referred to as *policy routing*

Policy Routing

- ★ Linux supports the idea of special routes and routing policies
 - ▶ Must be configured in the kernel and supports a maximum of 255 policy tables
- ★ Linux also supports the ability to create and manage alternate routing tables
 - ▶ There are three tables by default:
 - Local (cannot be modified or deleted by the user)
 - The local interfaces and broadcast addresses for the host itself
 - Main
 - Routes added by the administrator
 - Default
 - Reserved for post processing if previous rules did not find a match
 - ★ There are also default rules for each of the tables

Policy Routing #2

* In addition to the normal routing, Linux provides mechanisms for other types of actions

- ▶ Black hole
 - Packets matching this type are silently dropped
- ▶ Unreachable
 - Packets are dropped and an ICMP host unreachable message is generated
- ▶ Prohibit
 - Packet is dropped and ICMP packet filtered message is generated
- ▶ Throw
 - Causes the route lookup to abandon the current table and move to the next one

Creating a Custom Policy Routing Table

- * To create a custom routing table with name “custom” and an ID of “200”:

```
# echo 200 custom >> /etc/iproute2/rt_tables
```

- ▶ This will be persistent across reboots

- * We'll create a rule for looking at the source of the packet and using that in the routing decision:

```
# ip rule add from 192.168.20.200 lookup custom
```

- * To see the current rules:

```
# ip rule list
```

- * To make rules persist across reboots, you need to add the rule in `/etc/network/interfaces` or `/etc/sysconfig/network/ifup-<interface>`
`post-up ip rule add from 192.168.20.200 lookup custom`

Custom Policy Routing Table #2

* Next, tell the system to use the custom table by adding a route:

```
# ip route add default via 192.168.30.1 dev eth1 table custom
```

* Packets from 192.168.20.200 will now route via 192.168.30.1 on the eth1 device

* We can add alternate actions based on the DSCP or other various elements of the packet

- ▶ E.g., drop non-secure traffic

Dynamic Routing Protocols

★ Distance Vector Protocols

▶ RIP/RIPv2/RIPng

- Routing choice is made from the number of hops to get to the destination
 - Shorter is better
- Updates required sending the entire routing table to all nodes
- Updates typically happened every 30 seconds
- Could take some time for routing table convergence
- RIP is easy to implement and still found in modern devices such as IoT edge sensors

▶ Border Gateway Protocol (BGP)

- BGP is an exterior gateway protocol designed to exchange routing and reachability information between autonomous systems (AS)
- Routing decisions are based on paths, network policies or rule sets made by the system administrator



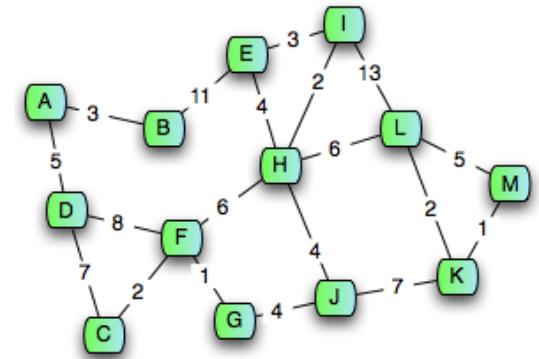
Source: phonecan.com

Dynamic Routing Protocols #2

Link-State Protocols

Open Shortest Path First (OSPF)

- Link-states take into account more than just hop count (distance, throughput, availability, etc.)
- Each router constructs a network topology in a link-state database (LSDB)
- Link-state updates are multicast to neighboring routers immediately
 - Only changes are sent, not the entire routing table
- Node reachability is determined via Dykstra's Algorithm



Source: scienceblogs.com

Dynamic Routing Protocols #3

Link-State Protocols

▶ Link-State Routing Protocol (LSRP)

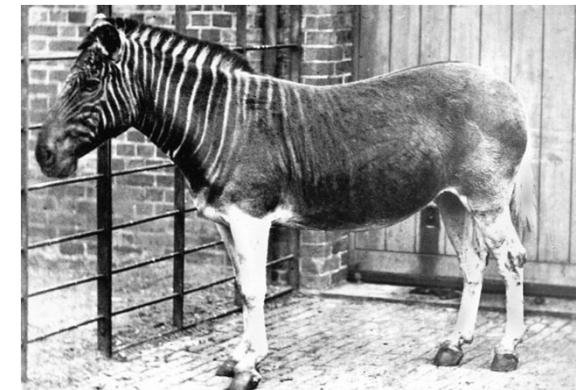
- Like OSPF, an LSRP node sends updates to all of its nearest neighbors
 - Uses simplified Dykstra's Algorithm
- Optimized for mobile, ad-hoc mesh networks

▶ Intermediate System to Intermediate System (ISIS)

- Classified as an interior gateway protocol
- Referred to as the de facto standard for large service provider network backbones
- Operated by periodically flood-routing link-state changes through the network
- Also uses Dykstra's Algorithm to build the internal link-status tables

Supporting Dynamic Routing

- ★ Ideally, we would like to learn routes dynamically
- ★ Linux supports a dynamic routing subsystem called quagga
 - ▶ Derived from the original zebra project before zebra was taken private
- ★ By the way, a quagga is an extinct sub-species of the plains zebra



Source: wikipedia.com

Quagga Architecture

★ Quagga uses a core daemon called “zebra”

- ▶ Acts as an abstraction layer to the underlying Linux kernel
- ▶ Presents a consistent API over a Unix-domain socket or TCP socket to quagga clients
 - Communicates with the routing table via netlink sockets (AF_NETLINK)
- ▶ Quagga clients implement the routing protocol and communicate routing updates to the zebra daemon

Quagga Clients

* Quagga has several existing clients that are bundled with the source:

- ▶ ospfd
 - Implements OSPFv2
- ▶ ripd
 - Implements RIP v1 and v2
- ▶ ospf6d
 - Implements OSPFv3 (IPv6)
- ▶ ripngd
 - Implements RIPng (IPv6)
- ▶ bgpd
 - Implements BGPv4+ (including address family support for multicast and IPv6)

* Quagga also offers a development library to facilitate the implementation of custom clients

- ▶ Plugins exist for LSRP, MPLS and Bidirectional Forwarding Detection (BFD)

Getting Quagga

★ Prepackaged and available for:

- ▶ Debian/Ubuntu/Mint (.deb)
- ▶ RHEL/Fedora/Suse (.rpm)

★ Source is available from www.quagga.net

- ▶ Build via standard procedure:
 - configure, make, make install
- ▶ Builds all clients by default, use configure options to disable
 - E.g., --disable-ripngd

Configuring Quagga

★ Two ways of configuring the quagga daemons:

- ▶ Configuration file

- Each daemon has its own file
- For quagga version 0.99, the default location for all the configuration files is /etc/quagga/
- You can use “-f” on the daemon command line to specify a different configuration file

- ▶ Virtual TeletYpe (vty) interface

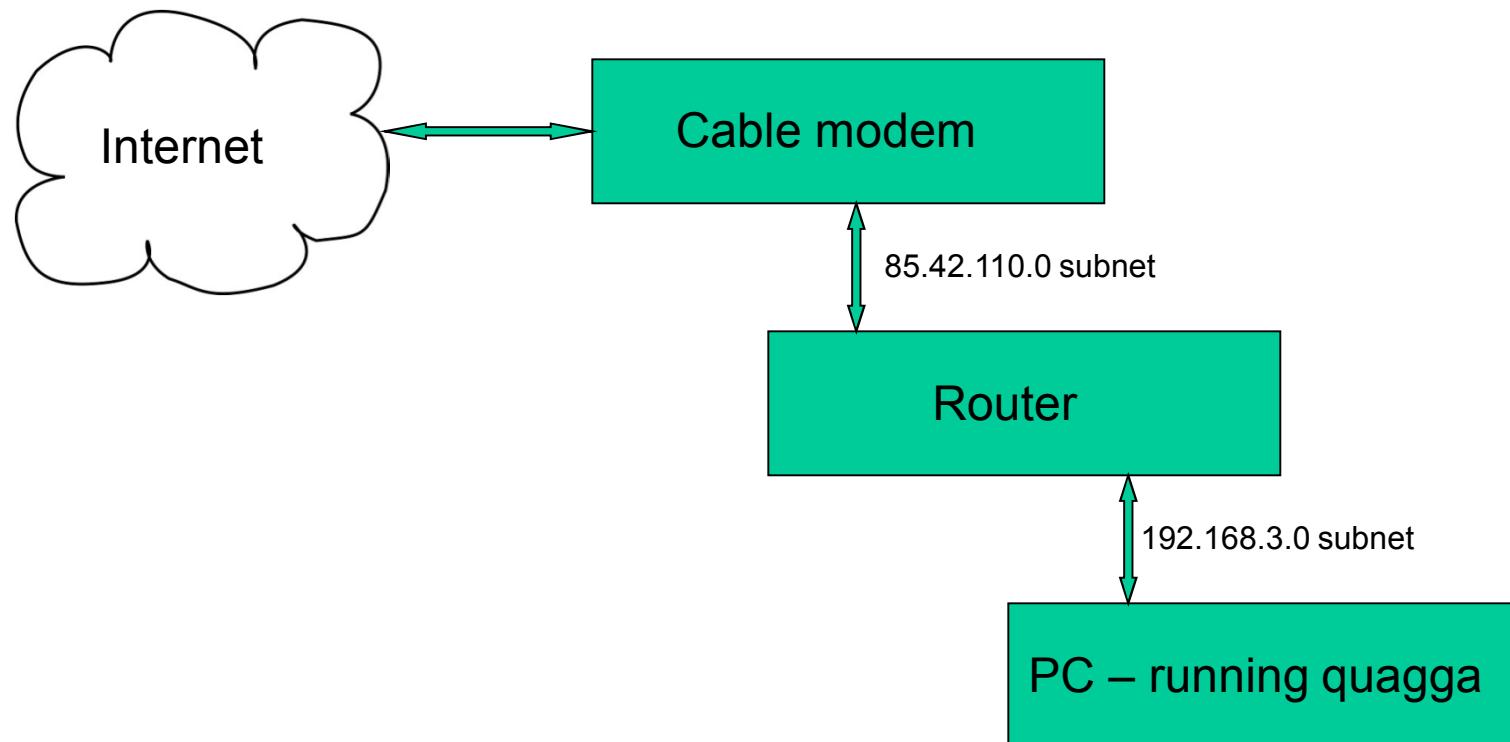
- Command line interface to running daemon
- Each daemon listens on a specific port
 - 2601 for zebra, 2602 for ripd, etc.

Quagga Configuration Example

- ★ We'll use zebra and ripd
- ★ To use the vty interface, each daemon must have a configuration file
 - ▶ The vty interface must have a password, and it must be specified in the configuration file
 - Good idea to make the configuration file readable only by root
 - ▶ You can also specify the hostname that appears as the prompt for the vty user interface

Our Example System

*Very basic setup



Zebra Configuration File

* Our zebra.conf contains:

```
password abc123
enable password abc123
hostname quagga-zebra
```

- ▶ Obviously, not a production password

* Start the zebra daemon on the command line: “**zebra -d**”

- ▶ The “**-d**” says to run as a daemon

* Connect to the zebra vty interface

- ▶ **telnet 127.0.0.1 2601**

The Zebra Vty Interface

★ Connecting to the zebra vty interface

- ▶ It is listening on the default port of 2601
- ▶ At the “Password:” prompt, we use abc123 as specified in the configuration file
- ▶ Once logged in, we have a command line interface, with the prompt as specified in the configuration file

```
# telnet 127.0.0.1 2601
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.20.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
quagga-zebra>
```

Showing Routes Using Zebra

* Show the current routes

```
quagga-zebra> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 127.0.0.0/8 is directly connected, lo
C>* 192.168.3.0/24 is directly connected, em1
```

* Note there is no default route

* “em1” is the Ethernet interface

Enabling RIP

- ★ Next, we need to enable RIP
- ★ Our RIP configuration file (**ripd.conf**) is also simple:

```
password abc456
hostname quagga-ripd
router rip
network em1
```

- ▶ Again we specify the vty password and prompt
- ▶ We also need the “router rip” command to enable RIP
- ▶ The “network em1” specifies the interface RIP should use

- ★ Start the RIP daemon

- ▶ “**ripd -d**”

The Ripd Vty Interface

- ★ Connect to the ripd vty interface on port 2602 and use the abc456 password
- ★ Use the command **show ip rip** to show the routes
- ★ Still no default route
 - ▶ Can be added using policy routing

```
# telnet 127.0.0.1 2602
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.20.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
quagga-ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

      Network          Next Hop          Metric From      Tag Time
C(i) 192.168.3.0/24    0.0.0.0           1 self          0
```

Debugging the Routing Protocol

- ✖ Our **ripd** configuration file is not set up for logging
- ✖ We can't enable logging via the vty interface
 - ▶ We need to add a line to the configuration file to enable logging and set the destination:
 - Either send log messages to stdout (only if **ripd** not in daemon mode):
 - “log stdout”
 - Or to a file:
 - “log file /tmp/riplogfile”

What are Netlink Sockets?

- ★ Netlink sockets are an IPC mechanism for communicating from user-space to kernel-space
 - ▶ Several different variants depending on the requirement
- ★ These use the AF_NETLINK protocol family
 - ▶ Protocol variants determine the application
- ★ Netlink sockets are how the zebra daemon communicates to the routing table

Netlink Protocol Variants

★ Protocol variants determine the application

- NETLINK_ROUTE
 - Access to routing daemon communications channel
- NETLINK_FIREWALL
 - Received packets sent by firewall code
- NETLINK_NFLOG
 - Communications channel for user-space iptable management and Netfilter logging
- NETLINK_ARPD
 - Manage ARP table from user space
- NETLINK_USER
 - Generic communications channel from user to kernel space and back

Example Quagga Code – rt_zebra.c

```
/* Make socket for Linux netlink interface. */
static int
netlink_socket (struct nlsock *nl, unsigned long groups)
{
    int ret;
    struct sockaddr_nl snl;
    int sock;
    int namelen;
    int save_errno;

    if (zserv_privs.change (ZPRIVS_RAISE))
    {
        zlog (NULL, LOG_ERR, "Can't raise privileges");
        return -1;
    }

    sock = socket (AF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
    if (sock < 0)
    {
        zlog (NULL, LOG_ERR, "Can't open %s socket: %s",
              nl->name,
              safe_strerror (errno));
        return -1;
    }

    memset (&snl, 0, sizeof snl);
    snl.nl_family = AF_NETLINK;
    snl.nl_groups = groups;
```

Example Quagga Code #2

```
/* Bind the socket to the netlink structure for anything. */
ret = bind (sock, (struct sockaddr *) &snl, sizeof snl);
save_errno = errno;
if (zserv_privs.change (ZPRIVS_LOWER))
    zlog (NULL, LOG_ERR, "Can't lower privileges");

if (ret < 0)
{
    zlog (NULL, LOG_ERR, "Can't bind %s socket to group 0x%lx: %s",
          nl->name, snl.nl_groups, safe_strerror (save_errno));
    close (sock);
    return -1;
}

/* multiple netlink sockets will have different nl_pid */
namelen = sizeof snl;
ret = getsockname (sock, (struct sockaddr *) &snl, (socklen_t *) &namelen);
if (ret < 0 || namelen != sizeof snl)
{
    zlog (NULL, LOG_ERR, "Can't get %s socket name: %s", nl->name,
          safe_strerror (errno));
    close (sock);
    return -1;
}

nl->snl = snl;
nl->sock = sock;
return ret;
}
```

Summary

- ✖ IP routing is the process of getting an IP packet from its source to its destination, potentially across multiple networks
- ✖ At each hop, the routing table is searched for the best match, next-hop address
- ✖ Routing may be configured statically or dynamically
 - ▶ Several dynamic routing algorithms exist that automatically calculate lowest cost paths
- ✖ Quagga is Linux's dynamic routing suite
 - ▶ It provides clients/daemons to support most of the current routing protocols

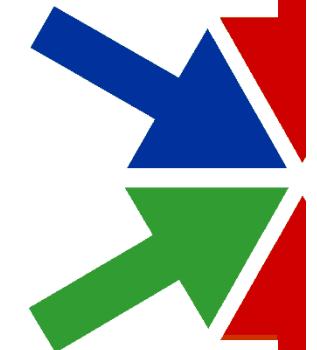
Questions

- * The IP address 0.0.0.0 is also known as “default”
 - ▶ True or False?
- * RIP is a distance vector routing protocol
 - ▶ True or False?
- * OSPF uses the Dijkstra’s algorithm for calculating the routes
 - ▶ True or False?
- * When a link state changes, OSPF rebroadcasts the entire routing table
 - ▶ True or False?
- * We can configure quagga daemons like ripd via configuration files or via a vty interface
 - ▶ True or False?

Chapter Break

Ethernet Bonding

Adding performance and reliability to networking



What We Will Cover

✖ Overview

✖ Failover Mode

✖ Bonding Policies

✖ Linux Kernel Support

✖ Configuration Examples

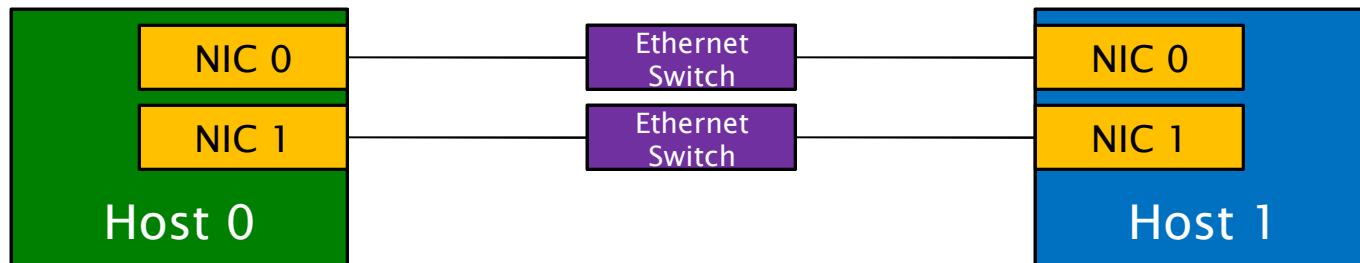
✖ Performance Testing Example

Virtual Devices

- ★ Virtual devices are an abstraction built on top of one or more real devices
- ★ There are several uses of virtual network devices in Linux:
 - ▶ Bonding
 - Bundling groups of physical interfaces to make them behave as one
 - ▶ IEEE 802.1Q
 - VLAN devices
 - ▶ Bridging
 - ▶ Aliasing Interfaces
 - Virtual NICs to handle special routing or subnet requirements
 - ▶ True Equalizer (TEQL)
 - A queuing discipline related to QoS that requires a virtual device
 - ▶ Tunnel Interfaces
 - IP-over-IP and GRE tunneling are based on virtual devices

Ethernet Bonding Overview

- Ethernet bonding is a networking configuration in which two or more network interfaces on a host are combined for redundancy and/or increased throughput
- Also known as ‘channel bonding’, ‘link aggregation’, ‘port trunking’, ‘link bundling’ or ‘NIC teaming’
- Ethernet bonding works at layer 2 of the OSI networking model (link-layer)



Network Failure Modes

- ★ At the physical level, we can detect failures in the interface card, the cable and the switch port
 - ▶ The PHY will indicate the status change to the NIC driver
- ★ Failures that occur elsewhere in the network path require end-to-end heartbeat verification in software
 - ▶ Linux has several examples of these applications
- ★ Once a failure is detected, what do you do about it?
 - ▶ Take action before the failure occurs...

Network Failover

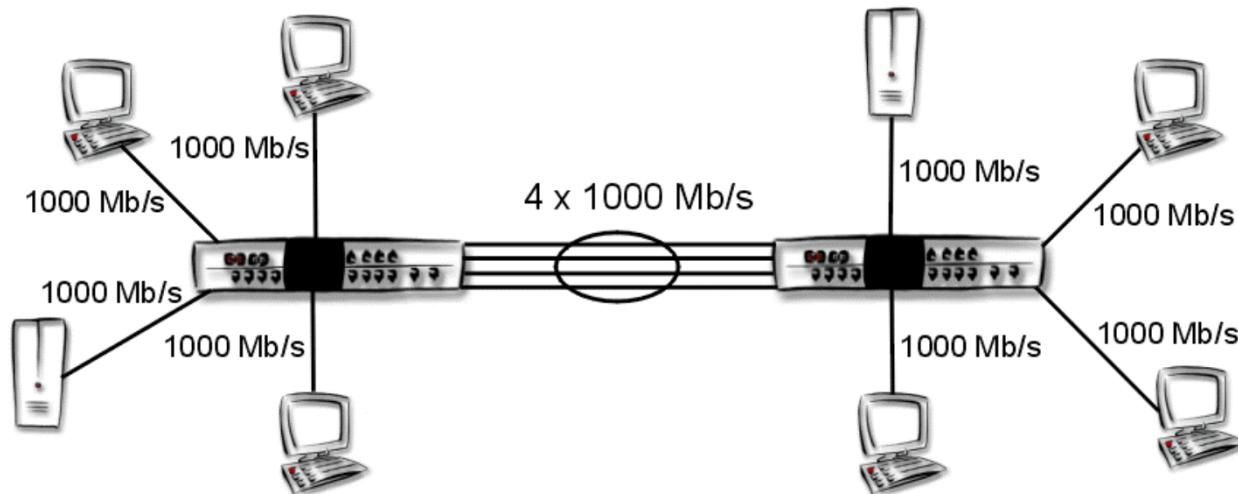
★ Link failover refers to the process by which a failure of one component of a redundant link can be transparently masked by having the link's traffic routed to another link

★ Bonding Ethernet drivers

- ▶ Permit redundant LAN connections to be grouped together to provide for link failover, link throughput aggregation, or both
- ▶ Failover is very fast (< 1ms), but detection can be slower
 - Configurable at the system jiffy rate

Network Throughput

- ★ In addition to providing fault tolerance, Ethernet bonding can provide more bandwidth to a network
- ★ It can also reduce costs by removing the need to upgrade equipment (e.g., from 1 to 10 GbE switches)
- ★ Can bond up to 8 slave ports



Source: syskconnect.com

Link Aggregation Control Protocol

- ★ LACP is part of the IEEE 802.3ad specification
- ★ Provides a method to control the bundling of several physical ports together to form a single logical channel
- ★ Allows a network device to negotiate an automatic bundling of links by sending LACP packets to a directly connected LACP-enabled peer
 - ▶ Less prone to trouble than static link aggregation

Linux Supports 7 Bonding Policies

★ 7 policies that fall into 3 categories:

- ▶ Failover only
- ▶ Requiring switch support
- ▶ Generic

★ Failover only

- ▶ Active Backup: One port is active until the link fails, then the other takes over the MAC address and becomes active
- ▶ Driver mode = 1 or active-backup



Source: stormgrounds.com

Policies Requiring Switch Support

★ Round robin

- ▶ Transmit packets in sequential order from the first available slave through the last
- ▶ Provides load balancing and fault tolerance
- ▶ Driver mode = 0 or balance-rr (default)

★ IEEE 802.3ad dynamic link aggregation

- ▶ Official standard for link aggregation, and includes many configurable options for how to balance the traffic
- ▶ Driver mode = 4 or 802.3ad

★ XOR

- ▶ Traffic is hashed and balanced over available links
- ▶ This mode is also available as part of 802.3ad
- ▶ Provides load balancing and fault tolerance
- ▶ Driver mode = 2 or balance-xor

Generic Policies

Broadcast

- ▶ Not really link aggregation
- ▶ Broadcasts all traffic out all interfaces in the bond
- ▶ Can be useful when sending data to partitioned broadcast domains for high availability
- ▶ If using broadcast mode on a single network, switch support is recommended
- ▶ Driver mode = 3 or broadcast

Generic Policies (2)

★ Adaptive Transmit Load Balancing

- ▶ Outgoing traffic is distributed according to the current load on each slave
- ▶ Incoming traffic is received by current slave
- ▶ If receiving slave fails, another slave takes over the MAC address of the failed receiving slave
- ▶ Driver mode = 5 or balance-tlb

★ Adaptive Load Balancing

- ▶ Includes adaptive transmit and receive load balancing
- ▶ Driver mode = 6 or balance-alb

Which Policy is Right for You?

- ★ If you have switches that support 802.3ad and you have no special needs, use 802.3ad mode
- ★ If you have no switch support and just want to increase throughput and enable failover, use balance-alb mode
- ★ If you just need a data replication link between two servers, use balance-rr mode
- ★ Sometimes experimentation is required to find the right mode for your application

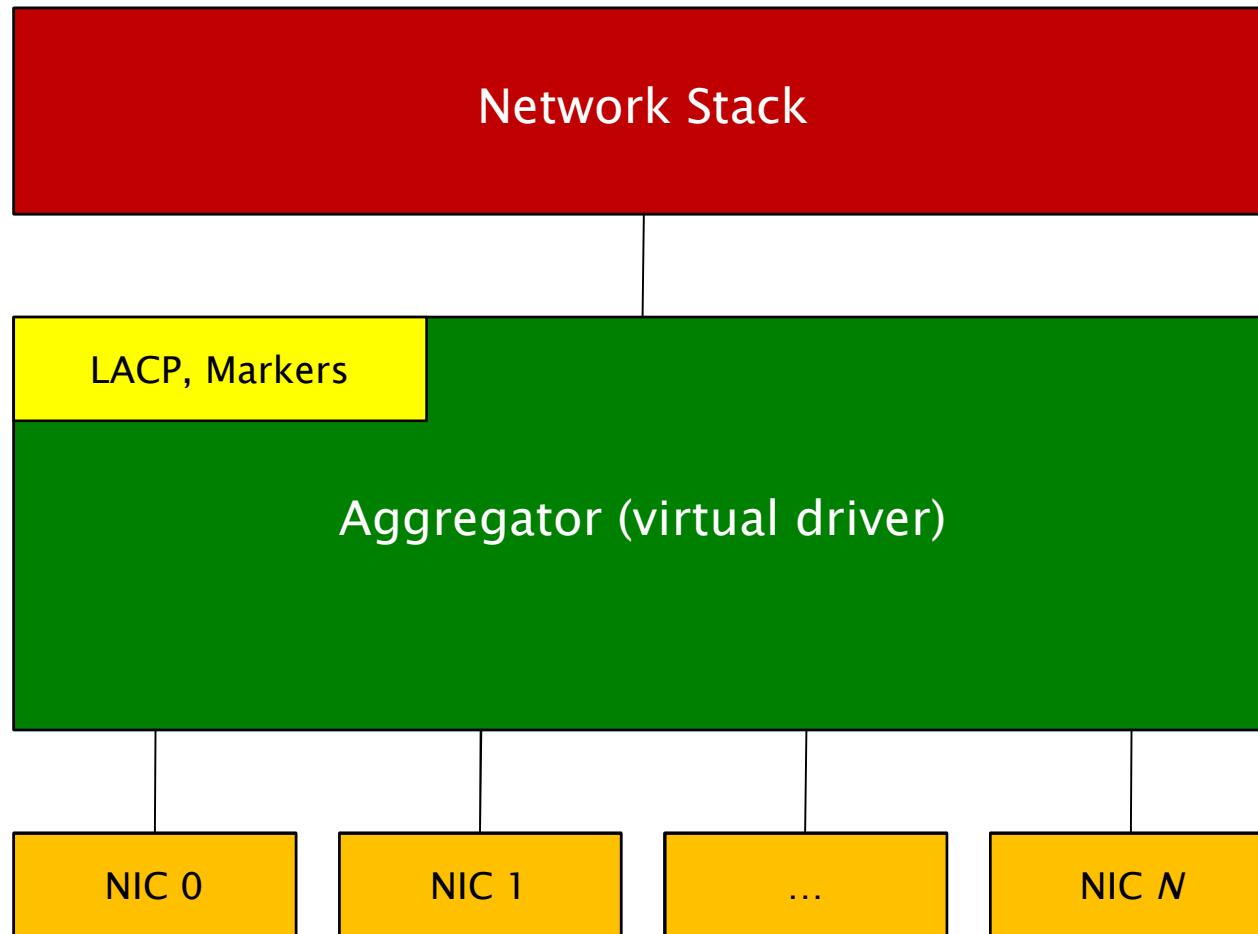
Linux Support

★ Linux supports bonding with the bonding kernel module and the **ifenslave** utility

- ▶ Configuration methods vary per Linux distribution and versions of a distribution
 - **ifenslave** is already included in the RedHat distro, but may need to be installed for Debian-based systems

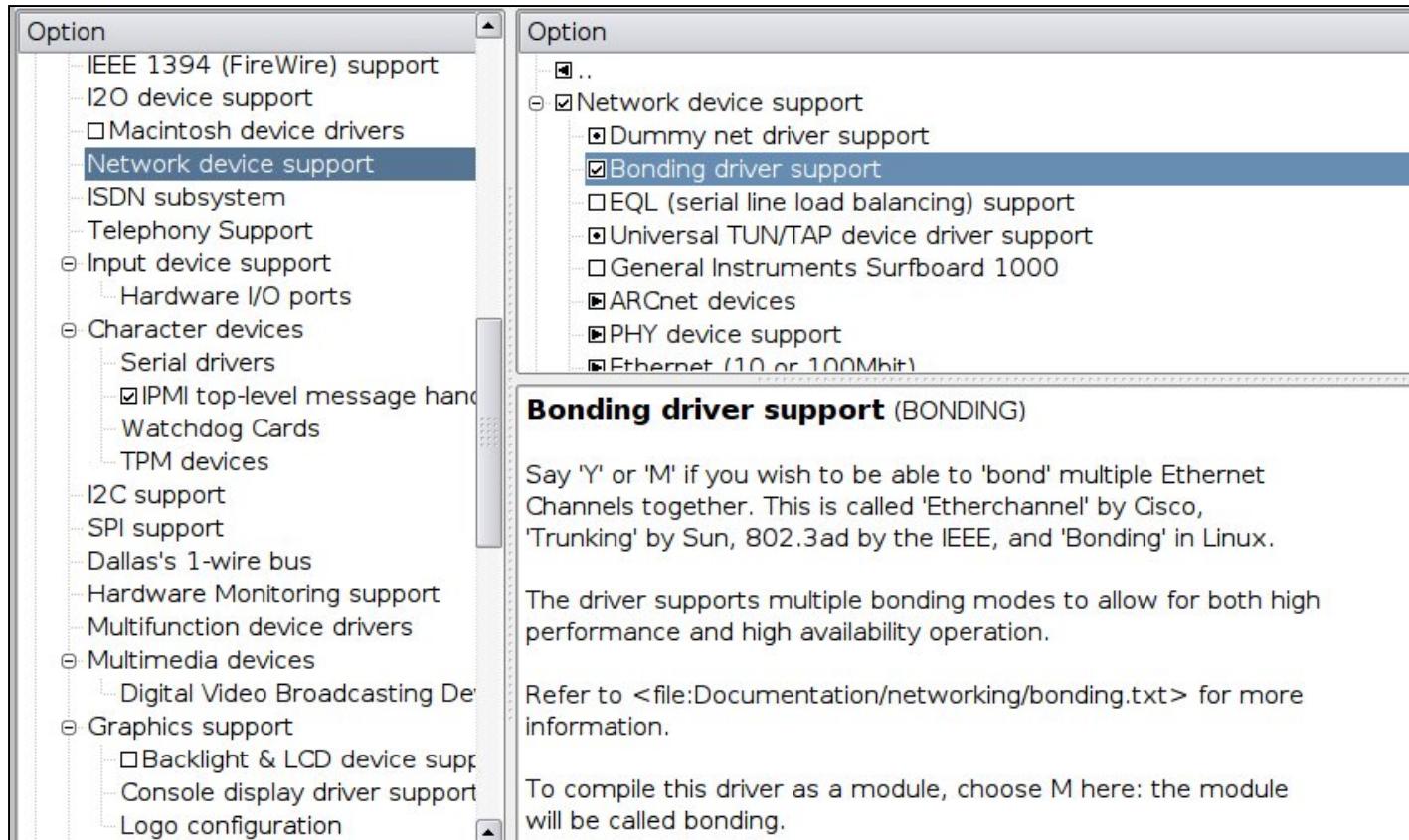
★ Note that the bonding policy is set at the time the bonding module is loaded

Driver Implementation



Enabling in the Linux Kernel

- ★ Bonding support is in the network device support section of the kernel configuration parameters



Bonding Configuration

- ★ Bonding can be configured manually or automatically
 - ▶ Manual configuration uses either **ifenslave** or the **sysfs** interface
 - Options are supplied as parameters to the bonding module at load time
 - ▶ Network initialization scripts are used for automatic configuration
 - E.g., **/etc/sysconfig/network-scripts/ifcfg-bondX**

Bonding Options

★ Supported options

- ▶ arp_interval
 - The ARP link monitoring period (in ms)
- ▶ arp_ip_target
 - The IP addresses to use as ARP monitoring peers when arp_interval is greater than zero
- ▶ arp_validate
 - Specifies whether or not ARP probes and replies should be validated in the active-backup mode
- ▶ downdelay
 - The time (in ms) to wait before disabling a failed slave
- ▶ fail_over_mac
 - Specifies whether active-backup mode should set all slaves to the same MAC address
- ▶ lacp_rate
 - The rate at which the link partner is asked to transmit LACPDU packets in 802.3ad mode



Source: wizbangblog.com

Bonding Options (2)

★ Supported options (continued)

- ▶ `max_bonds`
 - The number of bonding devices to create
- ▶ `miimon`
 - The MII link monitoring period (in ms)
- ▶ `mode`
 - Specifies the bonding policy
- ▶ `primary`
 - A string specifying which slave is the primary device
- ▶ `updelay`
 - The time (in ms) to wait before enabling a recovered slave
- ▶ `use_carrier`
 - Specifies whether MII or ethtool is used to determine link status
- ▶ `xmit_hash_policy`
 - Selects the transmit hash policy to use for slave selection in balance-xor and 802.3ad modes

Configuration Examples

- ★ Assumes RHEL 5 or greater
- ★ Manual and automatic cases
- ★ Bind eth2 & eth3 together in Active Backup mode
- ★ Use 00:80:c8:e7:ab:5c as the aggregated channel's MAC and 192.168.100.33 as its IP address
- ★ Monitor link status every 100 ms
- ★ Wait 200 ms to react to a change in a slave port's state

Manual Configuration Example

```
[root]# modprobe bonding mode=1 miimon=100 downdelay=200 updelay=200
[root]# ip link set dev bond0 addr 00:80:c8:e7:ab:5c
[root]# ip addr add 192.168.100.33/24 brd + dev bond0
[root]# ip link set dev bond0 up
[root]# ifenslave bond0 eth2 eth3
The interface eth2 is up, shutting it down to enslave it.
The interface eth3 is up, shutting it down to enslave it.
[root]# ip link show eth2 ; ip link show eth3 ; ip link show bond0
4: eth2: <BROADCAST,MULTICAST,SLAVE,UP> mtu 1500 qdisc pfifo_fast
    master bond0 qlen 100
    link/ether 00:80:c8:e7:ab:5c brd ff:ff:ff:ff:ff:ff
5: eth3:
    <BROADCAST,MULTICAST,NOARP,SLAVE,DEBUG,AUTOMEDIA,PORTSEL,NOTRAILERS,
     UP> mtu 1500 qdisc pfifo_fast master bond0 qlen 100
    link/ether 00:80:c8:e7:ab:5c brd ff:ff:ff:ff:ff:ff
58: bond0: <BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc noqueue
    link/ether 00:80:c8:e7:ab:5c brd ff:ff:ff:ff:ff:ff
```

Automatic Configuration

- * Create a file named:
`/etc/sysconfig/network-scripts/ifcfg-eth2`
with the following contents:

```
DEVICE=eth2  
USERCTL=no  
ONBOOT=yes  
MASTER=bond0  
SLAVE=yes  
BOOTPROTO=none
```

- * Create a similar file for eth3 where:

```
DEVICE=eth3
```

Automatic Configuration (2)

- * Create a file named
`/etc/sysconfig/network-scripts/ifcfg-bond0`
with the following contents:

```
DEVICE=bond0
IPADDR=192.168.100.33
NETMASK=255.255.255.0
NETWORK=192.168.100.0
BROADCAST=192.168.100.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
BONDING_OPTS="mode=1 miimon=100
               downdelay=200 updelay=200"
```

- * Finally restart networking as root

```
$ sudo /etc/rc.d/init.d/network restart
```

Verifying the Configuration

*Check /proc/net/bonding/bond0

- ▶ Example: mode=1 with miimon=100

```
Ethernet Channel Bonding Driver: 2.6.1 (October 29, 2004)
```

```
Bonding Mode: Active Backup
```

```
Currently Active Slave: eth2
```

```
MII Status: up
```

```
MII Polling Interval (ms): 100
```

```
Up Delay (ms): 200
```

```
Down Delay (ms): 200
```

```
Slave Interface: eth2
```

```
MII Status: up
```

```
Link Failure Count: 1
```

```
Slave Interface: eth3
```

```
MII Status: up
```

```
Link Failure Count: 1
```

Performance Testing

- ★ Uses iperf to generate and collect data, and Cacti to plot it
- ★ Test network setup:
 - ▶ 1 server host
 - ▶ 2 client hosts
 - ▶ Cisco 802.3ad compatible switch
- ★ Switch was configured to place the ports into a dynamic LACP bond
- ★ Bonding configured via the kernel module parameters:
 - ▶ mode=4 miimon=100 max_bonds=4 xmit_hash_policy=1

Performance Testing (2)

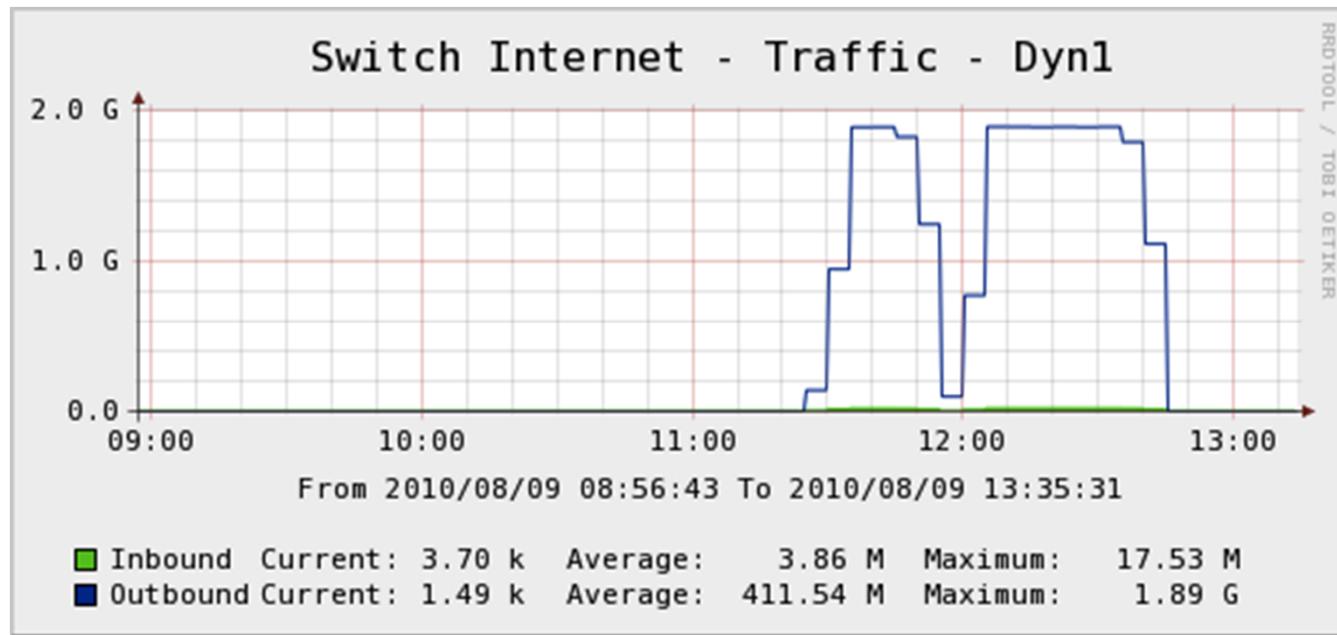
- * Run iperf on the server host

```
$ iperf -s -i 2
```

- * Run iperf on each client

```
$ cat /dev/zero | iperf -c svr.example.com -t 2400 -i 2
```

- * Aggregating 2 ports almost provides 2x peak performance increase



Summary

- ★ Ethernet bonding provides a low-cost solution for increasing network reliability and throughput
- ★ There are many bonding policies
 - ▶ The right choice for you may have to be found through experimentation
- ★ Linux fully supports bonding in the kernel network stack
- ★ Linux distros provide tools to create and manage bonds

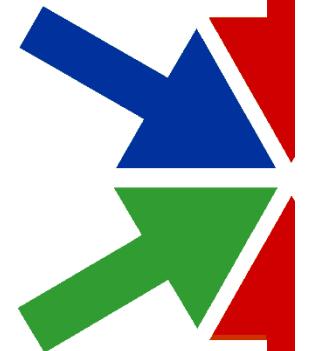
Questions

- ★ Bonding is also known as “link-aggregation” or “port trunking”
 - ▶ True or False?
- ★ Bonding supports 7 basic policies that fall into 3 categories
 - ▶ True or False?
- ★ In order to enable support for bonding, you need to enable bonding support in the Linux kernel and rebuild the kernel
 - ▶ True or False?
- ★ Active-backup is one of the supported bonding modes
 - ▶ True or False?
- ★ The bonding parameter “updelay” is the time (in ms) to wait before disabling a failed slave
 - ▶ True or False?

Chapter Break

IP Multicasting

Getting the message out...

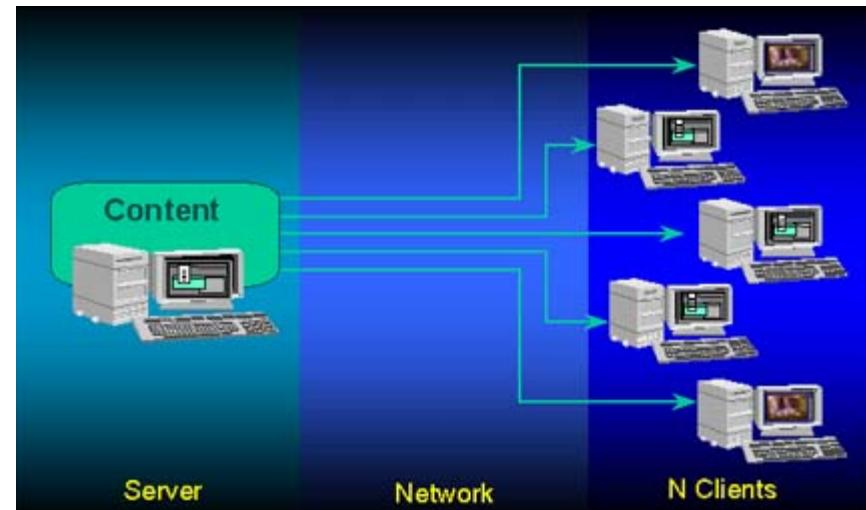


What We Will Cover

- ★ Unicast vs. multicast
- ★ Host groups
- ★ Multicast addresses
- ★ Any-source vs source-specific multicast
- ★ Internet Group Management Protocol
- ★ Multicast routing

Unicast

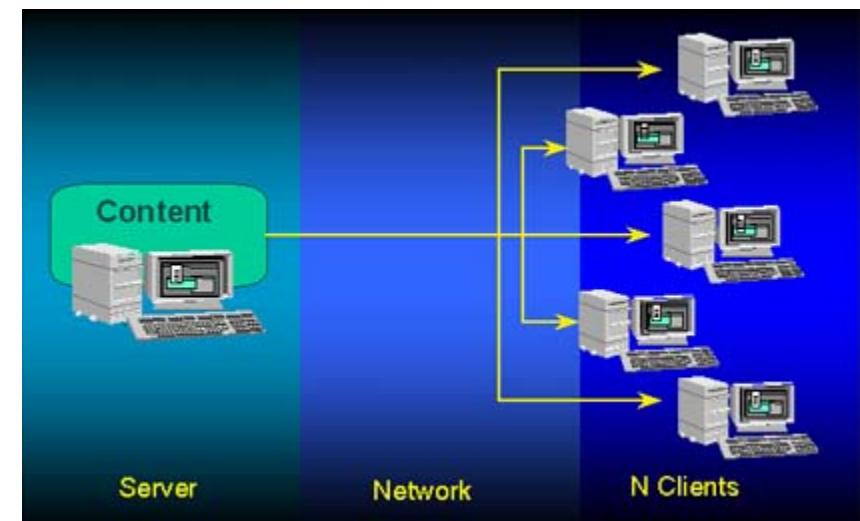
- ★ Each packet goes to just one receiver
- ★ Sending data, say a video stream, to N receivers means N video streams on the network



Source: Cisco

Multicast

- ★ Each packet can go to multiple receivers
- ★ Sending our video stream to N receivers means a single video stream on the network
- ★ Reduces network traffic
- ★ Reduces server workload
- ★ Uses UDP as the transport protocol



Source: Cisco

Multicast Basics – Host Group

- ★ A set of hosts identified by a single multicast address
- ★ Can be permanent or transient
 - ▶ A permanent group has a well-known, administratively-assigned multicast address
 - May have zero members
 - ▶ A transient group is given a dynamically-assigned multicast address
 - A transient group exists only as long as it has members

More About Host Groups

- ✖ Membership in the group is dynamic
 - ▶ Hosts may join or leave at any time
- ✖ Datagrams sent to the destination address are delivered to all hosts in the group
 - ▶ Delivery is not guaranteed
- ✖ Senders have no knowledge of receivers
 - ▶ Getting the datagram to the receiver is the job of the multicast-aware routers between the sender and the receivers

Host Group / Multicast Addresses

* IPv4 multicast address range is
224.0.0.0 – 239.255.255.255

- ▶ In CIDR notation: 224.0.0.0/4

* IPv6 multicast address range is FF00::/8

* Ethernet

- ▶ A ‘1’ in the LSB of the first octet indicates a multicast address
- ▶ The broadcast address (FF:FF:FF:FF:FF:FF) is actually a multicast address
 - Generally handled by Ethernet hardware the same as any other multicast address

Specific IPv4 Addresses

* 224.0.0.0 – 224.0.0.255 (assigned by IANA)

- ▶ Local subnet only
- ▶ Examples
 - 224.0.0.5: Open Shortest Path First (OSPF)
 - 224.0.0.9: Routing Information Protocol (RIP) v2
 - 224.0.0.22: Internet Group Management Protocol (IGMP) v3
 - 224.0.0.251: Multicast DNS (mDNS)

* 224.0.1.0 – 224.0.1.255 (assigned by IANA)

- ▶ Internetwork Control Block, permitted on Internet
- ▶ Example
 - 224.0.1.1: Network Time Protocol (NTP)

Specific IPv4 Addresses (2)

* 232.0.0.0 – 232.255.255.255

- ▶ Source-specific multicast

- More on this later

* 233.0.0.0 – 233.255.255.255

- ▶ GLOP Addressing

- Middle octets formed from 16-bit Autonomous System Number (ASN) assigned to organization
 - Every organization with an ASN gets 256 multicast addresses to use
 - 32-bit ASNs are *not* supported

Specific IPv4 Addresses (3)

* 234.0.0.0 – 234.255.255.255

► Unicast–Prefix–Based (RFC 6034)

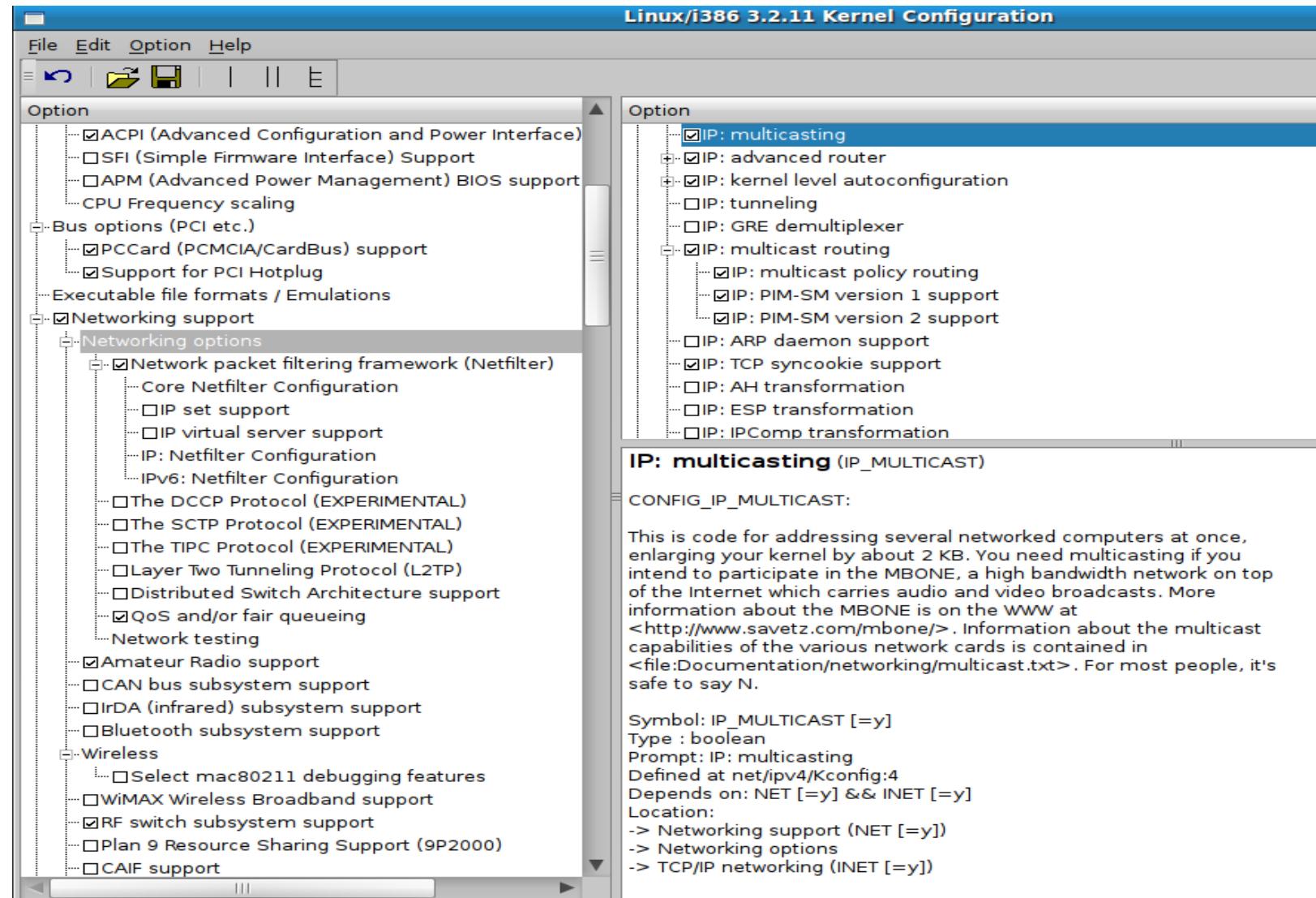
- Addresses some shortcomings of GLOP
- Given to organizations with a /24 prefix or shorter
 - In other words, must own at least 256 IP addresses
- Does not require coordination with other entities
 - Will route across the Internet
- The bigger the unicast block owned, the bigger the multicast group
 - 192.0.2.0/24 would get 234.192.0.2
 - 192.10.0.0/16 would get 234.192.10.0/24

* 239.0.0.0 – 239.255.255.255

► Administratively scoped

- Traffic must not cross administrative boundaries
- Addresses locally assigned, not globally unique

Configuring IP-Multicast in Kernel



- IP: multicasting
- IP: advanced router
- IP: kernel level autoconfiguration
- IP: tunneling
- IP: GRE demultiplexer
- IP: multicast routing
 - IP: multicast policy routing
 - IP: PIM-SM version 1 support
 - IP: PIM-SM version 2 support
- IP: ARP daemon support
- IP: TCP syncookie support
- IP: AH transformation
- IP: ESP transformation
- IP: IPComp transformation

IP: multicasting (IP_MULTICAST)

CONFIG_IP_MULTICAST:

This is code for addressing several networked computers at once, enlarging your kernel by about 2 KB. You need multicasting if you intend to participate in the MBONE, a high bandwidth network on top of the Internet which carries audio and video broadcasts. More information about the MBONE is on the WWW at <<http://www.savetz.com/mbone/>>. Information about the multicast capabilities of the various network cards is contained in <file:Documentation/networking/multicast.txt>. For most people, it's safe to say N.

