Linux System Administration SSU

Networks and Firewalls

1

This talk is intended to give you a foundation for understanding computer networks. The emphasis is on Linux, but much of this (especially the first part of the talk) will apply to any operating system.

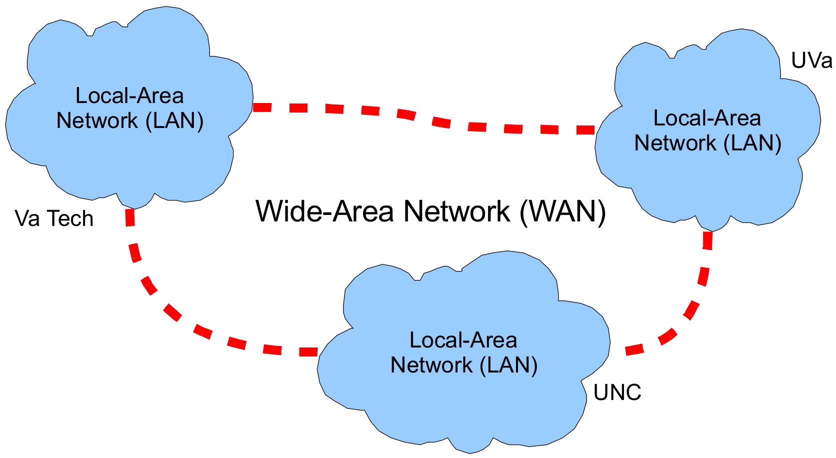
Even if you don't make direct use of this knowledge, it provides important background information you can use when configuring networks and firewalls, troubleshooting network problems, trying to understand what's possible on the network, and planning the future of your department's IT infrastructure.

Part 1: Network Hardware

This is a broad topic, but I'll really only be talking about the most common type of network hardware, Ethernet over twisted-pair copper wire.

Local-Area and Wide-Area Networks:

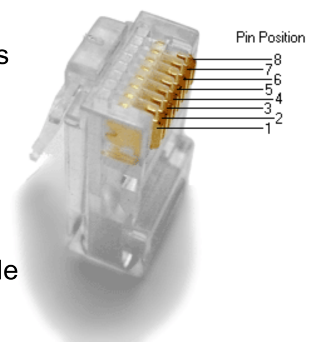
A LAN is a network that covers a relatively small, well-defined physical area like a department or a college campus. LANs are typically linked together by WANs. LANs tend to connect to WANs at only a few points, through expensive leased lines. LANs are optimized for speed over short distances. WANs are optimized for reliability over long distances. Because of all this, WANs and LANs tend to use different technologies.

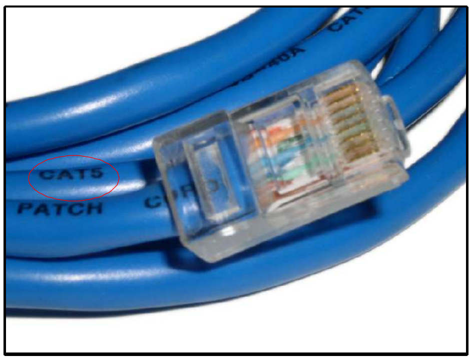


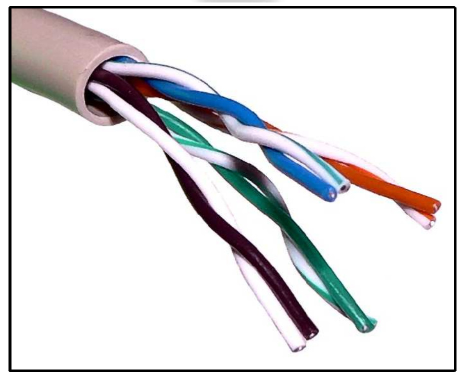
Local networks are created by local IT staff, or by hired contractors, who string cables and put in other network hardware as necessary. You can't usually connect two widely-separated networks this way, though. It would be unreasonable to send your IT people walking down the road from UVa to Va Tech, reeling out cable behind them, to create a link between the two universities. Instead, local networks are typically linked by leased lines, rented from phone companies or other providers. These companies have the infrastructure already in place to connect distant locations and maintain that connection.

Ethernet:

The most commonly used technology for wired LANs is currently “Ethernet” carried over twisted-pair cables. The twisted pairs of wires help reduce crosstalk. Ethernet devices can communicate at several different standard speeds: 10, 100 or 1000 Mbits/sec. Cable is designated “Category 5”, “5e” or “6” depending on its characteristics, with Category 6 cable being the best quality, and the only one suitable for reliable communication at 1000 Mbits/sec.