Symbol: IP_MULTICAST [=y]
Type : boolean
Prompt: IP: multicasting
Defined at net/ipv4/Kconfig:4
Depends on: NET [=y] && INET [=y]
Location:
-> Networking support (NET [=y])
-> Networking options
-> TCP/IP networking (INET [=y])

Any-source vs. Source-specific

* Any-source multicast

- ▶ In general, after joining a host group, the receiver receives any traffic destined for that multicast address, regardless of the source

* Source-specific multicast

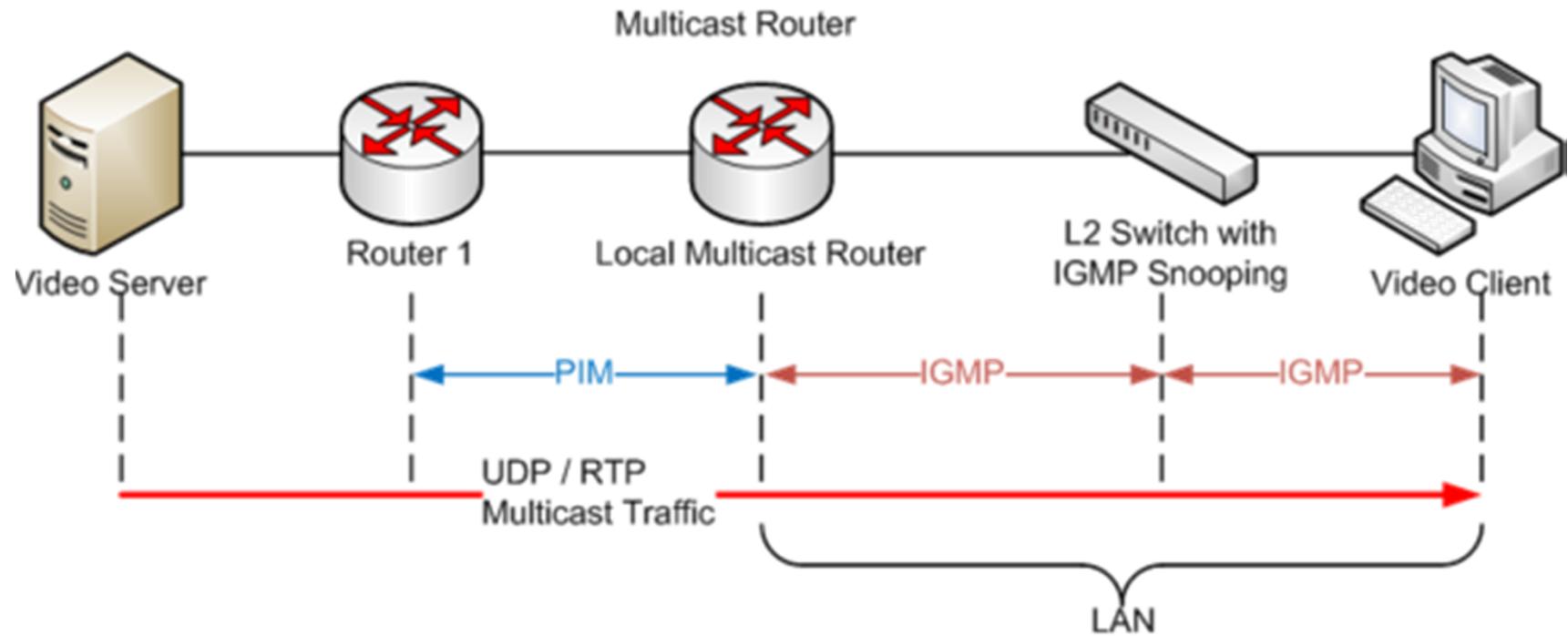
- ▶ The receiver specifies not just the destination address, but also a source address
- ▶ The destination address must be in the range 232.0.0.0/8 for IPv4, FF3x::/32 for IPv6

Internet Group Management Protocol

- ★ IGMP is used by hosts and immediately neighboring multicast routers
 - ▶ TTL is set to 1
- ★ Allows the multicast router to keep track of multicast group membership of hosts connected to it
- ★ Uses IP datagrams with a protocol of 2
 - ▶ (Protocol 1 is ICMP, 6 is TCP, 17 is UDP)
- ★ IGMP is IPv4 only
 - ▶ IPv6 uses Multicast Listener Discovery (MLD)

IGMP (2)

*IGMP traffic in a multicast network



Source: wikipedia.com

IGMP Message Types

*Membership query, type 0x11

- ▶ Periodically sent by multicast routers

*Membership report

- ▶ Type number depends on IGMP version
 - Type 0x12 in v1
 - Type 0x16 in v2
 - Type 0x22 in v3
- ▶ Sent by hosts in response to query, or whenever an interface state change occurs, such as when joining or leaving a group

*Leave group, type 0x17

IGMPv3 Membership Query

- ★ Sent by multicast routers to determine which multicast addresses are of interest to systems attached to this network segment
- ★ Routers periodically send General Queries to refresh the group membership state
- ★ Group-Specific Queries are used to determine the reception state for a given multicast address
- ★ Group-and-Source-Specific Queries are used to determine which systems want messages from a given multicast group

IGMPv3 Membership Query

- ★ Max Resp Code
 - ▶ Time in .1 secs allowed before sending a report
- ★ Group Address
 - ▶ Multicast address being queried
- ★ S
 - ▶ Routers should suppress the normal timer updates
- ★ Querier's Robustness Variable
 - ▶ Router's should update their Robustness Variable to match the latest Query
- ★ Querier's Query Interval Code
 - ▶ Time in seconds for the Query Interval
- ★ Number of Sources
 - ▶ Number of source addresses in the Query
 - For General or Group-Specific Queries, this value is 0
 - For Group-and-Source-Specific Queries, this is non-0
- ★ Source Addresses
 - ▶ Limited by the network MTU, but contains IP unicast addresses of interested parties

bit offset	0–3	4	5–7	8–15	16–31
0	Type = 0x11			Max Resp Code	Checksum
32				Group Address	
64	Resv	S	QRV	QQIC	Number of Sources (N)
96				Source Address [1]	
128				Source Address [2]	
				...	
				Source Address [N]	

Source: wikipedia.com

IGMP Snooping

- When multicast traffic arrives at a switch on the local network, it will by default flood all its ports with the traffic
- For ports with no listeners, this is a waste of bandwidth
- A switch that has IGMP snooping will monitor IGMP traffic between the hosts and router to know where listeners are
 - It will only forward traffic to links with listeners

Multicast Routing

- ❖ Purpose is to configure the multicast forwarding state in the routers between multicast listener and the source
- ❖ Routes between a source and its listeners are managed as trees
 - ▶ The source is the root
 - ▶ Listeners are leaves
 - ▶ Branches can be pruned when there are no more leaves listening
- ❖ View routes via “**ip mroute show**”



Source: all-hit-movies.blogspot.com

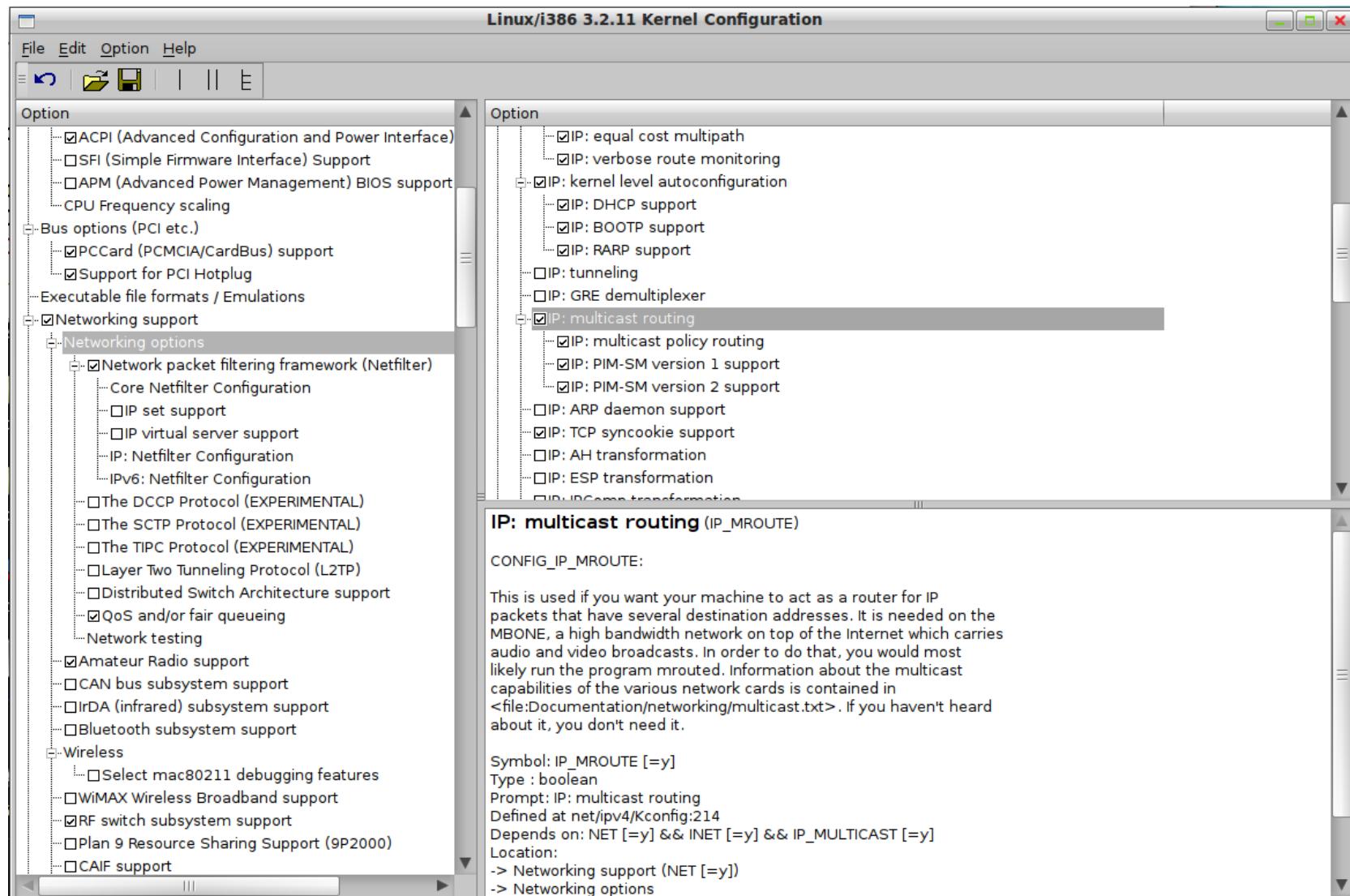
Multicast Routing Protocols

- ★ Protocol Independent Multicast (PIM)
 - ▶ The de-facto standard
- ★ There were multiple alternative multicast routing protocols that existed at one point or another
 - ▶ However, they are all now defunct
- ★ PIM does not require a specific topology discovery mechanism
 - ▶ Uses underlying unicast routing information
 - ▶ Also may use Multiprotocol Border Gateway Protocol (MP-BGP or MBGP) to distribute topology information when the multicast topology differs from the unicast topology
- ★ Two basic flavors
 - ▶ Dense mode (PIM-DM)
 - ▶ Sparse mode (PIM-SM)
 - SM is the most commonly used protocol
 - Bidirectional PIM (BIDIR-PIM) is a variant

PIM-SM

- ★ The root of the tree is at the sender for source-specific multicast, otherwise at a rendezvous point (RP)
 - ▶ The RP receives datagrams from multiple senders and distributes them to receivers
- ★ In initial state, no one receives multicast traffic
- ★ When a receiver joins a group, the forwarding state is configured, and traffic will be routed to the receiver

Enabling PIM-SM in the Kernel



PIM-DM

- Unlike SM, initially assumes that everyone wants all multicast traffic
- Everybody is joined by default
- Branches pruned when all hosts on that branch have indicated they don't want the multicast traffic
- Makes sense in a network densely-populated with interested hosts
 - ▶ Hence the name

IP Multicast API

- ★ In order to work with multicast addresses, user code must join/leave the multicast group
 - ▶ This uses the `setsockopt()` / `getsockopt()` interface
- ★ This works only for `SOCK_DGRAM` sockets
- ★ The option names are:

`IP_MULTICAST_LOOP`

`IP_MULTICAST_IF`

`IP_DROP_MEMBERSHIP`

`IP_MULTICAST_TTL`

`IP_ADD_MEMBERSHIP`

Multicast Options

* **IP_MULTICAST_LOOP**

- ▶ Set if you want a copy of your multicast messages

* **IP_MULTICAST_TTL**

- ▶ If not specified, multicast messages have a TTL of 1

* **IP_MULTICAST_IF**

- ▶ Address of local interface for use in multicast

* **IP_ADD_MEMBERSHIP**

- ▶ Used to tell the kernel what multicast groups you want to join

* **IP_DROP_MEMBERSHIP**

- ▶ Tell the kernel to drop this multicast group

Example Code

- * Before joining a group, you will need to fill out an `ip_mreq` structure (from `in.h`)

```
struct ip_mreq {  
    struct in_addr imr_multiaddr; /* address of group */  
    struct in_addr imr_interface; /* local interface */  
};
```

- * Then use `setsockopt()` to register with the kernel:

```
setsockopt (socket, IPPROTO_IP, IP_ADD_MEMBERSHIP,  
            &mreq, sizeof(mreq));
```

- * To drop membership, use the `IP_DROP_MEMBERSHIP` option in the above code snippet

Summary

- ★ Multicast is a very efficient way to distribute content to multiple receivers
- ★ There are numerous protocols to handle multicast routing, group management, etc.
 - ▶ The implementation details of these are very complex
- ★ IPv6 handles multicast natively, but draws on protocols used for IPv4 multicast

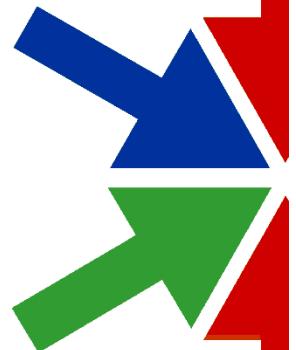
Questions

- * IP multicast is TCP based
 - ▶ True or False?
- * With GLOP addressing, 32-bit ASNs are permitted
 - ▶ True or False?
- * IPv4 multicast addresses range from 224.0.0.0 to 239.255.255.255
 - ▶ True or False?
- * IGMP is IPv4 only
 - ▶ True or False?
- * IP Multicast guarantees reliable message delivery
 - ▶ True or False?

Chapter Break

Introduction to the Frame Queuing Model

Frame Queuing Model



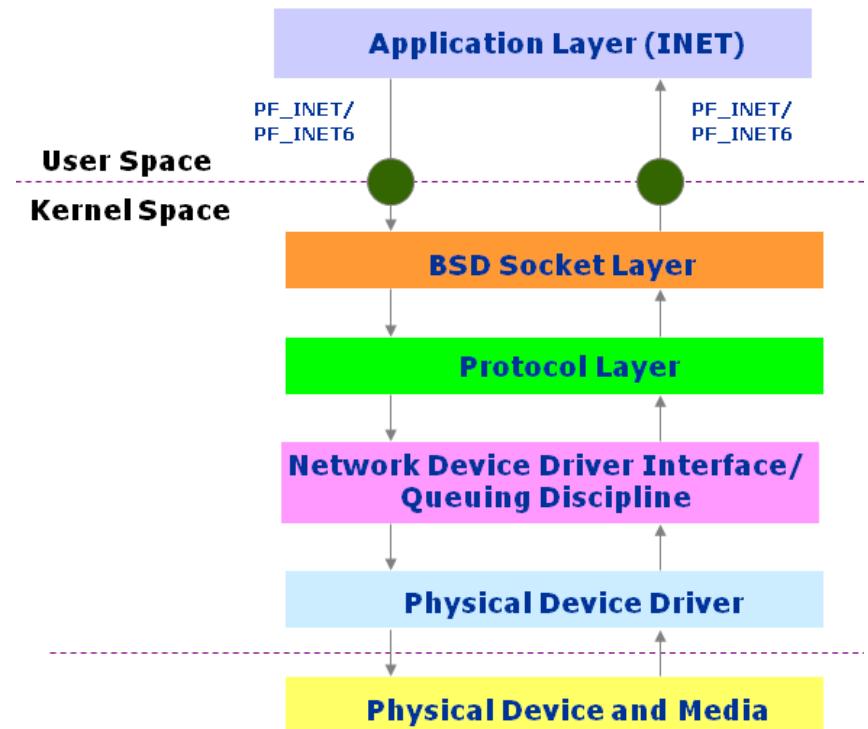
Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

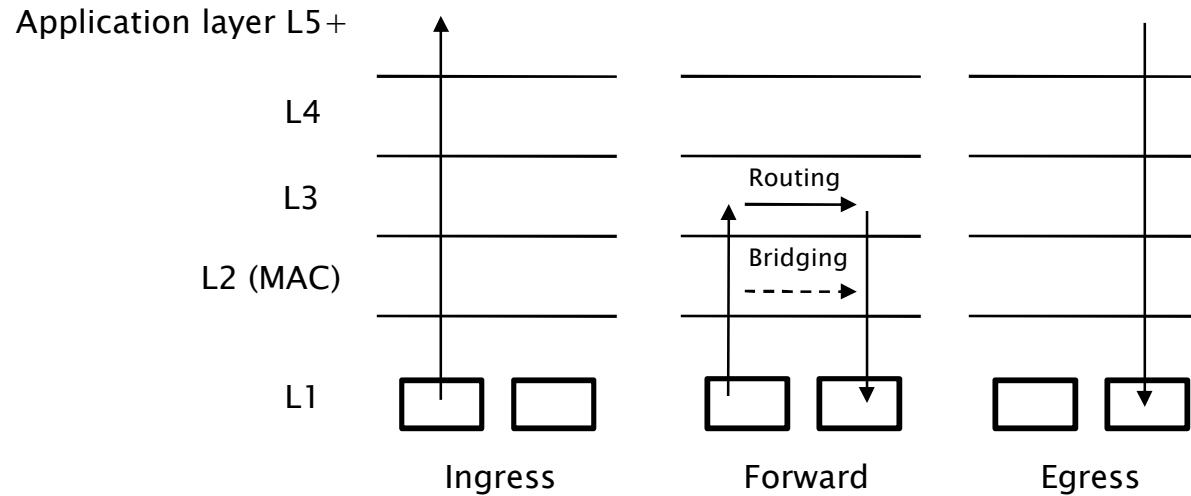
- ★ Introduction to the frame queuing model
- ★ Ingress frame flow
- ★ Egress frame flow
- ★ Services supported in this model

Where are we?

- We've already talked about the interface to the POSIX/BSD socket layer and the physical driver
- Next, we will look at the driver interface and the network queuing model
 - The inbound and outbound frame handling

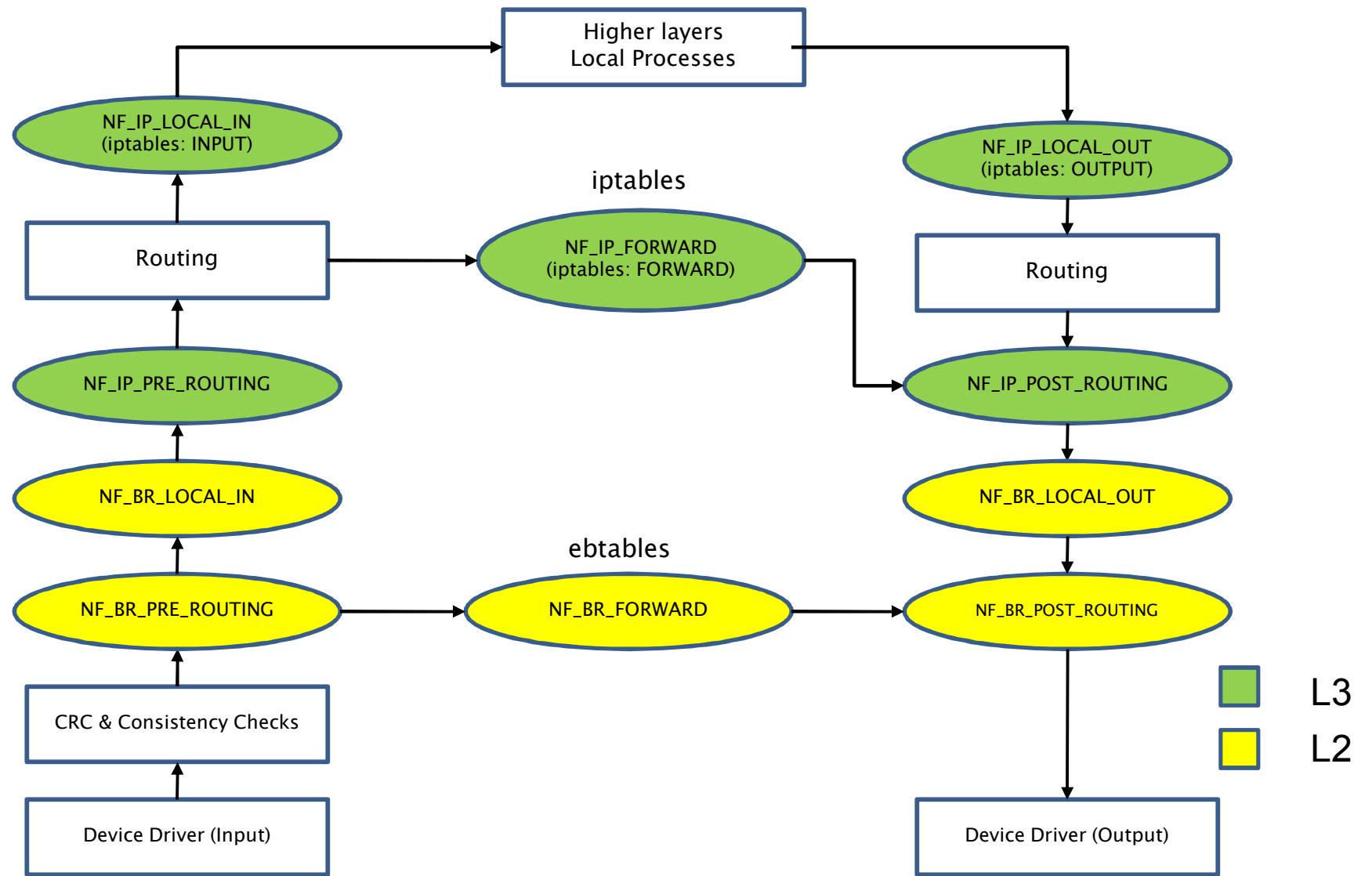


Traffic Direction

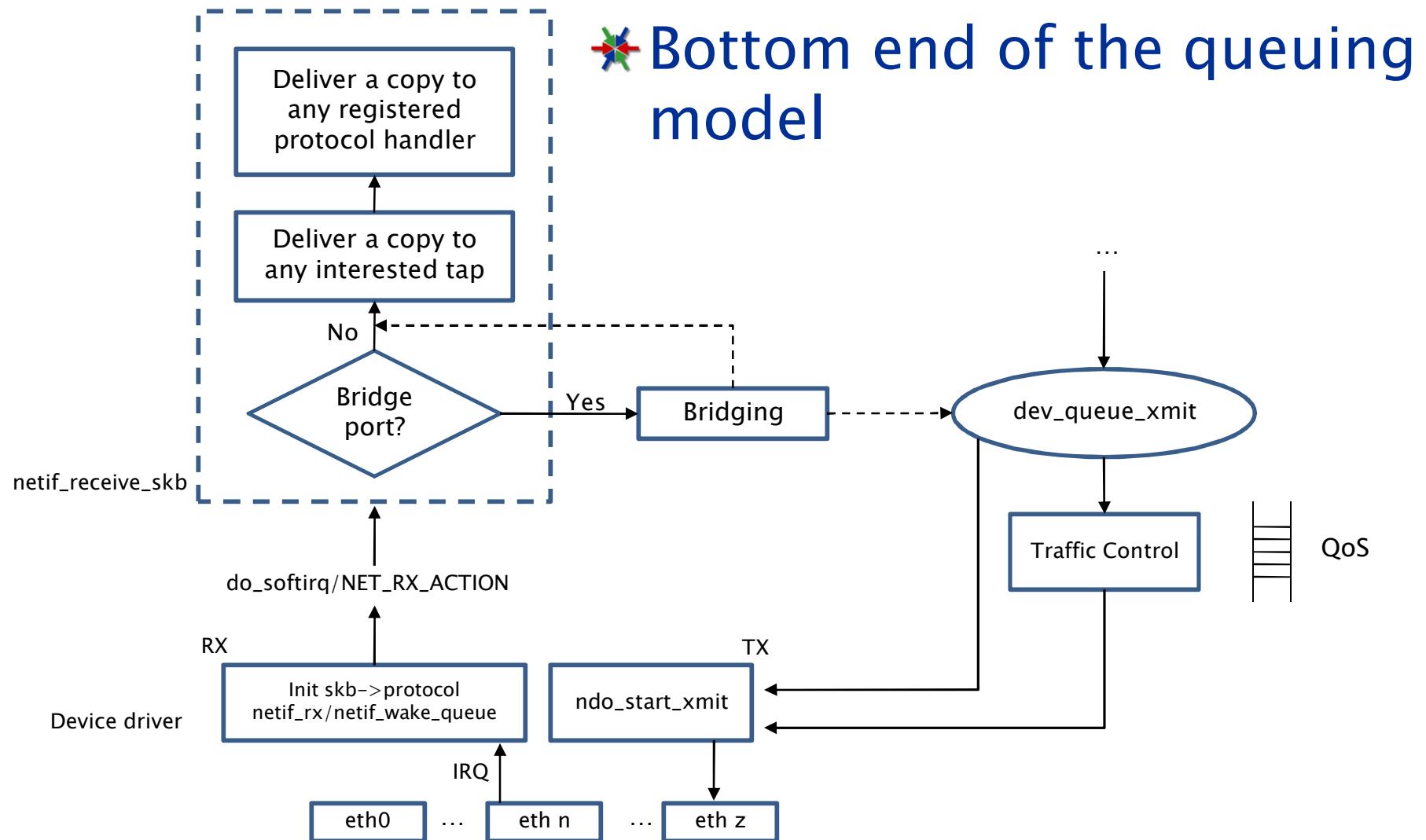


- * The paths taken in the stack by network frames differ for the inbound (ingress), routed/bridged and outbound (egress) data streams
- * There are multiple decision points along the way that determine the fate and path of network frames
- * These include the potential of simply dropping the packet at several decision points

Deciding the Fate of Frames



Low-Level Frame Handling Detail



Bottom end of the queuing model

Hooks in the Stack

- ★ Each of the green/yellow ovals on the earlier page represent places where we can hook into the stack-processing flow
 - ▶ We need to write a KLM to perform the hook and insert our code
- ★ However, there are already several services that use these mechanisms:
 - ▶ Quality of Service
 - ▶ Firewalls/NAT
 - ▶ VLANs
 - ▶ Bridging
- ★ We will be addressing these topics in the next few chapters

Example Hook Module

- Here is an example of hooking in the stack to simply drop all UDP traffic (source <http://www.paulkiddie.com>)

```
// 'Hello World' v2 netfilter hooks example that drops UDP (protocol 17)

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/skbuff.h>
#include <linux/udp.h>
#include <linux/ip.h>

static struct nf_hook_ops nfho;      //net filter hook option struct
struct sk_buff *sock_buff;
struct udphdr *udp_header;          //udp header struct (not used)
struct iphdr *ip_header;            //ip header struct

int init_module() {
    nfho.hook = hook_func;
    nfho.hooknum = NF_IP_PRE_ROUTING;
    nfho(pf = PF_INET;
    nfho.priority = NF_IP_PRI_FIRST;
    nf_register_hook(&nfho);

    return 0;
}
```

Example Hook Module #2

```
unsigned int hook_func(unsigned int hooknum, struct sk_buff **skb,
                      const struct net_device *in,
                      const struct net_device *out,
                      int (*okfn)(struct sk_buff *)) {
    sock_buff = *skb;

    // grab network header using accessor
    ip_header = (struct iphdr *)skb_network_header(sock_buff);

    if(!sock_buff) { return NF_ACCEPT; }

    // grab transport header
    if (ip_header->protocol==17) {
        udp_header = (struct udphdr *)skb_transport_header(sock_buff);
        // log we've got udp packet to /var/log/messages
        printk(KERN_INFO "got udp packet \n");

        return NF_DROP;
    }

    return NF_ACCEPT;
}

void cleanup_module() {
    nf_unregister_hook(&nfho);
}
```

Summary

- ★ Packet/frame flows are split into ingress (input) and egress (output) flows
- ★ There are several places in the queuing model where the fate of a packet is decided
 - ▶ We will examine these in greater detail in subsequent chapters
- ★ We will need to write a kernel loadable module in order to hook into the stack

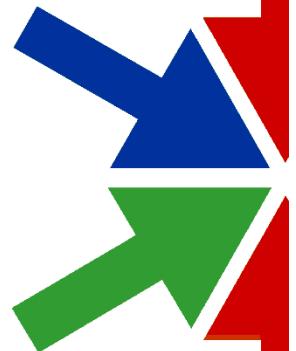
Questions

- ★ Bridge filtering happens before the routing decision on the ingress path
 - ▶ True or False?
- ★ User-space code can directly hook into the protocol processing at one of the **NF_** hook locations
 - ▶ True or False?
- ★ The traffic control code is located on the egress side of the stack
 - ▶ True or False?
- ★ Bridging decisions are handled at layer 2
 - ▶ True or False?
- ★ Routing decisions are handled at layer 4
 - ▶ True or False?

Chapter Break

Quality of Service

PTR



What We Will Cover

- ★ Quality of service
- ★ Traffic control
- ★ Queuing disciplines
- ★ Linux user-space traffic control tools
- ★ Linux kernel traffic control processing

Quality of Service (QoS)

- Provides guarantees on the ability of a network to deliver predictable and desirable results
- Involves prioritization of different traffic types (e.g., web surfing vs. BitTorrent)
- QoS metrics include bandwidth, jitter, latency and packet loss
 - ▶ The key is to achieve the right balance of these attributes for the needs of your system

Example QoS Requirements

★ Voice over IP (VoIP)

- ▶ Packet loss should be less than 1%
- ▶ Latency should be less than 150 ms
- ▶ Jitter should be targeted at 30 ms
- ▶ Bandwidth can range from 21 to 320 kbps depending on the CODEC

★ Interactive video requirements are very similar to VoIP but require higher bandwidth depending on the video CODEC used

★ Streaming video

- ▶ Packet loss should be less than 5%
- ▶ Latency should be less than 5 secs
- ▶ Jitter has no significant requirements
- ▶ Bandwidth is relatively large but subject to the CODEC used

Example QoS Requirements (2)

- ★ For good interactive performance, **rlogin**, **telnet** and **ssh** should have a high priority
 - ▶ Latency less than 200 ms
 - ▶ Low percentage of dropped packets
 - ▶ Jitter & bandwidth requirements are less important
- ★ But **ssh** is also used for **scp** and tunneling
 - ▶ Need good interactivity, but throttle bulk data
 - ▶ Filter based on packet lengths?
 - Imperfect solution since **scp** packets could become fragmented
 - ▶ Some **ssh** implementations provide different Type of Service/Differentiated Service Code Point (TOS/DSCP) values in the IP header for the different functions

TOS vs. DSCP

- * The type of service (TOS) field was originally defined in RFC 791 and subsequently reinterpreted in RFC 1349
- * In RFC 3168, the field is now a 6-bit value (DSCP) and a 2-bit Explicit Congestion Notification (ECN)
- * However, as of the 4.2 kernel, the field is still being interpreted as per RFC 1349
 - ▶ There is no backward compatibility between RFC 3168 and RFC 1349

0	1	2	3	4	5	6	7
Precedence				Type of Service			

Source: wikipedia.com

0	1	2	3	4	5	6	7
DSCP field				ECN field			

Source: wikipedia.com

DSCP<-> TOS QoS Precedence Conversion

DSCP Class	DSCP (bin)	DSCP (hex)	DSCP (dec)	ToS (dec)	ToS (hex)	ToS (bin)	ToS (bin)	Prec.	ToS (dec)	ToS Delay Flag	ToS Throughput Flag	ToS Reliability Flag	ToS Network Format
none	000000	0x00	0	0	0x00	00000000	000	0	0	0	0	0	Routine
cs1	001000	0x08	8	32	0x20	00100000	001	1	0	0	0	0	Priority
af11	001010	0x0A	10	40	0x28	00101000	001	1	0	1	0	0	Priority
af12	001100	0x0C	12	48	0x30	00110000	001	1	1	0	0	0	Priority
af13	001110	0x0E	14	56	0x38	00111000	001	1	1	1	0	0	Priority
cs2	010000	0x10	16	64	0x40	01000000	010	2	0	0	0	0	Immediate
af21	010010	0x12	18	72	0x48	01001000	010	2	0	1	0	0	Immediate
af22	010100	0x14	20	80	0x50	01010000	010	2	1	0	0	0	Immediate
af23	010110	0x16	22	88	0x58	01011000	010	2	1	1	0	0	Immediate
cs3	011000	0x18	24	96	0x60	01100000	011	3	0	0	0	0	Flash
af31	011010	0x1A	26	104	0x68	01101000	011	3	0	1	0	0	Flash
af32	011100	0x1C	28	112	0x70	01110000	011	3	1	0	0	0	Flash
af33	011110	0x1E	30	120	0x78	01111000	011	3	1	1	0	0	Flash
cs4	100000	0x20	32	128	0x80	10000000	100	4	0	0	0	0	FlashOverride
af41	100010	0x22	34	136	0x88	10001000	100	4	0	1	0	0	FlashOverride
af42	100100	0x34	36	144	0x90	10010000	100	4	1	0	0	0	FlashOverride
af43	100110	0x26	38	152	0x98	10011000	100	4	1	1	0	0	FlashOverride
cs5	101000	0x28	40	160	0xA0	10100000	101	5	0	0	0	0	Critical
ef	101110	0x2E	46	184	0xB8	10111000	101	5	1	1	0	0	Critical
cs6	110000	0x30	48	192	0xC0	11000000	110	6	0	0	0	0	Internetworkcontrol
cs7	111000	0x38	56	224	0xE0	11100000	111	7	0	0	0	0	Networkcontrol

Source: bytesolutions.com

Application <-> TOS Correspondence

- ★ Various applications set the TOS values via `setsockopt()`
- ★ Here are some sample TOS values for selected applications:

TELNET		1000	(minimize delay)
FTP	Control	1000	(minimize delay)
FTP	Data	0100	(maximize throughput)
TFTP		1000	(minimize delay)
SMTP	Command phase	1000	(minimize delay)
SMTP	DATA phase	0100	(maximize throughput)
DNS	UDP Query	1000	(minimize delay)
DNS	TCP Query	0000	
DNS	Zone Transfer	0100	(maximize throughput)
NNTP		0001	(minimize monetary cost)
ICMP	Errors	0000	
ICMP	Requests	0000	(mostly)
ICMP	Responses	0000	(mostly)

Traffic Control Overview

- ★ Linux utilizes traffic control principles to support QoS
- ★ All ingress and egress network traffic is influenced
- ★ Traffic control framework in Linux integrates components of
 - ▶ Policing
 - ▶ Scheduling
 - ▶ Shaping

Traffic Control Terminology

★ Policing

- ▶ Applies to ingress
- ▶ Primarily used for throttling the rate at which flows may arrive
- ▶ Dropping is a severe form of policing
- ▶ Actions: accept, drop, mark or reclassify
- ▶ Policing does not have the capability to delay packets (traffic shaping)
- ▶ Classifying separate packets into different queues for different treatment
- ▶ Marking is a mechanism for packet altering



Source: blogspot.com

Traffic Control Terminology (2)

★ Scheduling

- ▶ Reorders/prioritizes packets on egress
- ▶ Used to improve interactivity for traffic that needs it while still guaranteeing bandwidth to bulk transfers

★ Shaping

- ▶ Modifies the rate of transmission on egress
- ▶ Used to lower bandwidth due to network congestion or to smooth out bursts to promote better network behavior
- ▶ Involves buffering data, assigning priorities and setting classification

Queuing Disciplines

- ★ A queuing discipline represents the way a queue is organized
 - ▶ Specifies the rules of inserting and removing packets to and from a queue
- ★ Also referred to as a “qdisc”
- ★ Defines the variance of the waiting time
- ★ Can be categorized as “classless” or “classful”



Source: ineedmotivation.com

Queuing Disciplines – Classless

★ Accept data and only reschedule, delay or drop it

★ Examples

- ▶ First In, First Out (FIFO)
- ▶ PFIFO_FAST
- ▶ Token Bucket Flow (TBF)
- ▶ Random Early Detection (RED)
- ▶ Stochastic Fair Queuing (SFQ)
- ▶ Generalized RED (GRED)
- ▶ Asynchronous Transfer Mode (ATM)

Queuing Disciplines – Classful

- ★ Can have filters attached to them, allowing packets to be directed to particular classes and sub-queues
 - ▶ Can shape the flows

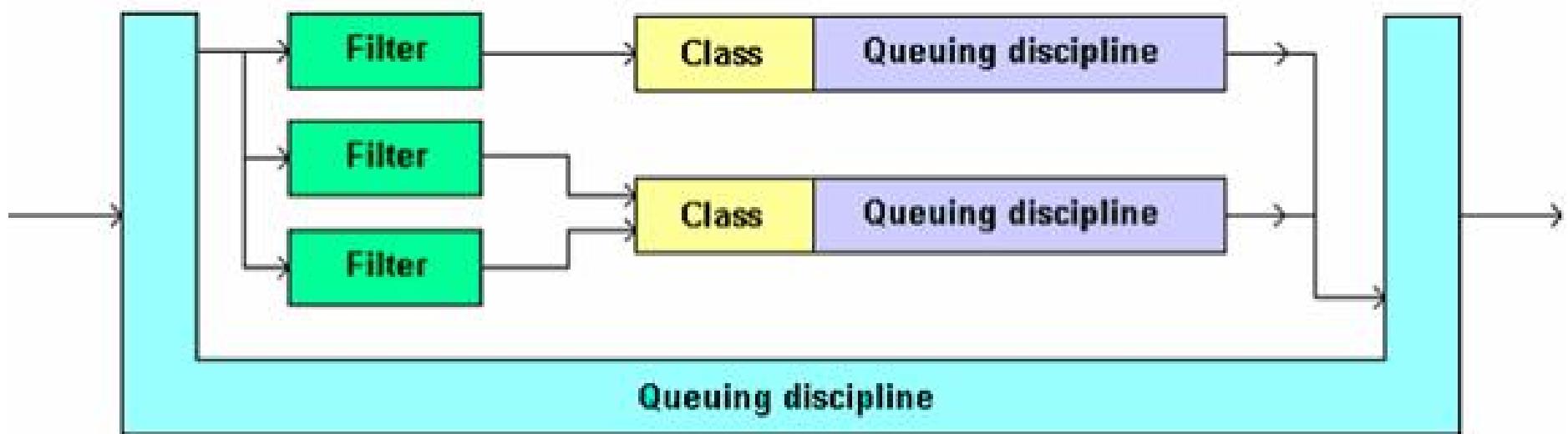


Source: hackcollege.com

★ Examples:

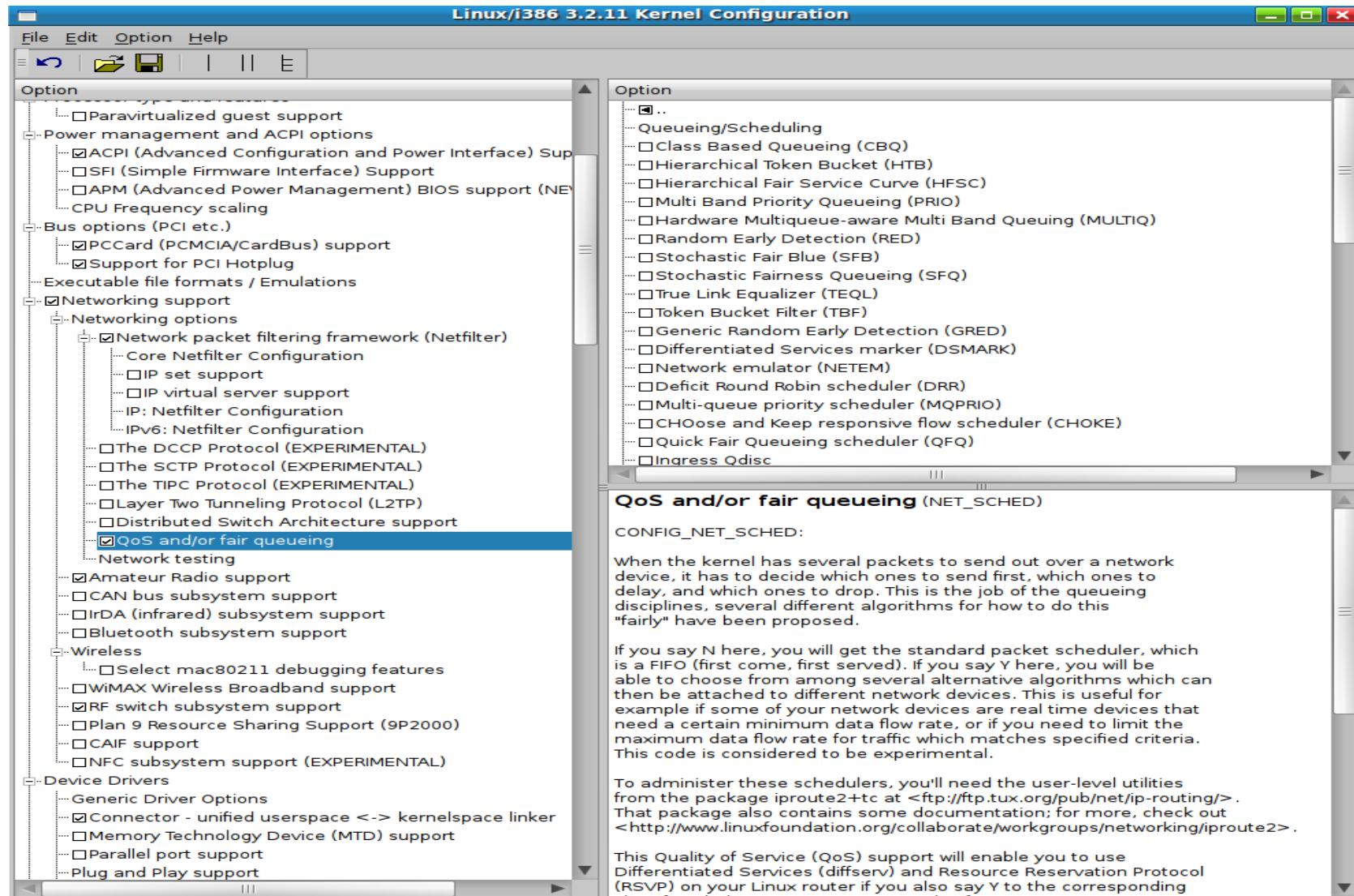
- ▶ Class-Based Queuing Discipline (CBQ)
- ▶ Priority (PRIO)
- ▶ Hierarchy Token Bucket Flow (HTB)
- ▶ Diff-Serv Marker (DS_MARK)
- ▶ Clark-Shenker-Zhang (CSZ)

Generic Classful Qdisc



Source: mpltutorial.com

QoS Kernel Configuration



- Option**
- ..
 - Queueing/Scheduling**
 - Class Based Queueing (CBQ)
 - Hierarchical Token Bucket (HTB)
 - Hierarchical Fair Service Curve (HFSC)
 - Multi Band Priority Queueing (PRIO)
 - Hardware Multiqueue-aware Multi Band Queuing (MULTIQ)
 - Random Early Detection (RED)
 - Stochastic Fair Blue (SFB)
 - Stochastic Fairness Queueing (SFQ)
 - True Link Equalizer (TEQL)
 - Token Bucket Filter (TBF)
 - Generic Random Early Detection (GRED)
 - Differentiated Services marker (DSMARK)
 - Network emulator (NETEM)
 - Deficit Round Robin scheduler (DRR)
 - Multi-queue priority scheduler (MQPRIO)
 - CHOOSE and Keep responsive flow scheduler (CHOKE)
 - Quick Fair Queueing scheduler (QFQ)
 - Ingress Qdisc

QoS and/or fair queueing (NET_SCHED)

CONFIG_NET_SCHED:

When the kernel has several packets to send out over a network device, it has to decide which ones to send first, which ones to delay, and which ones to drop. This is the job of the queueing disciplines, several different algorithms for how to do this "fairly" have been proposed.

If you say N here, you will get the standard packet scheduler, which is a FIFO (first come, first served). If you say Y here, you will be able to choose from among several alternative algorithms which can then be attached to different network devices. This is useful for example if some of your network devices are real time devices that need a certain minimum data flow rate, or if you need to limit the maximum data flow rate for traffic which matches specified criteria. This code is considered to be experimental.

To administer these schedulers, you'll need the user-level utilities from the package iproute2+tc at <[ftp://ftp.tux.org/pub/net/ip-routing/](http://ftp.tux.org/pub/net/ip-routing/)>. That package also contains some documentation; for more, check out <<http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2>>.

This Quality of Service (QoS) support will enable you to use Differentiated Services (diffserv) and Resource Reservation Protocol (RSVP) on your Linux router if you also say Y to the corresponding

First In, First Out (FIFO) Qdisc

* Linux supports two FIFO queuing disciplines:

- ▶ Packet FIFO (PFIFO)
 - The size of the queue is defined in number of packets
- ▶ Byte FIFO (BFIFO)
 - The size of the queue is defined in number of bytes
- ▶ If the queue becomes full, incoming packets are dropped

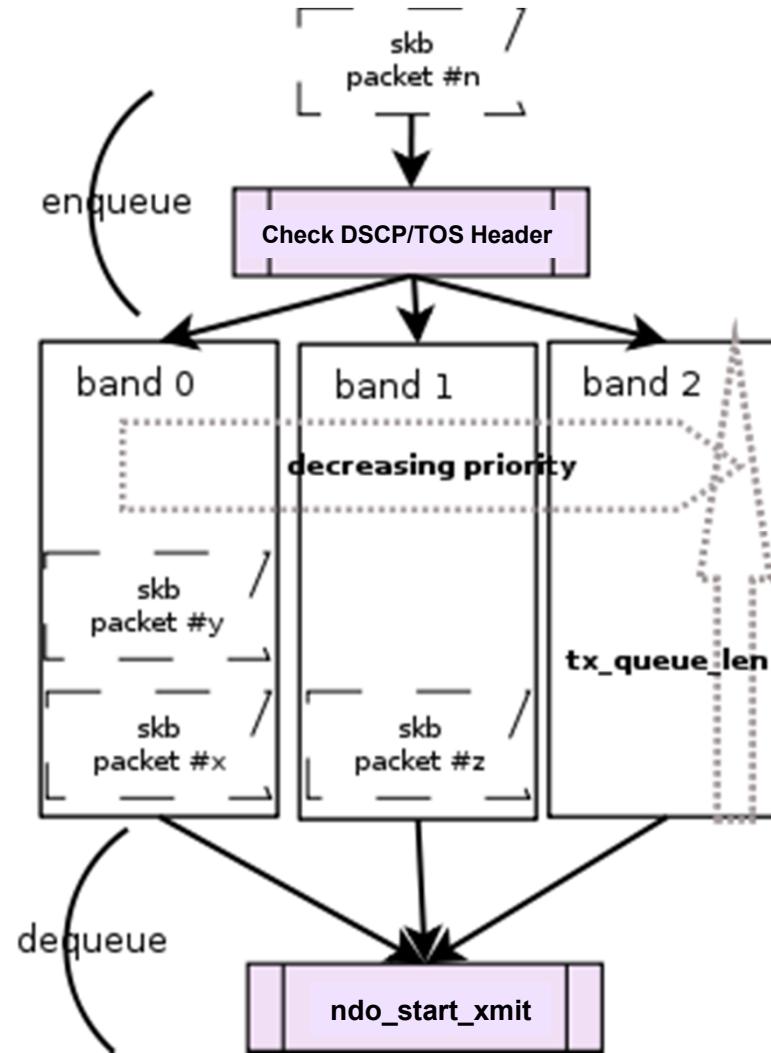
* The source code for the FIFO queuing discipline is in `linux/net/sched/sch_fifo.c`

PFIFO_FAST Qdisc

- ★ Default qdisc for all Linux interfaces is PFIFO_FAST
- ★ Based on conventional FIFO with 3 levels of prioritization (bands 0, 1 & 2)
 - ▶ Highest priority traffic (interactive flows) are placed into band 0 and are serviced first
 - ▶ Likewise, band 1 is emptied before band 2
- ★ No configurable parameters for this qdisc
 - ▶ However, you can assign the packet queue length
- ★ For changing or assigning a value for txqueuelen, use the ip command

```
# ip link set eth0 txqueuelen 1000
```

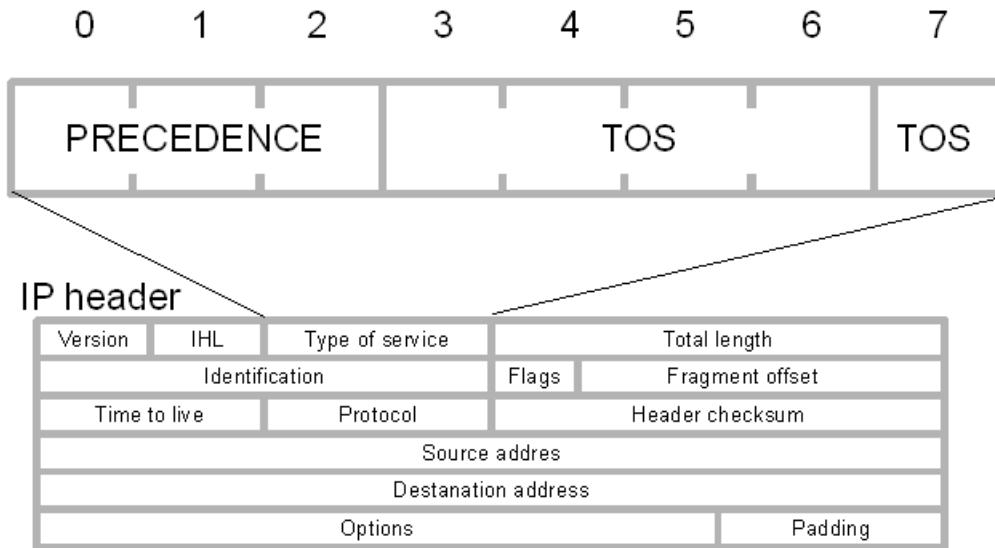
PFIFO_FAST Qdisc Diagram



Source: linuxwall.info

PFIFO_FAST Band Select Rules

- Kernel uses Type of Service (RFC 1349) field in IP header to select band



- The TOS field bits are defined as:

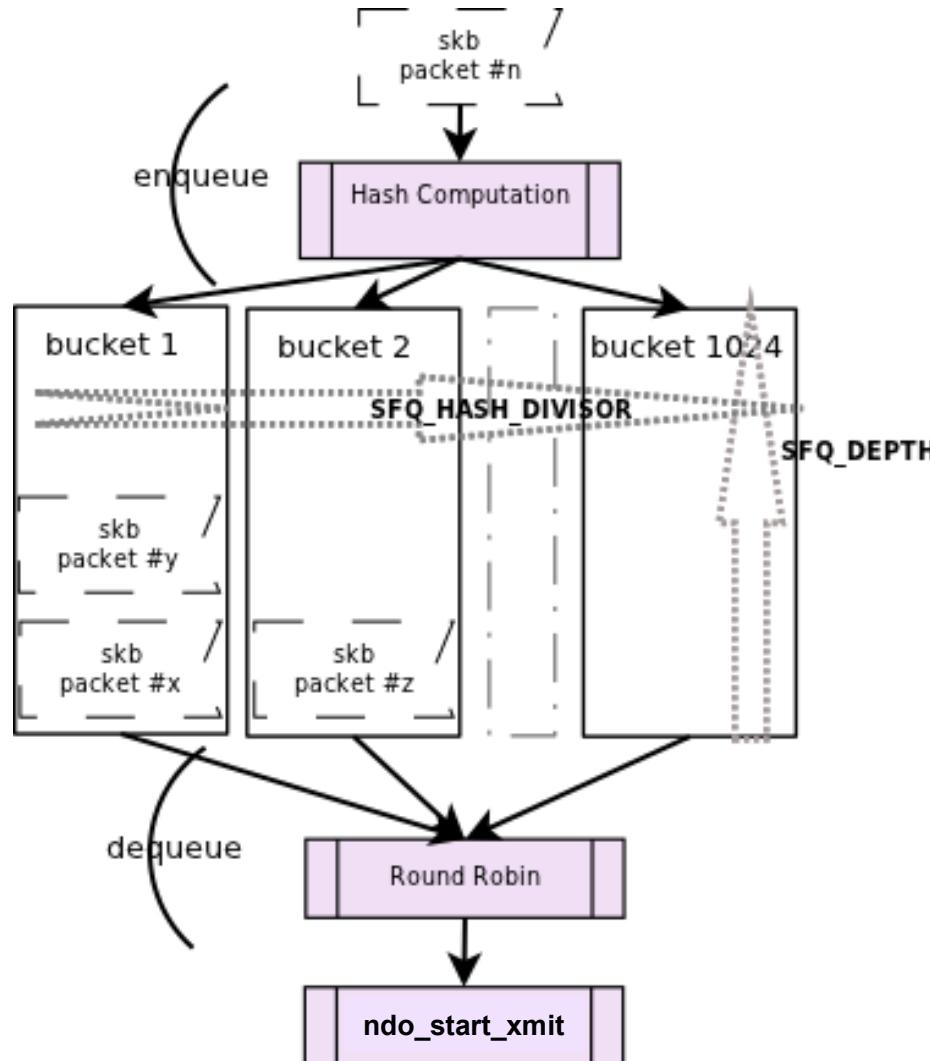
- 10000 Minimize delay
- 01000 Maximize throughput
- 00100 Maximize reliability
- 00010 Minimize monetary cost
- 00000 Normal Service

TOS	Meaning	Band
0x00	Normal Service	1
0x02	Minimize Monetary Cost	2
0x04	Maximize Reliability	1
0x06	mmc+mr	1
0x08	Maximize Throughput	2
0xa	mmc+mt	2
0xc	mr+mt	2
0xe	mmc+mr+mt	2
0x10	Minimize Delay	0
0x12	mmc+md	0
0x14	mr+md	0
0x16	mmc+mr+md	0
0x18	mt+md	1
0x1a	mmc+mt+md	1
0x1c	mr+mt+md	1
0x1e	mmc+mr+mt+md	1

Stochastic Fairness Queuing (SFQ)

- ★ An algorithm that shares the bandwidth without giving any privilege
 - ▶ Takes a fingerprint of the packet based on its header
 - ▶ Uses the fingerprint as a hash to select one of 1024 buckets for the packet
 - ▶ The buckets are emptied in a round robin fashion
- ★ This distribution is somewhat random so no packet or connection has priority over another
- ★ We can add a perturbation to SFQ so packets that hash to the same bucket don't monopolize the communications
 - ▶ **perturb** parameter specified in seconds before forcing a switch

SFQ Diagram



Source: linuxwall.info

Token Bucket Flow (TBF)

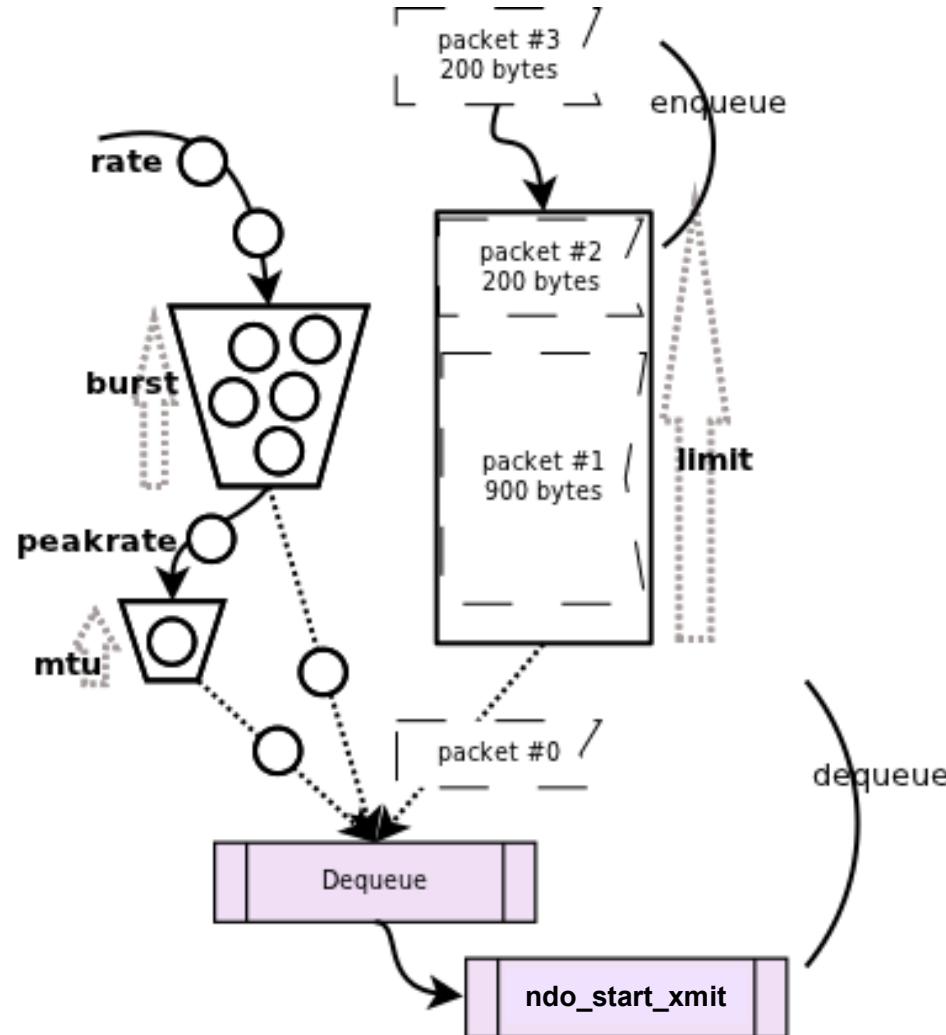
★ An algorithm that attempts to control bandwidth

- ▶ Processes packets based on number of bits in the packets, defining a bit rate
- ▶ Tokens are added to buckets at a separate “token rate”
- ▶ Each bit “consumes” a token when packet is transmitted

★ Scenarios

- ▶ Bit rate = token rate: packet passes with no delay
- ▶ Bit rate < token rate: data can become “bursty”
- ▶ Bit rate > token rate: data is throttled

TBF Diagram

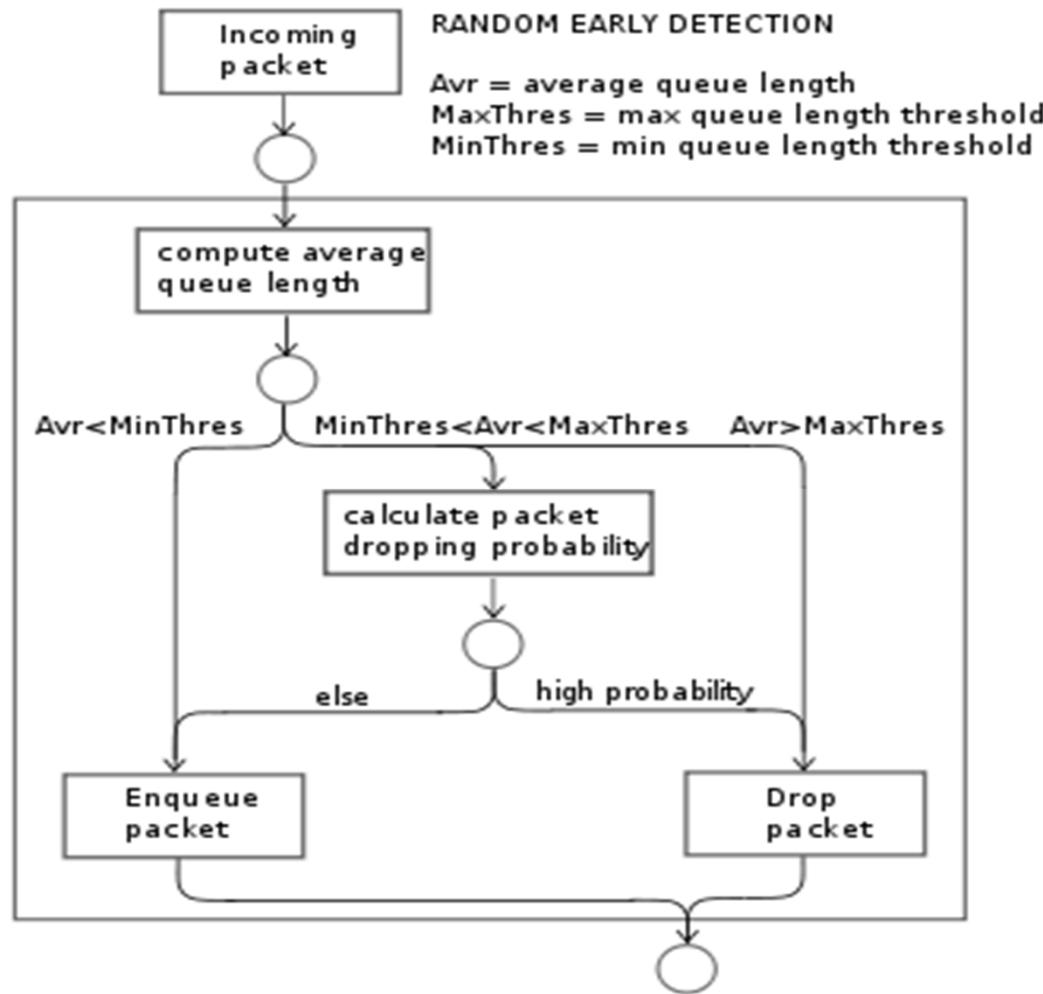


Source: linuxwall.info

Random Early Detection (RED)

- ★ RED has the following algorithm:
 - ▶ Packet drop probability is based on queue length
 - Larger queue equals higher probability of a packet to get dropped
- ★ RED is fair
 - ▶ Probability of a flow's packet to be lost is proportional to its share of link bandwidth
- ★ RED can mark packets with ECN instead of dropping them
 - ▶ ECN in IP header informs sources about the traffic jam and provides a way to prevent packet loss
 - ▶ RED focuses on congestion avoidance by controlling the average packet queue size
- ★ Numerous variations of RED exist
- ★ Source code for RED: [linux/net/sched/sch_red.c](#)

RED Processing Flow

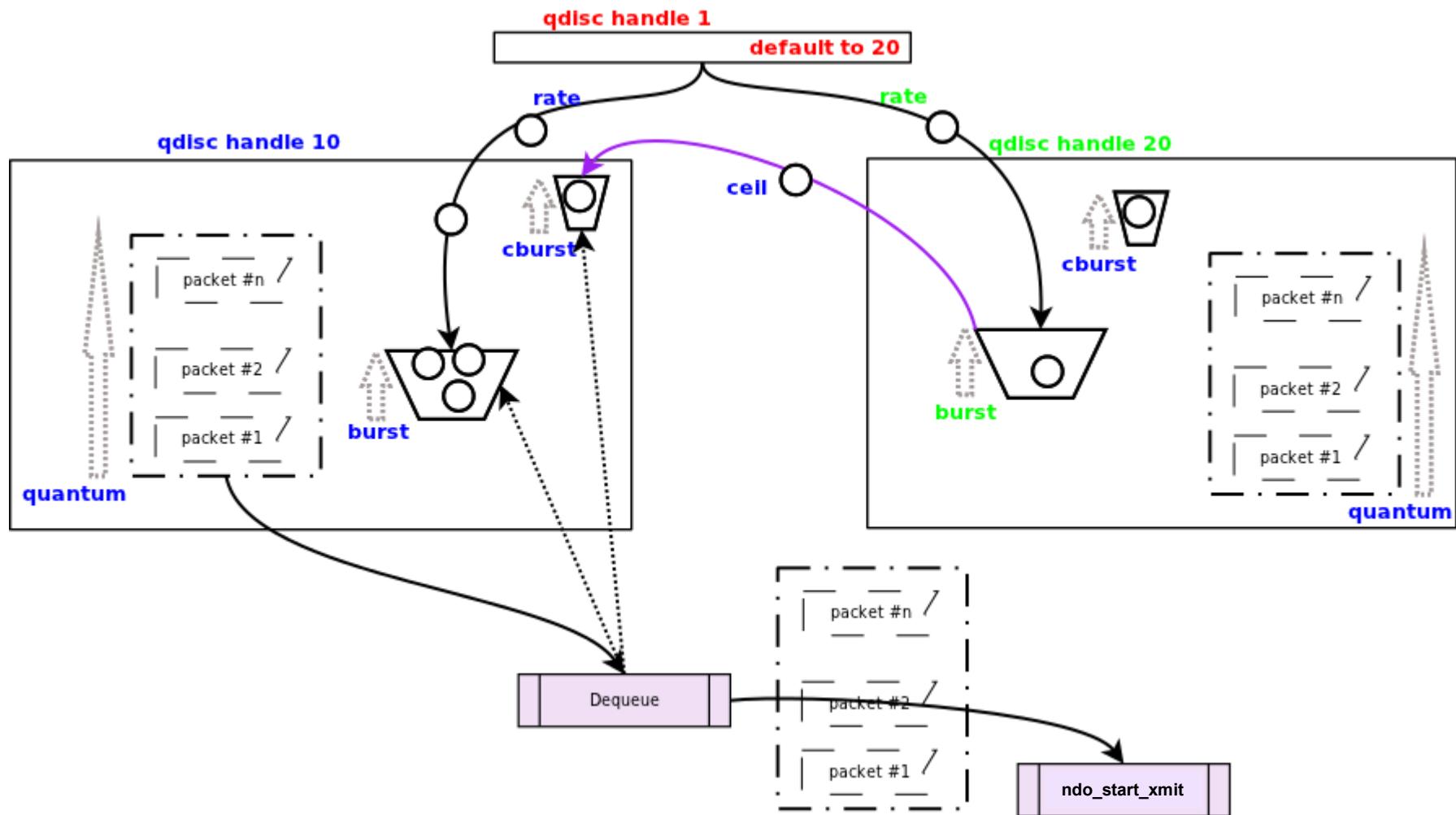


Source: wikipedia.org

Hierarchy Token Bucket Flow (HTB)

- ★ Version of TBF with classes
- ★ Each class is “TBF–like” and is linked in a tree structure
- ★ Improves bandwidth management
 - ▶ Priority between classes
 - ▶ Classes can borrow bandwidth from others
 - ▶ Can plug in another qdisc (e.g., SFQ) at the tree leaf tier
- ★ Parameters:
 - ▶ rate: token production rate
 - ▶ ceil: ceiling for borrowing from other classes
 - ▶ burst: amount of bytes that can be burst at the ceiling speed

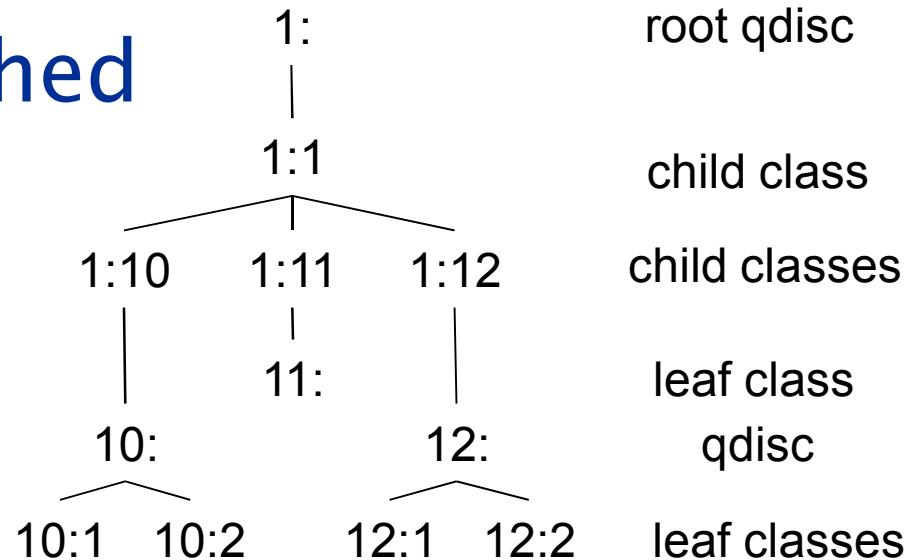
HTB Diagram



Source: linuxwall.info

Roots, Handles, Siblings and Parents

- ★ Each interface has one egress “root qdisc”
- ★ The handles of these qdiscs consist of two parts, a major and a minor number
 - ▶ By convention, the root qdisc is called “1:” which is equal to “1:0”
- ★ A filter can be attached to each node
- ★ A typical hierarchy may look like this:



How Packets Dequeue to the H/W

- ★ When the kernel decides it needs to dequeue a packet, the root qdisc get a dequeue request
- ★ That request is then passed to 1:1 which in turn is passed to 10:, 11: and 12:
 - ▶ Each will talk to their siblings trying to find a packet to dequeue
 - This might require that the entire tree is traversed
- ★ Nested classes only talk to their parent qdiscs – never an interface
 - ▶ Classes will never dequeue faster than their parents will allow
 - E.g., we can have SFQ as an inner class (no shaping, only scheduling) and have HTB as an outer class that does the shaping

Linux Traffic Control Tools

* **tc** (traffic control)

- ▶ Included in the iproute2 package of tools

* **iptables**

- ▶ Provides simpler but limited control
- ▶ Can be used in conjunction with **tc**

Linux tc Command

- ★ Used to configure traffic control in the Linux kernel
 - ▶ First appeared in Linux 2.2
- ★ Linux traffic control consists of shaping, scheduling, policing
 - ▶ Can be attached at either ingress or egress
- ★ Processing of traffic is controlled by three kinds of objects: qdiscs, classes and filters
 - ▶ The following **tc** commands are available for objects: add, remove, change, replace and link

Filter Classifiers

★ Filters can be constructed from various classifiers:

- ▶ fw
 - Uses the firewall markings for the decision
- ▶ u32
 - Bases decisions on fields within the packet
- ▶ route
 - Bases decision on the route
- ▶ rsvp, rsvp6
 - Bases decision on the RSVP protocol

Filter Arguments

★ Various filters have arguments:

▶ **protocol**

- The protocol this classifier will accept (ip, icmp, etc.)

▶ **parent**

- The handle this classifier is to be attached to

▶ **prio**

- The priority of this classifier
 - Lower numbers get tested first

▶ **handle**

- Means different things to different classifiers

★ Example u32 selector:

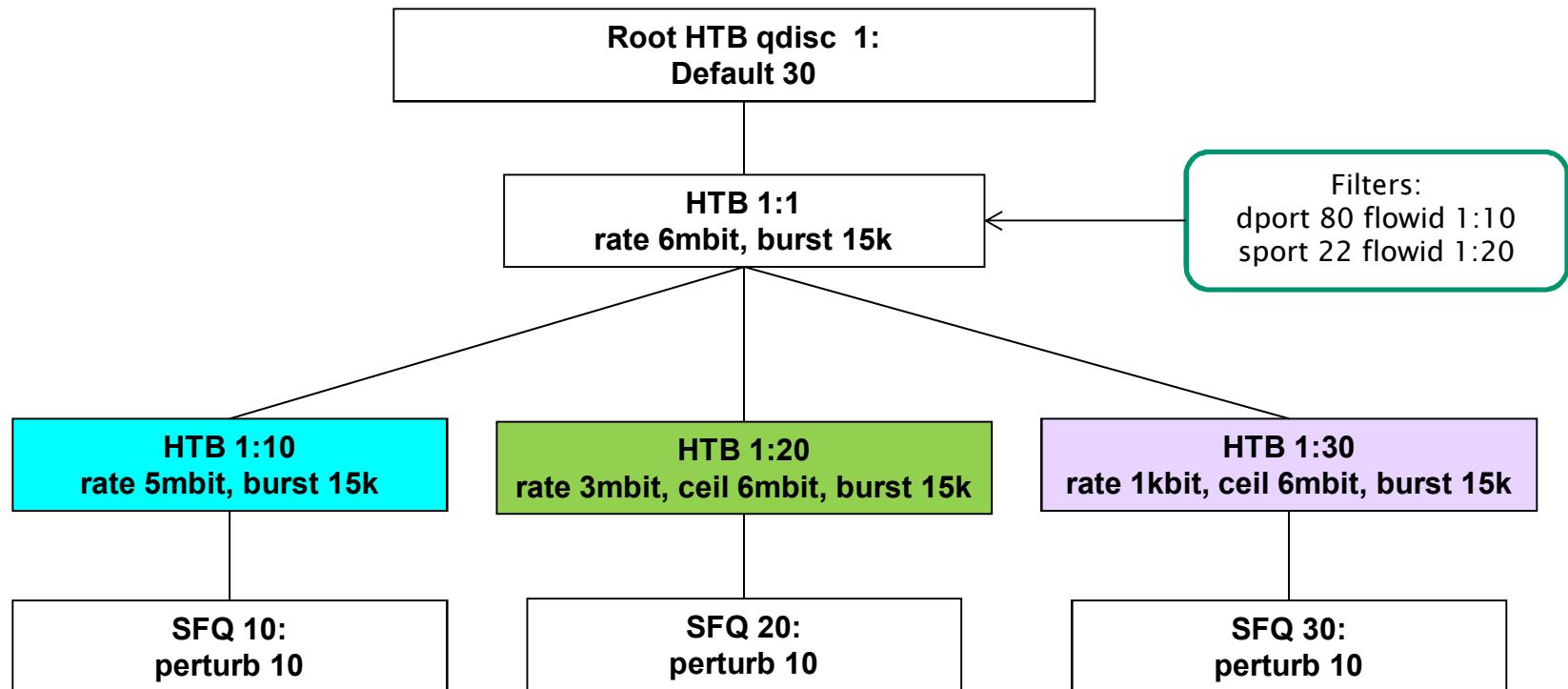
```
# tc filter add dev eth0 protocol ip parent 1:0 prio 10 u32 \
  match ip tos 0x10 0xff flowid 1:10
```

▶ Matches if the TOS field has minimize delay bits set

0	4	8	16	19	31
Version	Header Length	Service Type	Total Length		
		Identification	Flags	Fragment Offset	
TTL	Protocol		Header Checksum		
		Source IP Addr			
		Destination IP Addr			
		Options		Padding	

Source: thegeekstuff.com

Example Traffic Control Policy



Source: linuxwall.info

TC Configuration Example

Define the root qdisc:

```
# tc qdisc add dev eth0 root handle 1: htb default 30
```

Add the classes:

```
# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

Add the leaf nodes:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Define the filters:

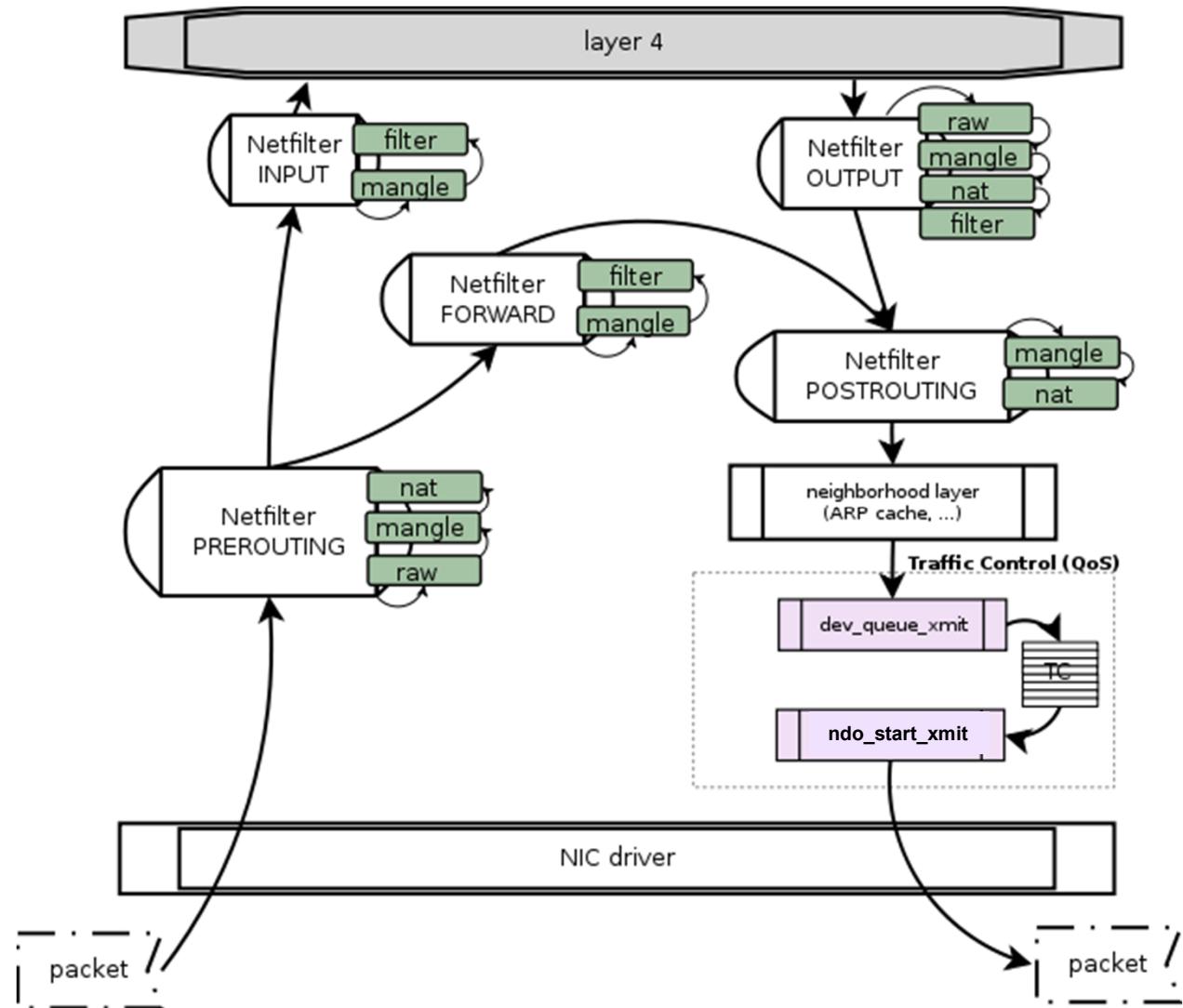
```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 22 0xffff flowid 1:20
```

Using iptables for Traffic Shaping

- * The **iptables** command can also be used for *traffic shaping*
- * Here, we classify ssh packets ala the example:

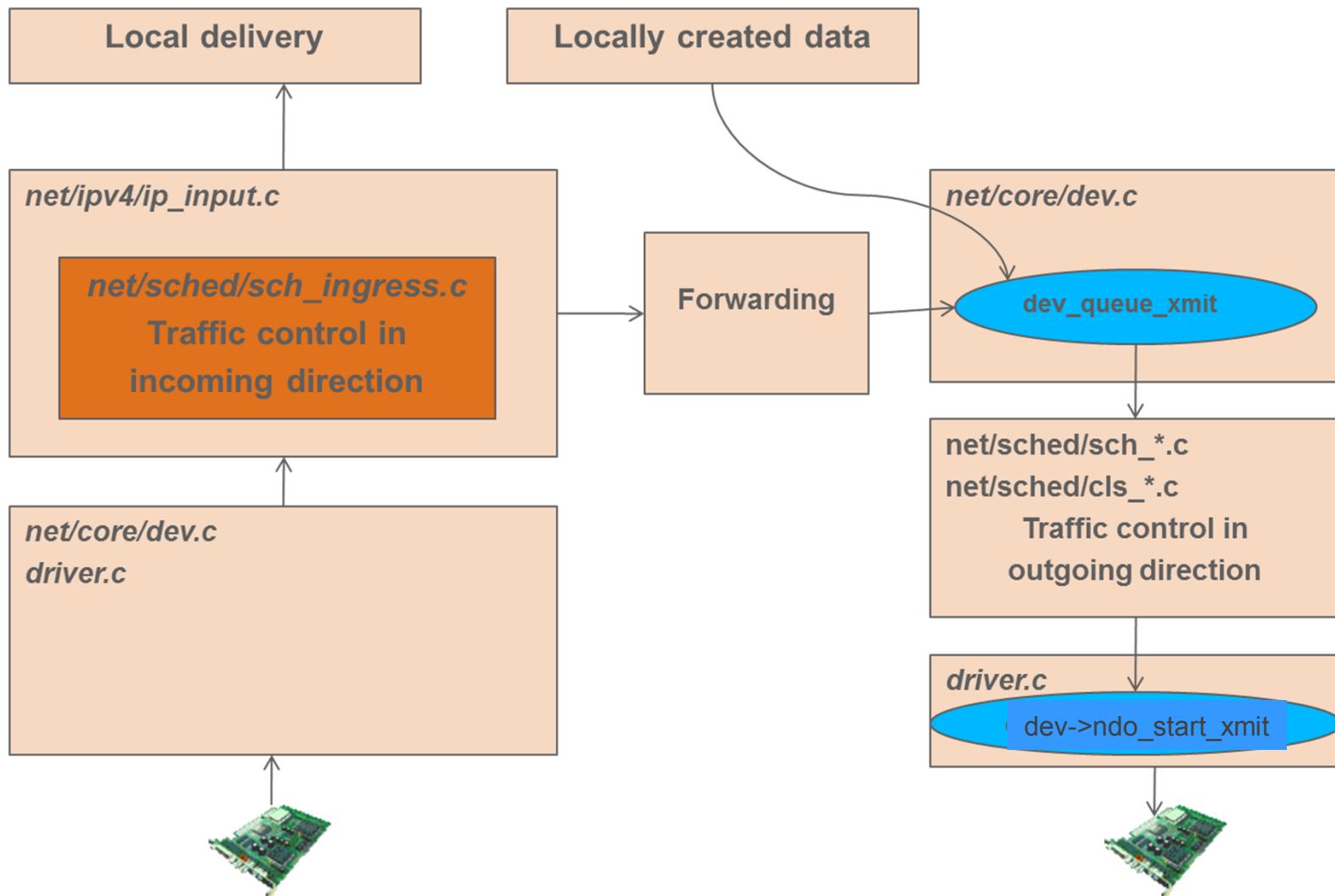
```
$ iptables -t mangle \
    -A POSTROUTING -p tcp \
    --dport 22 --syn \
    -m state --state NEW \
    -m length --length 40:68 \
    -j CLASSIFY --set-class 1:20
```

Linux Kernel Packet Processing Flow

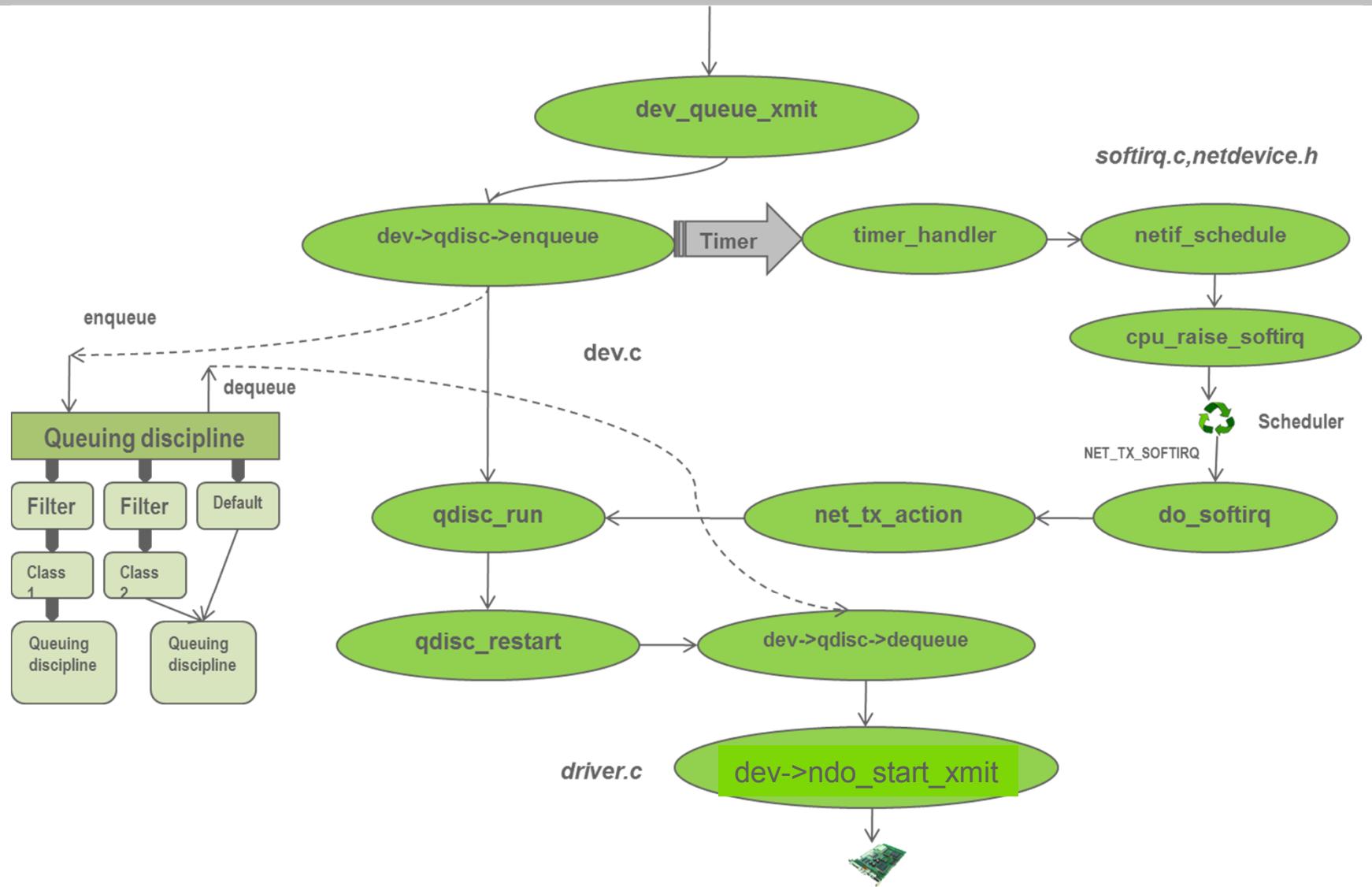


Source: linuxwall.info

Linux Source Code Control Flow



Linux Source Code Control Flow (2)



net_device Structure

- ★ Struct `net_device` provides a uniform interface between software-based protocols and network adapters

```
struct net_device {  
    ...  
    struct netdev_queue *_tx;  
};  
  
struct netdev_queue {  
    struct net_device *dev;  
    struct Qdisc *qdisc;  
    unsigned long state;  
    spinlock_t _xmit_lock;  
    int xmit_lock_owner;  
    struct Qdisc *qdisc_sleeping;  
};
```

net_device Structure (Older Kernel)

```
struct net_device {  
    ...  
    spinlock_t          queue_lock      __cacheline_aligned_in_smp;  
    struct Qdisc        *qdisc;  
    struct Qdisc        *qdisc_sleeping;  
    struct list_head    qdisc_list;  
    unsigned long       tx_queue_len; // Max frames per  
                                // queue allowed  
  
    /* ingress path synchronize */  
    spinlock_t          ingress_lock;  
    struct Qdisc        *qdisc_ingress;  
};
```

Qdisc Structure

```
struct Qdisc {
    int                         (*enqueue)(struct sk_buff *skb, struct Qdisc *dev);
    struct sk_buff *            (*dequeue)(struct Qdisc *dev);
    unsigned                    flags;
#define TCQ_F_BUILTIN      1
#define TCQ_F_THROTTLED   2
#define TCQ_F_INGRESS     4
#define TCQ_F_WARN_NONWC (1 << 16)
    int                         padded;
    struct Qdisc_ops           *ops;
    struct qdisc_size_table    *stab;
    struct list_head            list;
    u32                         handle;
    u32                         parent;
    atomic_t                   refcnt;
    struct gnet_stats_rate_est rate_est;
    int                         (*reshape_fail)(struct sk_buff *skb, struct Qdisc *q);
    *u32_node;
    struct Qdisc                __parent;          /* This field is deprecated */
    *dev_queue;
    struct Qdisc                *next_sched;
    struct sk_buff               *gso_skb;
    state;
    struct sk_buff_head          q;
    struct gnet_stats_basic     bstats;
    struct gnet_stats_queue      qstats;
};