The connectors at the ends of the cables are called RJ-45 connectors. They're a type of modular connector similar to modular telephone connectors (called RJ-11), but wider.

Each pair of wires carries signals. At the end of the line, the voltage difference between the two wires in the pair is measured. By measuring this differential voltage, noise that affects both wires equally is eliminated. The twists in the wires help ensure that sources of noise along the way do affect both wires equally.

This twisted-pair scheme has been in use since the early days of the phone system. It's well-understood technology, and cheap to produce.

Ethernet Hardware Standards:

There are many varieties of ethernet hardware. Here are a few of them:

Twisted-Pair Copper:

- 10BaseT: 10 Mbits/sec over twisted-pair copper cable.
- 100BaseT: 100 Mbits/sec over twisted-pair.
- 1000BaseT: 1000 Mbits/sec (1 Gbit/sec) over twisted-pair.
- 10GBaseT: 10 Gbit/sec over twisted-pair (future).

Optical Fiber:

- 1000BaseSX: 1000 Mbits/sec over optical fiber.
- 10GBaseSR: 10 Gbit/sec over optical fiber (future).

Coaxial (obsolete):

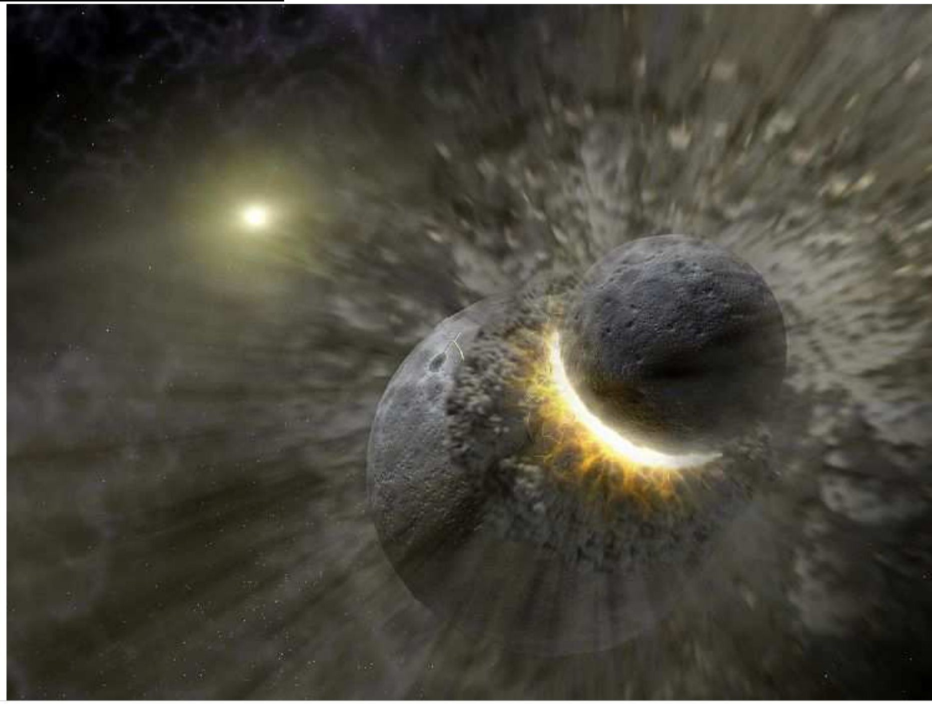
- 10Base5: 10 Mbits/sec over 50-ohm RG-8 coaxial cable.
- 10Base2: 10 Mbits/sec over 50-ohm RG-58 coaxial cable.

A note on nomenclature: The “base” in names like “10BaseT” comes from “baseband”, meaning that these transmission standards use a band of frequencies starting at zero and going up to some maximum, cutoff, frequency.

Optical fibers are better for long distances, but they're more expensive to deploy and maintain. They're usually used for connecting buildings together.

Typical network connections to desktop computers currently use either 10BastT or 100BaseT Ethernet. 1000 Mbit/set Ethernet is often used between buildings or as the backplane of clusters of computers.

Part 2: Ethernet



Now that we've seen the hardware, let's look at how Ethernet works. As we'll see, collisions are an integral part of it.

MAC Addresses:

Every Ethernet device has a unique 6-byte (48-bit) address, called a "Media Access Control" (MAC) address. Typically, the first 3 bytes of the address identify the network device's manufacturer, leaving 3 bytes (24 bits) to identify the device uniquely in that manufacturer's address space.

00	:	1A	:	A0	:	BF	:	6B	:	5F
(00-FF)		(00-FF)		(00-FF)		(00-FF)		(00-FF)		(00-FF)

Dell, Inc.

One of the 2^{24} addresses available to Dell.

(2^{24} = 16,777,216)

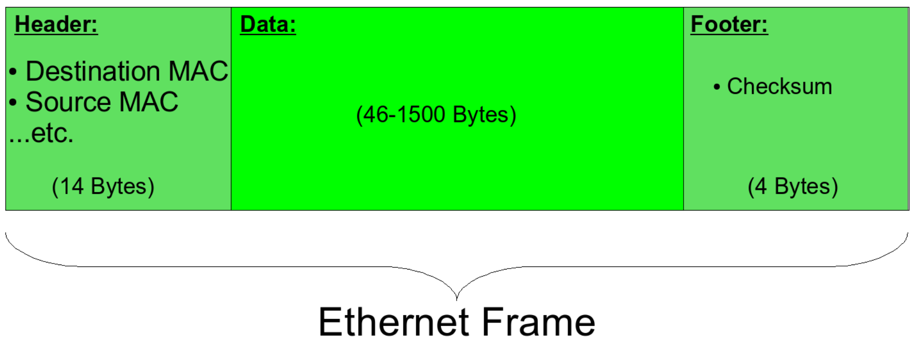
A tool for looking up vendors, by MAC address: http://www.coffer.com/mac_find/

A single manufacturer may actually be allocated several different 3-byte prefixes, especially a large manufacturer like Dell. There are plenty to go around: almost 17 million.

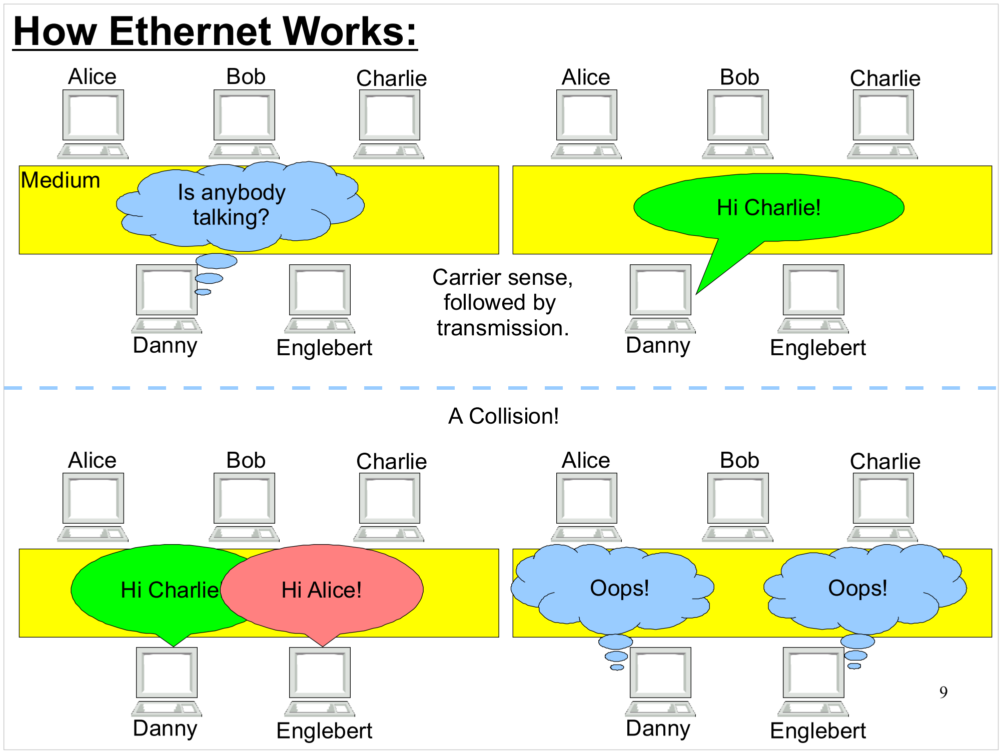
There's no expectation that we'll run short of MAC addresses anytime soon. The total number of them is about 280 trillion. This isn't the case with some other network addresses, as we'll see.