};
```

- Defined in [include/net/sch_generic.h](#) file

Qdisc_ops Structure

```
struct Qdisc_ops {
    struct Qdisc_ops           *next;
    const struct Qdisc_class_ops *cl_ops;
    char                      id[IFNAMSIZ];          // Name of the qdisc
    int                       priv_size;

    int                      (*enqueue) (struct sk_buff *, struct Qdisc *); // Add
                                            // a new packet to the queue
                                            // for transmission
    struct sk_buff *          (*dequeue) (struct Qdisc *);   // Gets next packet to send
    struct sk_buff *          (*peek) (struct Qdisc *);
    unsigned int               (*drop) (struct Qdisc *);

    int                      (*init) (struct Qdisc *, struct nlattr *arg); // Init qdisc
    (*reset) (struct Qdisc *);
    (*destroy) (struct Qdisc *);
    (*change) (struct Qdisc *, struct nlattr *arg);

    int                      (*dump) (struct Qdisc *, struct sk_buff *);
    int                      (*dump_stats) (struct Qdisc *, struct gnet_dump *);

    struct module             *owner;
};
```

- Defined in [include/net/sch_generic.h](#) file

`Qdisc_ops` Callbacks

enqueue

- ▶ Puts packet in qdisc queue
- ▶ Parameters: the packet and the destination qdisc
- ▶ Returns the result of the queuing: success, success with congestion, or drop

dequeue

- ▶ Takes the next packet from the qdisc for transmitting on the network interface
- ▶ Parameter: the desired source qdisc
- ▶ Returns the dequeued packet, or NULL if no packet is available

peek

- ▶ Peek at packet without removing from the queue

`qdisc_ops` Callbacks (2)

★ `drop`

- ▶ Drops a packet from the queue
- ▶ Parameter: the desired qdisc
- ▶ Return value indicates whether a packet was dropped

★ `init`

- ▶ Initializes and configures the qdisc
- ▶ Parameters: the target qdisc and the struct describing the desired configuration
- ▶ Return value indicates success or failure

★ `reset`

- ▶ Resets the qdisc
- ▶ Clears the queue, sets its state variables to their initial values
- ▶ Parameter: the target qdisc
- ▶ No return value (should always succeed)

qdisc_ops Callbacks (3)

* **destroy**

- ▶ Performs removal of the qdisc
- ▶ Parameter: the target qdisc
- ▶ No return value (should always succeed)

* **change**

- ▶ Changes the configuration of the qdisc, but without full Init
- ▶ Parameters: the target qdisc and the desired configuration
- ▶ Returns success or failure

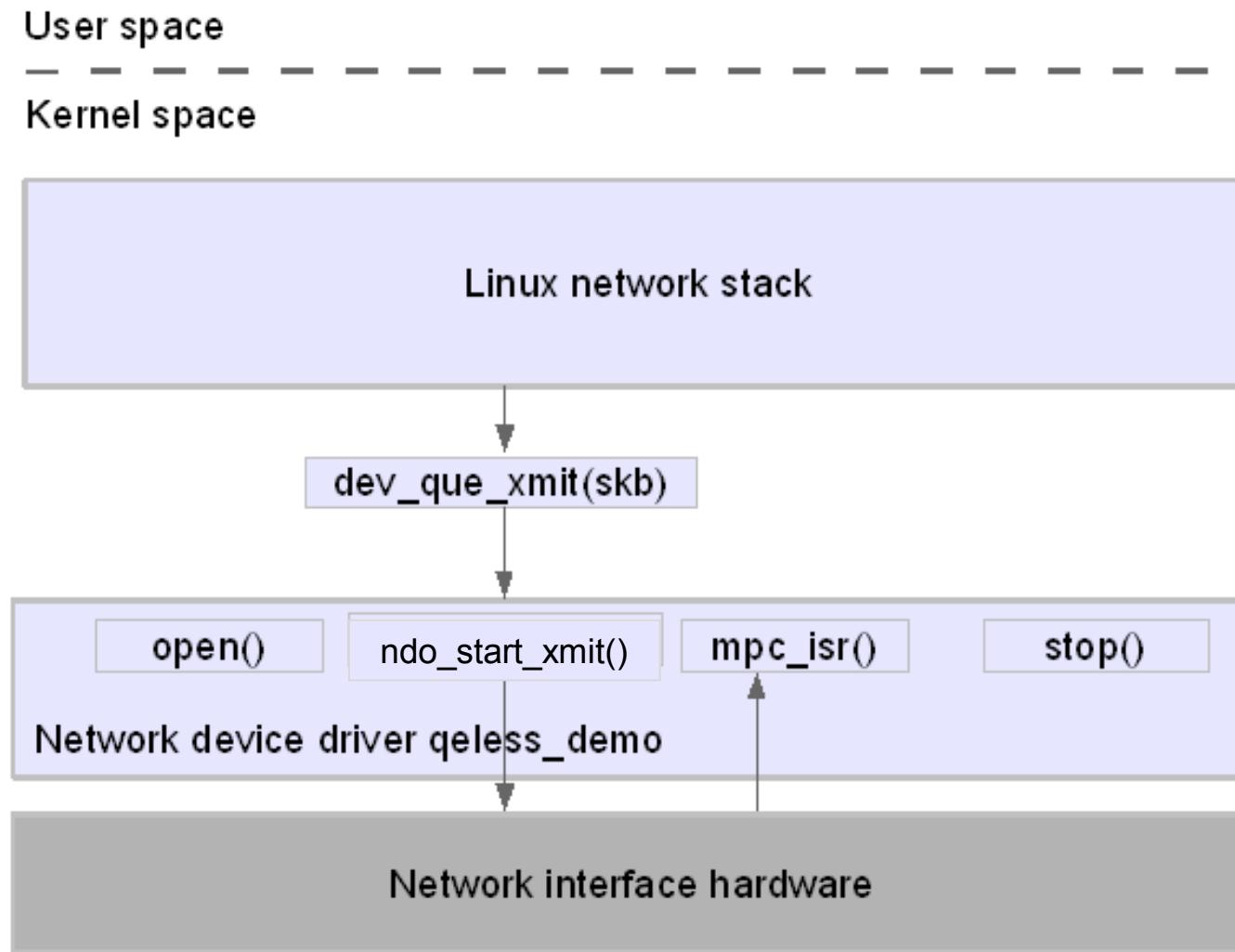
* **dump**

- ▶ Dumps diagnostic data

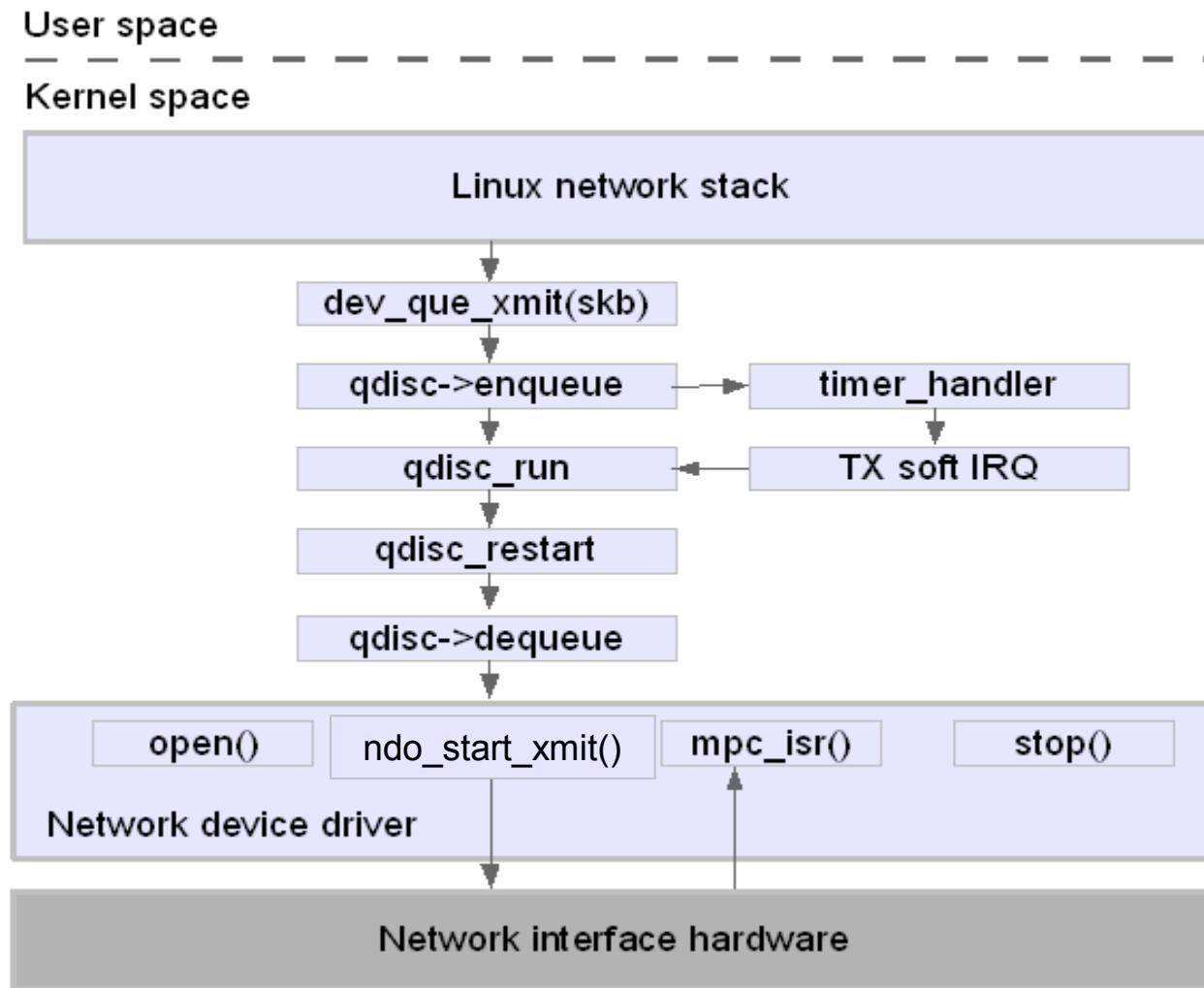
Devices Without Qdisc

- ✖ Some networking devices will not have a qdisc
 - ▶ Loopback device
 - ▶ Tunneling devices
- ✖ You will not be able to control the quality of service on these types of devices

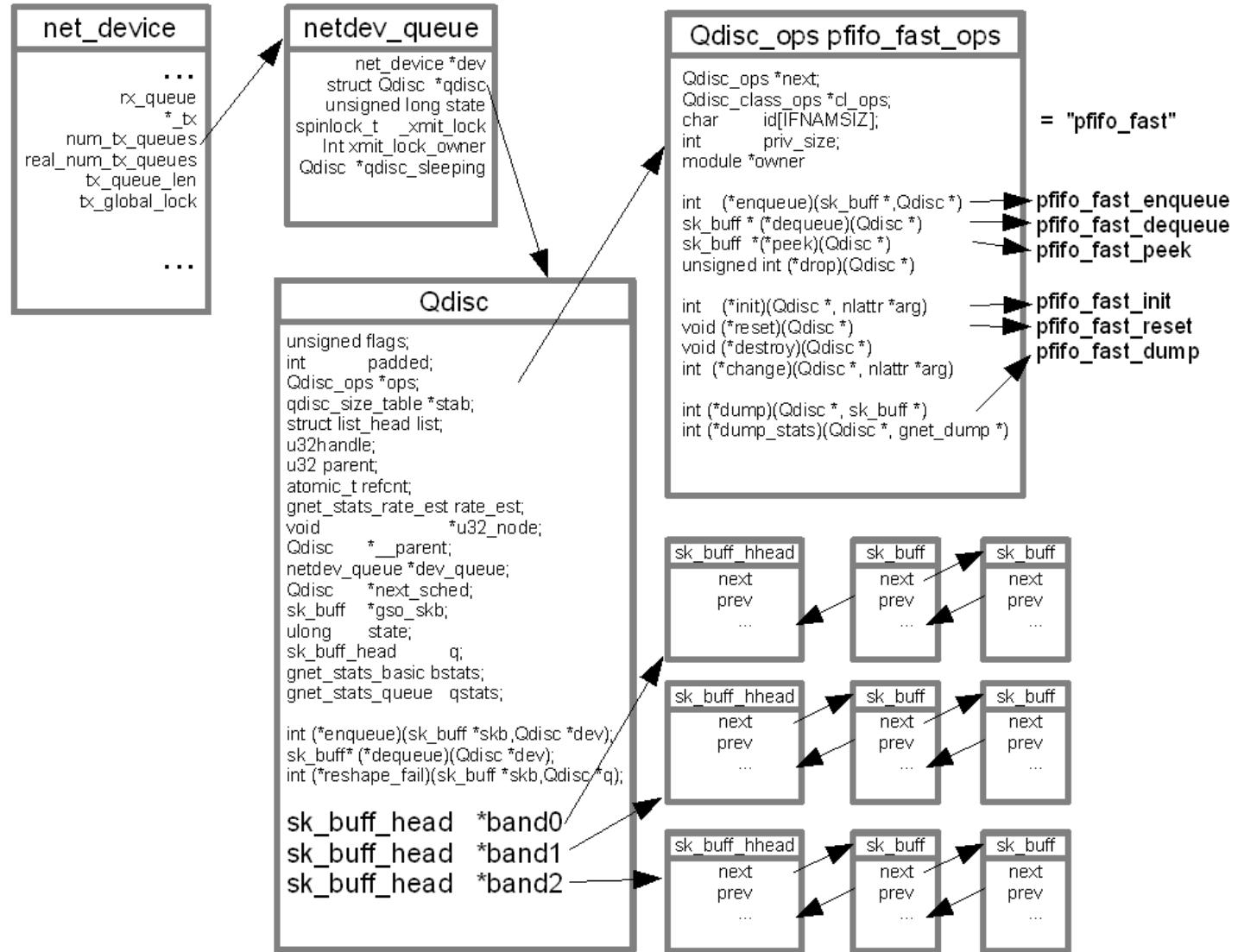
Transmit – Devices Without Qdisc



Transmit – Devices With Qdisc



Example: PFIFO_FAST Data Structures



PFIFO_FAST Qdisc_ops

```
static struct Qdisc_ops pfifo_fast_ops __read_mostly =
{
    .id          = "pfifo_fast",
    .priv_size   = PFIFO_FAST_BANDS *
                    sizeof(struct sk_buff_head),
    .enqueue     = pfifo_fast_enqueue,
    .dequeue     = pfifo_fast_dequeue,
    .peek        = pfifo_fast_peek,
    .init        = pfifo_fast_init,
    .reset       = pfifo_fast_reset,
    .dump        = pfifo_fast_dump,
    .owner       = THIS_MODULE,
};
```

Registering Qdisc

- * The following functions can be used to register and unregister a qdisc type in the Linux kernel (defined in include/net/pkt_sched.h)

```
int register_qdisc (struct Qdisc_ops *qops) ;  
int unregister_qdisc (struct Qdisc_ops *qops) ;
```

- * Example:

```
static int __init pfifo_fast_module_init(void)  
{  
    return register_qdisc(&pfifo_fast_ops);  
}  
  
static void __exit pfifo_fast_module_exit(void)  
{  
    unregister_qdisc(&pfifo_fast_ops);  
}
```

Traffic Control – Resources

- ★ The different queuing disciplines provide substantial flexibility and, unfortunately, complexity when defining traffic control rules
- ★ Two excellent resources are:
 - ▶ Linux Advanced Routing & Traffic Control HOWTO
 - <http://lartc.org/howto/index.html>
 - ▶ Traffic Control HOWTO
 - <http://linux-ip.net/articles/Traffic-Control-HOWTO/>

Summary

- ★ A good QoS policy allocates resources (balancing response times, bandwidth and reliable delivery) to different traffic types appropriately
- ★ QoS is instrumental in optimizing network performance
- ★ Linux contains many traffic control tools and features to implement sophisticated QoS policies
- ★ Default Linux policies are sufficient for most situations

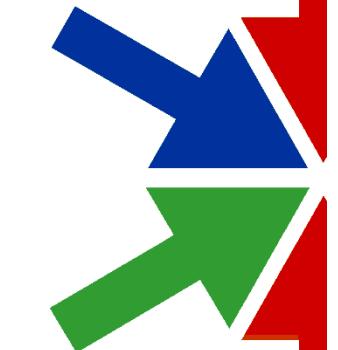
Questions

- * The concept of traffic shaping is used to smooth out bursts of packets to provide more consistent network behavior
 - ▶ True or False?
- * The command for configuring traffic control is the SFQ command
 - ▶ True or False?
- * QoS queuing disciplines are referred to as classless or classful
 - ▶ True or False?
- * HTB is an example of a classless qdisc
 - ▶ True or False?
- * The iptables interface can be used for traffic shaping
 - ▶ True or False?

Chapter Break

Understanding Linux Networking

L3/L4 Firewalls and IPtables



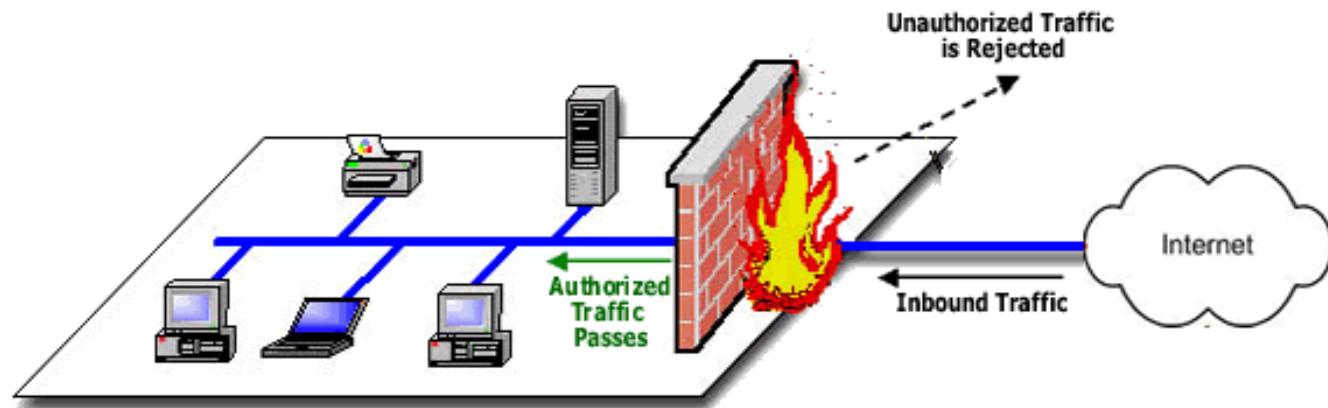
Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ❖ Types of firewalls
- ❖ Linux netfilter/iptables
- ❖ NAT via iptables
- ❖ Example iptables configuration
- ❖ Example firewall code
- ❖ Netfilter changes in new kernels

Firewalls

- ★ Isolate an internal network from an external network allowing some packets to pass and blocking others
- ★ Provided as server software or as a standalone 'security appliance'
- ★ Can consist of one or more devices employing various techniques



Source: ccnpsecurity.blogspot.com

Firewall Taxonomy

★ Most firewalls contain elements of each of the following

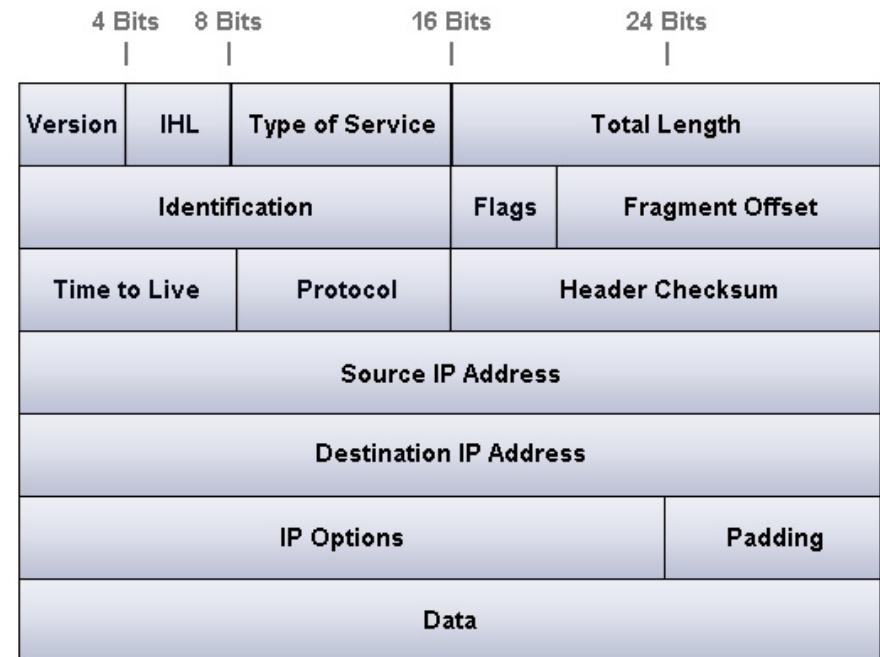
- ▶ Packet filters
 - Also known as network-layer firewalls
- ▶ Stateful filters
 - Also known as circuit-level firewalls
- ▶ Application gateways
 - Also known as proxies



Source: furnitureanddesignideas.com

Packet Filters

- ★ Validates a packet based mostly on the contents of its IP header
 - ▶ IP Addresses
 - ▶ Protocol
 - ▶ Type of service
 - ▶ Hardware Interface
 - ▶ Direction
- ★ Each packet is an entity onto themselves
 - ▶ I.e., the filter is stateless
- ★ Little visibility into the packet payload/data
- ★ Packet filters *can* look into the TCP header
 - ▶ Typically only for the TCP/UDP port, however



Source: learn-networking.com

Well Known Port Examples

★ A Linux system has 65535 ports

- ▶ Ports are bound using the **bind()** call for servers or automatically for clients

★ Not all are used

★ Modification of well-known ports (1–1023) requires superuser permissions

- ▶ Reserved for privileged system processes

Protocol	Port	Protocol	Port
FTP	20, 21	NNTP	119
SSH, SFTP, SCP	22	IMAP	143
Telnet	23	SNMP	161
SMTP	25	LDAP	389
TACACS	49	ISAMP (VPN)	500
DNS	53	Syslog	514
TFTP	69	LDAP/TLS	636
HTTP	80	L2TP	1701
Kerberos	88	PPTP	1723
POP3	110	Remote access	3389

Source: certapps.com

Other Ports

- ★ Registered Ports (1024 -> 49151) are for applications that do not need superuser privileges
 - ▶ OpenVPN:1194
 - ▶ NFS: 2049
 - ▶ SVN: 3690
- ★ Ephemeral Ports (49152 -> 65535) are for applications that need a temporary communications port
- ★ Many Linux kernels use the port range (32768 -> 61000) for sockets

Problems With Port Blocking

FTP example

- ▶ Client sends command from an arbitrary port to port 21 on the server
- ▶ Server sends data from port 20 to the client on a dynamically allocated port
- ▶ Depending on the firewall configuration (usually default deny) the dynamically-allocated port is probably blocked

Example Packet Filtering Rules

Packet filter behavior is defined by the use of rules

Policy/Rule	Firewall Setting
No outside Web access	Drop all outgoing packets to any IP address using port 80
Prevent IPTV from eating up the available bandwidth	Drop all incoming UDP packets except DNS and router broadcasts
Prevent your network from being used for a DoS attack	Drop all ICMP packets going to a 'broadcast' address (e.g. 222.22.255.255)
Prevent your network from being tracerouted	Drop all outgoing ICMP

Source: cis.poly.edu

Packet Filter Rule Definitions

- Rules are processed from top to bottom until the packet matches one
- If no rule matches, the default policy is followed

Action	Source Address	Dest Address	Protocol	Source Port	Dest Port
Deny	222.22/16	outside of 222.22/16	TCP	> 1023	80
Allow	outside of 222.22/16	222.22/16	TCP	80	>1023
Allow	222.22/16	outside of 222.22/16	UDP	>1023	53
Allow	outside of 222.22/16	222.22/16	UDP	53	>1023
Deny	All	All	All	All	All

Source: cis.poly.edu

Packet Filters: Pros & Cons

★ Advantages

- ▶ Effective against worms and Trojan horses
- ▶ Can be very fast if filtering rules are not too complex
- ▶ Built into Linux kernel

★ Disadvantages

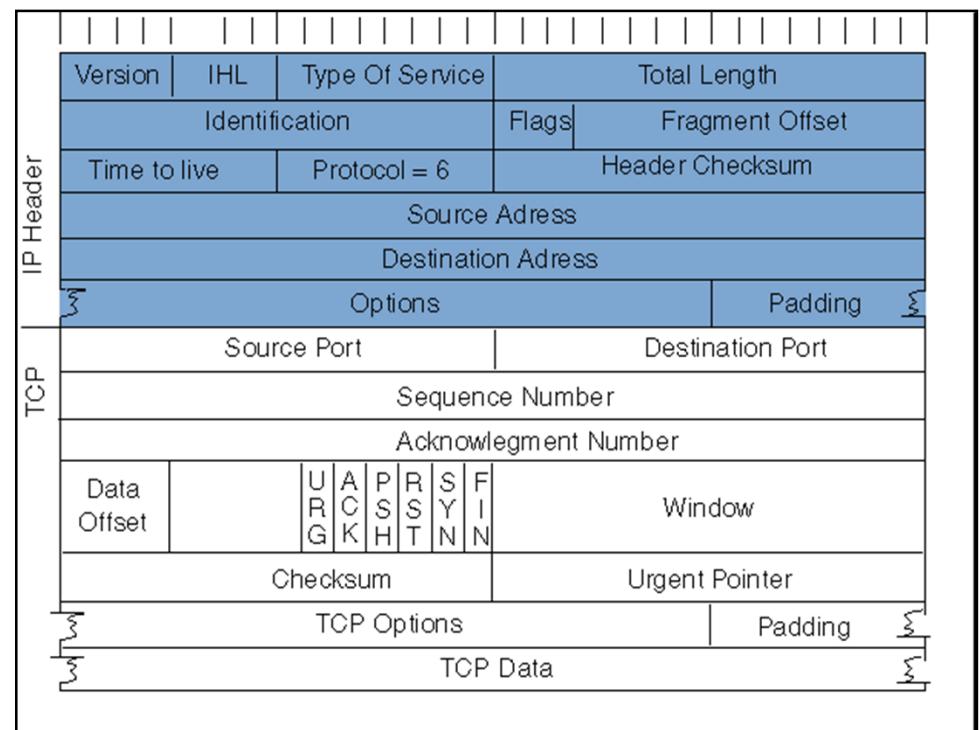
- ▶ Does not protect against attacks with malformed packets (e.g., spoofing)
- ▶ Does not protect against protocol-based attacks (e.g., buffer overflows)
- ▶ Not effective against attacks using authorized channels (e.g., email viruses)
- ▶ Cannot enforce some policies like excluding certain users
- ▶ Rules can get complicated and difficult to test

★ Necessary but not sufficient

- ▶ Packet filters are the first line of defense in a multi-layered approach

Stateful Filters

- ★ Extension of basic packet filters
- ★ Remembers the state of communication sessions
- ★ Using TCP header information, it permits connections only from trusted clients
- ★ Can configure to only allow in packets from established sessions



Source: cisco.com

Connection State Table

- ★ Keeps a dynamic table of active sessions
- ★ Assigns connection states to each packet

► NEW, ESTABLISHED, RELATED, INVALID

IPTState - IPTables State Top					
Version: 2.2.2	Sort: SrcIP	b: change sorting	h: help	Prt	State
Source	Destination				TTL
192.168.1.102:34098	202.188.1.5:53		udp		0:00:4
192.168.1.102:36089	202.188.1.5:53		udp		0:02:4
192.168.1.102:39257	209.85.175.18:443		tcp	ESTABLISHED	119:59:1
192.168.1.102:33297	202.71.102.178:80		tcp	TIME_WAIT	0:00:4
192.168.1.102:34813	209.85.175.83:443		tcp	TIME_WAIT	0:01:1
192.168.1.102:42555	209.85.175.18:443		tcp	TIME_WAIT	0:01:1
192.168.1.102:50345	202.188.1.5:53		udp		0:01:4
192.168.1.102:42553	209.85.175.18:443		tcp	TIME_WAIT	0:01:1
192.168.1.102:33298	202.1.102.178:80		tcp	TIME_WAIT	0:01:4
192.168.1.102:60630	209.85.175.17:443		tcp	TIME_WAIT	0:01:1
192.168.1.102:40759	64.4.9.254:80		tcp	ESTABLISHED	119:59:1
192.168.1.102:39219	209.85.175.18:443		tcp	ESTABLISHED	119:58:4
192.168.1.102:54933	207.46.124.98:80		tcp	TIME_WAIT	0:01:0

Source: johnboy60.com

Stateful Filters: Pros & Cons

★ Advantages

- ▶ Relatively easy to implement
 - Standard protocols are very easy to configure
- ▶ Among the fastest filter types
 - Needs to check packet rules for connection requests (OSI layer 7)
 - Otherwise only needs to check state table (OSI layer 4 and below)
- ▶ Protects against 'answer' session exploits and some DoS like SYN-flooding
- ▶ Built into Linux kernel

★ Disadvantages

- ▶ Does not protect against attacks with malformed packets (e.g., spoofing)
- ▶ Does not protect against protocol-based attacks (e.g., buffer overflows)
- ▶ Not effective against attacks using authorized channels (e.g., email viruses)

Application Gateways (ALG)

- ★ Uses application and protocol-specific knowledge to enhance the protection provided by a firewall or NAT
- ★ Also provides customized NAT traversal filters for “control/data” protocols
 - ▶ Prevents FTP control channel timing out on large data transfers
 - ▶ Avoids keeping many firewall ports open by managing dynamic ephemeral ports for active or passive mode FTP transfers
- ★ Filtering performed at the OSI Application Layer
 - ▶ Possible performance degradation
- ★ Usually hosted on dedicated proxy servers
- ★ Similar to a proxy but “invisible”
 - ▶ Does not require configuration by the client

Application Layer Filters

- ★ Also known as socket filters
 - ▶ Filters the connection between the application layer and the lower layers of the OSI model
- ★ Applies filtering rules on a per process basis instead of on a per port basis
- ★ Can detect protocol trying to use wrong port or a protocol being “abused”
- ★ Typically used in conjunction with packet filters

L7-filter

- ★ L7-filter is a kernel-level packet classifier that uses application-layer data to identify packets
- ★ Main goal is to identify P2P communications that use unpredictable port numbers
- ★ Regardless of the port being used, it can classify packets for many protocols including HTTP, Jabber, BitTorrent and FTP
- ★ Best used in conjunction with Linux QoS to perform traffic shaping and accounting
- ★ See <http://l7-filter.sourceforge.net/>

Proxy

- ★ A client is explicitly aware of the proxy and connects to it, rather than the real server
- ★ A proxy server may act as a firewall by responding or not responding to a packet
- ★ Makes tampering with an internal system from the external network more difficult
- ★ Misuse of one internal system would not necessarily cause a security breach exploitable from the outside
- ★ Runs at the application layer
- ★ Hides the internal network
- ★ A proxy is very similar to an ALG
 - ▶ An ALG works without the application being configured to use it
 - ▶ A proxy usually needs to be configured in the client application

SOCKS Proxy protocol

- ★ SOCKet Secure (SOCKS)

- ★ Generic proxy protocol
 - ▶ Don't have to modify code to use a proxy

- ★ Can be used by HTTP, FTP, telnet, SSL,...

- ▶ Independent of application layer protocol

- ★ Includes authentication, limiting which users, apps or IP addresses can pass through the firewall



Source: wikimedia.org

Application Gateways: Pros & Cons

★ Advantages

- ▶ Proxy can log all connections, activity in connections
- ▶ Proxy can provide caching
- ▶ Proxy can do intelligent filtering based on content
- ▶ Proxy can perform user-level authentication

★ Disadvantages

- ▶ Not all services have proxied versions
- ▶ May need different proxy server for each service
- ▶ Requires modification of client
- ▶ Performance

Linux Internals: Netfilter & Iptables

★ All packets pass through netfilter

▶ Consists of callback hooks within the kernel

- `NF_IP_PRE_ROUTING`
- `NF_IP_LOCAL_IN`
- `NF_IP_LOCAL_OUT`
- `NF_IP_POST_ROUTING`
- `NF_IP_FORWARD`

▶ Filters packets based on a rule set

▶ Callbacks invoked for every packet that traverses the respective hook within the network stack

★ Firewall managed by iptables

▶ Complex command that supplies rules to netfilter

- Command line interface
- GUI alternatives available (UFW, firestarter)

iptables – Description

- ❖ Admin tool for IPv4 packet filtering and NAT
- ❖ Used to manage the tables of IPv4 packet filter rules in the Linux kernel
- ❖ Several different tables may be defined
 - ▶ Each table contains a number of built-in chains and may also contain user-defined chains
- ❖ Each chain is a list of rules which can match a set of packets
- ❖ Each rule specifies criteria for a packet and a target
 - ▶ Target specifies what to do with the packet

iptables – Targets

- ★ If a packet does not match a rule, then the next rule in the chain is examined
- ★ If a packet matches a rule, then packet handling is determined by the rule's target
- ★ The target is either the name of a user-defined chain or one of the special values **ACCEPT**, **DROP**, **QUEUE**, **REJECT**, **LOG** or **RETURN**

iptables – Targets (2)

ACCEPT

- ▶ Let the packet through

DROP

- ▶ Drop the packet on the floor

QUEUE

- ▶ Pass the packet to user space

REJECT

- ▶ Drop the packet, but send an ICMP message back to the source telling them that the packet was administratively dropped

LOG

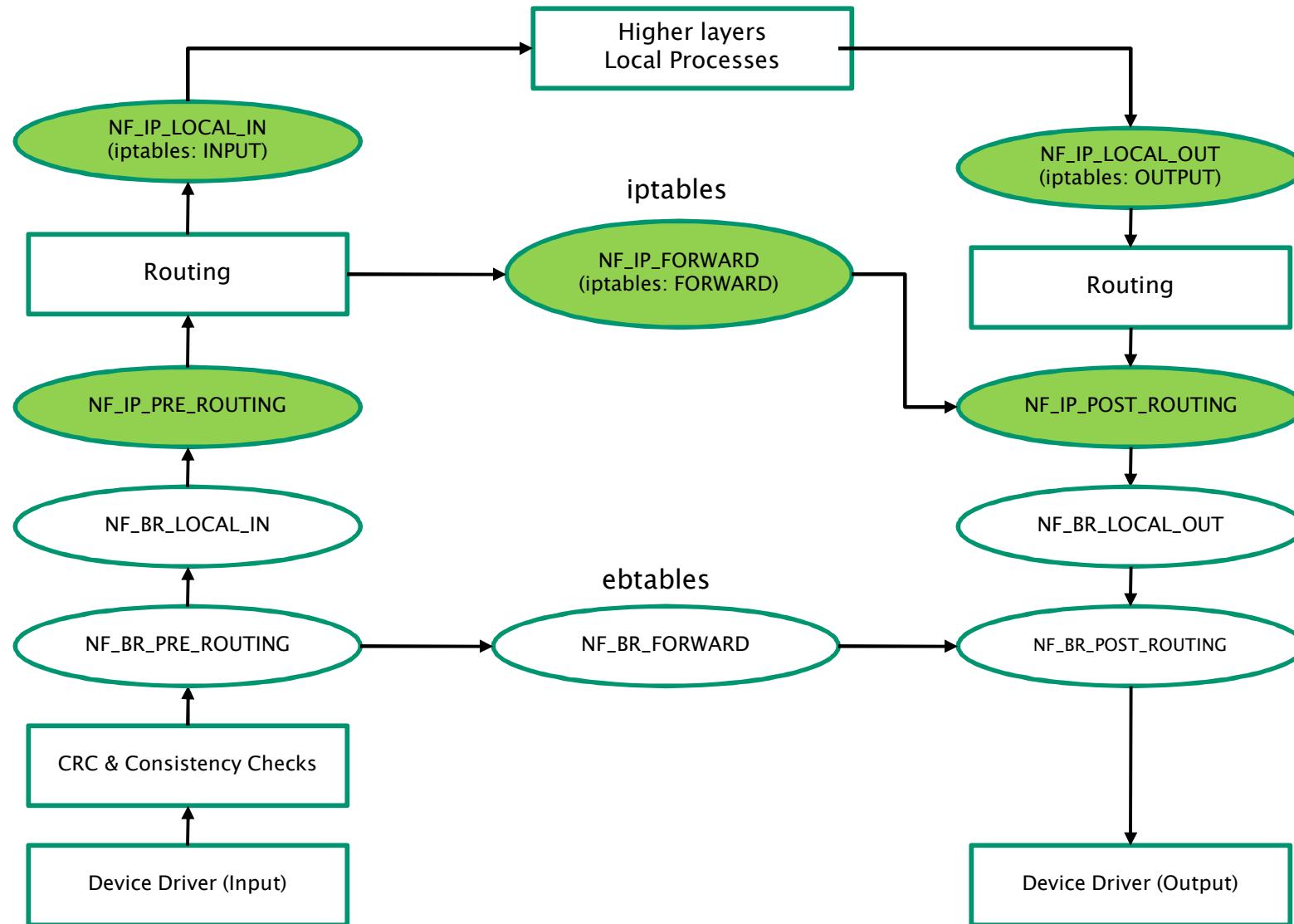
- ▶ Send a copy of the packet to the logging facility

RETURN

- ▶ Stop traversing current chain and resume at the next rule in the previous (calling) chain

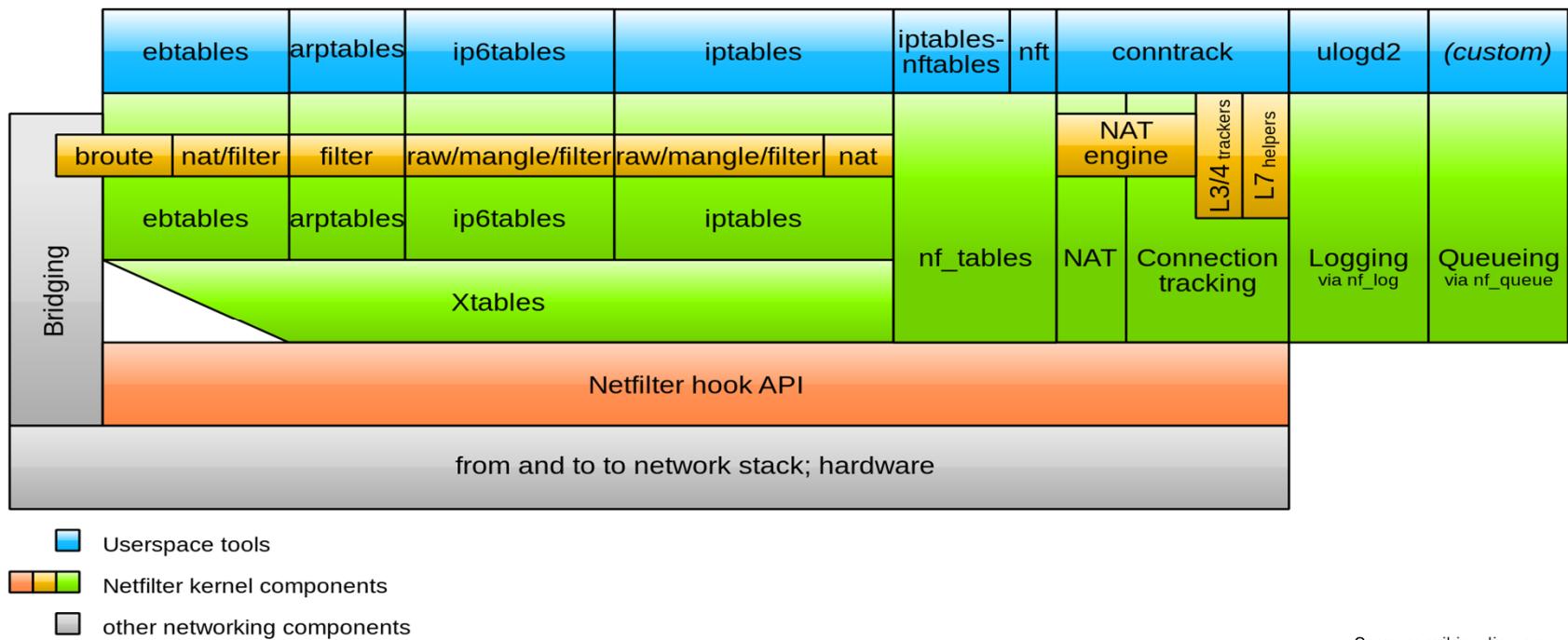
If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain's default policy determines the fate of the packet

Netfilter Kernel Hooks for L3



Netfilter Architecture

- ★ ebttables: Manages ruleset for Ethernet packet frames
- ★ arptables: Manages ruleset for ARP packet frames
- ★ ip6tables: iptables for IPv6
- ★ conntrack: Manage in-kernel connection state table



Iptables Ruleset Hierarchy

* Iptables -> Tables -> Chains -> Rules

* Table types: Filter, NAT, Mangle & Raw

TABLE 1

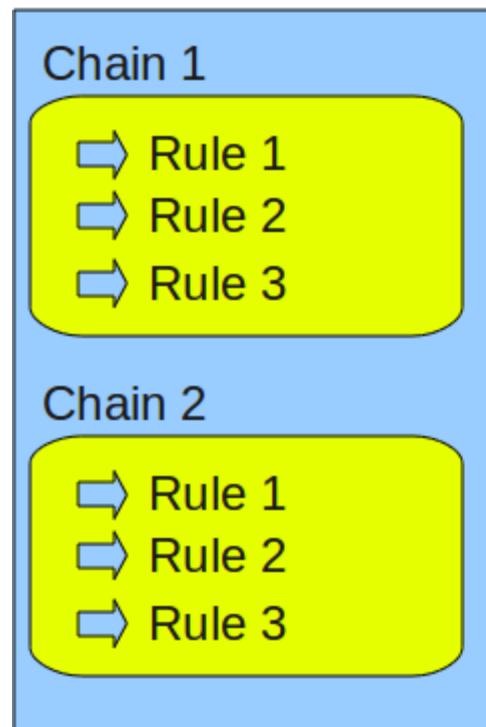
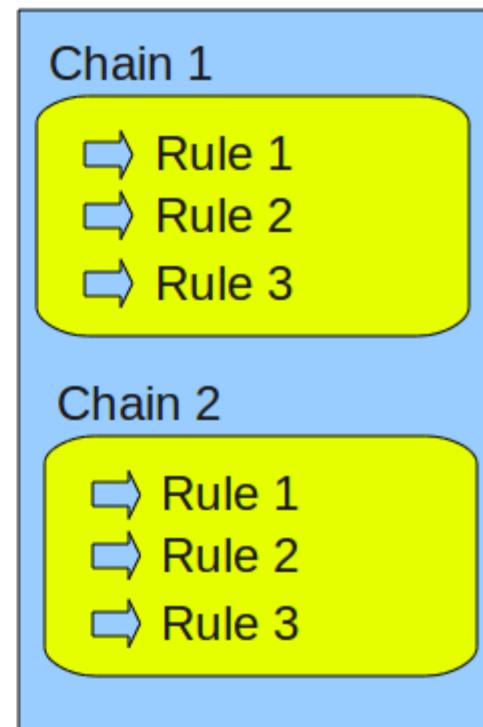


TABLE 2



Source: thegeekstuff.com

Filter Table Chains

- ★ A filter table is the default table type for the iptables command
 - ▶ No '-t' option is required
- ★ Built-in chains
 - ▶ INPUT – packets coming in from the network
 - ▶ OUTPUT – packets generated locally and going out to the network
 - ▶ FORWARD – packet for another host
 - Packets routed through the local host

NAT Table Chains

★ Network Address Translation (NAT)

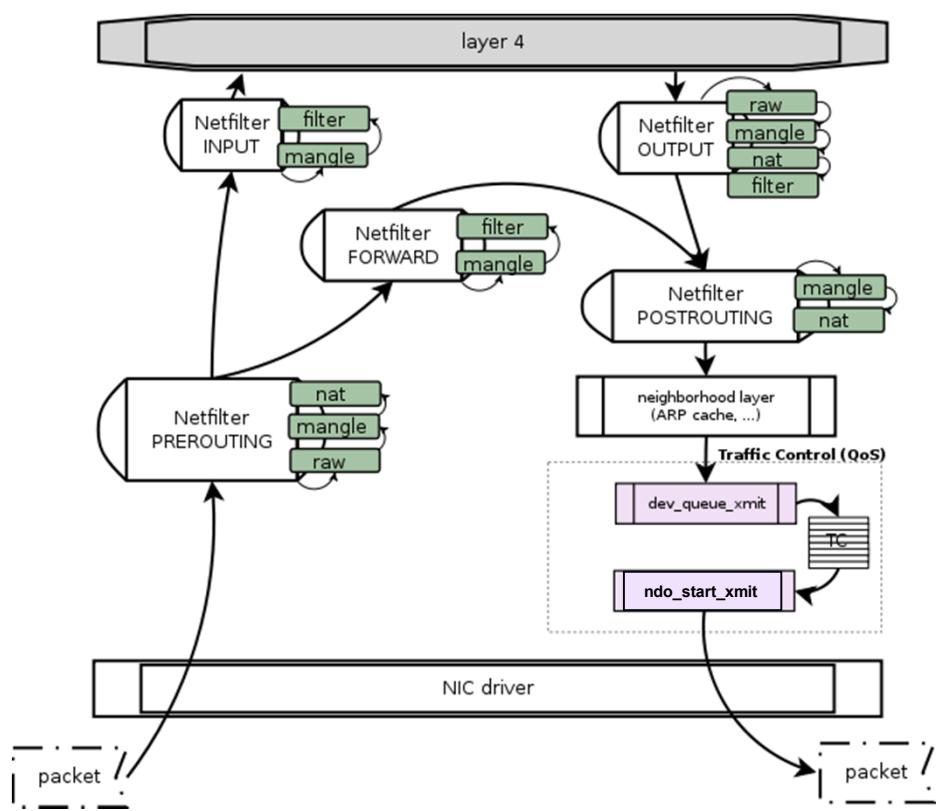
- ▶ Translates IP addresses and port numbers between external network and local network
- ▶ Specified via '`-t nat`' option to `iptables` command

★ Built-in chains

- ▶ PREROUTING – Alters packets immediately after arrival and before routing (used for DNAT)
- ▶ POSTROUTING – Alters packets after routing when they are leaving the system (used for SNAT)
- ▶ OUTPUT – NAT for packets generated on the local host

Using Iptables to Perform NAT

- ★ Incoming packets looking for routing will be passed from the NIC thru PREROUTING to the FORWARD stage
 - ▶ Connection requests from the Internet are not accepted
 - ▶ The remaining packets are passed to the POSTROUTING stage
- ★ At POSTROUTING, the source address is modified to be the router's address



Source: linuxwall.info

Iptables Targets for NAT

★ Target: An action associated with a rule

- ▶ Source NAT (SNAT): Rewrites the source IP of the packet with the router's static IP
 - Only meaningful in the POSTROUTING chain
- ▶ MASQUERADE: Like SNAT for routers with dynamic addresses (i.e., DHCP)
- ▶ DNAT: Rewrites the destination IP of the packet
 - Meaningful in the PREROUTING and OUTPUT chains

★ Remember that rule order matters

Configuring NAT on Linux

- ★ Ensure that the `iptable_nat` kernel module is loaded

```
$ lsmod | grep iptable_nat
```

- ▶ Otherwise, modify `/etc/rc.local` or `/etc/modules` to install it

- ★ Ensure that routing is enabled

```
$ cat /proc/sys/net/ipv4/ip_forward
```

- ▶ If not, set `net.ipv4.ip_forward` to 1 in `/etc/sysctl.conf`

- ★ Configure the kernel to change the source address on all packets that are going to the Internet

- ▶ Example:

```
$ iptables -A POSTROUTING -t nat -o eth0  
          -s 192.168.0.0/24 -d 0/0  
          -j MASQUERADE
```

Configuring NAT on Linux (2)

- ★ Ensure that all packets from the Internet belong to **ESTABLISHED** or **RELATED** connections

```
$ iptables -A FORWARD -t filter -i eth0  
        -m state --state ESTABLISHED,RELATED  
        -j ACCEPT
```

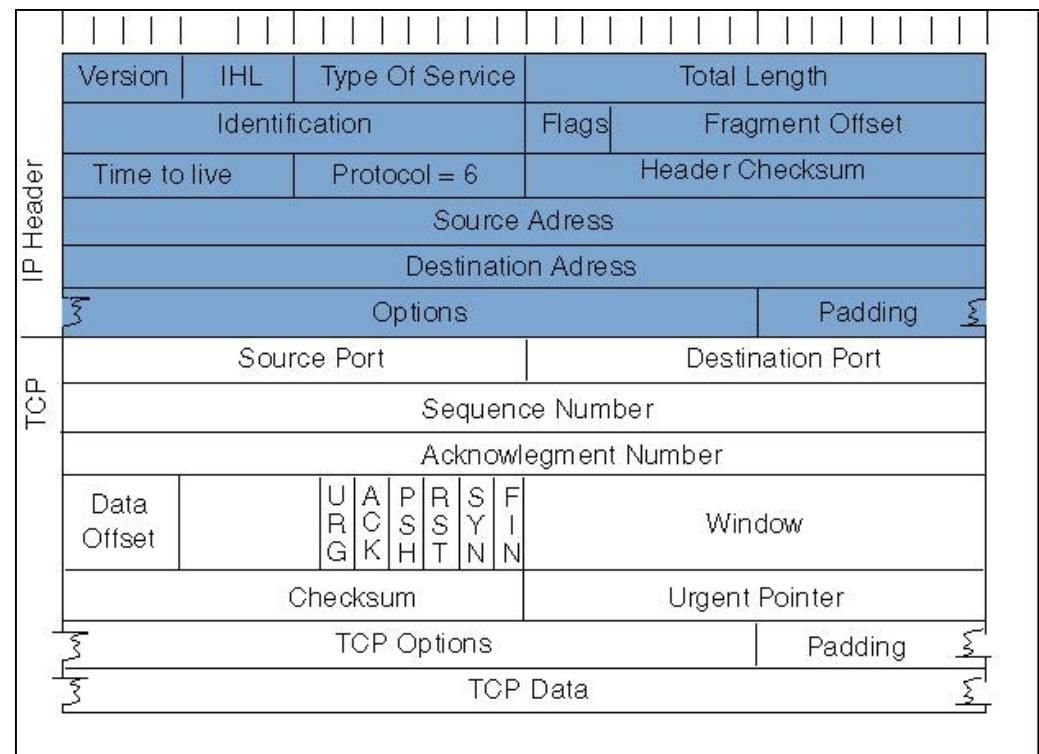
- ★ Ensure that packets for **NEW** connections can be sent to the Internet