Ethernet Frames:

Data is transmitted across an Ethernet network within packages called “frames”. The structure of an Ethernet frame looks like this:



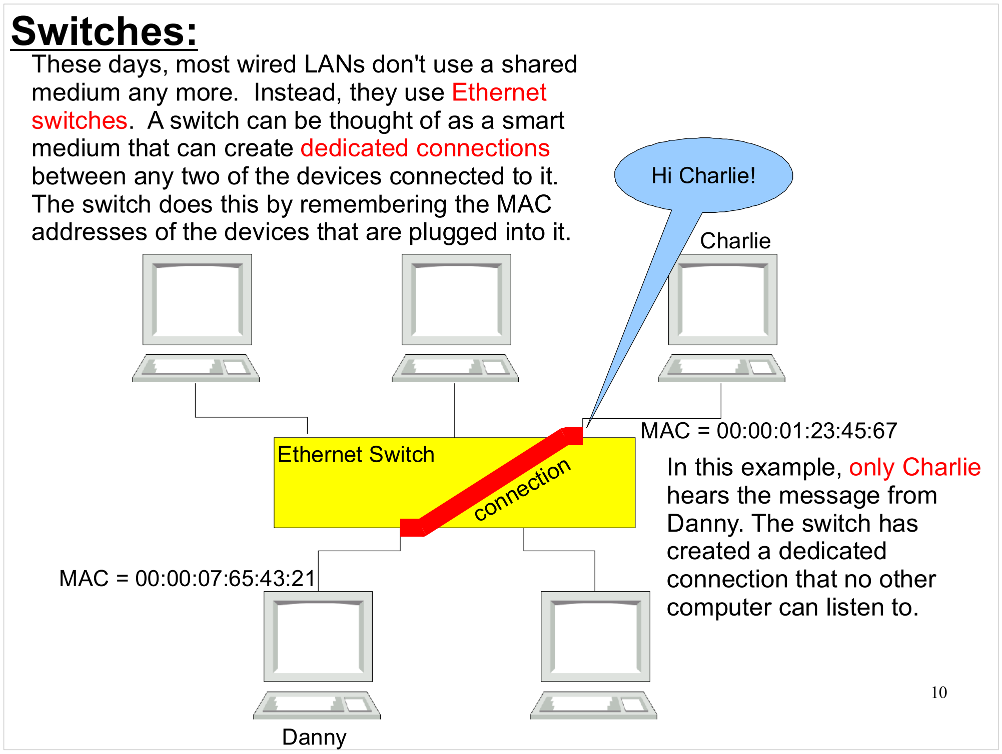
The checksum at the end of the frame is just a number computed from the frame's data. When the frame is composed, some function is applied to the data that produces a “hash”. The function is chosen so that any small change in the data will produce a different hash. This hash is stored as the checksum value in the frame's footer. When the frame arrives, the same function is applied again to the data. If the resulting hash doesn't match the hash stored in the checksum, then the receiver knows that the frame has been mangled during transmission.



You can think of the original Ethernet design as a bunch of rooms along a corridor. Whenever one of the occupants (a computer) wants to communicate with another, it shouts the message down the corridor (a “shared medium”). Everyone except the intended recipient ignores the message.

Originally, Ethernet used a design called “Carrier-Sense, Multiple Access with Collision Detection” (CSMA/CD). When an Ethernet device wants to start talking, it first listens to see if the shared medium is free, then it transmits its message. If two devices talk at the same time, that's a “collision”, and each device shuts up for some random time, before trying again. Early on, this simple-minded system was (amazingly) shown to perform much better than more sophisticated networking schemes.

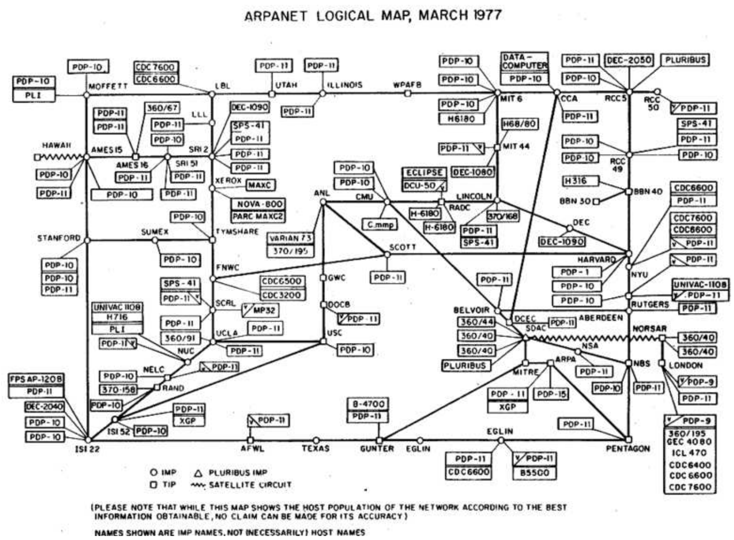
It's important to remember that collisions are a natural part of the way Ethernet works. There's nothing wrong with a few of them.