```
$ iptables -A FORWARD -t filter -o eth0  
        -m state  
        --state NEW,ESTABLISHED,RELATED  
        -j ACCEPT
```

- ★ Note: If you configure your firewall to do masquerading, then it should be used as the default gateway for all hosts on your network

Additional Packet Header Changes

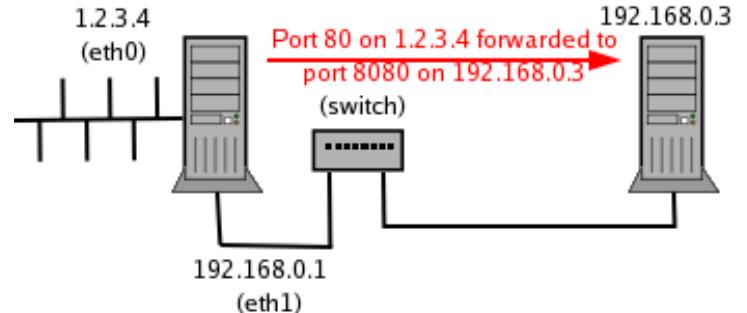
- Since ‘NATing’ changes the packet’s IP address, the IP header and TCP header checksums need to be recalculated as well
- Netfilter hooks are responsible for this unless offloading is enabled



Source: cisco.com

Port Forwarding

- Setting up a server (i.e., Web/HTTP server) behind a NAT router is problematic
- NAT router has no way of telling which host on the private network should receive the packet from the public client, so the packet is dropped
- Port forwarding allows you to route all traffic to port 80 from the Internet to a specific private network host
 - Port forwarding is handled by the PREROUTING chain of the NAT table



Source: linuxgazette.net

Configuring Port Forwarding

- * As before, ensure that the `iptables_nat` module is loaded, and that the `ip_forward` `sysctl` is set
- * Get the dynamic IP address (DHCP) of the router's Internet interface (eth0 in this example)

```
$ external_ip=`ifconfig "eth0" | grep 'inet addr'  
| awk '{print $2}' | sed -e 's/.*/\``'
```

- * Allow traffic to port 80 of the firewall's external IP address to be forwarded to port 8080 on host 192.168.0.3

```
# iptables -t nat -A PREROUTING -p tcp -i eth0  
-d $external_ip --dport 80 --sport 1024:65535  
-j DNAT --to 192.168.0.3:8080
```

Configuring Port Forwarding (2)

- ★ After DNAT packets are routed via the filter table's FORWARD chain, accept NEW connections on port 8080 for the target machine on the private network (eth1)

```
# iptables -A FORWARD -p tcp -i eth0 -o eth1  
          -d 192.168.0.3 --dport 8080  
          --sport 1024:65535 -m state --state NEW  
          -j ACCEPT
```

- ★ On the Internet interface allow all connections out but don't accept any other NEW connections

```
# iptables -A FORWARD -t filter -o eth0 -m state  
          --state NEW,ESTABLISHED,RELATED -j ACCEPT  
# iptables -A FORWARD -t filter -i eth0 -m state  
          --state ESTABLISHED,RELATED -j ACCEPT
```

NAT Drawbacks

- ✖ Adds complexity to system configuration and management (i.e., port forwarding)
- ✖ Hurts performance
 - ▶ Each packet requires address translation, checksum recalculation, etc.
- ✖ Many protocols require full end-to-end connectivity
 - ▶ IPsec headers are digitally signed, preventing modification
 - Workaround via NAT-T (NAT traversal in IKE), but more overhead
 - ▶ FTP client expects the FTP server to initiate creation of the data channel by default
 - New incoming connection from the server will be dropped
 - Workaround is to use the non-default “passive” mode
 - ▶ And many more ...
- ✖ Each solution is application and protocol specific

Mangle Table Chains

★ Mangle table is for specialized packet alteration

- ▶ Specified via the '`-t mangle`' option

★ Built-in chains

- ▶ PREROUTING
- ▶ OUTPUT
- ▶ FORWARD
- ▶ INPUT
- ▶ POSTROUTING

What Fields Can Be Mangled?

* Time-To-Live (TTL)

- ▶ Also known as Hop Limit
- ▶ Can be used to hide the number of computers on a LAN

* Explicit Congestion Notification (ECN)

- ▶ Used to turn off ECN if communicating with older networking equipment

* Differentiated Services Code Point (DSCP)

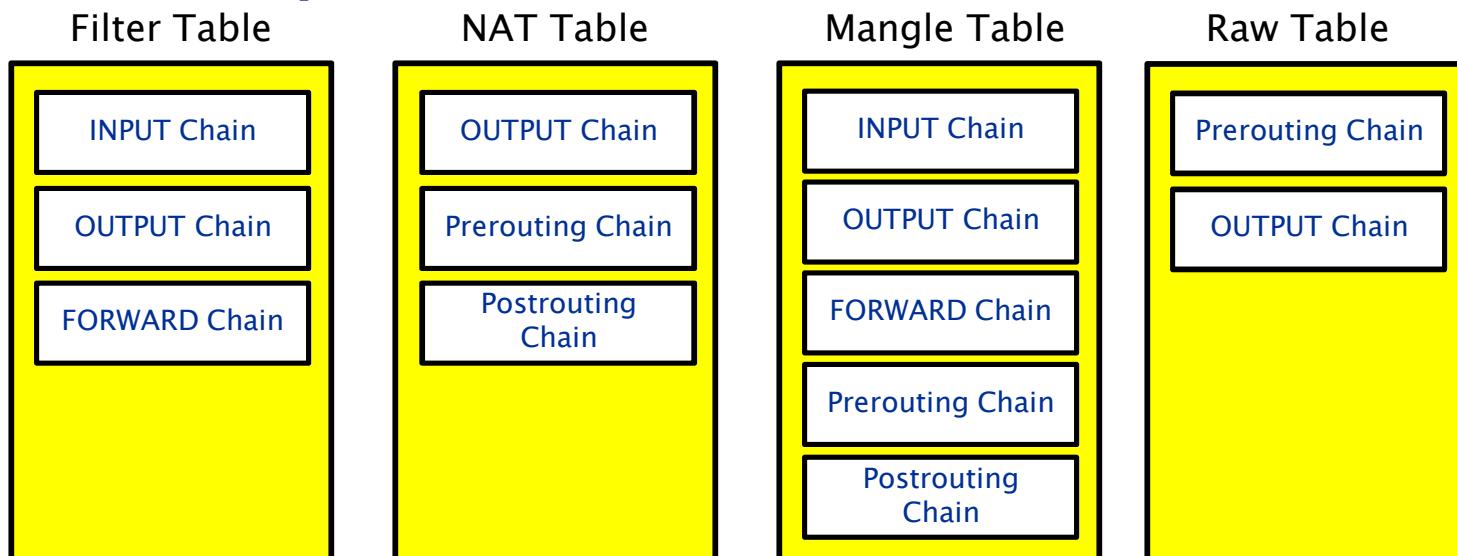
- ▶ When combined with QoS, effectively changes the packet stream priority

Raw Table Chains

* Raw table for configuration exceptions

- ▶ Specified via '`-t raw`' option
- ▶ Built-in chains: PREROUTING & OUTPUT

* Filter, NAT & Mangle tables are the most commonly used



Example Rules

* Block a specific IP address

```
$ SPAMMER_IP="206.46.232.39"  
$ iptables -A INPUT -s "$SPAMMER_IP" -j DROP
```

* Allow all incoming ssh connections on the eth0 interface

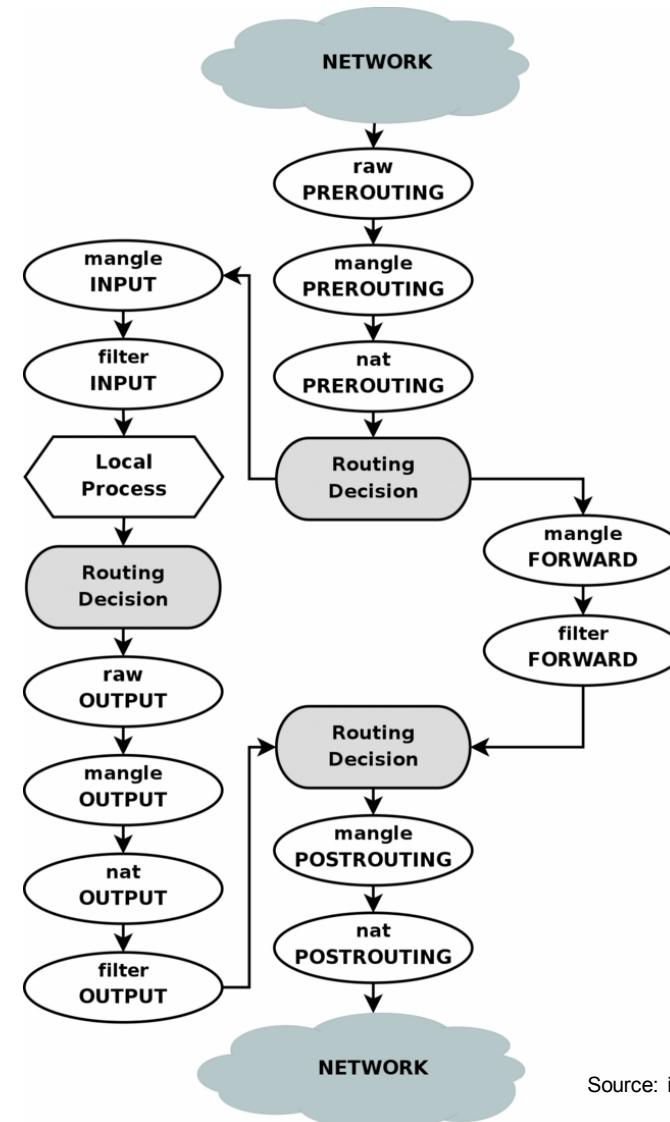
```
$ iptables -A INPUT -i eth0 -p tcp --dport 22  
      -m state --state NEW,ESTABLISHED -j ACCEPT  
$ iptables -A OUTPUT -o eth0 -p tcp --sport 22  
      -m state --state ESTABLISHED -j ACCEPT
```



Source: halloweencostumes.com

Traversing Chains and Tables

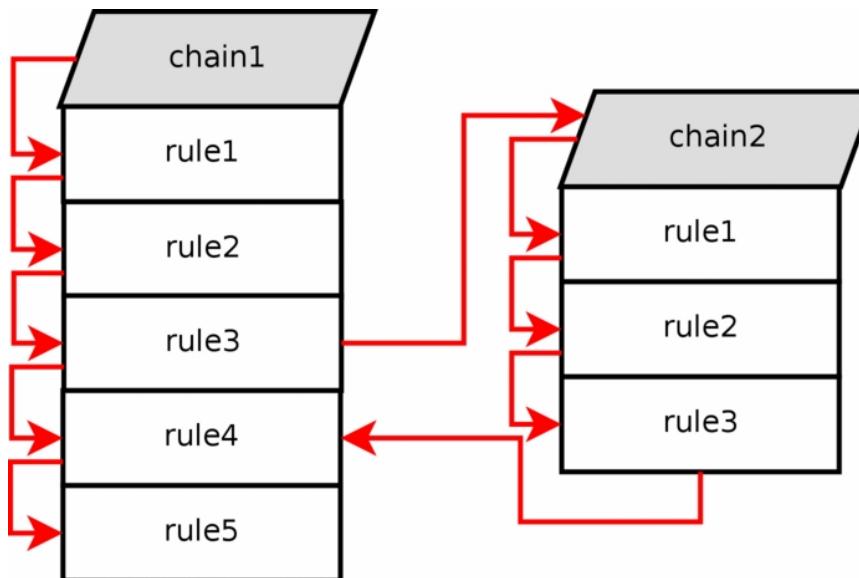
- ★ Each chain has a specific order of tables
 1. raw
 2. mangle
 3. nat
 4. filter
- ★ Packets can be dropped by any chain



Source: iptables.info

Jumping Chains

- * Jumps to a user-defined chain (chain 2) within the same table are allowed
- * Traversal ended by a target or the previous chain (chain 1)



Source: iptables.info

Example Traversal

Step	Table	Chain	Comment
1			On the wire (e.g., Internet)
2			Comes in on the interface (e.g., eth0)
3	raw	PREROUTING	This chain is used to handle packets before the connection tracking takes place. It can be used to set a specific connection not to be handled by the connection tracking code for example.
4			Connection tracking occurs here
5	mangle	PREROUTING	This chain is normally used for mangling packets, e.g., changing TTL
6	nat	PREROUTING	This chain is used for DNAT mainly. Avoid filtering in this chain since it will be bypassed in certain cases.
7			Routing decision. Is the packet destined for our local host or to be forwarded, and if forwarded, where.
8	mangle	INPUT	At this point, the mangle INPUT chain is hit. We use this chain to mangle packets, after they have been routed, but before they are actually sent to the process on the machine.
9	filter	INPUT	This is where we do filtering for all incoming traffic destined for our local host. Note that all incoming packets destined for this host pass through this chain, no matter what interface or in which direction they came from.
10			Local process or application (server or client program).

Source: iptables.info

Linux Firewall Configuration

- ★ A firewall is useless until it is properly configured
- ★ Default-allow policy
 - ▶ Everything is allowed from inside the firewall
 - ▶ Prone to omissions
 - But, you'll never know because everything will still work
- ★ Default-deny policy
 - ▶ Proper security practice
 - ▶ May initially inconvenience users since their network-centric application may stop working

Firewall Configuration Example

- ★ The following slide sequence takes us through a simple example of setting up a firewall on a host connected directly to the Internet
 - ▶ Not a standalone firewall fronting a private network
 - ▶ No NAT, mangling, etc.
- ★ Rules are case sensitive
- ★ Order is important in rule chains

Enable Logging

* Enable logging to help with debugging

- ▶ To avoid cluttering up the syslog file add the following line to `/etc/syslog.conf`:

```
kern.=debug /var/log/iptables
```

- ▶ Restart syslog daemon:

```
$ sudo service syslog restart
```

- ▶ Later add logging options to rules of interest

Start Fresh

- ★ Cleaning out all chains will block packets from all sources
 - ▶ Add an INPUT chain policy to leave the front door open
`$ iptables -P INPUT ACCEPT`
- ★ A chain policy tells the firewall the default behavior to take for a packet on that chain, if there is no rule that matches the packet
- ★ Flush the built-in filter chains
 - `$ iptables -F INPUT`
 - `$ iptables -F OUTPUT`
 - `$ iptables -F FORWARD`
- ★ If applicable, flush and delete any user-defined chains
 - `$ iptables -X <user-defined>`

Display Fresh Tables

- * A fresh iptables configuration should look very similar to the results below:

```
$ iptables -L -n -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
```

- * Option '**-L**' displays the list
- * Option '**-n**' suppresses DNS lookups of IP addresses

Create Custom Chains

* Use these user-defined chains as rule targets

- ▶ SPAM – List of known spammers
- ▶ WEB – List of known website attackers
- ▶ BLACKLIST – A catch-all list used to block specific users on specific (or all) ports
- ▶ THRU – This chain will verify desired ports for desired services
- ▶ LOGDROP – A simple chain that logs packets to the log file and then drops the packet

* Create the chains

```
$ iptables -N SPAM  
$ iptables -N WEB  
$ iptables -N BLACKLIST  
$ iptables -N THRU  
$ iptables -N LOGDROP
```

The First Rule

★ Accept input packets for “established” (not new) connections

```
$ iptables -A INPUT -i eth0 -m state  
          --state RELATED,ESTABLISHED -j ACCEPT
```

- ▶ Append a rule to the INPUT chain (`iptables -A INPUT`)
- ▶ Rule criteria 1: If packet is input from eth0 (`-i eth0`)
- ▶ Rule criteria 2: And if packet belongs to an existing connection (`-m state --state RELATED,ESTABLISHED`)
- ▶ Rule target: Then jump to the ACCEPT target (`-j ACCEPT`)

Allow localhost

* Allow all incoming packets from the localhost interface ‘lo’ so that all local processes can communicate with each other

```
$ iptables -A INPUT -i lo -j ACCEPT
```



Source: thinkgeek.com

What to Do With Oddball Packets?

* Drop incoming malformed TCP packets

```
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -j DROP  
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags FIN,SYN FIN,SYN -j DROP  
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags SYN,RST SYN,RST -j DROP  
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags FIN,RST FIN,RST -j DROP  
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags FIN,ACK FIN -j DROP  
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
    --tcp-flags ACK,URG URG -j DROP
```



Source: typepad.com

* Equivalent experimental command

```
$ iptables -A INPUT -i eth0 -m unclean -j DROP
```

Process User-Defined Rules

- ★ Send any incoming packet on TCP port 25 (SMTP mail server) to the SPAM chain

```
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
--dport 25 -j SPAM
```

- ★ Validate web client access (WEB chain)

```
$ iptables -A INPUT -i eth0 -p tcp -m tcp  
--dport 80  
--tcp-flags SYN,RST,ACK SYN -j WEB
```

- ★ The SPAM and WEB chain rules will be set up to block specific IPs, like one of our previous example rules

Blacklist IP Sources

- * There are sites which keep up-to-date lists of bad IP addresses
- * The following site provides a list of bad mail sources (port 25) from China and Korea in iptables rule format:

<http://www.okean.com/antispam/iptables/rc.firewall.sinokorea>



Source: bestvpnservice.com

Continue the INPUT Chain

- ★ Check against the general purpose blacklist chain

```
$ iptables -A INPUT -j BLACKLIST
```

- ★ Move on to the THRU chain to check for packets to explicitly allow

```
$ iptables -A INPUT -j THRU
```

- ★ Log any packets that fail to get addressed by our other chains

```
$ iptables -A INPUT -m limit --limit 1/sec  
          -j LOGDROP --log-prefix "drop_packet:"  
          --log-level 7
```

The THRU Chain

- * The THRU chain is used to explicitly state which ports we allow through
- * Limit pings to one per second

```
$ iptables -A THRU -p icmp -m limit --limit 1/sec  
          -m icmp --icmp-type 8 -j ACCEPT
```

- * Accept incoming connections to SSH, mail, HTTP, FTP & POP3

```
$ iptables -A THRU -i eth0 -p tcp -m tcp --dport 22 -j ACCEPT  
$ iptables -A THRU -i eth0 -p tcp -m tcp --dport 25 -j ACCEPT  
$ iptables -A THRU -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT  
$ iptables -A THRU -i eth0 -p tcp -m tcp --dport 21 -j ACCEPT  
$ iptables -A THRU -i eth0 -p tcp -m tcp --dport 110 -j ACCEPT
```

As Configured So Far

- ★ With this current configuration, the system is reasonably well protected
- ★ We have:
 - ▶ Allowed in the specific packets that we have checked
 - ▶ Ensured that the localhost processes can communicate
 - ▶ Blocked malformed packets
 - ▶ Logged packets that make it through the INPUT rule set
 - Logging them allows further analysis
 - These will be dropped by a rule we are about to add...
 - ▶ Opened up specific ports that we want to use, and implicitly blocked all other ports

Finish the Configuration

- * Add the last line to the INPUT chain
 - ▶ Drop the packet if it passed through all the previous rules

```
$ iptables -A INPUT -j DROP
```
- * To be safe, add the following policy in case we add more rules later

```
$ iptables -P INPUT DROP
```

Resulting Iptables Listing

* Filter table INPUT chain listing

1. Accept packets for established connections from eth0 interface
2. Accept all packets from lo interface
3. Drop any malformed TCP packets
4. If destination port is 25, jump to SPAM chain
5. If destination port is 80, jump to WEB chain
6. Jump to BLACKLIST and THRU chains
7. Log and drop any packets that fall through

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
pkts bytes target     prot opt in     out      source          destination
194K  41M ACCEPT    all  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      state RELATED,ESTABLISHED
45321 76M ACCEPT    all  --  lo     ^       0.0.0.0/0        0.0.0.0/0
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x3F/0x00
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x03/0x03
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x06/0x06
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x05/0x05
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x11/0x01
      0    0 DROP      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp flags:0x30/0x20
465 27669 SPAM      tcp  --  *      ^       0.0.0.0/0        0.0.0.0/0      tcp dpt:25
1255 61764 WEB      tcp  --  eth0   ^       0.0.0.0/0        0.0.0.0/0      tcp dpt:80 flags:0x16/0x02
3801 229K BLACKLIST all  --  *      ^       0.0.0.0/0        0.0.0.0/0
3762 227K THRU      all  --  *      ^       0.0.0.0/0        0.0.0.0/0
1378 96990 LOG      all  --  *      ^       0.0.0.0/0        0.0.0.0/0      limit: avg 1/sec burst 5 LOG flags 0 level 7 prefix 'drop_packet'
1551 105K DROP      all  --  *      ^       0.0.0.0/0        0.0.0.0/0
```

Source: pettingers.org

Resulting Iptables Listing (2)

- ★ FORWARD chain defaults to its policy (DROP)
- ★ The OUTPUT chain defaults to its policy (ACCEPT)
- ★ Log and reject packets for the 3 SSH clients on the BLACKLIST chain
 - ▶ Limit the logging rate to 1 Hz

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out    source          destination

Chain OUTPUT (policy ACCEPT 249K packets, 115M bytes)
  pkts bytes target     prot opt in     out    source          destination

Chain BLACKLIST (1 references)
  pkts bytes target     prot opt in     out    source          destination
      16   960 LOGDROP    tcp  --  *       *    218.8.127.193    0.0.0.0/0        tcp dpt:22
      18  1080 LOGDROP    tcp  --  *       *    205.209.180.130    0.0.0.0/0        tcp dpt:22
       0    0 LOGDROP    tcp  --  *       *    213.25.122.250    0.0.0.0/0        tcp dpt:22

Chain LOGDROP (538 references)
  pkts bytes target     prot opt in     out    source          destination
      18   864 LOG       tcp  --  *       *    0.0.0.0/0        0.0.0.0/0        tcp dpt:25 limit: avg 1/sec burst 5 LOG flags 0 level 7 prefix 'spam_blacklist'
       0    0 LOG       tcp  --  *       *    0.0.0.0/0        0.0.0.0/0        tcp dpt:80 limit: avg 1/sec burst 5 LOG flags 0 level 7 prefix 'web_blacklist'
      15   900 LOG       tcp  --  *       *    0.0.0.0/0        0.0.0.0/0        tcp dpt:22 limit: avg 1/sec burst 5 LOG flags 0 level 7 prefix 'ssh_blacklist'
      52  2904 REJECT    all  --  *       *    0.0.0.0/0        0.0.0.0/0        reject-with icmp-host-prohibited
```

Source: pettingers.org

Resulting Iptables Listing (3)

- ★ Log and drop all packets with IPs on the WEB and SPAM chains
- ★ Accept all packets for the 5 services listed on the THRU chain

```
Chain SPAM (1 references)
pkts bytes target  prot opt in     out    source          destination
   0    0 LOGDROP  all  --  +      +      222.249.0.0/17  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      222.248.0.0/16  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      222.240.0.0/13  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      222.232.0.0/13  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      59.0.0.0/11    0.0.0.0/0

Chain THRU (1 references)
pkts bytes target  prot opt in     out    source          destination          limit: avg 1/sec burst 5 icmp type 8
  15   621 ACCEPT  icmp  --  +      +      0.0.0.0/0    0.0.0.0/0
  42  2088 ACCEPT  tcp   --  eth0   +      0.0.0.0/0    0.0.0.0/0          tcp dpt:22
 438 25776 ACCEPT  tcp   --  eth0   +      0.0.0.0/0    0.0.0.0/0          tcp dpt:25
1255 61764 ACCEPT  tcp   --  eth0   +      0.0.0.0/0    0.0.0.0/0          tcp dpt:80
  10   492 ACCEPT  tcp   --  eth0   +      0.0.0.0/0    0.0.0.0/0          tcp dpt:21
 269 16164 ACCEPT  tcp   --  eth0   +      0.0.0.0/0    0.0.0.0/0          tcp dpt:110

Chain WEB (1 references)
pkts bytes target  prot opt in     out    source          destination
   0    0 LOGDROP  all  --  +      +      212.135.188.179  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      203.31.169.16   0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      213.253.126.91  0.0.0.0/0
   0    0 LOGDROP  all  --  +      +      202.38.216.127  0.0.0.0/0
```

Source: pettingers.org

Save Firewall Settings

- ★ The iptables configuration is lost on system reboot, so save it to a file

```
$ iptables-save -c > iptables.bak
```

- ★ To re-establish it on the next boot, place the following command in a startup script (such as /etc/rc.d/rc.local)

```
$ iptables-restore -c < iptables.bak
```

- ★ There are also GUI interfaces for creating and maintaining firewalls
 - ▶ E.g., UFW, Firestarter and others

Changes to the Netfilter Internals

- ★ Large servers may have thousands of netfilter rule
 - ▶ Searching for rule matches takes a lot of time
- ★ The Netfilter project has rewritten the searching code to improve search performance
 - ▶ Incorporated in 3.13+
 - ▶ Backwards compatible with **iptables**
 - New **nft** command is available with additional features
 - But, this is not generally available in the repos yet
 - ▶ No change in the **NF_** hooks in the kernel

Summary

- ★ A firewall is one part of a complete network security strategy
- ★ This chapter focused on L3+ firewalls and ALGs
 - ▶ We will address L2 firewalls in a later chapter
- ★ Also need to address:
 - ▶ Encryption
 - ▶ Anti-virus
 - ▶ Authentication & authorization
- ★ Support for firewalls is built into the Linux kernel
- ★ Linux distros also provide many user space capabilities that support firewalls

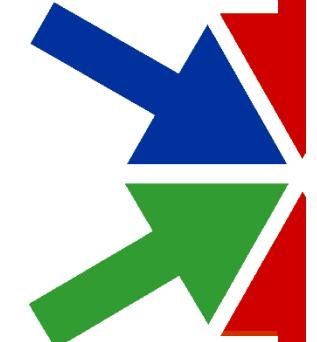
Questions

- * Ports below 1024 are reserved for system use
 - ▶ True or False?
- * Packet filters validate a packet based largely on their L3/L4 headers
 - ▶ True or False?
- * Stateful filters validate connection states based on ICMP host unreachable traffic
 - ▶ True or False?
- * The **iptables** command can be used to manipulate the various netfilter chains
 - ▶ True or False?
- * A “default deny” firewall policy means that any packet not explicitly allowed is dropped
 - ▶ True or False?

Chapter Break

Bridging Networks

Jumping Gaps and L2 Firewalls

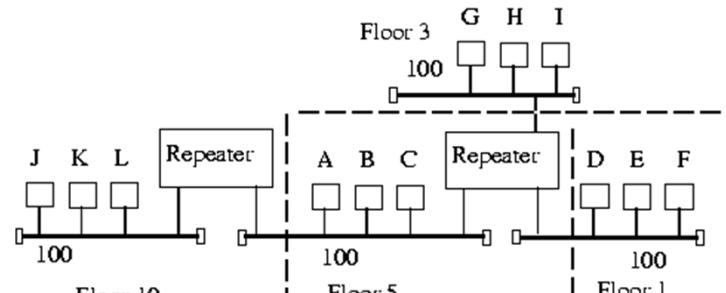


What We Will Cover

- * The players
- * Ethernet background
- * **etables**

The Players

- ✖ A repeater is a device, typically equipped with two ports, that simply copies what it received on one port to the other
 - ▶ Bit-by-bit copy of the data
 - ▶ Not encountered much these days
- ✖ A bridge understands link-layer protocols and copies data frame by frame rather than bit by bit
 - ▶ Our focus for this chapter
 - ▶ Can make filtering decisions
 - ▶ Often implemented as a multi-port bridge referred to as a *switch*
- ✖ A router is a device that understands L3 network protocols and forwards ingress packets based on a routing table
 - ▶ The older term, *gateway*, is still used to refer to this type of device



Source: epq.com.co

Ethernet Background

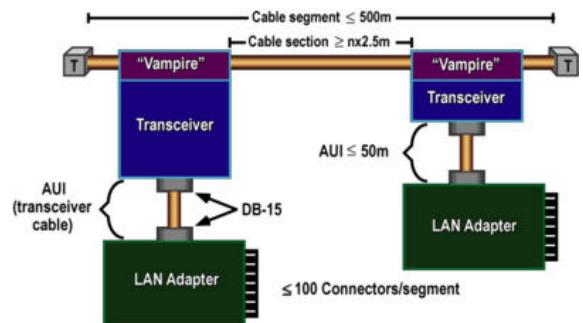
★ Ethernet is a passive, contention-based broadcast technology

★ Collision domain

- ▶ All nodes in the domain see all frames
- ▶ Hubs, bridges, and repeaters preserve the collision domain
- ▶ On a switch, each port is a separate collision domain -- fewer collisions for better throughput

★ Broadcast domain

- ▶ All nodes see each frame sent to the Ethernet broadcast address



Source: hill2dot0.com

Ethernet Background (2)

★ As nodes are added to a collision domain, performance degrades

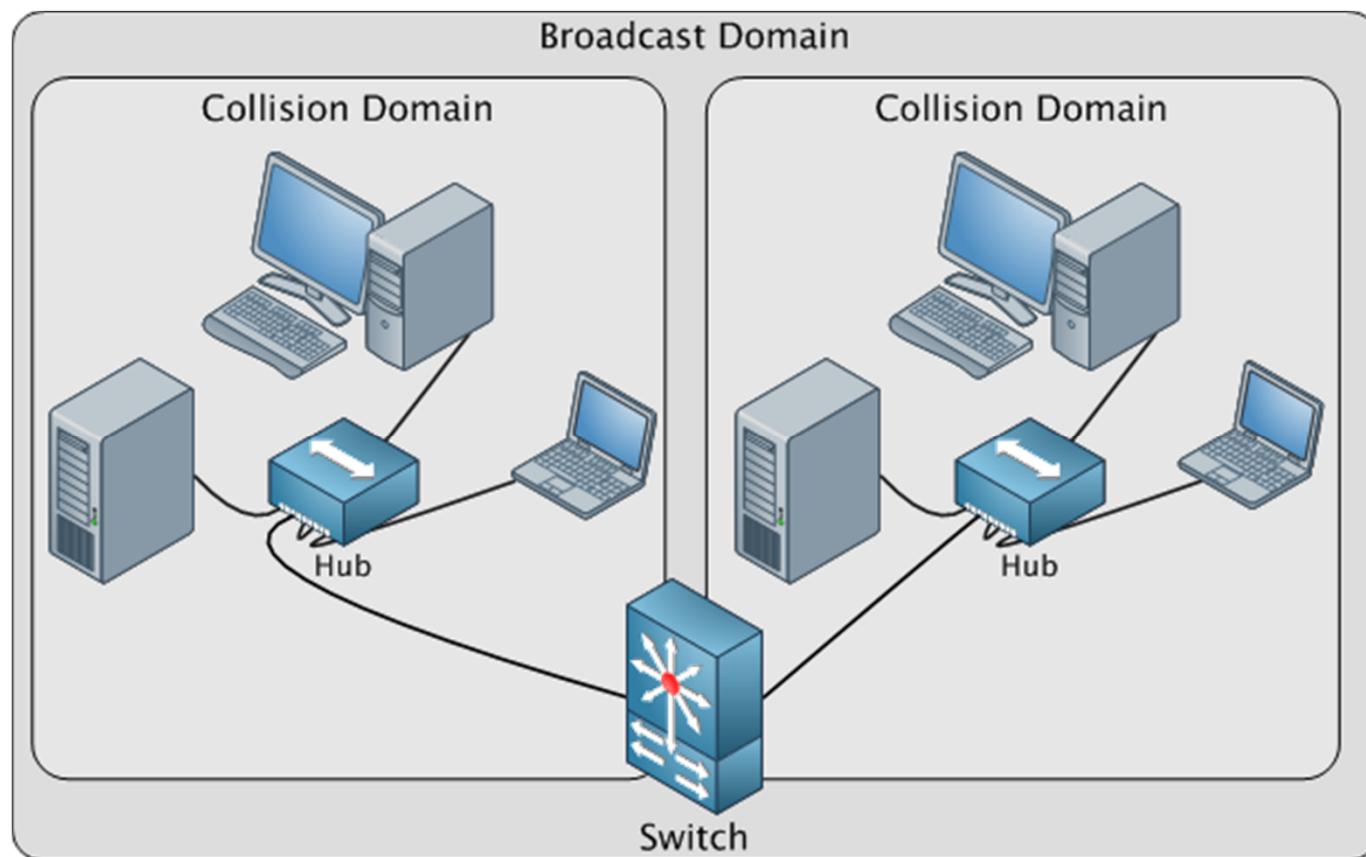
- ▶ Only one node may transmit at a time
- ▶ Switches solve this problem

★ As nodes are added to a broadcast domain, performance degrades

- ▶ Each broadcast on a switched LAN must be sent to each node on the LAN
 - This forces some packet processing even if the packet isn't meant for a given destination machine
- ▶ This can be dealt with using ebttables or VLANs

Collision and Broadcast Domains

- Each switch port is a separate collision domain
- All ports are in the same broadcast domain



Address Learning

- ★ Fortunately, most bridges don't blindly copy frames
 - ▶ They learn which L2 addresses are on which ports
 - Called *passive learning*
- ★ Each port becomes its own collision domain
- ★ Most bridges assume that each port has a unique L2 address
 - ▶ Not the case with Ethernet bonding
 - Requires a special, IEEE 802.3ad (link aggregation) extension to the switch

Broadcast and Multicast Addresses

- ★ When a bridge receives frame addressed to the L2 broadcast address (FF:FF:FF:FF:FF:FF) or to an L2 multicast address, the frame is copied to every port except the one it came from
 - ▶ Multicast and broadcast addresses cannot be used as a source L2 address, so the switch won't get confused
- ★ The switch's knowledge of which L2 addresses are on which ports ages over time
 - ▶ This causes the switch to relearn the port addresses periodically
 - Requires the periodic use of flood routing to all ports
 - Aging period is configurable in managed switches

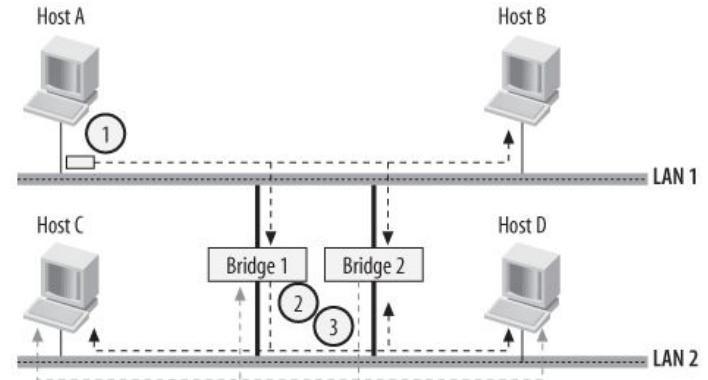
Bridging Loops

* In order to ensure connectivity, it is possible to connect multiple bridges between two LAN segments

- ▶ The bridges are typically transparent and blindly copy the frames

* This can lead to bridging loops where the bridges near-simultaneously copy the frame just as their aging expires

- ▶ Leads to a *broadcast storm* as each bridge sees the packet from the other bridge and thinks that the originating host has moved to the other LAN segment

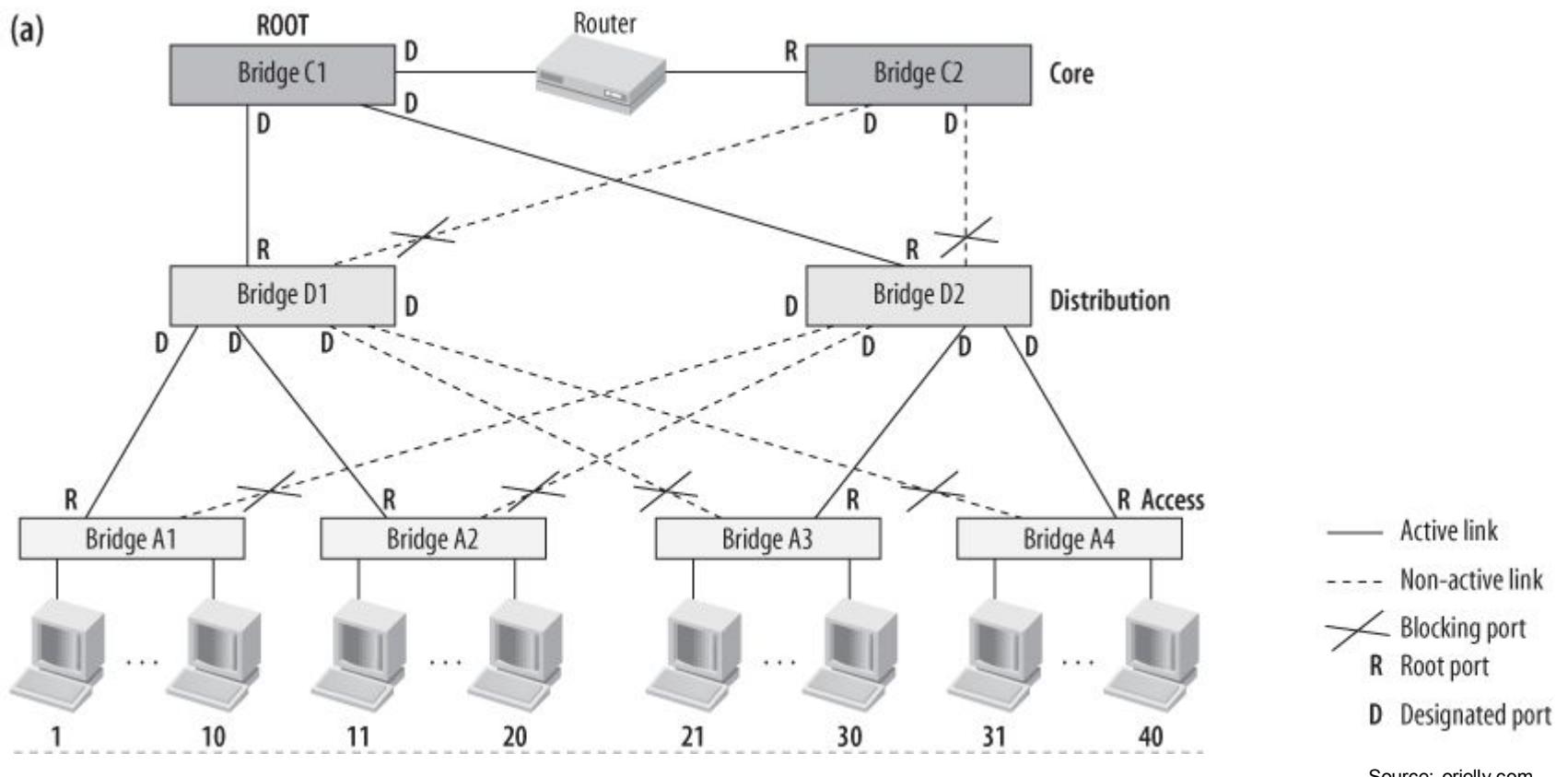


Source: oreilly.com

Handling Loops with STP

- ★ To address the need for redundancy and expansion without introducing bridging loops, IEEE 802.1d Spanning Tree Protocol (STP) was introduced
- ★ With STP, multiple bridges can be arranged in a hierarchy of access bridges, distribution bridges and core bridges
 - ▶ Redundant links are automatically disabled by the STP until a failure is detected

Example L2 Bridge Hierarchy w/ STP



★ STP traffic between the switches ensures that failover is transparent to the hosts

Advantages/Disadvantages of STP

- ★ Use of multiple bridges helps to segregate traffic
- ★ Avoids the need for connecting all hosts to one giant switch
- ★ Configuration is more of an issue
 - ▶ Requires more expensive managed switches
- ★ Convergence time can be significant if the aging is not addressed properly
 - ▶ Changes in topology can take considerable time to propagate
 - ▶ Newer Rapid STP (RSTP) and Multiple STP (MSTP) protocols are now also supported in Linux

Linux and Bridges

- ★ Linux has support for 802.1d STP and is fully cooperative with commercial 802.1d-capable switches
 - ▶ Linux also supports IGMP/MLD snooping for learning about multicast groups
- ★ L2 firewall rules can be combined with iptables to create a bridging IP firewall
 - ▶ Uses the ebtables interface to create “brouters”
 - ▶ Also supports arptables support for limiting ARP broadcasts to specific broadcast domains

Bridging Firewalls

★ **e**bttables – Filtering tool for a Linux-based bridging firewall

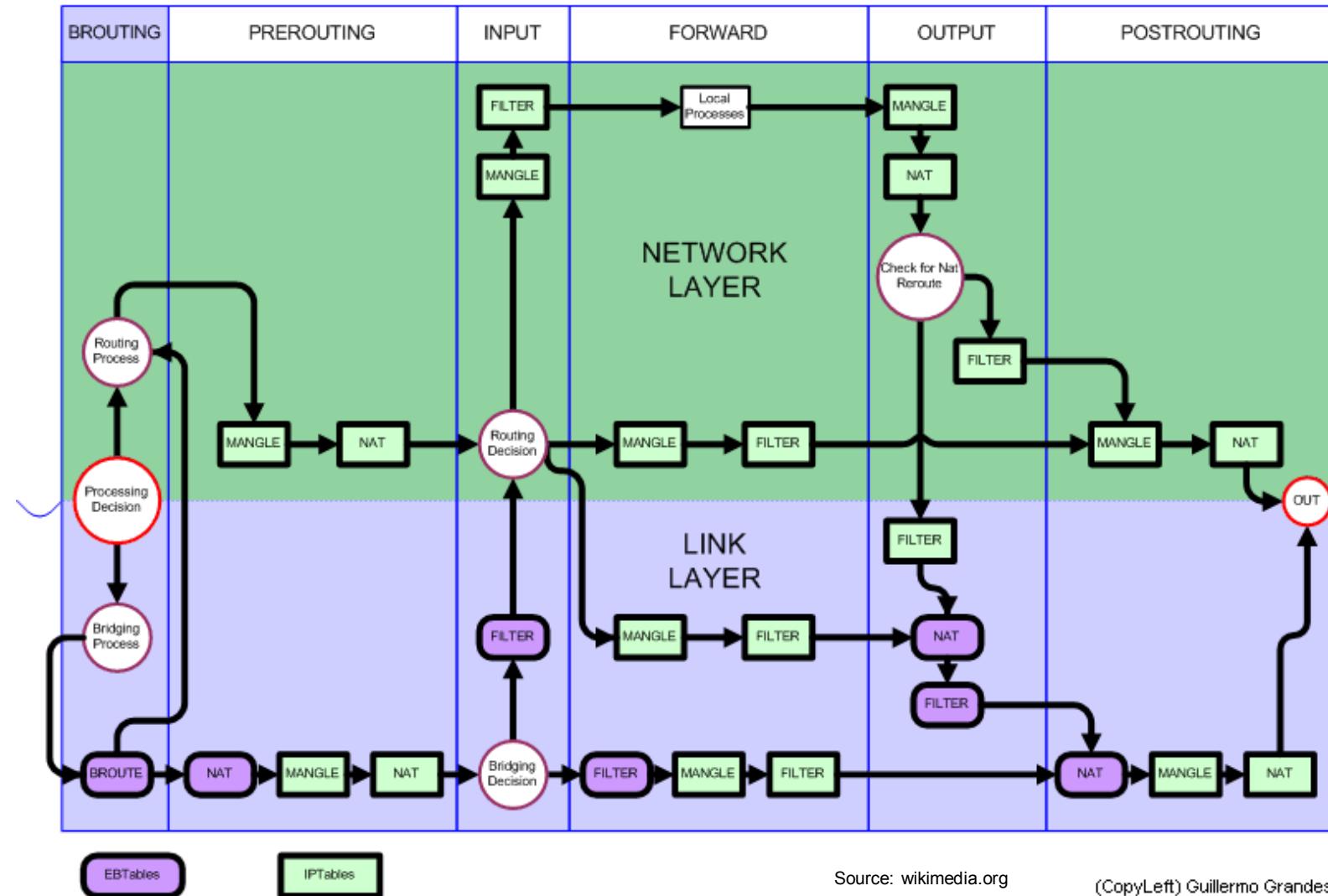
★ Enables transparent filtering of network traffic

- ▶ Link layer filtering
 - Simple, fast, stealthy
- ▶ Basic, higher-layer filtering
 - Can be combined with other tools (iptables, ip6tables, arptables) to provide more advanced higher-layer filtering

ebtables Tables

- ★ There are three **ebtables** tables in the kernel
 - ▶ Filter, NAT and broute
- ★ The filter table is the default table for commands
 - ▶ Used to implement iptables-like matching at L2
- ★ The NAT table allows rewriting of L2 addresses
- ★ The broute table is where we decide to bridge the frame or forward it to L3 netfilter code
 - ▶ **DROP** rule says to route the frame to L3 and **ACCEPT** rule says to bridge the frame

Bridging Firewall Flow



Source: wikimedia.org

(CopyLeft) Guillermo Grandes

Example: Creating a Brouter

* Given a bridge br0 with ports eth0 and eth1:

```
ifconfig br0 0.0.0.0
ifconfig eth0 172.16.1.1 netmask 255.255.255.0
ifconfig eth1 172.16.2.1 netmask 255.255.255.0
ebtables -t broute -A BROUTING -p ipv4 -i eth0 --ip-dst 172.16.1.1 -j DROP
ebtables -t broute -A BROUTING -p ipv4 -i eth1 --ip-dst 172.16.2.1 -j DROP
ebtables -t broute -A BROUTING -p arp -i eth0 -d $MAC_OF_ETH0 -j DROP
ebtables -t broute -A BROUTING -p arp -i eth1 -d $MAC_OF_ETH1 -j DROP
ebtables -t broute -A BROUTING -p arp -i eth0 --arp-ip-dst 172.16.1.1 -j DROP
ebtables -t broute -A BROUTING -p arp -i eth1 --arp-ip-dst 172.16.2.1 -j DROP
```

- * The first 2 **ebtables** commands force packets sent to eth0 and eth1 explicitly to be routed
- * The last 4 are needed to ensure that ARP works by forcing ARP packets to L3

Summary

- ★ Bridges are used to link broadcast domains to make large LAN segments appear to be contiguous
- ★ Linux supports IEEE 802.1d STP for creating bridge hierarchies
- ★ **ebtables** commands change the definition of the DROP and ACCEPT commands in the broute chain
 - ▶ DROP routes while ACCEPT bridges

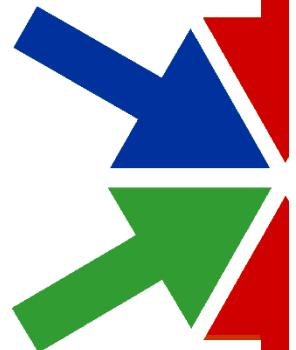
Questions

- * Routers are often referred to as gateways
 - ▶ True or False?
- * For a typical switch, each port is considered to be a unique collision domain
 - ▶ True or False?
- * Broadcast domains can never span more than one switch
 - ▶ True or False?
- * There are three ebtables tables known as filter, NAT and broute
 - ▶ True or False?
- * The **ebtables** command can be used to create a bridging router known as a brouter
 - ▶ True or False?

Chapter Break

Debugging Network Communications

NET
TCP
IP



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ★ Common Network Problems

- ★ Debugging Tools

- ▶ ifconfig
- ▶ ethtool
- ▶ arp
- ▶ ping
- ▶ telnet
- ▶ wget and curl
- ▶ netstat
- ▶ iptables
- ▶ traceroute, tracert and mtr
- ▶ tcpdump, tshark and wireshark
- ▶ nslookup
- ▶ nmap
- ▶ netcat and nc

Common Network Related Problems

★ Slow Connection

- ▶ Possible causes: bad cabling, NIC speed setting (10/100/1000), duplex mismatch, poor routing, remote server overloaded, misconfigured DNS

★ Loss of Communication

- ▶ The slow connection causes may also lead to complete loss of communication
- ▶ Remote server or application down, misconfigured firewall, other host misconfigurations

Viewing Interfaces with ip

★ The **ip addr show** command shows the active interfaces

- ▶ This includes multi-homed interfaces and MAC addresses

★ Example: interface (wlan0) is down

- ▶ UP and RUNNING flags not present
- ▶ Note the counters for overruns, loss of carrier, dropped packets and frame errors

```
# ip addr show wlan0
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state
DOWN mode DEFAULT group default qlen 1000
    link/ether 24:77:03:bd:33:80 brd ff:ff:ff:ff:ff:ff
```

Enabling Interfaces With ip

* Enable NIC and assign static IP

- ▶ Requires root privileges

```
# ip addr add 192.168.101.5/24 dev wlan0  
# ip link set dev wlan0 up
```

* Example wireless interface up

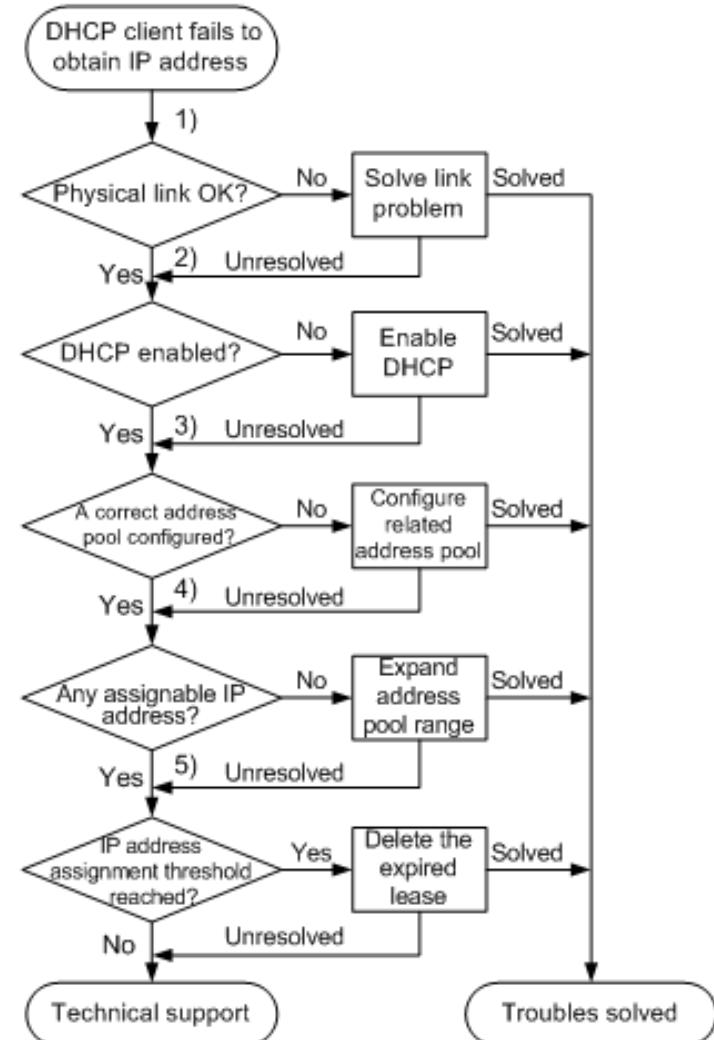
```
# ip addr show wlan0  
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP  
group default qlen 1000  
    link/ether 24:77:03:bd:33:80 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.101.5/24 brd 192.168.101.255 scope global wlan0  
        valid_lft forever preferred_lft forever  
    inet6 fd00::341f:28e:2e1c:eb97/64 scope global temporary dynamic  
        valid_lft 3597sec preferred_lft 597sec  
    inet6 fd00::2677:3ff:febd:3380/64 scope global dynamic  
        valid_lft 3597sec preferred_lft 597sec  
    inet6 fe80::2677:3ff:febd:3380/64 scope link  
        valid_lft forever preferred_lft forever
```

DHCP Troubleshooting

★ DHCP clients automatically assign their interfaces an IP address starting with 169.254.x.x

► This is an Automatic Private IP Address (APIPA) range

★ Until the client can make contact with the DHCP server the IP address will remain the same



Source: h3c.com

Duplex Mismatch

★ Duplex settings must match on both ends of the connection

- ▶ Half-duplex or full-duplex but not a mix
- ▶ Slow and bursty communications (i.e., ping) will not show symptoms of duplex mismatch
- ▶ Duplex is usually autonegotiated

★ Effect: Slow connection

★ Symptoms

- ▶ Source is full-duplex, destination is half-duplex
 - Data packets lost; ACK packets delayed or lost
- ▶ Source is half-duplex, destination is full-duplex
 - ACK packets lost; data packets delayed or lost

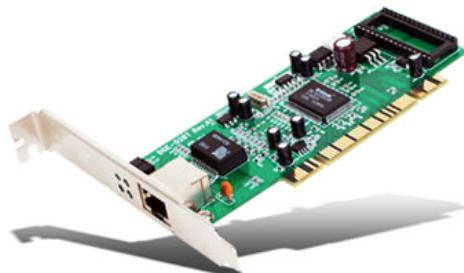
Testing Link Status With ethtool

* Command **ethtool** provides a means to interrogate the device PHY

```
$ sudo ethtool eth0
Settings for eth0:
  Supported ports: [ TP ]
  Supported link modes:  10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Advertised link modes:   10baseT/Half 10baseT/Full
                         100baseT/Half 100baseT/Full
                         1000baseT/Full
  Advertised pause frame use: No
  Advertised auto-negotiation: Yes
  Speed: 1000Mb/s
  Duplex: Full
  Port: Twisted Pair
  PHYAD: 2
  Transceiver: internal
  Auto-negotiation: on
  MDI-X: off (auto)
  Supports Wake-on: pumbg
  Wake-on: g
  Current message level: 0x00000007 (7)
                         drv probe link
  Link detected: yes
```

Dumping `net_stats` via ethtool

* Use the ‘-S’ option for stats



Source: imaginux.com

```
$ sudo ethtool -S eth0
NIC statistics:
  rx_packets: 38406476      tx_deferred_ok: 0
  tx_packets: 19526147      tx_single_coll_ok: 0
  rx_bytes: 49945533180     tx_multi_coll_ok: 0
  tx_bytes: 2629762845     tx_timeout_count: 0
  rx_broadcast: 323901      tx_restart_queue: 0
  tx_broadcast: 111798      rx_long_length_errors: 0
  rx_multicast: 499508     rx_short_length_errors: 0
  tx_multicast: 50583      rx_align_errors: 0
  rx_errors: 0              tx_tcp_seg_good: 159080
  tx_errors: 0              tx_tcp_seg_failed: 0
  tx_dropped: 0             rx_flow_control_xon: 0
  multicast: 499508         rx_flow_control_xoff: 0
  collisions: 0             tx_flow_control_xon: 0
  rx_length_errors: 0       tx_flow_control_xoff: 0
  rx_over_errors: 0         rx_csum_offload_good: 37775855
  rx_crc_errors: 0          rx_csum_offload_errors: 73
  rx_frame_errors: 0        rx_header_split: 0
  rx_no_buffer_count: 0     alloc_rx_buff_failed: 0
  rx_missed_errors: 0       tx_smbus: 0
  tx_aborted_errors: 0      rx_smbus: 390
  tx_carrier_errors: 0      dropped_smbus: 0
  tx_fifo_errors: 0          rx_dma_failed: 0
  tx_heartbeat_errors: 0     tx_dma_failed: 0
  tx_window_errors: 0        rx_hwstamp_cleared: 0
  tx_abort_late_coll: 0     uncorr_ecc_errors: 0
                           corr_ecc_errors: 0
```

Possible Ethernet Error Causes

- ★ Collisions: NIC detects itself and another host attempting transmissions at the same time
 - ▶ Due to heavy traffic, bad NIC or cable issues
- ★ CRC errors: Frames sent but were corrupted in transit
 - ▶ Due to electrical noise or cable issues
- ★ Frame errors: Incorrect CRC and a non-integer number of bytes are received
 - ▶ Due to collisions or bad NIC

Possible Ethernet Error Causes (2)

- ★ FIFO and overrun errors: Number of times the NIC was unable to transfer data to its buffers
 - ▶ Usually a sign of excessive traffic
- ★ Length errors: Received frame length does not match the Ethernet standard
 - ▶ Usually due to incompatible duplex settings
- ★ Carrier errors: NIC card loses its link to the hub or switch
 - ▶ Usually faulty cabling or bad interfaces on the NIC or other networking equipment

ARP Table

★ Another cause of connectivity problems can be the ARP table

- ▶ The ‘`arp -a`’ command displays the local ARP table
 - Cached MAC addresses for hosts on the network with which we have communicated
 - If a MAC address is not visible, check the remote firewall

```
$ sudo arp -a
HP8610 (192.168.101.125) at fc:8f:90:45:35:db [ether] on eth0
? (192.168.101.4) at b8:8d:12:20:14:12 [ether] on eth0
media-server (192.168.101.200) at 00:0a:e4:89:b9:3b [ether] on eth0
fios-gw (192.168.101.1) at 18:1b:eb:22:d4:94 [ether] on eth0
HP8620 (192.168.101.22) at 9c:b6:54:5e:dc:da [ether] on eth0
```

★ If having problems, verify the MAC addresses are correct

- ▶ ‘`arp -d`’ allows you to remove a stale table entry

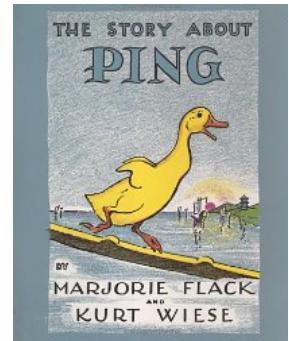
The ping Command

- ★ The most popular method to test connectivity across multiple networks
- ★ Reasons for lack of response:
 - ▶ Bad IP or the host does not exist
 - ▶ Remote server not configured to respond
 - ▶ Firewall or intermediate router blocking ICMP
 - ▶ Incorrect routing
 - A symptom of bad routes is the ability to ping servers only on your local network and nowhere else

Successful ping Results

- ★ Sends an ICMP echo packet every second to the specified IP, and waits for ICMP echo-reply until <ctrl-C>

```
$ sudo ping 192.168.101.22
PING 192.168.101.22 (192.168.101.22) 56(84) bytes of data.
64 bytes from 192.168.101.22: icmp_seq=1 ttl=255 time=58.3 ms
64 bytes from 192.168.101.22: icmp_seq=2 ttl=255 time=48.3 ms
64 bytes from 192.168.101.22: icmp_seq=3 ttl=255 time=49.8 ms
64 bytes from 192.168.101.22: icmp_seq=4 ttl=255 time=47.6 ms
^C
--- 192.168.101.22 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4005ms
rtt min/avg/max/mdev = 47.611/51.050/58.389/4.322 ms
```



Source: amazon.com

Bad ping Results

*'Destination Host Unreachable' message is caused by a router or a server knowing that the target IP address is part of a valid network; however, it is not responding

```
$ sudo ping 192.168.101.122
PING 192.168.101.122 (192.168.101.122) 56(84) bytes of data.
From 192.168.101.8 icmp_seq=1 Destination Host Unreachable
From 192.168.101.8 icmp_seq=2 Destination Host Unreachable
From 192.168.101.8 icmp_seq=3 Destination Host Unreachable
From 192.168.101.8 icmp_seq=4 Destination Host Unreachable
^C
--- 192.168.101.122 ping statistics ---
5 packets transmitted, 0 received, +4 errors, 100% packet loss,
time 4000ms
```

Type 3 ICMP Codes

Destination Unreachable Codes	
3	
Net Unreachable	The sending device knows about the network but believes it is not available at this time. Perhaps the network is too far away through the known route.
Host Unreachable	The sending device knows about host but doesn't get ARP reply, indicating the host is not available at this time
Protocol Unreachable	The protocol defined in IP header cannot be forwarded.
Port Unreachable	The sending device does not support the port number you are trying to reach
Fragmentation Needed and Don't Fragment was Set	The router needs to fragment the packet to forward it across a link that supports a smaller maximum transmission unit (MTU) size. However, application set the Don't Fragment bit.
Source Route Failed	ICMP sender can't use the strict or loose source routing path specified in the original packet.
Destination Network Unknown	ICMP sender does not have a route entry for the destination network, indicating this network may never have been available.
Destination Host Unknown	ICMP sender does not have a host entry, indicating the host may never have been available on connected network.

Source: linuxhomenetworking.com

Type 5 ICMP Codes

5	Redirect Codes	
	Redirect Datagram for the Network (or subnet)	ICMP sender (router) is not the best way to get to the desired network. Reply contains IP address of best router to destination. Dynamically adds a network entry in original sender's routing tables.
	Redirect Datagram for the Host	ICMP sender (router) is not the best way to get to the desired host. Reply contains IP address of best router to destination. Dynamically adds a host entry in original sender's route tables.
	Redirect Datagram for Type of the Service and Network	ICMP sender (router) does not offer a path to the destination network using the TOS requested. Dynamically adds a network entry in original sender's route tables.
	Redirect Datagram for the Type of Service and Host	ICMP sender (router) does not offer a path to the destination host using the TOS requested. Dynamically adds a host entry in original sender's route tables.

Source: linuxhomenetworking.com

Other ICMP Codes

6	Alternate Host Address Codes	
	Alternate Address for Host	Reply that indicates another host address should be used for the desired service. Should redirect application to another host.
11	Time Exceeded Codes	
	Time to Live exceeded in Transit	ICMP sender (router) indicates that originator's packet arrived with a Time To Live (TTL) of 1. Routers cannot decrement the TTL value to 0 and forward the packet.
	Fragment Reassembly Time Exceeded	ICMP sender (destination host) did not receive all fragment parts before the expiration (in seconds of holding time) of the TTL value of the first fragment received.
12	Parameter Problem Codes	
	Pointer indicates the error	Error is defined in greater detail within the ICMP packet.
	Missing a Required Option	ICMP sender expected some additional information in the Option field of the original packet.
	Bad Length	Original packet structure had an invalid length.

Source: linuxhomenetworking.com

Connectivity Testing With telnet

- ★ Allows specifying any TCP port number
 - ▶ Defaults to port 23
- ★ For example, Telnet can be used to verify that the **ssh** service is activated on a remote server by specifying its port number (22)
- ★ Reasons for refused connection
 - ▶ Service not running on remote server
 - ▶ Firewall configured to reject or drop requests

```
$ telnet 192.168.101.8 22
Trying 192.168.101.8...
Connected to 192.168.101.8.
Escape character is '^]'.
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.3
^]
telnet> quit
Connection closed.
```

Causes of telnet Failure

Reasons for refused connection

- ▶ Service not running on remote server
- ▶ Firewall configured to reject request

```
$ telnet 192.168.101.22 22
Trying 192.168.101.22...
telnet: Unable to connect to remote host: Connection refused
```

Reasons for timed out connection

- ▶ Remote server is powered down or does not exist
- ▶ Firewall configured to drop request

```
$ sudo telnet 216.10.100.12 22
Trying 216.10.100.12...
telnet: Unable to connect to remote host: Connection timed out
```

Testing Websites

- ✖ A browser does not display many useful HTTP error codes
- ✖ **telnet**, **wget** and **curl** are better diagnostic tools
 - ▶ Text-based
 - ▶ Fast telnet responses with slow **wget** or **curl** responses indicate web server configuration problems, not network problems
 - ▶ **wget** can request just the web page header which contains HTTP error codes (i.e., OK = 200)

```
$ curl -I www.theptrgroup.com
HTTP/1.1 200 OK
Date: Tue, 13 Oct 2015 15:24:06 GMT
Server: Apache
X-Powered-By: PHP/5.3.29
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Pragma: no-cache
Cache-control: private
Set-Cookie: PHPSESSID=805594772b0194e3e7fe00014da29b69; path=/
Connection: close
Content-Type: text/html; charset=utf-8
```

Testing Websites With wget

- * Option ‘-r’ recursively downloads pages
- * Option ‘-N’ includes download speed, file size and timestamps

```
$ sudo wget -N www.theptrgroup.com
--2015-10-13 11:25:12--  http://www.theptrgroup.com/
Resolving www.theptrgroup.com (www.theptrgroup.com) ... 205.134.170.50
Connecting to www.theptrgroup.com
(www.theptrgroup.com)|205.134.170.50|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

[ <=> ] 29,961 --.-K/s in 0.01s

Last-modified header missing -- time-stamps turned off.
2015-10-13 11:25:12 (1.95 MB/s) - 'index.html' saved [29961]
```

The ss Command

- ★ **ss** is the socket status command
 - ▶ Replacement for the **netstat** command
- ★ Shows the status of open sockets:
 - ▶ Open TCP ports and their states
 - Common states: **LISTEN**, **ESTABLISHED**, **TIME_WAIT**
 - ▶ Open UDP ports
 - ▶ UNIX domain sockets
- ★ Example:

```
# > ss -t -a
      State      Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
      LISTEN      0        128          127.0.0.1:17600      *:*
      LISTEN      0        128          *:39649            *:*
      LISTEN      0        64           *:nfs              *:*
      LISTEN      0        1           127.0.0.1:openvpn    *:*
      LISTEN      0        50           127.0.0.1:mysql     *:*
...
      ESTAB       0        0           192.168.101.8:46478  192.155.48.48:http
      CLOSE-WAIT   38       0           192.168.101.8:34333  54.192.54.115:https
      ESTAB       0        0           192.168.101.8:58373  199.16.156.120:https
      ESTAB       0        0           127.0.0.1:53866    127.0.0.1:openvpn
```

Firewalls and iptables

- ★ Firewalls can be a source of connectivity problems
 - ▶ Installed by default on RedHat and Fedora Linux
- ★ Use `# service iptables status` to determine if the firewall is active
 - ▶ Works for RH-derived distros
 - ▶ For Debian-based distros, add pre-up and post-down entries to `/etc/network/interfaces`
- ★ Use '`iptables -v --list`' to list the active iptables rules

The traceroute Command

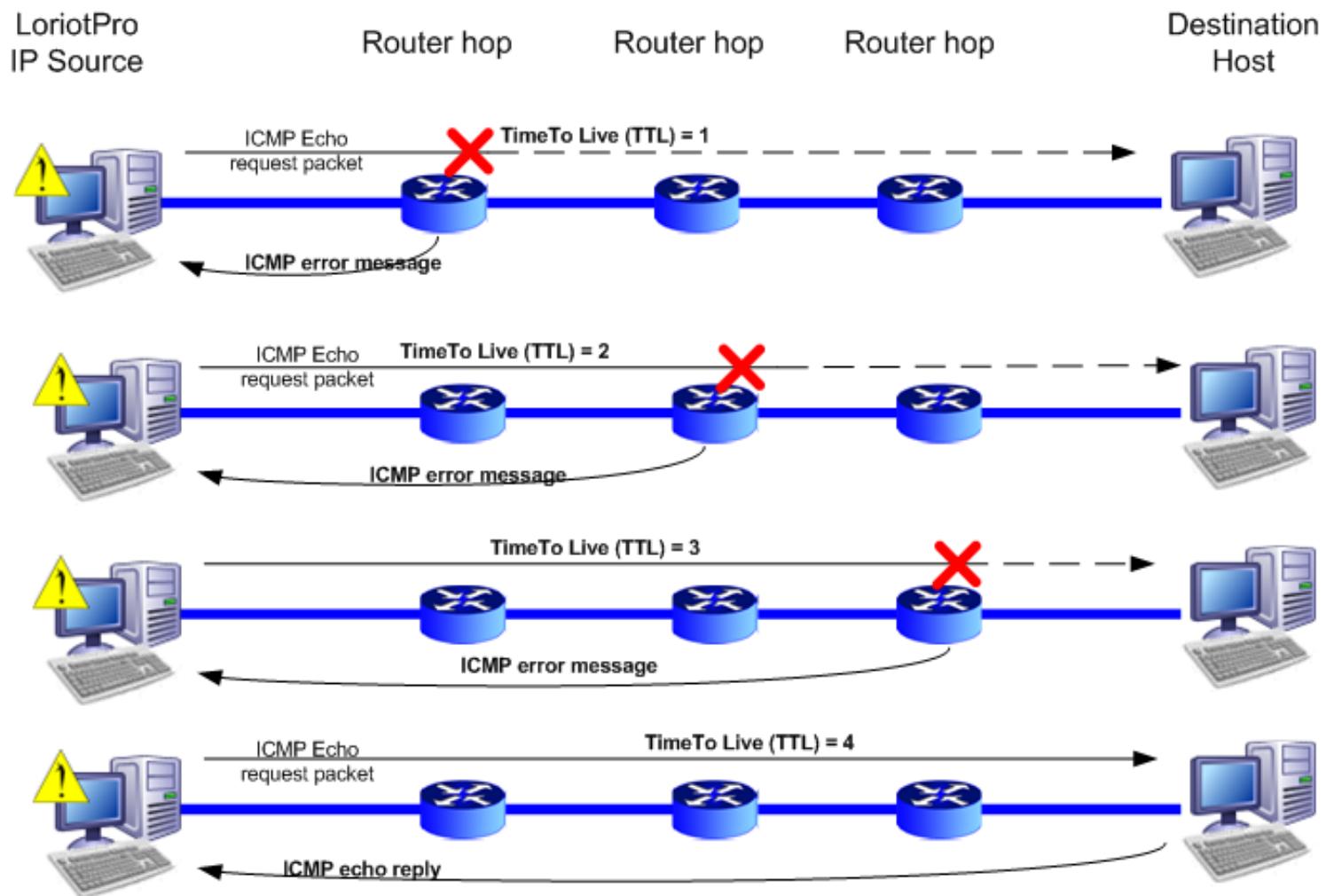
- ★ Displays the list of router hops between the local host and the specified remote host
 - ▶ Windows version is **tracert**
- ★ Increments IP packet's TTL field from 0 in order to query the next router
- ★ The following example shows that all the hop times are under 50 ms

```
$ traceroute mail.vpn.theptrgroup.com
traceroute to mail.vpn.theptrgroup.com (10.11.12.9), 64 hops max
 1  10.11.21.1  13.542ms  12.720ms  15.941ms
 2  10.11.12.9  10.821ms  14.955ms  12.416ms
```

- ★ Sometimes it's useful to have a traceroute from the reverse direction for comparison
 - ▶ Helps find routing problems

Source: linuxhomenetworking.com

traceroute Steps



Source: loriotpro.com

LUTEUS Copyrights 2008

traceroute Return Codes



- ▶ Expected 5 second response time exceeded.
Could be caused by:

- A router on the path not sending back the ICMP "time exceeded" messages
- A router or firewall in the path blocking the ICMP "time exceeded" messages
- The target IP address not responding



- ▶ Host, network or protocol unreachable



- ▶ Communication administratively prohibited. A router Access Control List (ACL) or firewall is in the way



- ▶ Source route failed. Source routing attempts to force traceroute to use a certain path. Failure might be due to a router security setting

False Alarm: Timeouts

- ★ If there is no response within 5 seconds, then an asterisk (*) is printed

```
$ traceroute www.theptrgroup.com
traceroute to www.theptrgroup.com (205.134.170.50), 64 hops max
 1  192.168.101.1      0.478ms   0.379ms   0.443ms
 2  71.127.46.1       8.295ms    *          *
 3  130.81.223.130    6.097ms    *          12.409ms
 4  *  *  *
 5  140.222.227.195   12.455ms   41.989ms   10.312ms
 6  154.54.10.197     9.480ms   10.384ms   9.528ms
 7  154.54.31.105     10.298ms   9.456ms   10.437ms
 8  154.54.31.41      164.857ms  212.325ms  205.032ms
 9  154.24.12.2       9.519ms   10.099ms   9.643ms
10  38.104.29.34      92.638ms  50.177ms   246.953ms
11  205.134.160.234   14.872ms   12.487ms   12.284ms
12  205.134.176.50    9.602ms   10.216ms   9.650ms
13  *  *  *
14  205.134.191.90    210.473ms  219.909ms   12.658ms
15  *  *  *
16  205.134.191.177   10.859ms   10.180ms   9.770ms
17  *  *  *
18  *  *  *
```

False Alarm: Timeouts (2)

- To get around some firewalls, ICMP messages can be sent instead of UDP messages, by using the '-I' option

```
$ traceroute -I www.theptrgroup.com
traceroute to www.theptrgroup.com (205.134.170.50), 64 hops max
 1  192.168.101.1      0.465ms   0.396ms   0.366ms
 2  71.127.46.1        8.346ms   6.319ms   7.335ms
 3  100.41.4.30        12.474ms  10.180ms  9.638ms
 4  * * *
 5  140.222.227.197   9.812ms   9.380ms   7.526ms
 6  154.54.10.197     8.060ms   9.680ms   10.229ms
 7  154.54.31.105     9.611ms   10.211ms  9.644ms
 8  154.54.31.41      10.156ms  10.215ms  9.427ms
 9  154.24.12.2       10.425ms  9.472ms   10.356ms
10  38.104.237.194    9.516ms   10.362ms  9.518ms
11  205.134.176.54    10.353ms  9.467ms   10.399ms
12  205.134.176.50    9.508ms   10.368ms  9.527ms
13  205.134.191.181   10.393ms  9.522ms   10.388ms
14  205.134.191.90    9.527ms   9.655ms   10.256ms
15  205.134.185.46    10.440ms  9.442ms   9.724ms
16  205.134.191.177   10.228ms  9.663ms   10.196ms
17  205.134.160.102   11.853ms  12.325ms  12.533ms
18  205.134.170.50    13.074ms  14.225ms  12.590ms
```

False Alarm: Slow Internet

- Example: Hops 6 and 7 were slow to respond
- Note that hops 8 through 11 were fine
 - ▶ Thus, this is not an Internet congestion issue
 - ▶ Some Internet routers put **traceroute** requests at a low priority via QoS

```
$ traceroute 80.40.118.227
```

1	66.134.200.97	1.123 ms	2.834 ms	1.699 ms
2	172.31.255.253	43.812 ms	15.491 ms	44.417 ms
3	192.168.21.65	15.176 ms	16.429 ms	8.557 ms
4	64.200.150.193	26.132 ms	13.012 ms	16.733 ms
5	64.200.151.229	38.912 ms	12.470 ms	14.495 ms
6	64.200.149.14	239.723 ms	255.182 ms	253.510 ms
7	64.200.150.110	254.164 ms	252.263 ms	252.221 ms
8	192.174.250.34	24.073 ms	20.277 ms	20.274 ms
9	192.174.47.6	91.224 ms	89.286 ms	60.895 ms
10	80.40.96.12	17.163 ms	20.331 ms	20.401 ms
11	80.40.118.227	30.334 ms	16.653 ms	23.563 ms

TTL Timeout

- ★ Ping TTL timeouts are usually caused by a routing loop
 - ▶ A packet will bounce between 2 routers until TTL = 0

```
G:\>ping 186.9.17.153

Pinging 186.9.17.153 with 32 bytes of data:

Reply from 186.40.64.94: TTL expired in transit.

Ping statistics for 186.9.17.153:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
  Minimum = 0ms, Maximum = 0ms, Average = 0ms

G:\>tracert 186.9.17.153

Tracing route to lostserver.confusion.net [186.9.17.153]
over a maximum of 30 hops:
  1  <10 ms    <10 ms    <10 ms  186.217.33.1
  2  60 ms     70 ms     60 ms  rtr-2.confusion.net [186.40.64.94]
  3  70 ms     71 ms     70 ms  rtr-1.confusion.net [186.40.64.93]
  4  60 ms     70 ms     60 ms  rtr-2.confusion.net [186.40.64.94]
  5  70 ms     70 ms     70 ms  rtr-1.confusion.net [186.40.64.93]
  6  60 ms     70 ms     61 ms  rtr-2.confusion.net [186.40.64.94]
  7  70 ms     70 ms     70 ms  rtr-1.confusion.net [186.40.64.93]
  8  60 ms     70 ms     60 ms  rtr-2.confusion.net [186.40.64.94]
  9  70 ms     70 ms     70 ms  rtr-1.confusion.net [186.40.64.93]
...
...
Trace complete.
G:\>
```

Source: linuxhomenetworking.com

My traceroute (mtr)

- ★ Performs repeated traceroute calls in real time
- ★ Good for finding intermittent problems or bursts of congestion

```
My traceroute [v0.85]
valiant (:)
12:04:39 2015                                              Tue Oct 13
Keys: Help   Display mode   Restart statistics   Order of fields   quit
                                                Packets          Pings
                                                Loss%    Snt    Last     Avg   Best   Wrst  StDev
Host
1. PTRGroupHerndon-3.tunnel.tserv13.ash1.ipv6.he.net        0.0%     2   14.9   15.2  14.9  15.5  0.0
2. ge5-4.core1.ash1.he.net                                  0.0%     2   16.9   16.5  16.1  16.9  0.0
3. 100ge5-1.core1.nyc4.he.net                            0.0%     2   16.9   20.6  16.9  24.2  5.1
4. as7018-att.10gigabitethernet2-3.core1.nyc4.he.net      0.0%     2   19.3   19.1  18.9  19.3  0.0
5. n54ny22crs.ipv6.att.net                             0.0%     2   54.7   54.6  54.6  54.7  0.0
6. wswdc22crs.ipv6.att.net                           0.0%     2   56.9   56.7  56.6  56.9  0.0
7. attga21crs.ipv6.att.net                         0.0%     2   61.8   59.5  57.3  61.8  3.0
8. dlstx22crs.ipv6.att.net                      0.0%     2   58.9   57.0  55.1  58.9  2.6
9. dlstx405me3.ipv6.att.net                    0.0%     2   74.4   66.2  58.1  74.4  11.5
10. 2001:1890:c00:8701::11b7:3f7f                0.0%     2   53.5   53.5  53.5  53.5  0.0
11. rcdn9-cd2-dmzbb-gw2-ten1-1.cisco.com        0.0%     2   53.7   54.8  53.7  56.0  1.4
12. rcdn9-cd2-dmzdcc-gw2-por2.cisco.com       0.0%     2   51.5   52.6  51.5  53.7  1.4
13. rcdn9-16b-dcz05n-gw2-por2.cisco.com     0.0%     2   54.0   53.0  51.9  54.0  1.4
14. www1.cisco.com                                0.0%     2   52.0   53.5  52.0  54.9  2.0
```

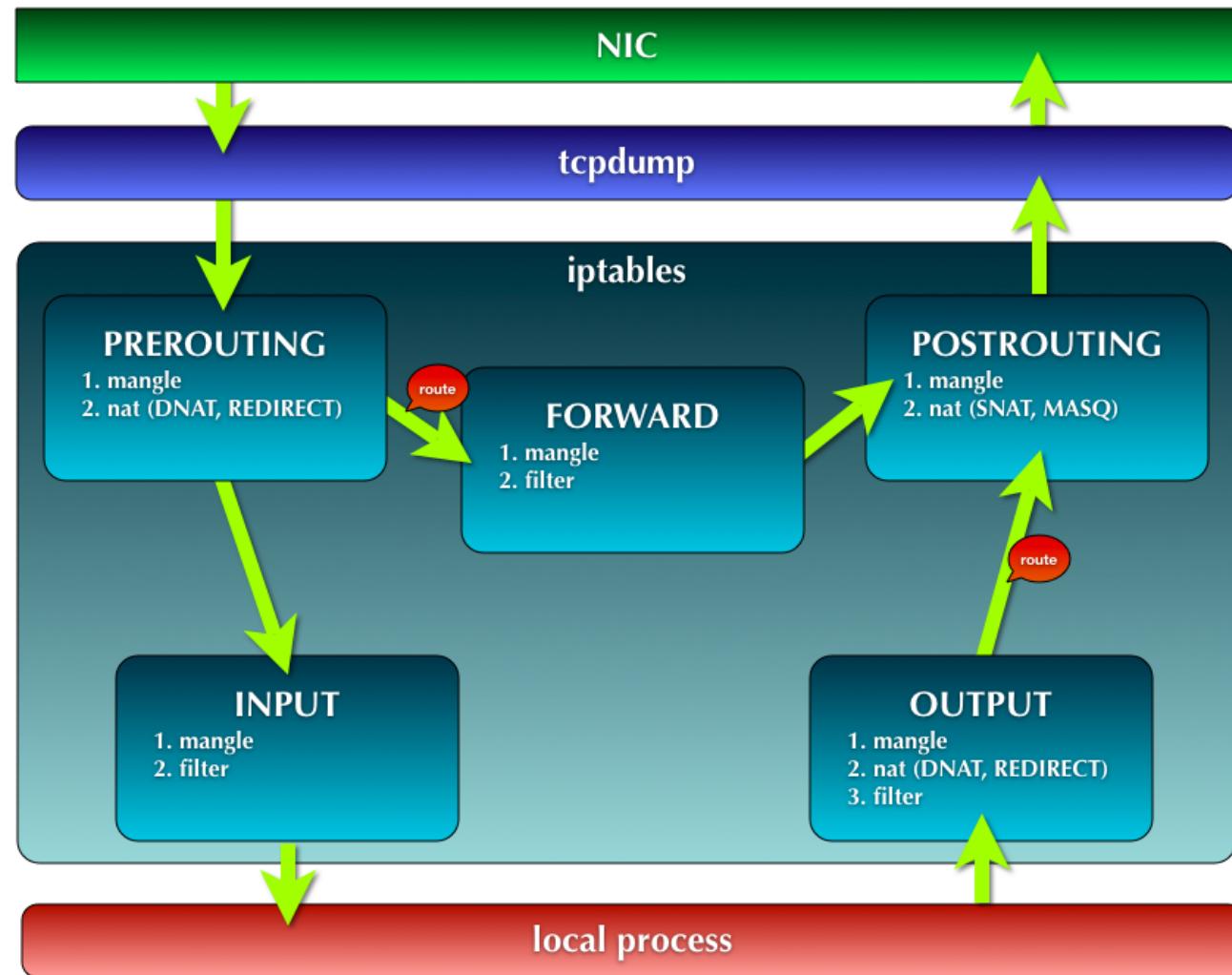
Packet Capture and Analysis Tools

- ★ These tools place interfaces in promiscuous mode to allow viewing of all network traffic
- ★ Analyze network behavior and performance
- ★ Determine whether there are routing problems
- ★ Allow the user to isolate the source of problems
- ★ Intercept and display communications from other users or computers on the network
 - ▶ Can see other users' login ID, password, URLs visited, viewed website content or any other information over unencrypted channels (e.g., telnet, HTTP)

tcpdump

- ❖ Command line tool that captures and displays network packets
- ❖ Can read from and write to 'pcap' files
- ❖ Can filter packets based on multiple and specific criteria
 - ▶ By IPs, MACs, port number, protocol, message type, interface, TTL, etc.

Where does tcpdump Capture Packets?



Source: d.hatena.ne.jp

Example tcpdump Command

- ★ View only ICMP packets traversing through interface eth0

- ▶ Column 1: timestamp
- ▶ Column 2: source and destination
- ▶ Column 3: message type

```
$ sudo tcpdump -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
12:08:19.303595 IP valiant > fios-gw: ICMP valiant udp port 1873 unreachable, length 103
12:08:26.460507 IP valiant > fios-gw: ICMP echo request, id 21014, seq 1, length 64
12:08:26.460989 IP fios-gw > valiant: ICMP echo reply, id 21014, seq 1, length 64
12:08:27.460396 IP valiant > fios-gw: ICMP echo request, id 21014, seq 2, length 64
12:08:27.460861 IP fios-gw > valiant: ICMP echo reply, id 21014, seq 2, length 64
12:08:28.460379 IP valiant > fios-gw: ICMP echo request, id 21014, seq 3, length 64
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
```

More tcpdump Examples

- * Print all packets arriving from or departing to host 'sundown':

```
$ tcpdump host sundown
```

- * Print traffic between hosts 'helios' and either 'hot' or 'ace':

```
$ tcpdump host helios and \(\ hot or ace \)
```

- * Print all IP packets between host 'ace' and any host except 'helios':

```
$ tcpdump ip host ace and not helios
```

- * Print all ftp traffic through Internet gateway 'snup':

```
$ tcpdump 'gateway snup and (port ftp or ftp-data) '
```

- * Print IP traffic neither sourced from nor destined to local hosts:

```
$ tcpdump ip and not net localnet
```

- * Print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host:

```
$ tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0  
and not src and dst net localnet'
```

tshark and wireshark

* tshark previously known as **tethereal**

* Very similar to **tcpdump**

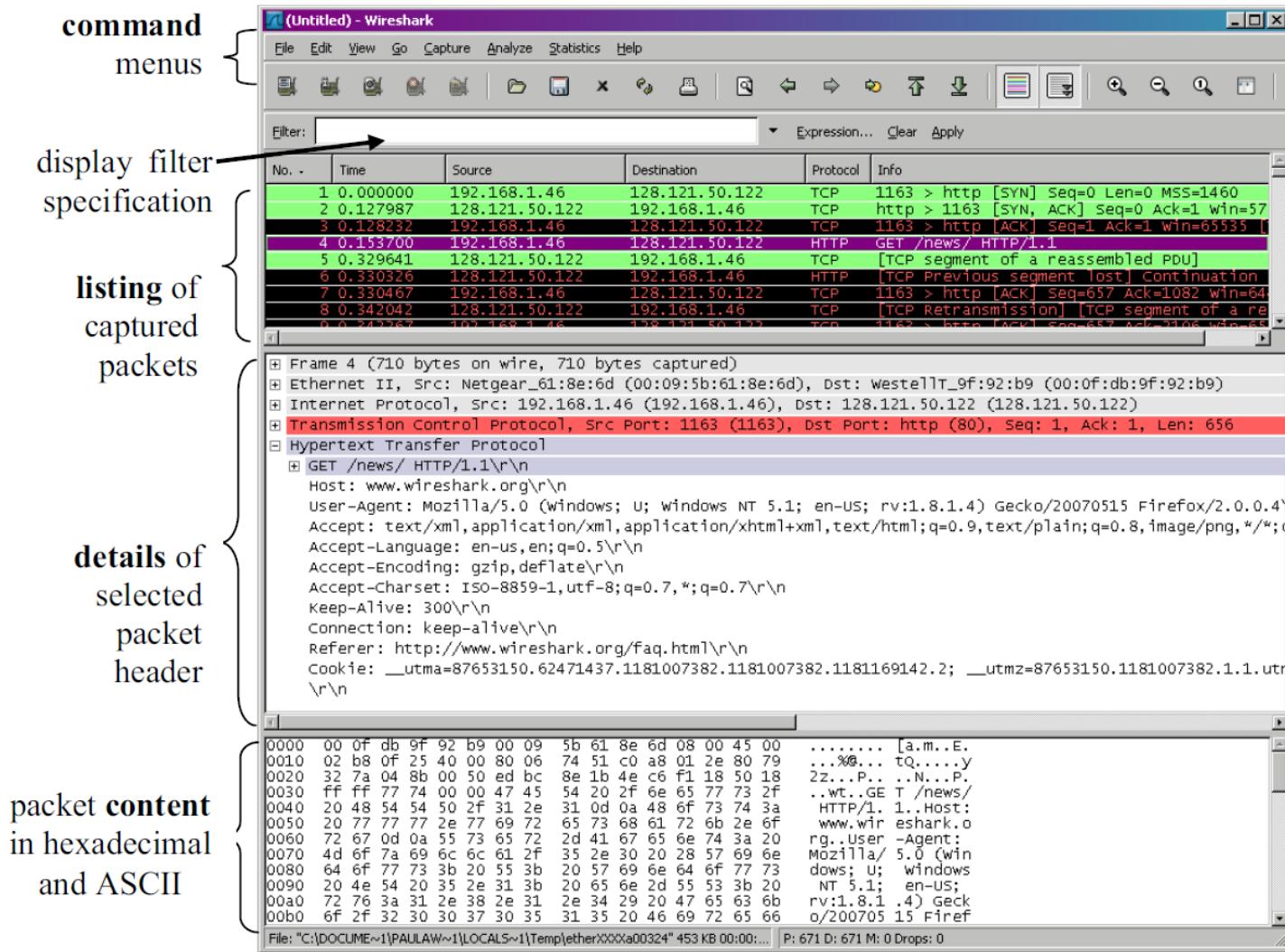
- ▶ Similar command syntax and output display
- ▶ Files saved in ‘pcap’ file format as well as others
 - Uses **libpcap** to access the NIC

* Advantages over **tcpdump**

- ▶ Can create a new file with a new filename extension when a size limit is reached
- ▶ Can limit the total number of files created before overwriting the first file (a ring of files)

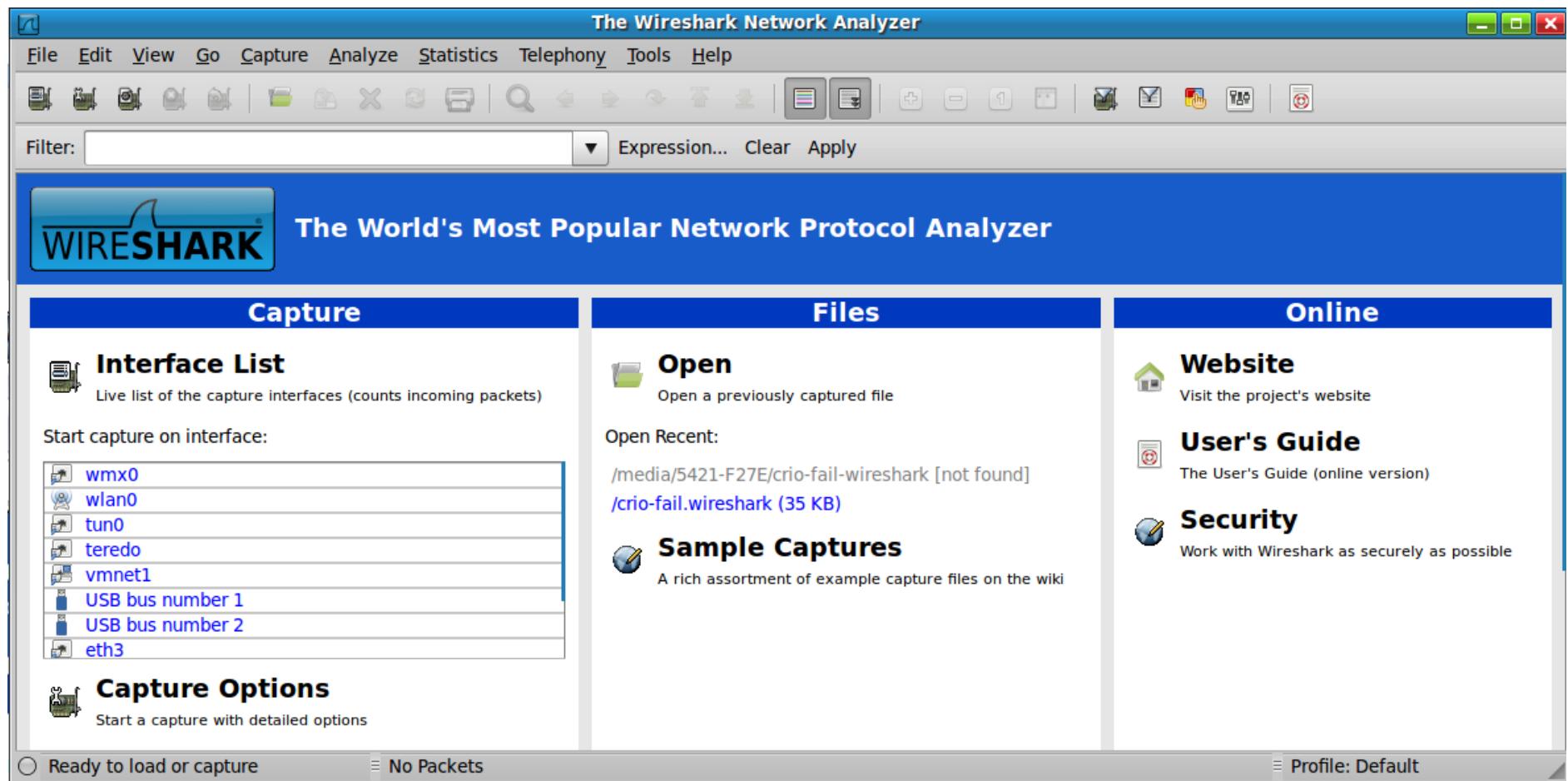
* **wireshark**: tshark with a GUI front end

wireshark: GUI Layout



Source: www.engr.siu.edu

wireshark: Select HW Interface



wireshark: Bad Packets (in Black)

No.	Time	Source	Destination	Protocol	Info
26	10.665135	192.168.101.28	74.125.227.112	TCP	36123 > http [ACK] Seq=1 Ack=2 Win=1092 Len=0 TSYN
27	11.338360	71.178.251.7	192.168.101.28	UDP	Source port: https Destination port: 47318
28	12.019857	192.168.101.28	173.194.64.105	TLSv1	Application Data
29	12.021251	192.168.101.28	173.194.64.105	TCP	[TCP segment of a reassembled PDU]
30	12.021273	192.168.101.28	173.194.64.105	TLSv1	Application Data
31	12.063525	173.194.64.105	192.168.101.28	TLSv1	Application Data
32	12.063568	192.168.101.28	173.194.64.105	TCP	59511 > https [ACK] Seq=1822 Ack=38 Win=1002 Len=0
33	12.068134	173.194.64.105	192.168.101.28	TCP	[TCP Dup ACK 31#1] https > 59511 [ACK] Seq=38 Ack=38 Win=980 Len=0
34	12.068380	173.194.64.105	192.168.101.28	TCP	https > 59511 [ACK] Seq=38 Ack=1822 Win=980 Len=0
35	12.100829	173.194.64.105	192.168.101.28	TLSv1	Application Data
36	12.100864	192.168.101.28	173.194.64.105	TCP	59511 > https [ACK] Seq=1822 Ack=89 Win=1002 Len=0
37	12.100994	173.194.64.105	192.168.101.28	TLSv1	Application Data
38	12.101018	192.168.101.28	173.194.64.105	TCP	59511 > https [ACK] Seq=1822 Ack=225 Win=1002 Len=0
39	13.020404	192.168.101.28	173.194.64.105	TLSv1	Application Data
40	13.064212	173.194.64.105	192.168.101.28	TLSv1	Application Data
41	13.064253	192.168.101.28	173.194.64.105	TCP	59511 > https [ACK] Seq=1859 Ack=262 Win=1002 Len=0
42	14.591858	192.168.101.24	239.255.255.250	SSDP	M-SEARCH * HTTP/1.1
43	14.593103	Wistron 89:b9:3b	Broadcast	ARP	Who has 192.168.101.24? Tell 192.168.101.200

► Frame 1: 111 bytes on wire (888 bits), 111 bytes captured (888 bits)
► Ethernet II, Src: Apple_9e:df:32 (00:24:36:9e:df:32), Dst: HsingTec_08:04:9d (00:d0:09:08:04:9d)
► Internet Protocol, Src: 71.178.251.7 (71.178.251.7), Dst: 192.168.101.28 (192.168.101.28)
► User Datagram Protocol, Src Port: https (443), Dst Port: 47318 (47318)
► Data (69 bytes)

Hex	Dec	Text
0000	00 d0 09 08 04 9d 00 24	36 9e df 32 08 00 45 00 ..\$ 6..2..E.
0010	00 61 3a 31 00 00 35 11	e2 dc 47 b2 fb 07 c0 a8 .a:1..5. ...G.....
0020	65 1c 01 bb b8 d6 00 4d	63 94 34 55 3b 6e 98 05 e.....M c.4U;n..
0030	30 b3 9d 84 4a ee f1 4a	a9 d9 98 76 00 05 b9 6f 0...J..J ...v....0
0040	01 61 15 00 00 00 00 00	00 00 00 00 00 00 00 00

eth3: <live capture in progress> File: Packets: 43 Displayed: 43 Marked: 0 Profile: Default

wireshark: Apply Filter

Filter: dns

Expression... Clear Apply

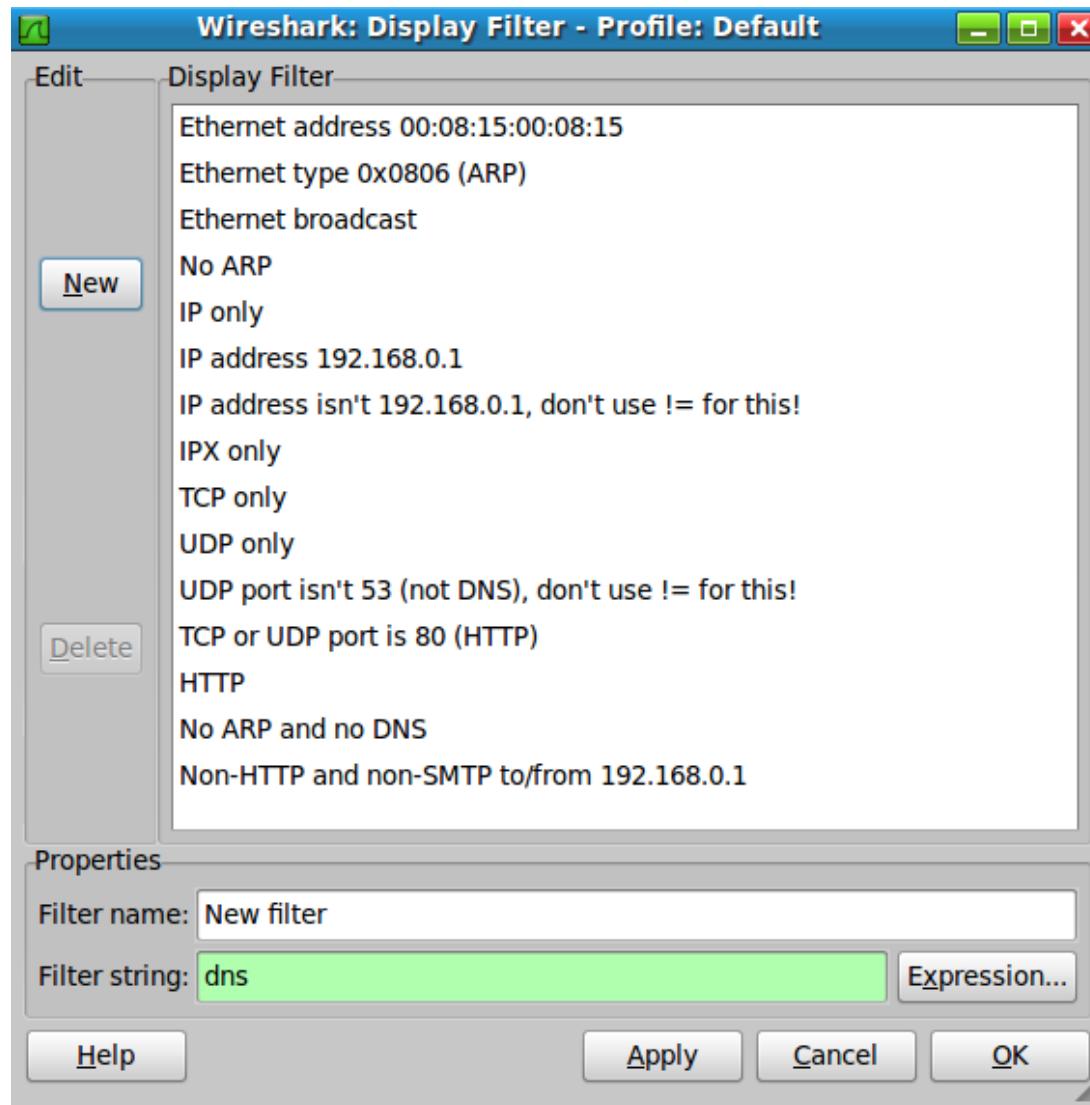
No.	Time	Source	Destination	Protocol	Info
481	161.014611	192.168.101.28	192.168.101.1	DNS	Standard query AAAA 0-149.channel.facebook.com
482	161.014707	192.168.101.28	192.168.101.1	DNS	Standard query AAAA s-static.ak.fcdn.net
485	161.063409	192.168.101.1	192.168.101.28	DNS	Standard query response
486	161.063552	192.168.101.28	192.168.101.1	DNS	Standard query AAAA 0-149.channel.facebook.com.dc.dc.cox.net
487	161.072405	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME s-static.ak.fcdn.net.edgekey.net CNAME e3353.ds
488	161.072555	192.168.101.28	192.168.101.1	DNS	Standard query A s-static.ak.fcdn.net
489	161.094278	192.168.101.1	192.168.101.28	DNS	Standard query response, No such name
490	161.094413	192.168.101.28	192.168.101.1	DNS	Standard query A 0-149.channel.facebook.com
491	161.095365	192.168.101.1	192.168.101.28	DNS	Standard query response A 66.220.151.80
495	161.127417	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME s-static.ak.fcdn.net.edgekey.net CNAME e3353.ds
505	161.348083	192.168.101.28	192.168.101.1	DNS	Standard query AAAA static.ak.facebook.com
506	161.366009	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME static.ak.facebook.com.edgesuite.net CNAME a749.
507	161.366230	192.168.101.28	192.168.101.1	DNS	Standard query A static.ak.facebook.com
509	161.425076	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME static.ak.facebook.com.edgesuite.net CNAME a749.
523	170.748043	192.168.101.28	192.168.101.1	DNS	Standard query AAAA ajax.googleapis.com
524	170.798402	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME googleapis.l.google.com AAAA 2607:f8b0:400e:c01:
525	170.798549	192.168.101.28	192.168.101.1	DNS	Standard query A ajax.googleapis.com
526	170.811019	192.168.101.1	192.168.101.28	DNS	Standard query response CNAME googleapis.l.google.com A 173.194.64.95

► Frame 370: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
► Ethernet II, Src: HsingTec_08:04:9d (00:d0:09:08:04:9d), Dst: Apple_9e:df:32 (00:24:36:9e:df:32)
► Internet Protocol, Src: 192.168.101.28 (192.168.101.28), Dst: 192.168.101.1 (192.168.101.1)
► User Datagram Protocol, Src Port: 34359 (34359), Dst Port: domain (53)
► Domain Name System (query)

0000 00 24 36 9e df 32 00 d0 09 08 04 9d 08 00 45 00 .\$.6..2..E.
0010 00 54 da 75 40 00 40 11 14 b5 c0 a8 65 1c c0 a8 .T.u@.a.e...
0020 65 01 86 37 00 35 00 40 df 79 a9 59 01 00 00 01 e..7.5.@ .y.Y....
0030 00 00 00 00 00 04 6d 61 69 6c 03 76 70 6e 0bm ail.vpn.
0040 71 00 05 70 71 70 07 70 06 70 02 02 01 00

eth3: <live capture in progress> Fi... Packets: 579 Displayed: 84 Marked: 0 Profile: Default

wireshark: Manage Filters



wireshark: Follow TCP Stream

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
701	164.623068	192.168.101.28	68.99.123.175	TCP	38404 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460
702	164.649388	68.99.123.175	192.168.101.28		38404 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0
703	164.649433	192.168.101.28	68.99.123.175		> http [ACK] Seq=1 Ack=1 Win=14600 Len=0
704	164.649569	192.168.101.28	68.99.123.175		Location/residential?s=filter&dest=http%3A%
705	164.680262	68.99.123.175	192.168.101.28		38404 [ACK] Seq=1 Ack=926 Win=10185 Len=0
706	164.685552	68.99.123.175	192.168.101.28		.1 302 Moved Temporarily
707	164.685586	192.168.101.28	68.99.123.175		> http [ACK] Seq=926 Ack=525 Win=15544 Len=0
708	164.685746	192.168.101.28	68.99.123.175		> http [FIN, ACK] Seq=236 Ack=1776 Win=1832
709	164.686103	192.168.101.28	68.99.123.175		ispatch/5622055160688078603/intercept.co
710	164.715473	68.99.123.161	192.168.101.28		57324 [FIN, ACK] Seq=1776 Ack=237 Win=8190
711	164.715508	192.168.101.28	68.99.123.175		> http [ACK] Seq=237 Ack=1777 Win=18340 Len=0
712	164.726534	68.99.123.175	192.168.101.28	SCTP	Segment of a reassembled PDU]
713	164.726662	68.99.123.175	192.168.101.28		Segment of a reassembled PDU]
714	164.726675	192.168.101.28	68.99.123.175		> http [ACK] Seq=1889 Ack=3445 Win=20440 Len=0
715	164.749043	192.168.101.200	239.168.101.200		* HTTP/1.1
716	164.754037	68.99.123.175	192.168.101.200		Segment of a reassembled PDU]
717	164.754286	68.99.123.175	192.168.101.200		Segment of a reassembled PDU]
718	164.754298	192.168.101.28	68.99.123.175		> http [ACK] Seq=1889 Ack=6365 Win=26280 Len=0

► Frame 701: 74 bytes on wire (592 bits), 74 bytes captured

► Ethernet II, Src: HsingTec_08:04:9d (00:d0:09:08:04:9d), Dst: 68:99:12:3:17:5 (00:0c:29:31:17:05)

► Internet Protocol Version 4, Src: 192.168.101.28 (192.168.101.28), Dst: 68.99.123.175 (68.99.123.175)

► Transmission Control Protocol, Src Port: 38404 (38404), Dst Port: http (80), Seq: 0, Len: 0

Follow TCP Stream

Follow UDP Stream

Follow SSL Stream

Copy

Decode As...

Print...

Show Packet in New Window

36:9e:df:32) (23.175)

0000 00 24 36 9e df 32 00 d0 09 08 04 9d 08 00 45 00 .\$.2..E.
0010 00 3c 1e bd 40 00 40 06 36 28 c0 a8 65 1c 44 63 .<..@. 6(..e.Dc
0020 7b af 96 04 00 50 a0 5b 9a 0a 00 00 00 00 a0 02 {....P.[.....
0030 39 08 e6 05 00 00 02 04 05 b4 04 02 08 0a 04 04 9.....
0040 24 17 00 00 00 00 01 02 02 02 ..

eth3: <live capture in progress> Fi... Packets: 2165 Displayed: 2165 Marked: 0 Profile: Default

wireshark: Complete Stream

wireshark: Complete Stream

Follow TCP Stream