The switch does its work by remembering the MAC addresses of the devices that are plugged into it. For each connector, the switch maintains a list of the MAC addresses it has recently seen talking on that connection.

Some traffic still needs to be broadcast, though. (ARP packets, for example, as we'll see.) The switch will send broadcast traffic to all devices connected to it.

Part 3: The Internet Protocol (IP):



In the 1960s the Advanced Research Projects Agency (ARPA) of the Department of Defense began building a nationwide network called “ARPANET”. The first link (at 50 Kbits/sec) was created between UCLA and Stanford in 1969. ARPANET grew into the Internet we know now.

A new protocol, called “Internet Protocol” (IP) was invented for transmitting data on the Internet. This protocol was intended to be layered on top of a variety of underlying protocols. Thus, IP was independent of the details of a site's local network hardware. The Internet Protocol was capable of binding together a heterogeneous collection of computers around the world.

IP Addresses:

Each host on an IP network should have a unique 4-byte (32-bit) IP address. An IP address uniquely identifies a host on the Internet. IP addresses are typically expressed in “dotted decimal” form, like this:

192	.	168	.	100	.	21
(0-255)		(0-255)		(0-255)		(0-255)

(2^{32} = 4,294,967,296 possible addresses)

IP address numbers are managed by the “Internet Assigned Numbers Authority” (IANA). Three address ranges are reserved for private networks:

From	To	Number
10.0.0.0	10.255.255.255	16777216
172.16.0.0	172.31.255.255	1048576
192.168.0.0	192.168.255.255	65536

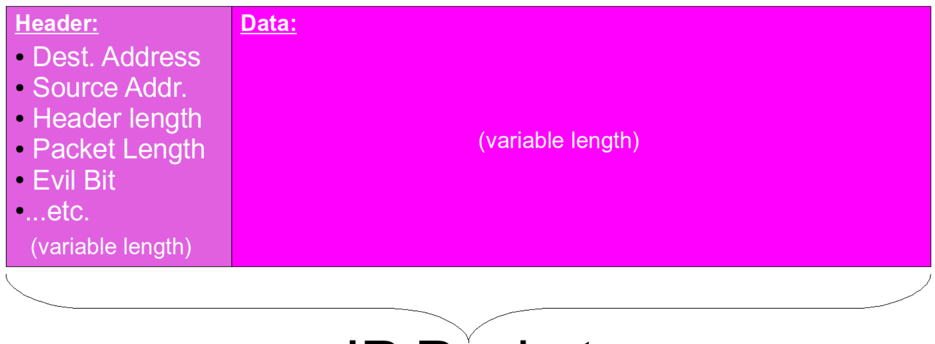
Note that I use the term “host” here instead of computer. By “host”, I mean “a thing with an IP address”. Usually, there will be a one-to-one mapping between IP addresses and computers, but not always by any means.

The IANA hands out blocks of addresses through regional registrars to internet service providers (ISPs). Each ISP is approved by the regional registrar, and must pay an annual fee to retain its IP addresses. UVa owns several address ranges: 128.143.*.*, 137.54.*.* and portions of the 199.111.*.* address space. The annual fee for a two-byte address range (called a /16 or a “Class B” network) like 128.143.*.* is $4,500.

Finally, note that everything I'm going to say about IP applies to the current version, IPv4. The successor, IPv6, is on its way, but probably won't affect you for a few more years. IPv6 has a much larger address space and a different notation.

IP Packets:

IP data is transmitted in “packets” that look like this:



IP Packet

(up to 65,535 bytes)