```
GET /location/residential?s=filter&dest=http%3A%2F%2Fwww.cox.com%2Fmyconnection%2Fhome.cox HTTP/1.0
User-Agent: Wget/1.12 (linux-gnu)
Accept: */
Host: intercept.cox.com
Connection: Keep-Alive
Cookie: SMIDENTITY=XqEIa6+ZMd+liMnbIfTs5fLiq8VkJSS6nwpw/UguCQ+CsRKgoeFWT6Fe9Ta4H003ryMSzeXpvs0YDyrD/K/T
+8oeAC1a29cPJ7z3DWrW+eQiRa1b9RXg3S2TwFqbDRTM1CLA6yBPeG31b
+fMtZl8a9UwAXET2dTB74X02Kto4mE9CbmpN0qjbek71hhuIbllVY2TcNJcmc94SDBVdps5uvI0U5Ms35lHIjdqm431l9/nvoTm5/
lUqHJi3GuACA7ru6ql0F8IZARbMG0RHXcyQfLJrHv3x8ee8udDnwoVH1zllGo7r/rZ1jyjR1
+YoZfa06sPXNGxucPfLzWydq9DspJh0vEM0VG7mnc5g9U4qFRJ0LrhzTp1Mv4SB35tN10KS0bzg57ljIzoI43077Lng/fmgqp/
S4IPQGJ5nz9KiP5ziaJjEoRZ9GbZ7cIpMSwQu6u1Y7sg4LFUaQ
+TQ570G0oa8/7Rdti8pawwXngJ7hNfftGmy2LNm0vhsJDYHUNJGNDxxG7AcrV4kwudAtqDtnjluJ1ihbk3AnVZzoI5i
+i/0DLRNLb4au95FT93DBkqYrb2vHBmnzABy+c1TxMdURNfhnfPL
+YkB5Lnuz0DfJS8FEMFoK08vms4C1rC7cp9Ul8aM0zBkWMygfaMyWynJab5iktaX3v4iSCg7AcopqyA8aqyflVRbUqtjPSNBY

HTTP/1.1 302 Moved Temporarily
Date: Thu, 06 Sep 2012 22:40:18 GMT
Server: Apache
Set-Cookie: JSESSIONID=0E1BE48AAC119FE175D48109556A7C6.cont_05; Path=/dispatch
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: http://intercept.cox.com/dispatch/5622055160688078603/intercept.cox?lob=residential&s=filter&dest=http%3A
%2F%2Fwww.cox.com%2Fmyconnection%2Fhome.cox
Content-Length: 0
```

Find Save As Print Entire conversation (62416 bytes) ▾ ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

wireshark: TCP Stream Data



wireshark: Inspect Packets

No.	Time	Source	Destination	Protocol	Info
701	164.623068	192.168.101.28	68.99.123.175	TCP	38404 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460
702	164.649388	68.99.123.175	192.168.101.28	TCP	http > 38404 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0
703	164.649433	192.168.101.28	68.99.123.175	TCP	38404 > http [ACK] Seq=1 Ack=1 Win=14600 Len=0
704	164.649569	192.168.101.28	68.99.123.175	HTTP	GET /location/residential?s=filter&dest=http%3A%2F%2Fwww.google.com%2F HTTP/1.1
705	164.680262	68.99.123.175	192.168.101.28	TCP	http > 38404 [ACK] Seq=1 Ack=926 Win=10185 Len=0
706	164.685552	68.99.123.175	192.168.101.28	HTTP	HTTP/1.1 302 Moved Temporarily
707	164.685586	192.168.101.28	68.99.123.175	TCP	38404 > http [ACK] Seq=926 Ack=525 Win=15544 Len=0
708	164.685746	192.168.101.28	68.99.123.161	TCP	57324 > http [FIN, ACK] Seq=236 Ack=1776 Win=18340 Len=0
709	164.686103	192.168.101.28	68.99.123.175	HTTP	GET /dispatch/5622055160688078603/intercept.coxi
710	164.715473	68.99.123.161	192.168.101.28	TCP	http > 57324 [FIN, ACK] Seq=1776 Ack=237 Win=8190 Len=0
711	164.715508	192.168.101.28	68.99.123.161	TCP	57324 > http [ACK] Seq=237 Ack=1777 Win=18340 Len=0
712	164.726534	68.99.123.175	192.168.101.28	TCP	[TCP segment of a reassembled PDU]
713	164.726662	68.99.123.175	192.168.101.28	TCP	[TCP segment of a reassembled PDU]
714	164.726675	192.168.101.28	68.99.123.175	TCP	38404 > http [ACK] Seq=1889 Ack=3445 Win=20440 Len=0
715	164.749043	192.168.101.200	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
716	164.754037	68.99.123.175	192.168.101.28	TCP	[TCP segment of a reassembled PDU]
717	164.754286	68.99.123.175	192.168.101.28	TCP	[TCP segment of a reassembled PDU]
718	164.754298	192.168.101.28	68.99.123.175	TCP	38404 > http [ACK] Seq=1889 Ack=6365 Win=26280 Len=0

▼ Frame 701: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)

Arrival Time: Sep 6, 2012 18:40:18.510430000 EDT
Epoch Time: 1346971218.510430000 seconds
[Time delta from previous captured frame: 0.000189000 seconds]
[Time delta from previous displayed frame: 0.000189000 seconds]
[Time since reference or first frame: 164.623068000 seconds]
Frame Number: 701
Frame Length: 74 bytes (592 bits)

0000	00 24 36 9e df 32 00 d0 09 08 04 9d 08 00 45 00	.\$.2..E.
0010	00 3c 1e bd 40 00 40 06 36 28 c0 a8 65 1c 44 63	.<@.6(..e.Dc
0020	7b af 96 04 00 50 a0 5b 9a 0a 00 00 00 a0 02	{....P.[.....
0030	39 08 e6 05 00 00 02 04 05 b4 04 02 08 0a 04 04	9.....
0040	

Frame (frame), 74 bytes Packets: 2628 Displayed: 2628 Marked: 0 Profile: Default

Testing DNS – nslookup

★ Verifying a name

```
$ nslookup www.theptrgroup.com  
Server:      127.0.0.1  
Address:     127.0.0.1#53
```

Non-authoritative answer:

```
Name:      www.theptrgroup.com  
Address:   205.134.170.50
```

★ Verifying an IP address

```
$ nslookup 156.154.70.1  
Server:      127.0.0.1  
Address:     127.0.0.1#53
```

Non-authoritative answer:

```
1.70.154.156.in-addr.arpa      name = rdns1.ultradns.net.
```

Authoritative answers can be found from:

nslookup and host

* nslookup using a specific DNS server

```
$ nslookup www.linuxhomenetworking.com 8.8.8.8 # Google's DNS server
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www.linuxhomenetworking.com canonical name = linuxhomenetworking.com.
Name:    linuxhomenetworking.com
Address: 162.249.37.221
```

* nslookup will be deprecated and replaced by the host command

- ▶ The host command syntax and output are very similar to that of nslookup

```
$ host www.linuxhomenetworking.com 8.8.8.8
Using domain server:
Name: 8.8.8.8
Address: 8.8.8.8#53
Aliases:

www.linuxhomenetworking.com is an alias for linuxhomenetworking.com.
linuxhomenetworking.com has address 162.249.37.221
linuxhomenetworking.com mail is handled by 10 nosmtp.linuxhomenetworking.com.
linuxhomenetworking.com mail is handled by 30 mailspoof.linuxhomenetworking.com.
linuxhomenetworking.com mail is handled by 20 mail.linuxhomenetworking.com.
```

Using nmap

- ✖ Can determine all the TCP ports on which a remote server is listening (that are not blocked by a firewall)
- ✖ Can also find open UDP ports, and much more
- ✖ Used to find network vulnerabilities
- ✖ Example: Scan using valid TCP connections (-sT) in extremely slow 'insane' mode (-T 5) from ports 1 to 5000

```
$ sudo nmap -sT -T 5 -p 1-5000 192.168.101.8
```

```
Starting Nmap 6.40 ( http://nmap.org ) at 2015-10-13 12:28 EDT
Nmap scan report for 192.168.101.8
Host is up (0.00055s latency).
Not shown: 4988 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
902/tcp   open  iss-realsecure
2000/tcp  open  cisco-sccp
2049/tcp  open  nfs

Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
```

Common nmap Arguments

Argument	Description
-p0	Nmap first attempts to ping a host before scanning it. If the server is being protected from ping queries, then you can use this option to force it to scan anyway.
-T	Defines the timing between the packets set during a port scan. Some firewalls can detect the arrival of too many non-standard packets within a predetermined time frame. This option can be used to send them from 60 seconds apart with a value of "5" also known as insane mode to 0.3 seconds with a value of "0" in paranoid mode.
-o	This will try to detect the operating system of the remote server based on known responses to various types of packets.
-p	Lists the TCP/IP port range to scan.
-s	Defines a variety of scan methods that use either packets that comply with the TCP/IP standard or are in violation of it.

Source: linuxhomenetworking.com

Testing Network Bandwidth

- ★ The **netcat** command can push the NIC to its capacity, but it does not report statistics
- ★ Use in conjunction with other tools
- ★ Listen on TCP port 7777, dumping the results on the floor

```
$ nc -l 7777 > /dev/null
```

- ★ Send OS-generated random data to the target host on port 7777

```
$ cat /dev/urandom | nc 192.168.2.50 7777
```

- ★ Note that disk latency is not incurred by either of these commands

Summary

- ★ The causes of network problems are not always obvious
 - ▶ Multiple approaches and tools are usually used when debugging a network problem
- ★ Linux provides many tools that allow you to analyze a network's configuration and traffic
- ★ Often times, we will need to combine the outputs from several commands to determine the problem

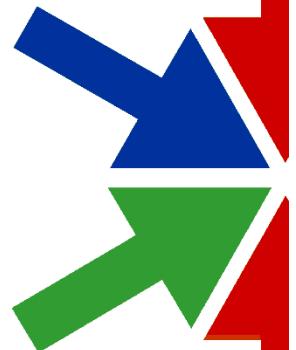
Questions

- ★ Hardware link status can be verified using the **linkstat** command
 - ▶ True or False?
- ★ ICMP return code 11 indicates that the packet has a bad message length
 - ▶ True or False?
- ★ Tools like **tcpdump** and **wireshark** use the libpcap interface for accessing packets
 - ▶ True or False?
- ★ Wireshark can read packet captures created by **tcpdump**
 - ▶ True or False?
- ★ The **qmap** tool can be used to find open ports on a server
 - ▶ True or False?

Chapter Break

Name Resolution and Service Discovery

ptr



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

★ Name resolution

- ▶ Linux name service
 - Resolver API
 - Nsswitch configuration
- ▶ Hostname resolution
- ▶ Sources of information

★ Service discovery

- ▶ Competing standards from Apple, Microsoft, and IETF

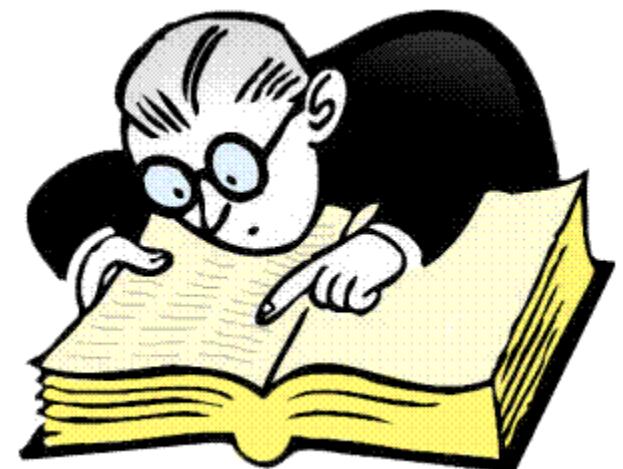
In the Dark Ages...

Humans once used phone books

- ▶ To find people or companies by name
- ▶ To find companies by name within service type

Name resolution and service discovery fulfill similar roles on IP networks:

- ▶ Find hosts by name
- ▶ Find services by name within type



Source: www.seoogle.com

Name Resolution

- ★ In general terms, name resolution refers to using a name to obtain other information about an object
- ★ On Linux systems, a name service can be used to obtain system and network information, such as:
 - ▶ Hosts
 - ▶ Networks, protocols, services
 - ▶ Users and groups

Linux Name Service

* Name service provides:

- ▶ Resolver API functions
- ▶ Ability to retrieve information from multiple sources
 - Local files, DNS, mDNS, LDAP, NIS, etc.

* Information sources and precedence are configured via the Name Service Switch

Name Service Switch

- ★ Resolver behavior is determined by `/etc/nsswitch.conf` entries
- ★ Each entry identifies a data type, and an ordered list of information sources for that data type
- ★ Information sources are consulted in order (from left to right) until an answer is found
- ★ Optional reaction clause (“[...]”) following an information source name in the list overrides normal search behavior

Default nsswitch.conf

Database Name	Lookup Rule
passwd:	compat
group:	compat
shadow:	compat
hosts:	files dns [!UNAVAIL=return]
networks:	nis [NOTFOUND=return] files
ethers:	nis [NOTFOUND=return] files
protocols:	nis [NOTFOUND=return] files
rpc:	nis [NOTFOUND=return] files
services:	nis [NOTFOUND=return] files

*Information Source Names
(Where to Look)*

Reaction

The diagram consists of two green arrows. One arrow originates from the text 'Information Source Names (Where to Look)' at the bottom left and points upwards towards the 'Lookup Rule' column. The other arrow originates from the text 'Reaction' at the bottom right and points upwards towards the 'Lookup Rule' column.

Common Service Options

Service Name	Type of Info Source	Typical Scope of Service
files	Local files, such as /etc/hosts	Small local network
db	Berkeley Database formatted files	Small local network
nis, nisplus	Sun's Network Information Service	Small enterprise
dns, dns6	Unicast Domain Name System	Wide area network
mdns, mdns_minimal, mdns4, mdns4_minimal, mdns6, mdns6_minimal	Apple's Multicast Domain Name Service (Avahi)	Small local plug-n-play network
ldap	Lightweight Directory Access Protocol	Enterprise or WAN
wins	Microsoft's NetBIOS Name Service (Samba)	Small to large Enterprise

Primary Resolver API Functions

Primary Function	Information Served
getaddrinfo(3)	host IP addresses
getnetbyname(3)	networks
getservbyname(3)	IP service ports
getprotobynumber(3)	IP protocols
getrpcbyname(3)	RPC program numbers
getpwent(3)	users and passwords
getrent(3)	groups

- Refer to each primary function's man page for the inverse function
- Note `getnameinfo(3)` replaced `gethostbyname(3)` in glibc version 2.1

Hostname Resolution

- ★ Names are easier to remember than IP addresses (especially for IPv6)
- ★ Name service helps translate names to/from IP addresses
 - ▶ Works for hostnames, aliases, or fully-qualified domain names
 - ▶ myhost -> 192.168.1.2
 - ▶ localhost -> 127.0.0.1
 - ▶ youtube.com -> 63.88.73.53
- ★ You see it in action when you type:
host www.youtube.com



Source: marchpr.com

Calling getaddrinfo()

```
// Resolving a given domain name to a list of address-info  
// structs  
  
struct addrinfo *info;  
status = getaddrinfo("www.youtube.com", NULL, NULL, &info);  
  
// IP version-agnostic extraction of address bytes  
addr_bytes = ADDR(info->ai_family, info->ai_addr);  
  
// Converting address bytes to printable ASCII string  
addr_str = inet_ntop(info->ai_family, addr_bytes,  
                     buf, sizeof buf);  
  
// Freeing the info list  
freeaddrinfo(info);
```

Example Address Resolution

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int main(int argc, char *argv[])
{
    struct addrinfo hints, *result, *rp;
    int sfd, s;
    struct sockaddr_storage peer_addr;
    socklen_t peer_addr_len;
    ssize_t nread;
    char buf[BUF_SIZE];

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;      /* Allow IPv4 or IPv6 */
    hints.ai_socktype = SOCK_DGRAM;   /* Datagram socket */
    hints.ai_flags = AI_PASSIVE;      /* For wildcard IP address */
    hints.ai_protocol = 0;            /* Any protocol */
    hints.ai_canonname = NULL;
    hints.ai_addr = NULL;
    hints.ai_next = NULL;
```

Example Address Resolution (2)

```
s = getaddrinfo(argv[1], NULL, &hints, &result);
if (s != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
    exit(EXIT_FAILURE);
}

/* getaddrinfo() returns a list of address structures.
   Try each address until we successfully bind(2).
   If socket(2) (or bind(2)) fails, we (close the socket
   and) try the next address. */

for (rp = result; rp != NULL; rp = rp->ai_next) {
    sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
    if (sfd == -1) {
        continue;
    }

    if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0) {
        break; /* Success */
    }

    close(sfd);
}

if (rp == NULL) { /* No address succeeded */
    fprintf(stderr, "Could not bind\n");
    exit(EXIT_FAILURE);
}

freeaddrinfo(result); /* No longer needed */
```

Example Address Resolution (3)

```
/* Read datagrams and echo them back to sender */

for (;;) {
    peer_addr_len = sizeof(struct sockaddr_storage);
    nread = recvfrom(sfd, buf, BUF_SIZE, 0, (struct sockaddr *) &peer_addr, &peer_addr_len);
    if (nread == -1) {
        continue; /* Ignore failed request */
    }

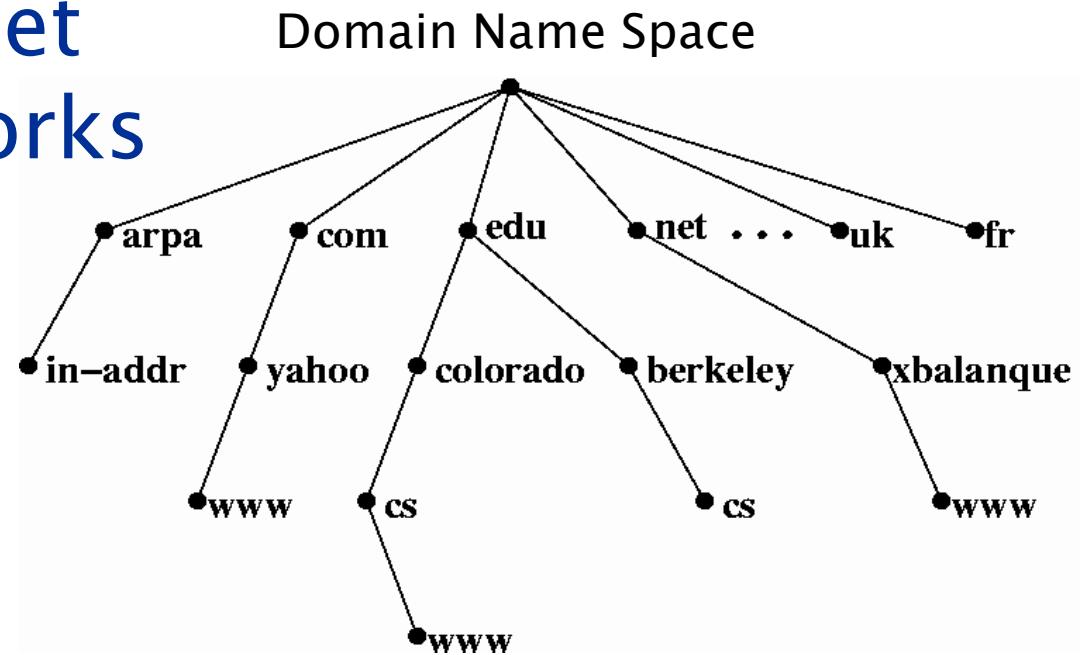
    char host[NI_MAXHOST], service[NI_MAXSERV];

    s = getnameinfo((struct sockaddr *) &peer_addr, peer_addr_len, host, NI_MAXHOST,
                    service, NI_MAXSERV, NI_NUMERICSERV);
    if (s == 0)
        printf("Received %ld bytes from %s:%s\n", (long) nread, host, service);
    else
        fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));

    if (sendto(sfd, buf, nread, 0, (struct sockaddr *) &peer_addr, peer_addr_len) != nread) {
        fprintf(stderr, "Error sending response\n");
    }
}
```

Domain Name System (DNS)

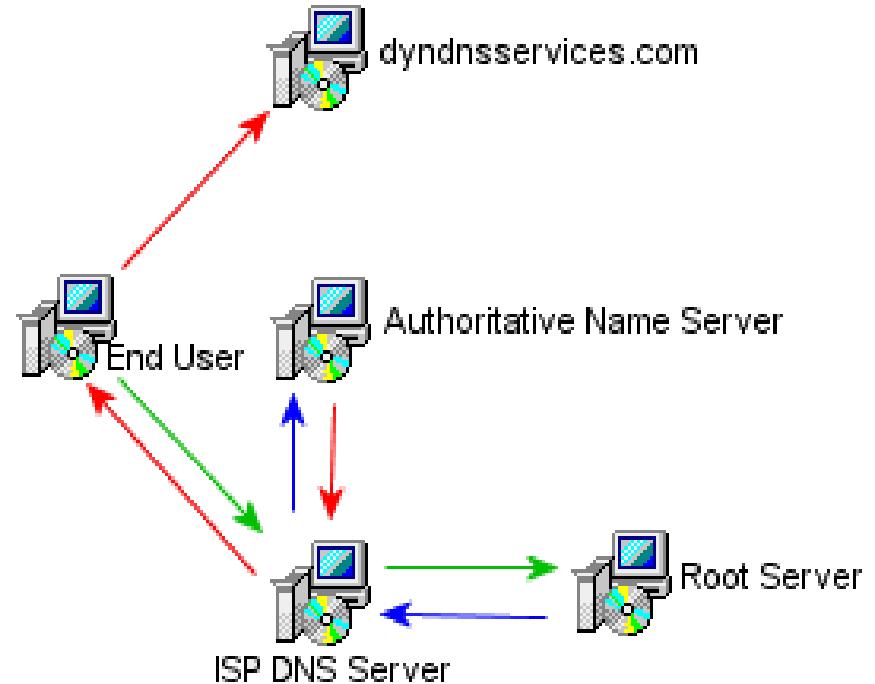
- ★ A distributed hierarchical naming system
- ★ Powers the internet and private networks
- ★ Resolves domain names to IP addresses



Source: <http://bio3d.colorado.edu>

DNS Server Roles

- ★ Root name server is at the top of the hierarchy
- ★ Authoritative name server records are explicitly configured
- ★ Intermediate servers may cache results of previous queries



Source: <http://www.dyndnsservices.com>

Key DNS Record Types

★ Queries are answered according to a server's DNS record database and cache entries

Record Type	Description
A	Address record, maps name to IP address
NS	Name server record, references another DNS server
MX	Mail server record
CNAME	Alias record, an alternate name for an entity

★ Many more record types exist, including AAAA for IPv6 address records

Configuring a DNS Client

- ★ Insert “dns” in the search list in /etc/nsswitch.conf

```
hosts:      files dns
```

- ▶ This example tells the resolver to make a DNS query if the name is not found in the local /etc/hosts file
- ★ Configure DNS options in /etc/resolv.conf
 - ▶ Read by the resolver the first time it is invoked by a process
 - ▶ Entries consist of keyword-value pairs
 - ▶ Only the local name server will be queried if /etc/resolv.conf does not exist

Configuring resolv.conf

* **nameserver ip-address**

- ▶ Nameservers to query
- ▶ Limit of 3 entries
- ▶ Queried in the order listed
- ▶ Defaults to local nameserver

```
domain home
search home
nameserver 192.168.1.1
nameserver 73.251.0.12
```

* **domain name**

- ▶ Local domain name
- ▶ Derived from host name if not provided

* **search name-list**

- ▶ Domains to search for hostnames
- ▶ Space or tab delimited
- ▶ Defaults to local domain

Configuring resolv.conf (2)

* sortlist ip-netmask-list

- ▶ Sort order of `getaddrinfo` (3) results
- ▶ IP–netmask pairs (netmask optional)
- ▶ Space or tab delimited
- ▶ Example:
 - `sortlist 130.155.160.0/255.255.240.0 130.155.0.0`

* options option-list

- ▶ Space or tab delimited list of resolver options
- ▶ Notably, `debug`
- ▶ See the man page for others

DNS Troubleshooting Commands

***host** [-aCdlnrsTwv] [-c class] [-N
ndots] [-R number] [-t type] [-W wait]
[-m flag] [-4] [-6] {name} [server]

- ▶ Single DNS query
- ▶ Example:

```
$ host -t hinfo -W 10 www.cisco.com
```

- Queries host info with 10 second initial timeout

***dig server**

- ▶ Domain Information Groper
- ▶ Command-line or batch modes
- ▶ Preferred for its flexibility, easy of use, and clarity of output
- ▶ Flexible, many options

Example dig command

```
$ dig www.cisco.com

; <>> DiG 9.9.5-3ubuntu0.5-Ubuntu <>> www.cisco.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49779
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;www.cisco.com.           IN      A

;; ANSWER SECTION:
www.cisco.com.        1989    IN      CNAME    www.cisco.com.akadns.net.
www.cisco.com.akadns.net. 300    IN      CNAME    wwwds.cisco.com.edgekey.net.
wwwds.cisco.com.edgekey.net. 13404 IN      CNAME
wwwds.cisco.com.edgekey.net.globalredir.akadns.net.
wwwds.cisco.com.edgekey.net.globalredir.akadns.net. 1937 IN CNAME
e144.dscb.akamaiedge.net.
e144.dscb.akamaiedge.net. 20    IN      A       172.232.16.170

;; Query time: 26 msec
;; SERVER: 192.168.101.1#53(192.168.101.1)
;; WHEN: Fri Dec 04 08:32:26 EST 2015
;; MSG SIZE  rcvd: 223
```

LDAP

★ Lightweight Directory Access Protocol

- ▶ Standard protocol for accessing and maintaining distributed directory information services

★ Logical directory structure

- ▶ Hierarchical set of entries, accessed by “distinguished name”
(think pathname)
- ▶ Entries have a set of attributes
- ▶ Attributes have names and one or more values
- ▶ Entries and attributes are unordered



Source: www.hitechologies.com

Uses for LDAP

★ Useful for any application requiring a distributed, hierarchical directory

- ▶ Hostname resolution
- ▶ Service discovery
- ▶ Directory aggregation
- ▶ “White Pages” directory
- ▶ Identity management
- ▶ Certificate mgmt. (LDAPv3)
- ▶ Knowledge management
- ▶ ... and many more



Source: www.imex.com

LDAP Servers

- ★ Directory contents governed by a schema, known as a DIT (Directory Information Tree)
- ★ May hold a sub-tree and refer clients to other LDAP servers
- ★ Support anonymous, plain-text, and SASL (Kerberos) authentication
- ★ Open source server for Linux: OpenLDAP



Source: www.foodandbeverageunderground.com

Configuring an LDAP Client

★ Install required packages

- ▶ libnss-ldap, libpam-ldap, nscd
 - PAM library is for authentication
 - nscd is a caching daemon

★ Configure /etc/nsswitch.conf

```
hosts:      files  ldap  dns
```

★ Configure server options in /etc/ldap.conf

★ Add server credentials to /etc/ldap.secret

★ If using LDAP for authentication, modify PAM configuration files

mDNS

- ❖ Multicast DNS provides lookup of DNS records without a managed DNS server
- ❖ Uses multicast messages to the mDNS multicast address (IPv4) 224.0.0.251
 - ▶ IPv6 address is ff02::fb
- ❖ Designates a local DNS namespace
- ❖ Local hosts form a mDNS multicast group and answer mDNS requests if they have the requested DNS record

What is Service Discovery?

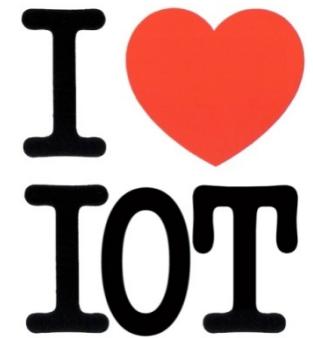
- ★ Service Discovery – automatic detection of network nodes and services
- ★ Broad topic with many protocols and applications, from discovering Java objects to web services
- ★ We will be discussing zeroconf and three competing standards:
 - ▶ Apple's mDNS / DNS-SD
 - ▶ Microsoft's UPnP
 - ▶ IETF's SLP



Source: testandtry.com

Zero Configuration Networking

- ★ In the “Internet of Things”, we want to bring home a new appliance, power it on, and instantly have it recognized by our smart phone, tablet, Smart TV, or media server
- ★ Devices must automatically obtain network addresses, make their presence known, and advertise their services to client devices



3 Pillars of Zeroconf

*Automatic:

- ▶ Assignment of network addresses
- ▶ Resolution and distribution of hostnames
- ▶ Service discovery



Source: www.wellhappypeaceful.com

Address Auto-Configuration

Link Local Addressing

- ▶ Self assigned, device checks for conflict
- ▶ IPv4
 - 169.254.0.0/16 APIPA block
 - Used when DHCP unavailable
- ▶ IPv6
 - Derived from factory-assigned MAC
 - fe80::/64 prefix
- ▶ Packets with these addresses are not forwarded by routers



Zeroconf Name Resolution

★ Multicast Domain Name Service

- ▶ IETF Internet-Draft

★ Apple's implementation, mDNS

- ▶ Compatible with DNS Service Discovery (DNS-SD)
- ▶ Device chooses domain name in local DNS namespace, announces it via multicast

★ Microsoft's implementation, Link Local Multicast Name Resolution (LLMNR)

- ▶ Incompatible with DNS-SD
- ▶ Device may choose any domain name
- ▶ IETF considers this a security risk

Service Discovery in Zeroconf

* IETF Service Location Protocol (SLP)

- ▶ Services described by URLs are announced on local network

* Apple's DNS Service Discovery (DNS-SD)

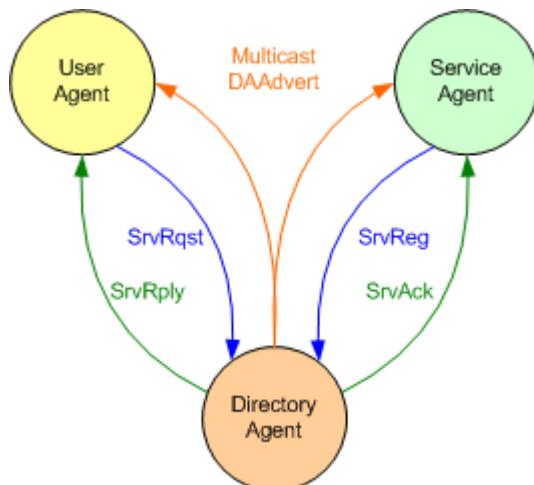
- ▶ Uses DNS SRV, TXT, PTR records to advertise services
- ▶ Compatible with mDNS

* Microsoft's UPnP Simple Service Discovery Protocol (SSDP)

- ▶ Service type and name sent via HTTP

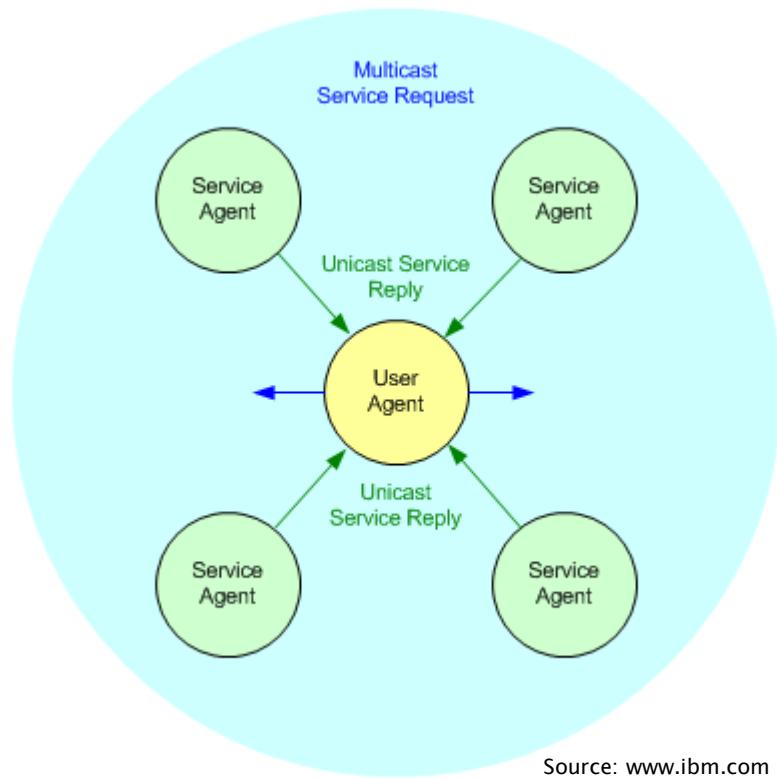
Service Location Protocol (SLP)

- Three device roles: Directory Agent (DA), Service Agent (SA), User Agent (UA)
- Uses UDP and TCP on port 427
- Uses multicast
- Active DA discovery:



Source: www.ibm.com

or



Source: www.ibm.com

Linux and SLP

- ★ Widely adopted for network attached storage (NAS) and network printers
 - ▶ Hewlett Packard, Novell, Storage Networking Industry Association
- ★ UA support for print and storage services built into many Linux distros
 - ▶ Common Unix Printing System (CUPS) supports SLP
- ★ OpenSLP provides a DA daemon and libraries to build SLP interfaces for other kinds of services

Applications by Implementation

★ mDNS / DNS-SD

- ▶ Apple Bonjour
- ▶ Printers, storage devices, any Apple-discoverable device
- ▶ Avahi on Linux

★ LLMNR / SSDP

- ▶ Small Office/Home Office (SOHO) devices
- ▶ Media Center systems
- ▶ Windows XP

★ SLP

- ▶ SUSE Linux
- ▶ Network printers
- ▶ NAS servers

Summary

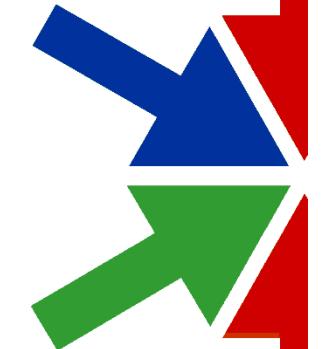
- ★ Linux provides flexible configuration of name resolution services
- ★ Depending on your system configuration, you also have choices for setting up service discovery
- ★ With the rollout of IPv6 and the rise of the Internet of Things, these services and zeroconf will become much more important

Questions

- ★ Name service configuration is set in:
 - /etc/dhcp3.conf
 - /etc/nsswitch.conf
 - /etc/passwd
- ★ DNS is the primary mechanism for name resolution on the Internet
 - ▶ True or False?
- ★ AAAA records are used in IPv6 DNS name resolution
 - ▶ True or False?
- ★ Name a tool for testing DNS name resolution
 - Bonjour
 - Dig
 - Dug
 - LDAP
- ★ Avahi is a Linux tool for implementing mDNS service discovery
 - ▶ True or False?

Chapter Break

Virtual Local Area Network (VLAN)



What We Will Cover

- ★ What is a VLAN?
- ★ Why use VLANs?
- ★ VLAN implementation
- ★ Using VLANs in Linux
- ★ Support for VXLAN

What is a VLAN?

- ★ Virtual Local Area Network
- ★ Set of hosts that are placed into the same Ethernet broadcast domain
 - ▶ Hosts do not have to be in the same physical broadcast domain (that is, attached to the same switch)
 - ▶ Excludes hosts not in the VLAN
 - Even if they are attached to the same switch
- ★ Like Ethernet bonding and bridges, VLANs are built on top of a virtual device in the kernel

Why Use VLANs?