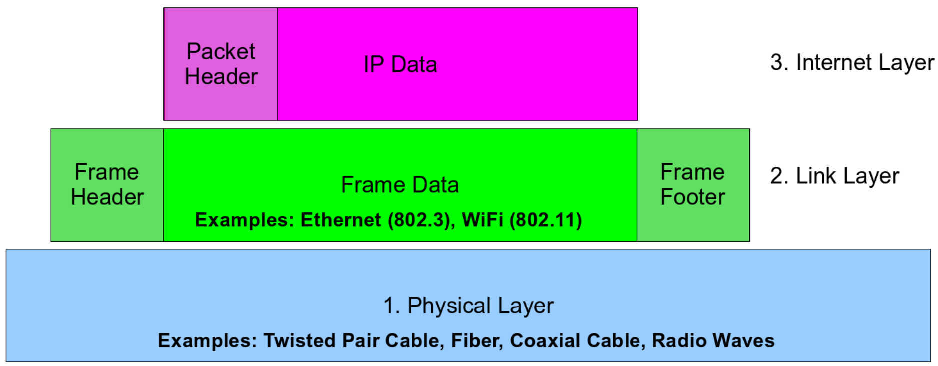
The source and destination address, in this case, are IP addresses. The “packet length” gives the total length of the packet. Other header entries give the length of the header itself, which can vary. The “Evil Bit” is an April Fool's joke. RFC 3514 proposes that an unused bit in the header be used to signify whether the sender is evil, making it easy to block malicious packets.

Notice that an IP packet can have a maximum size of 65,535 bytes, but an Ethernet frame can only carry up to 1,500 bytes of data. When a large IP packet arrives at an Ethernet network, it may be broken up into smaller packets (“fragments”) which are sent in separate Ethernet frames. The IP headers of these fragments will contain the information necessary to reassemble them later.

The maximum size of the underlying layer's “chunks” is called the “Maximum Transmission Unit” (MTU).

The IP Layer:

The Internet Protocol is layered on top of lower-level protocols. For example, on an Ethernet network, IP packets are transmitted within the data section of Ethernet frames. When two networks meet, the IP data may be re-packaged and transmitted further in some completely different way. In this way, IP packets can traverse heterogeneous networks.



We're starting to build up a stack of protocols, each of which adds features unavailable at lower levels. The physical layer gives us a mechanism for transmitting zeros and ones. The “link” layer (e.g., Ethernet) gives us a way to send a set of zeros and ones to a particular computer. The Internet layer gives us a way to transmit data across a heterogeneous network. We'll add two more layers before we're done.

You can imagine IP packets travelling across the network the way you'd take a plane trip. First, you get in your car and drive to the airport. There, you board a small plane and fly to another airport, where you get on a big plane and fly somewhere else. Once there, you rent a car and drive to your final destination. Just like you, IP packets can travel in a variety of vehicles on their way from one computer to another. Sometimes they'll be contained in Ethernet packets, sometimes they'll travel over other types of network. But eventually they'll arrive a their destination.

The “ping” Command:

The “ping” command sends small packets to a host on the Internet, then tells you if the host responded and how long it took the packet to get there and back. It will also tell you if any packets were lost in transmission. By default, ping will keep sending packets until you stop it with a “Ctrl-C”.

```
~/demo> ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.367 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=1.01 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.326 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.275 ms
```

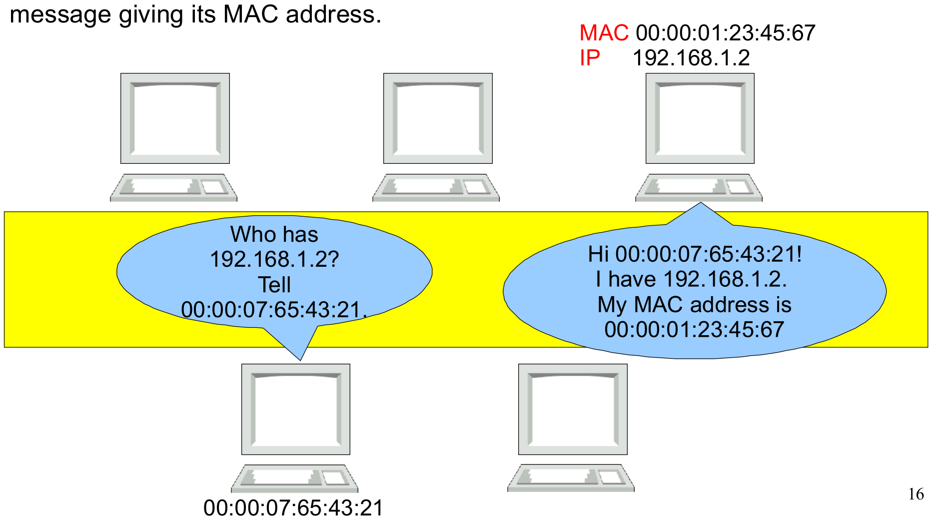
Ctrl-C

```
--- 192.168.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time
2998ms
rtt min/avg/max/mdev = 0.275/0.494/1.011/0.301 ms
```

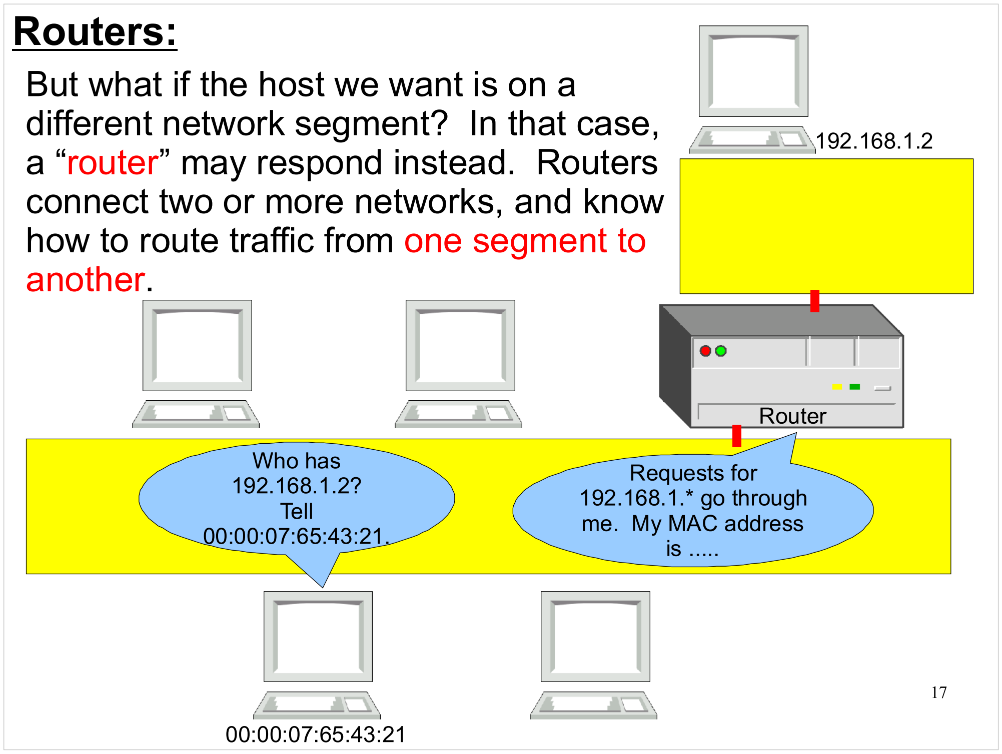
Note that a host may simply choose not to respond to ping requests. This is often done for security reasons. Bad Guys will often look for target computers by pinging, in numerical order, each IP address on a network. Addresses that don't respond may be ignored.

The Address Resolution Protocol (ARP):

In order to compose an Ethernet frame, we need to know the recipient's MAC address. If we know a host's IP address, we may be able to find its MAC address using a protocol called ARP. ARP packets are broadcast, so every host on the local network hears them (even if they're connected through a switch). If a host with the requested IP address is on the local network, it will respond with a message giving its MAC address.



To broadcast a request on an Ethernet network, you send it to the special MAC address “FF:FF:FF:FF:FF:FF”.



A network "segment" is a section of a network that is connected at the physical layer. Traffic can travel between all computers on the same segment without the help of routers or other intermediate devices that understand higher-level protocols.

When a router hears an ARP request for a host on one of the other segments it's connected to, the router responds with its own MAC address. The sending computer then knows to send Ethernet frames to the router, and the router will pass them along to the other network segment, doing its own ARPs there to find the destination host.

The ARP Cache:

Each computer maintains a cache of the results of recent ARP requests. Under Linux, you can use the "arp" command to view the contents of the cache. This is a good way to find the MAC address of another local computer. If it's not in the cache already, use "ping hostname" to send a packet to the host, and then look again. Note that remote hosts will all appear to have the MAC address of the router.

```
[root@demo ~]# arp

Address                  HWtype  HWaddress           Flags Mask  Iface
print.phys.Mydomain.Org  ether   00:16:3E:3E:8D:00   C           eth0
tracking.phys.Mydomain.  ether   00:04:75:06:E8:D7   C           eth0
data.phys.Mydomain.Org   ether   00:04:75:86:EA:5E   C           eth0
d-128-100-154.bootp.Myd  ether   00:21:70:DF:23:E0   C           eth0
vesna.phys.Mydomain.Org  ether   00:16:76:83:01:AE   C           eth0
galileo.phys.Mydomain.O  ether   00:15:C5:5D:58:72   C           eth0
memory-alpha.phys.Mydom  ether   00:04:75:86:EA:02   C           eth0
gilmer-router-all.acc.M  ether   00:D0:05:30:78:00   C           eth0
teleport.phys.Mydomain.  ether   00:20:AF:69:13:B5   C           eth0
```