★ Manageability

- ▶ Logical network layout driven by usage model, not physical network layout
 - E.g., Sales and HR on different logical networks, even though they share the same physical network
- ▶ Multiple classes of traffic (voice, data) over the same physical network cabling
 - Quite common in enterprises and hotels

Why Use VLANs? (2)

Performance

- ▶ Restrict Ethernet broadcasts to relevant hosts
 - One-to-one mapping between IP subnet and VLAN

Security

- ▶ Segregate traffic without needing to install additional switches
- ▶ Examples
 - Guest network on a separate VLAN
 - Restrict each switch port to a specific VLAN

Single Switch VLAN

★ Each frame received by the switch is associated with a particular VLAN

▶ Static VLAN / Port-based VLAN

- Each switch port is assigned to a VLAN
- All frames received on that port are in the VLAN
- Only frames for that VLAN are forwarded to the port

▶ Dynamic VLAN / MAC-based VLAN

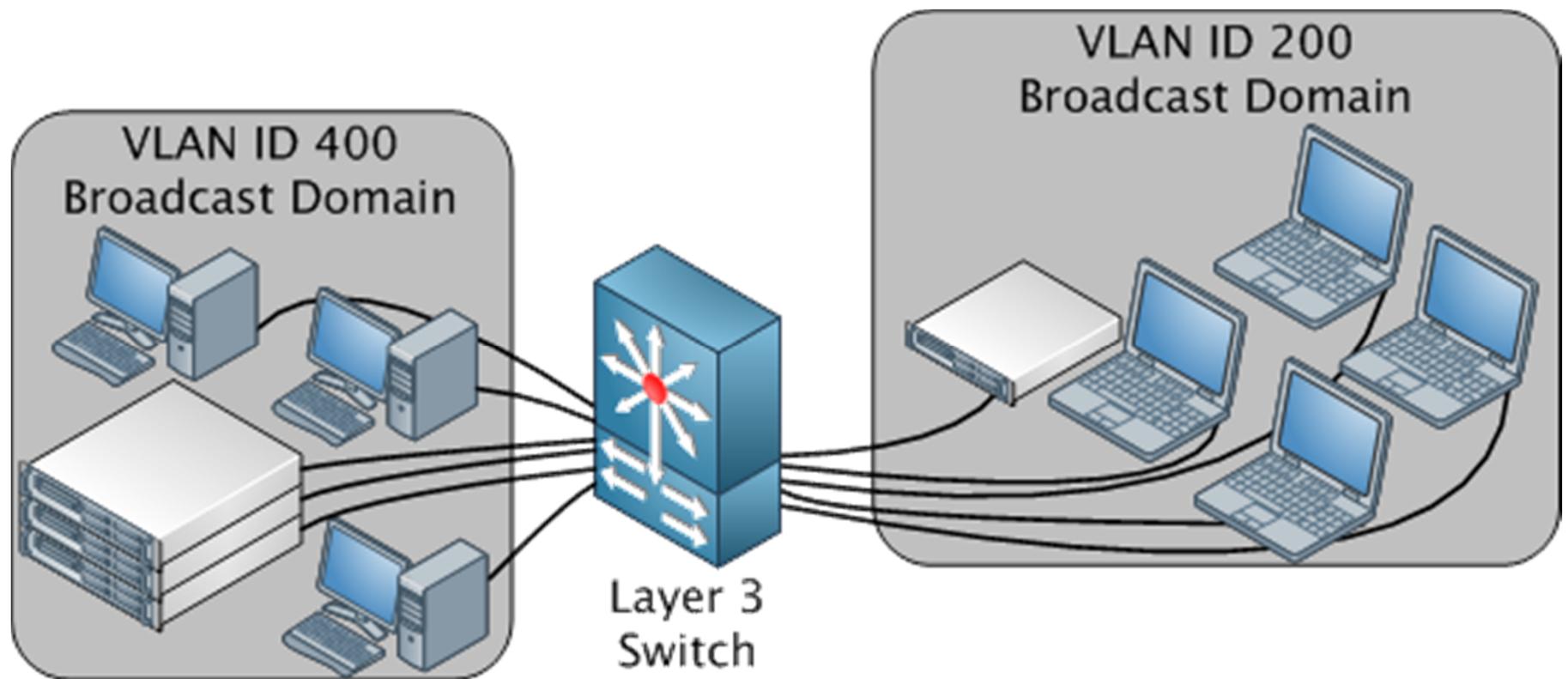
- Dynamically-configured, static VLAN
- When a device connects to the switch, the port is assigned to a VLAN according to a management database

▶ Protocol-based VLAN

- Different higher-layer protocols are assigned to different VLANs
 - E.g., keep IPX and IP traffic in different broadcast domains on the same switch

Basic Single Switch VLAN

- Each VLAN is a separate broadcast domain
- A Layer 3 switch or router must be used to pass traffic between VLANs

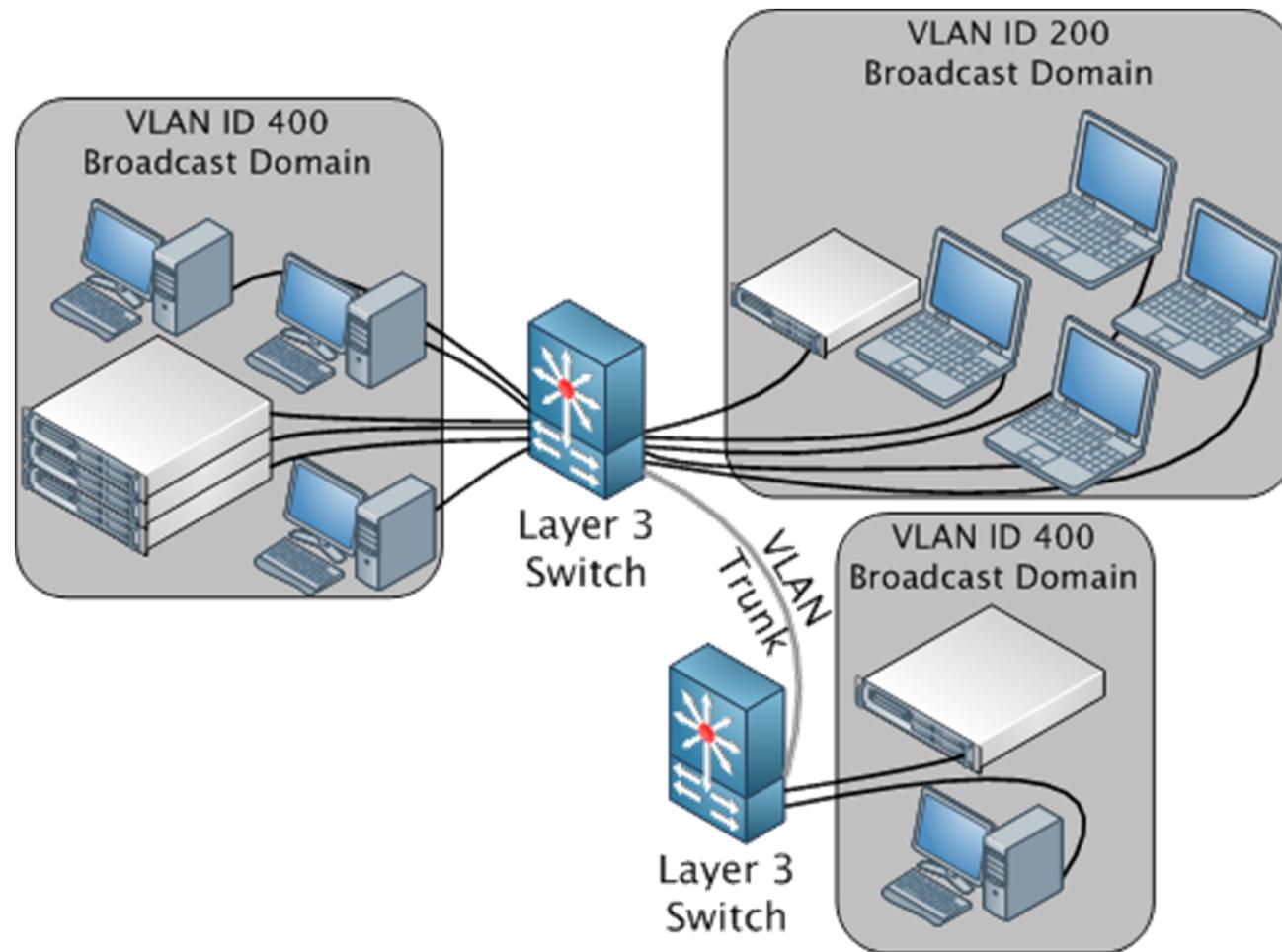


VLAN Trunking

- ★ A VLAN can span multiple switches by using “trunking”
 - ▶ IEEE 802.1Q specifies modification to the Ethernet frame to support trunking
- ★ Each Ethernet frame sent between the switches is tagged with the ID of the VLAN to which it belongs
- ★ A VLAN can span physically separate networks
- ★ Cisco’s proprietary ISL (Inter-Switch Link) protocol also handles VLAN trunking
 - ▶ PVSTP/PVSTP+ are also Cisco proprietary

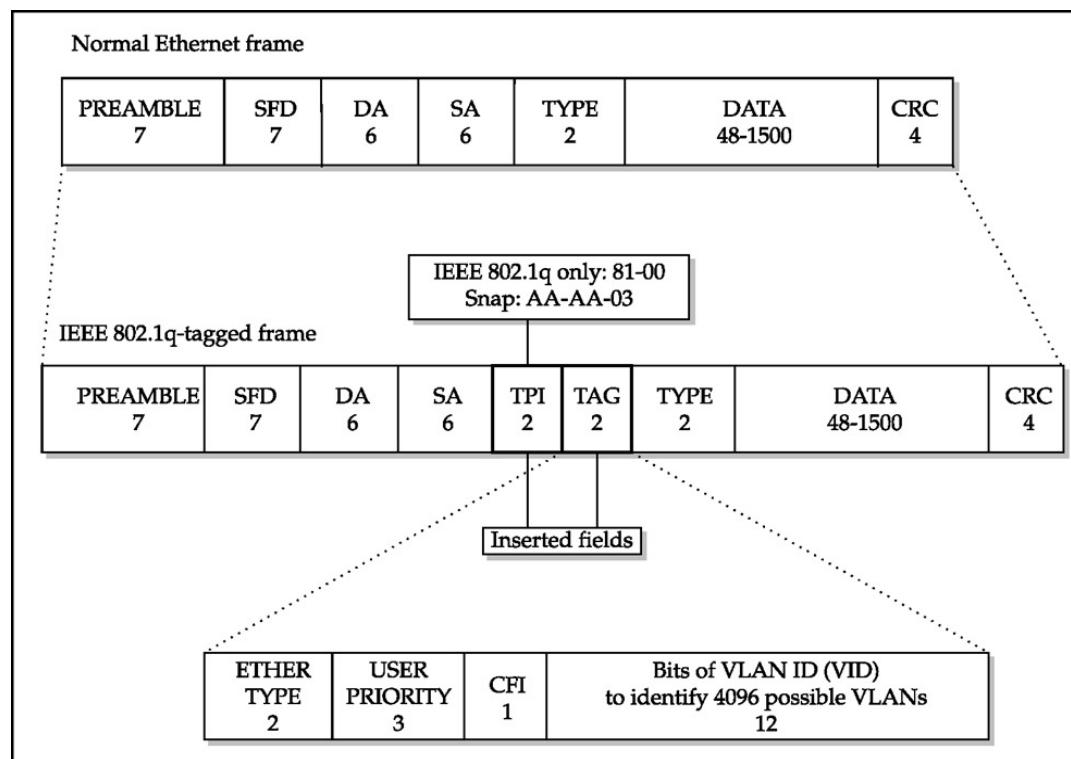
VLAN Trunking (2)

- ★ VLAN 400 spans two physical networks



VLAN Trunk Packet

- ★ Four-byte 802.1Q tag inserted between Source Address and the Ether-type field
- ★ 12 bit VLAN ID: 4096 possible VLANs



Source: cisco.iphelp.ru

Copyright 2007-2017, The PTR Group, Inc.

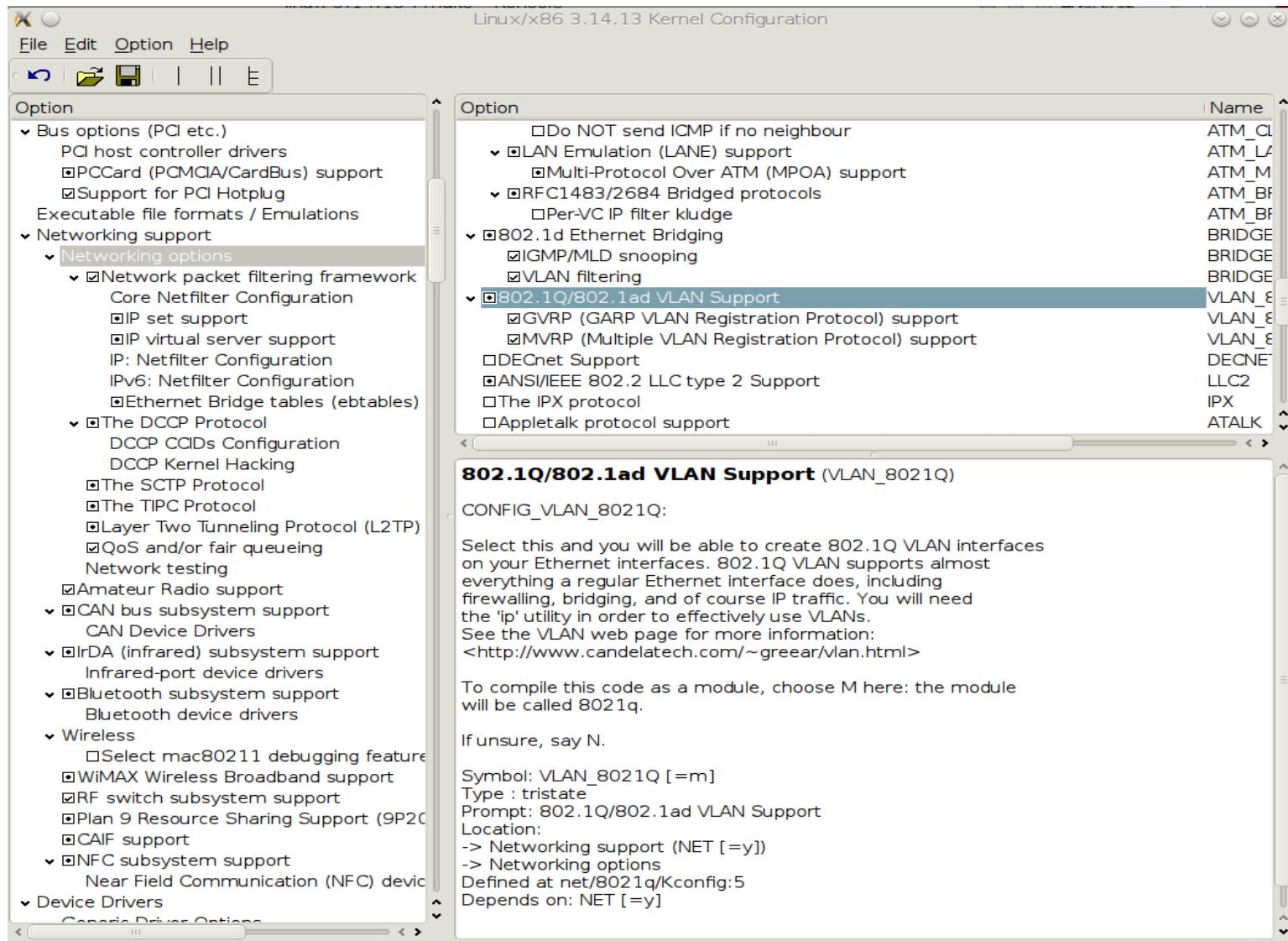
P2015-VLAN-10



Using VLANs in Linux

- ★ Access to a single VLAN (via a static, dynamic, or protocol-based VLAN) requires no changes for the host
 - ▶ The Ethernet frames the host sees are normal Ethernet frames
- ★ Access to a trunked VLAN segment, where the host sends and receives tagged frames, requires kernel support
 - ▶ 8021q kernel module
- ★ VLANs are configured using ip

Enabling VLANs in the Kernel



Linux Trunked VLAN Example

* Connect eth0 of the Linux box to a trunked VLAN port on the switch

- ▶ Access VLANs 200 and 400 as follows

* Add the kernel module

```
# modprobe 8021q
```

* Add 2 VLANs to eth0

```
# ip link add link eth0 name eth0.200 type vlan id 200
# ip link add link eth0 name eth0.400 type vlan id 400
```

* Configure IP addresses for each VLAN

```
# ip addr add 10.1.1.2/24 brd 10.1.1.255 dev eth0.200
# ip link set dev eth0.200 up
# ip addr add 10.1.2.2/24 brd 10.1.2.255 dev eth0.400
# ip link set dev eth0.400 up
```

Verify that the VLANs are up

- * We can now verify that the VLANs are up and running:

```
valiant ~ # ip -d addr show eth0.200
17: eth0.200@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc noqueue state UP group default
    link/ether d4:be:d9:6c:9e:b0 brd ff:ff:ff:ff:ff:ff
        inet 10.1.1.2/24 brd 10.1.1.255 scope global eth0.200
            valid_lft forever preferred_lft forever
```

```
        inet6 fe80::d6be:d9ff:fe6c:9eb0/64 scope link
            valid_lft forever preferred_lft forever
```

```
valiant ~ # ip -d addr show eth0.400
```

```
18: eth0.400@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
qdisc noqueue state UP group default
    link/ether d4:be:d9:6c:9e:b0 brd ff:ff:ff:ff:ff:ff
        inet 10.1.2.2/24 brd 10.1.2.255 scope global eth0.400
            valid_lft forever preferred_lft forever
        inet6 fe80::d6be:d9ff:fe6c:9eb0/64 scope link
            valid_lft forever preferred_lft forever
```

Stopping the VLAN and Removing the Device

* To turn down the devices:

```
# ip link set dev eth0.200 down  
# ip link set dev eth0.400 down
```

* To remove the devices:

```
# ip link delete eth0.200  
# ip link delete eth0.400
```

Red Hat Startup VLAN Config

- * Scripts in /etc/sysconfig/network-scripts/

- * ifcfg-eth0

```
DEVICE=eth0
```

```
ONBOOT=no
```

```
TYPE=Ethernet
```

- * ifcfg-eth0.200

```
DEVICE=eth0.200
```

```
IPADDR=10.1.1.2
```

```
NETMASK=255.255.255.0
```

```
VLAN=yes
```

```
ONBOOT=yes
```

```
BOOTPROTO=none
```

- * ifcfg-eth0.400

```
DEVICE=eth0.400
```

```
IPADDR=10.1.2.2
```

```
NETMASK=255.255.255.0
```

```
VLAN=yes
```

```
ONBOOT=yes
```

```
BOOTPROTO=none
```

Debian (Ubuntu) Startup VLAN Config

*Load module on boot

```
# echo 8021q >> /etc/modules
```

*Add the following to /etc/network/interfaces

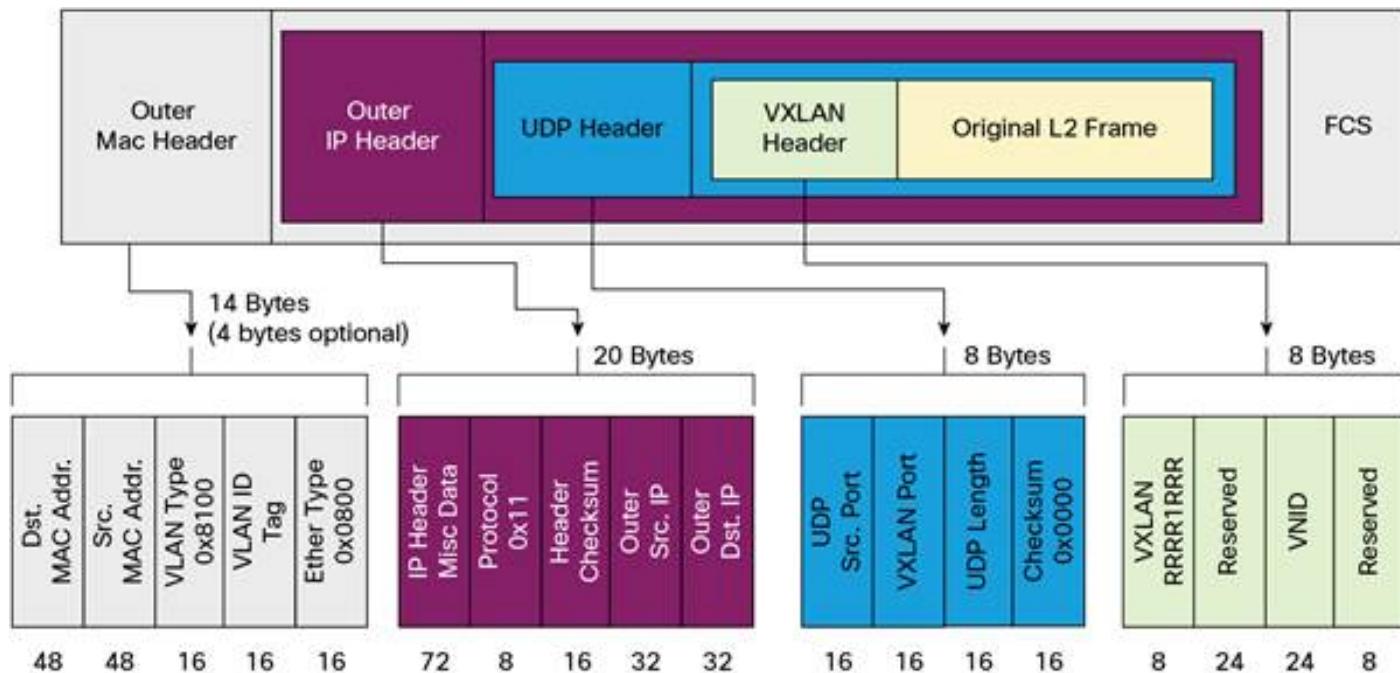
```
auto eth0.200
iface eth0.200 inet static
    address 10.1.1.2
    netmask 255.255.255.0
    vlan-raw-device eth0
auto eth0.400
iface eth0.400 inet static
    address 10.1.2.2
    netmask 255.255.255.0
    vlan-raw-device eth0
```

VXLAN Support

- ★ The Virtual Extensible LAN is a network virtualization technique that addresses the scalability problems found in large cloud deployments
 - ▶ Support added to the Linux kernel in 3.7 and it requires the iproute2 (e.g., **ip**) commands
- ★ VXLAN encapsulates L2 Ethernet frames in an L4 UDP packet using the IANA-assigned port 4789
- ★ VXLAN goes beyond the 4K virtual LANs of IEEE 802.1Q to support up to 16 million logical networks
 - ▶ Supports L2 adjacency across IP networks

VXLAN Packet

* VXLAN is a MAC-in-UDP encapsulation as shown below

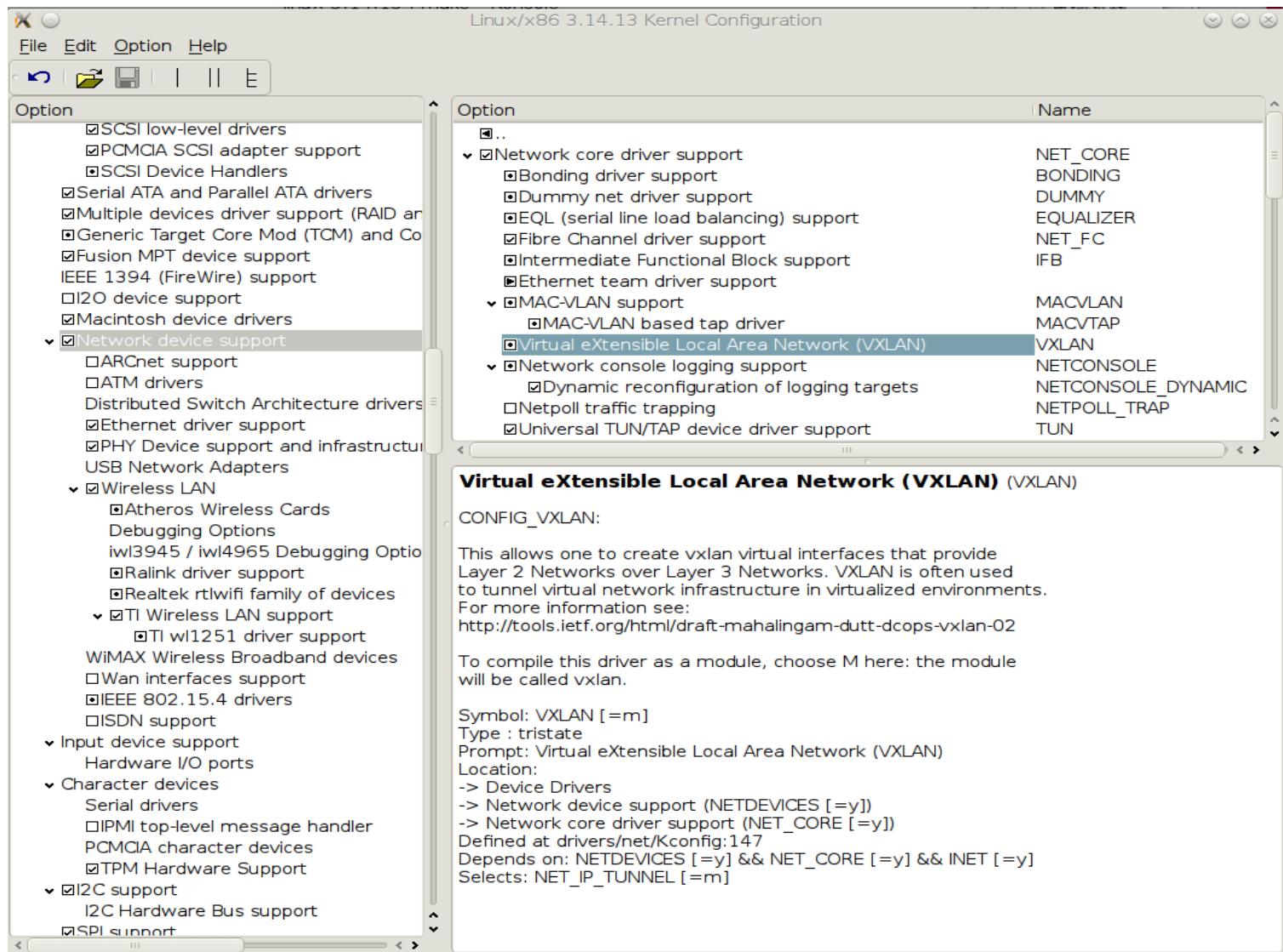


Source: cisco.com

VXLAN Support (2)

- ★ Multicast or unicast with Head-End Replication (HER) is used to flood broadcast, unknown destination address, multicast (BUM) traffic
- ★ The VXLAN specification was originally created by VMware, Arista Networks and Cisco
 - ▶ VXLAN is officially documented in RFC 7348

Enabling VXLAN in the Kernel



Linux Trunked VXLAN Example

- ★ Configure eth0 to have VXLAN support as follows
- ★ Add the kernel module

```
# modprobe vxlan
```

- ★ Add a VXLANs to eth0

```
# ip link add vxlan10 type vxlan id 10 group  
239.0.0.10 ttl 4 dev eth0
```

- ★ Configure an IP address for the VXLAN

```
# ip addr add 192.168.1.1/24 broadcast 192.168.1.255  
dev vxlan10  
# ip link set dev vxlan10 up  
# ip -d link show vxlan10  
21: vxlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state  
UNKNOWN mode DEFAULT group default  
    link/ether da:5f:f7:f1:f8:31 brd ff:ff:ff:ff:ff:ff promiscuity 0  
    vxlan id 10 group 239.0.0.10 dev eth0 port 32768 61000 ttl 4 ageing 300
```

VXLAN support in LXC and Open vSwitch

- ★ VXLAN support is available in both the Linux container interface (LXC) and in the Open vSwitch virtual switch implementation
- ★ The configuration of these features is covered in the Linux Virtualization with KVM class

Summary

- ✖ A VLAN is a Virtual Local Area Network
- ✖ Multiple VLANs may share a single physical network
- ✖ A VLAN may span multiple physical networks
- ✖ A trunked VLAN link carries tagged Ethernet frames for one or more VLANs
- ✖ Linux hosts can access simple (single VLAN ID) or trunked VLANs
- ✖ VXLAN support was added in the 3.7 kernel
 - ▶ Used in large-scale virtualization deployments

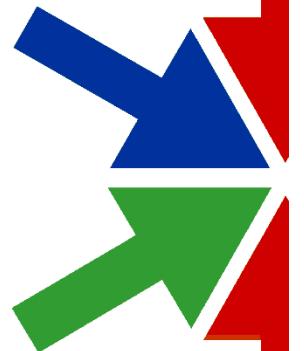
Questions

- ★ As nodes are added to a collision domain, network performance degrades
 - ▶ True or False?
- ★ IEEE 802.11V is the standard that describes VLAN support
 - ▶ True or False?
- ★ The VLAN tag mechanism supports 4096 different VLANs on a given segment
 - ▶ True or False?
- ★ VLANs can span physically separate networks
 - ▶ True or False?
- ★ VLAN support in Linux can be loaded dynamically via a kernel module
 - ▶ True or False?

Chapter Break

Network Security

PTR



Copyright 2007–2017,
The PTR Group, Inc.

What We Will Cover

- ★ Network Monitoring
- ★ Traffic Control
- ★ Network Encryption and VPN
- ★ Wireless
- ★ Tools and Resources



Source: thewheelwarehouse.com

Linux Networking Tools

- ★ Linux distributions come with a wealth of networking tools available as kernel modules, libraries, and applications
- ★ Available in a broad mix of languages: C, C++, Perl, Python, and Bash scripts
- ★ From a security perspective, we have tools that range from monitoring tools like **tcpdump** and **wireshark** to tools that actively probe the network interface like **Nessus**
 - ▶ The firewall interfaces can also be viewed as a network security feature

Network Monitoring

- ★ Network monitoring tools for Linux can be broadly grouped into two categories:
 - ▶ Low-level packet inspection and manipulation tools
 - ▶ Higher-level traffic and flow analysis and reporting
- ★ We will discuss some of the more important low-level tools in varying levels of detail, and provide a brief survey of the higher-level monitoring tools
 - ▶ We've already addressed **tcpdump** and **wireshark** in an earlier chapter

Packet Capture – TcpReplay Suite

- ★ A suite of tools for modifying packets captured in pcap format and then replaying them at arbitrary speeds
- ★ <http://tcpReplay.synfin.net>
- ★ Useful for repeatable testing
- ★ Excellent for regression testing fixes to packet handling bugs
- ★ Safer security testing – replaying traffic rather than rerunning exploits

Tcpreplay – Usage

★ Basic workflow:

- ▶ Capture packets with **tcpdump** or **wireshark**
- ▶ Run **tcpprep** (the Tcpreplay pre-processor) to “split” traffic into client/server streams
 - Split is done using tunable heuristics based on the traffic from each host in the capture file
- ▶ Run **tcprewrite** to edit packet data at various layers of the packets in the capture file
 - Can modify most elements of L2, L3 or L4
 - Automatically recalculates the checksums
- ▶ Run **tcpreplay** to send the captured packet data out onto the network
 - Great for regression testing of network vulnerabilities

Snort

* Open Source Network Intrusion Detection and Prevention System (IDS/IPS)

* Signature detection

- ▶ Compares packets against signatures of known malicious traffic
- ▶ Extensive signature database available

* Protocol based inspection

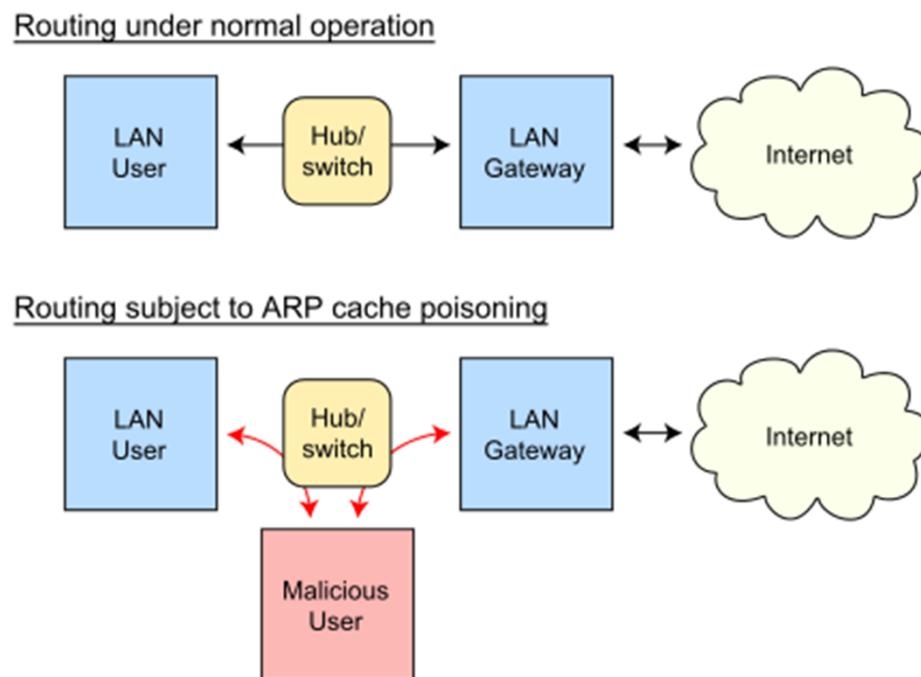
- ▶ Signatures can be confined to match only the appropriate location in protocol specific traffic

* Anomaly detection

- ▶ Detect network traffic anomalies using plugins

arpwatch

- ★ Detect ARP poisoning attempts
- ★ Monitors ARP traffic and reports changes in MAC:IP address pairs



Source: wikipedia.com

Active Network Monitoring

★ nmap

- ▶ Performs network scans to identify listening services on network nodes and the versions of those nodes

★ Nessus/OpenVAS, Nmap

- ▶ Scan network nodes for vulnerable applications
- ▶ OpenVAS is the open source fork of Nessus
 - Contains roughly half as many detection signatures
- ▶ Nmap is from Rapid7, makers of Metasploit

★ iperf

- ▶ Perform TCP and UDP bandwidth and latency testing

Passive Network Monitoring

* Argus

- ▶ Network flow monitoring, collect and report metrics such as delay, jitter, and packet loss

* Nagios

- ▶ Perform system and network monitoring
- ▶ Centralized display of network and host availability and performance data

* mrtg

- ▶ Monitor and graph network statistics available via SNMP from routers, servers, and other network nodes

* p0f

- ▶ Passive operating system identification by examining packet characteristics

* ClamAV

- ▶ Antivirus scanning engine and constantly refreshed signatures for viruses, trojans, and other malware
- ▶ De facto standard for mail-gateway AV in Linux

Simple Network Encryption

* OpenSSL

- ▶ Library implementing Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1)
- ▶ Typically used server-side for HTTPS
- ▶ Provides tools for certificate creation and management

* Stunnel

- ▶ Establishes SSL/TLS tunnels for wrapping unencrypted TCP or UDP traffic
- ▶ Listens for encrypted traffic on one port, passes unencrypted traffic to a separate port

Simple Network Encryption (2)

*socat

- ▶ Generic network traffic relay
- ▶ Establish encrypted (with TLS) or unencrypted connections between nearly arbitrary endpoints
- ▶ Easy proxies, tunnels, port forwarding, IP-to-serial connections

*ssh

- ▶ Secure shell protocol
- ▶ Encrypted remote command line access
- ▶ Simple encrypted tunnels with system-level authentication

Linux VPN Types

IPsec – IP Security

- ▶ Can be used to create a VPN by providing per-packet authentication and encryption of IP traffic
- ▶ Authentication using IKE (Internet Key Exchange) and Authentication Header
- ▶ Encryption of payload using ESP (Encapsulating Security Payload)

L2TP – Layer 2 Tunneling Protocol

- ▶ Encapsulates a tunneled protocol (typically PPP) within an IP tunnel where the tunneled packets are sent as UDP datagrams
- ▶ Provides no authentication or encryption
- ▶ May be combined with IPsec in the tunnel to provide authentication and encryption

Linux VPN Types (2)

★ PPTP – Point-to-Point Tunneling Protocol

- ▶ TCP session is used to initiate and manage a GRE (Generic Routing Encapsulation) tunnel between the endpoints
 - PPP is passed over the GRE tunnel
 - Packets encrypted with MPPE (Microsoft Point-to-Point Encryption)
- ▶ NATs must be configured to handle/forward both the initiating TCP session and the GRE tunnel

★ SSL/TLS – Secure Socket Layer / Transport Layer Security

- ▶ Encrypted traffic is tunneled over a TCP socket
- ▶ Asymmetric encryption for key exchange
- ▶ Symmetric key for packet encryption

IPsec – IP Security Protocols

* Suite of IETF protocols

- ▶ Authentication Header (AH)
- ▶ Encapsulating Security Payload (ESP)
- ▶ Internet Key Exchange (IKE & IKEv2)
 - Supported by the user-space **raccoon** daemon

* Operates at the Internet layer in the IP protocol stack (peer of IP, ICMP, IPv6)

* Each packet is authenticated

IPsec – IP Security Protocols (2)

Encryption – via ESP

- ▶ Payload only – transport mode
- ▶ Entire packet – tunnel mode
- ▶ Keys determined using IKE

Mutual authentication of endpoints – via AH

No need to modify application to use more secure communications

Developed for IPv6, backported to IPv4

IPsec – NAT-Traversal

- ★ IPsec not designed for use across NAT
 - ▶ Designed for IPv6 and IPv6 does not support NAT
 - ▶ NAT IP and port rewriting breaks key exchange and ESP
 - ▶ IKE uses the IP addresses as the identifiers for the endpoints

- ★ racoon supports NAT-Traversal in tunnel mode, for using IPsec behind a NAT

- ▶ Added to `/etc/racoon.conf`

```
nat_traversal on;      # enable NAT-Traversal
natt_keepalive 20;    # Send a keepalive every N seconds
isakmp_natt 10.1.1.2 [4500]; # IP and port to listen on
```

- ★ <http://www.ipsec-howto.org/x299.htm>

Cisco VPN

- ★ Cisco implements Virtual Private Networks (VPN) using IPsec
- ★ Supports NAT-Traversal
- ★ Linux supports Cisco's VPN using the `vpnc` package
 - ▶ 0.5.3 is most recent release
- ★ Runs entirely in user space
 - ▶ Uses the standard TUN kernel module
- ★ Path of least resistance to access Cisco VPNs
- ★ Also works on FreeBSD & OS/X

Linux vpnc - Configuration

* Install vpnc

- ▶ On Debian / Ubuntu: `apt-get install vpnc`
- ▶ On Red Hat / CentOS: `yum install vpnc`

* Create vpnc config file

- ▶ Default file is `/etc/vpnc/default.conf`, if not there, then `/etc/vpnc.conf`

```
IPSec gateway vpngateway.yourdomain.com
IPSec ID your_group_name
IPSec secret your_group_password
Xauth username your_username
Xauth password your_password
```

Linux vpnc - Connecting

* Install the tun kernel module

```
# modprobe tun
```

* Run the vpnc command to connect to the VPN

- ▶ It will prompt for configuration parameters that were not found in the config file
- ▶ E.g., if the Xauth password entry is omitted, vpnc will prompt for the password:

```
# vpnc
```

Enter password for your_username@vpngateway.yourdomain.com:

* Run the vpnc-disconnect command to disconnect from the VPN

OpenVPN

- ★ Built on OpenSSL
- ★ <http://openvpn.net>
- ★ Tunnel IP traffic over single UDP/TCP port
- ★ Uses PKI for authentication
 - ▶ Private and public keys for clients and servers
 - ▶ Master Certificate Authority cert used to sign the public keys of clients and server

OpenVPN: Routed vs. Bridged Mode

* Routed mode

- ▶ More efficient and scales better
- ▶ Recommended configuration
- ▶ But may not support all clients and all protocols
 - IPv6 support is only available in OpenVPN 2.3+

* Bridged mode places each client into the server's Ethernet broadcast domain

- ▶ Can support any protocol that works over Ethernet
- ▶ The current best option for IPv6 or IPX networks

Wireless – What We Will Cover

★ Supported protocols

★ Wi-Fi security

★ Wireless network connection tools

- ▶ Network Manager (**nm-applet**)
- ▶ **wicd**
- ▶ Wireless Tools: **iwconfig**, **iwlist**, **iwspy**, **iwpriv**

★ Monitoring

- ▶ **Kismet**
- ▶ **Wireshark**

Linux Wireless Protocols

★ Support for several wireless protocols

- ▶ Wi-Fi (802.11[a, b, g, n, ac]) with drivers for a variety of hardware
 - Supporting WEP, WPA, WPA-PSK, WPA2
 - Some support for emerging 802.11ac devices – typically from the device vendor, not in the mainline kernel yet
- ▶ WiMAX (802.16)
 - Driver donated by Intel
- ▶ Bluetooth
- ▶ ZigBee (802.15.4) / 6LoWPAN
 - Supports certain classes of devices (those where the MAC layer is implemented in the device)
- ▶ NFC
- ▶ Ultra Wide Band
- ▶ LTE
 - For certain USB-connected LTE modems
- ▶ 3G
- ▶ UMTS
 - For certain UMTS modems (serial and USB attached)

Wireless Security

- ✖ WEP – Wired Equivalency Protocol
 - ▶ Crackable in 5–10 minutes – DO NOT USE
- ✖ WPA/WPA2 Wi-Fi Protected Access
 - ▶ Intended to address the insecurity of WEP
 - ▶ Described by IEEE 802.11i
 - ▶ WPA is a subset of 802.11i specification (which wasn't final yet)
 - Interim replacement for WEP
 - ▶ Per packet encryption key
 - ▶ WPA uses RC4 for encryption (as did WEP)
 - But, changes encryption key for each data transmission
 - ▶ WPA2 uses AES for encryption
 - Better encryption, but requires more powerful hardware
 - ▶ Multiple authentication options
 - WPA-PSK (Pre Shared Keys)
 - Typical for a home or SOHO setup
 - Vulnerable to a dictionary attack attempting to brute-force the keys
 - Authentication via EAP (Extensible Authentication Protocol)
 - Supports plug-ins for PKI, RADIUS, IKEv2, or Diffie-Hellman
- ▶ WPS (Wi-Fi Protected Setup) vulnerability
 - WPS uses a preset PIN for each device to ease configuration
 - WPS is vulnerable to a brute-force attack to determine the PIN in a matter of hours
 - On some devices, you may not be able to disable WPS

Managing Wi-Fi in Linux

* ConnMan

- ▶ Daemon-based connection management designed for embedded Linux systems
- ▶ Plugin-based architecture
- ▶ Supports Wireless, Ethernet, WiMAX, Bluetooth, and VPNs

* wicd

- ▶ Graphical and curses-based management of connections

* NetworkManager

- ▶ Graphical “automagic” management of connections
- ▶ Supports Wireless, Ethernet, and PPP (including VPNs)
- ▶ Standard for RHEL and Ubuntu distributions

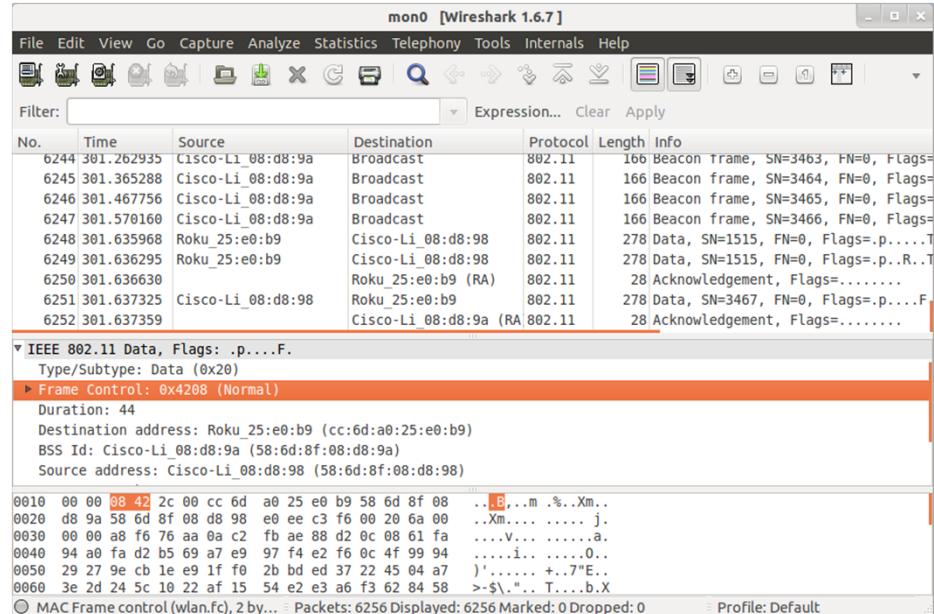
Wireless Monitoring – Kismet

- ★ <http://www.kismetwireless.net>
- ★ IEEE 802.11 L2 wireless sniffer
- ★ Passively identifies networks (named and hidden)
- ★ Finds non-beaconing networks via data traffic
- ★ Plug-in architecture to add additional features
- ★ Examine captured packets using the IDS plug-in
 - ▶ Or send captured packets to Snort via TUN/TAP interface

Wi-Fi Monitoring – Wireshark

- ★ Enable “monitor mode” on the radio to see raw 802.11 headers
- ★ The **iw** command is the current standard tool for manipulating wireless interfaces
 - ▶ Even though **airmon-ng** from the **aircrack-ng** package is often easier to use
- ★ Enable monitor mode on the appropriate device
 - ▶ Remember to delete the monitor interface when finished

```
# iw dev wlan0 interface add mon0 type monitor
# ifconfig mon0 up
# wireshark -i mon0
# iw dev mon0 del
```



Additional Utilities and Tools

- ★ netcat, socat, netpipes
- ★ ethtool
- ★ dsniff, ngrep
- ★ netstat, lsof
- ★ strace, ltrace
- ★ trickle
- ★ wget, curl
- ★ iftop, ethstatus, net-snmp
- ★ cutter, ettercap

Resources

- ★ <http://sectools.org>
- ★ <http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html>
- ★ <http://www.policyrouting.org/iproute2.doc.html>
- ★ <http://lxr.free-electrons.com/>
- ★ <http://freecode.com>

Summary

- ★ A broad array of network-related kernel features, libraries, applications and services are available for Linux
 - ▶ These are used to configure, secure, and monitor your network
- ★ Open source tools also serve as an excellent resource for investigating how to solve a particular task in Linux
 - ▶ What library or system call function(s) are used to accomplish the task

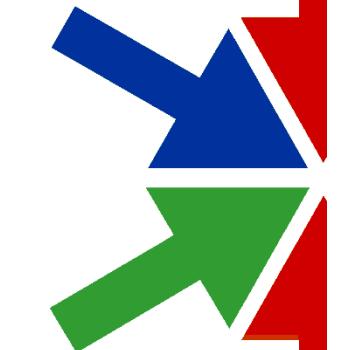
Questions

- ★ Linux supports the WPA/WPA2 encryption techniques used in Wi-Fi access points
 - ▶ True or False?
- ★ The ability to replay packet streams can be used to test for vulnerabilities
 - ▶ True or False?
- ★ Snort is an open-source intrusion detection and prevention system
 - ▶ True or False?
- ★ VPNs can be implemented using SSL via:
 - A. vpnc
 - B. arpwatch
 - C. OpenVPN
 - D. Mosaic
- ★ Kismet is an extensible wireless traffic monitoring facility in Linux
 - ▶ True or False?

Chapter Break

Final Thoughts

All's well that ends...



What We Will Cover

- ★ Where we've been
- ★ Where to next?
- ★ Recommended reading

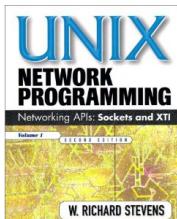
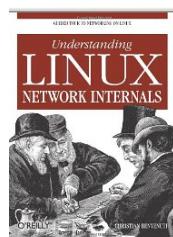
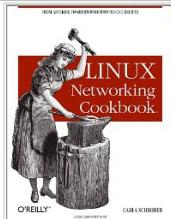
Where We've Been

- ⌘ Over the past four days, we've covered a lot of ground with Linux networking
- ⌘ We've seen some of the details of the protocol stack itself
- ⌘ We've examined many of the major features of the Linux networking model
- ⌘ We've also looked at debugging and security techniques

Where to Next?

- ★ There is no teacher better than experience when it comes to networking
 - ▶ Set up a couple of Linux systems or VMs and start experimenting
- ★ Next, get the kernel source code for your system and start looking at the kernel configuration
 - ▶ Rebuild the kernel if your default config doesn't support the options of interest

Good Reference Books



- ★ **Linux Networking Cookbook**
Carla Schroder
ISBN: 0596102488
- ★ **Understanding Linux Network Internals**
Christian Benvenuti
ISBN: 0596002556
- ★ **The Accidental Administrator: Linux Server Step-by-Step Configuration Guide**
Don R. Crawley
ISBN: 1453689923
- ★ **Unix Network Programming Vol 1**
W. Richard Stevens
ISBN: 013490012X

Web References

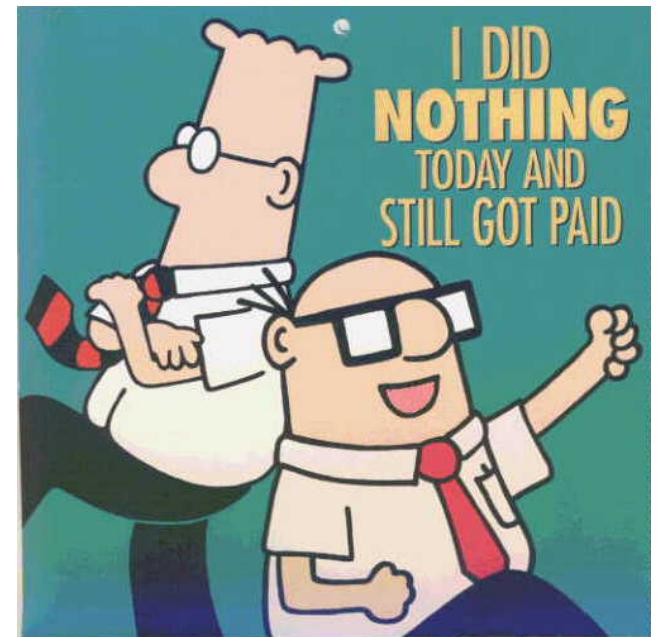
- ★ As usual, a good search engine is your friend
- ★ There are a few tutorials available on the web
 - ▶ <http://www.linuxhomenetworking.com>
- ★ Many of the tutorials are targeted at more generic server set up rather than the theory of operation
- ★ As you discover new answers, consider sharing them with the community at large

Final Words

* Now you know enough to really be dangerous...

- ▶ Take little baby steps and don't be afraid to experiment

* Go ye forth and network



Source: <http://www.dailydawdle.com>