One of those things you'll probably never need to do, but it's possible anyway:

You can also use the "arp" command to manually manipulate the ARP cache. This is sometimes necessary for configuring network devices that are only accessible through the local network segment (i.e., they don't have a keyboard or any other way of configuring them locally). Using the appropriate arp command, you can manually enter a MAC address into the ARP cache and associate it with an IP address. You can then use that IP address to communicate with the remote device.

The "traceroute" Command:

You can use the "traceroute" command to trace the path that packets would follow from one computer to another. In the example below, traceroute shows the path through various routers to a host at Google.

```
~/demo> traceroute  64.233.169.103
traceroute to 64.233.169.103 (64.233.169.103), 30 hops max, 40 byte packets
 1  gilmer-router-all.acc.Virginia.EDU (128.143.102.1)  0.578 ms  0.714 ms  0.684 ms
 2  carruthers-6509a-x.misc.Virginia.EDU (128.143.222.46)  0.407 ms  0.613 ms  0.589 ms
 3  new-internet-x.misc.Virginia.EDU (128.143.222.93)  0.564 ms  0.579 ms  0.731 ms
 4  192.35.48.26 (192.35.48.26)  3.937 ms  3.910 ms  3.919 ms
 5  te2-1--580.tr01-asbnva01.transitrail.net (137.164.131.177)  4.354 ms  4.567 ms
4.539 ms
 6   (137.164.130.154)  4.261 ms  4.250 ms  4.266 ms
 7  216.239.48.112 (216.239.48.112)  4.956 ms  4.910 ms  4.868 ms
 8  72.14.236.200 (72.14.236.200)  5.094 ms 64.233.175.171 (64.233.175.171)  5.304 ms
64.233.175.169 (64.233.175.169)  5.277 ms
 9  216.239.49.145 (216.239.49.145)  8.488 ms  7.596 ms  7.851 ms
10  yo-in-f103.google.com (64.233.169.103)  5.396 ms  5.478 ms  5.436 ms
```


Note that traceroute can't always identify all of the routers along the way. Firewall rules may prevent some intermediate hosts from responding to traceroute's queries.

Domain Name Servers:

Numbers are hard to remember, so we have Domain Name Servers (DNSs). A DNS is a server that maintains a list of computer names, and their associated numerical IP addresses. These names are organized into a hierarchical system of “Domains”. Name resolution information is kept in the file /etc/resolv.conf:

```
search phys.mydomain.org mydomain.org

nameserver 192.168.200.7
nameserver 192.168.50.7
nameserver 192.168.51.30
```

If I don't specify the full name, try appending these.

List of DNS servers to use.

Most network-aware commands will accept either a numerical address or a hostname. You can manually look up a host's address using the “host” command. You can also do the reverse.

```
~/demo> host www.virginia.edu
www.virginia.edu has address 128.143.22.36

~/demo> host 128.143.22.36
36.22.143.128.in-addr.arpa domain name pointer
www.Virginia.EDU.
```


Underneath a few top-level domains (TLDs) like “.com” and “.edu”, secondary domain names are given out to individuals or institutions, on a first-come-first-serve basis, through domain registrars. Registrars typically charge an annual fee for retaining a domain name. The annual fee for a domain name is typically only a few dollars.

The /etc/hosts File and “localhost”:

You can also maintain your own list of hostnames in the file /etc/hosts:

Address

List of Names

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1       localhost.localdomain   localhost

10.0.0.1 master1.private master1
10.0.0.2 master2.private master2 home.private mail.private

192.168.100.1 server1.cluster server1
192.168.100.2 server2.cluster server2
```

Note that there should always be an entry for the address 127.0.0.1. This is a special, reserved, address that will always refer to the local computer. It's called the “loopback” address. Many programs use the name “localhost” or “localhost.localdomain” to refer to it.

Applications will usually check the /etc/hosts file first, and then go to the DNS servers when trying to resolve a name. The order is configurable in /etc/nsswitch.conf. Type “man nsswitch.conf” for more information.

Part 4: Network Interfaces

A “network interface” is the device inside your computer that is actually plugged into the local network. This may be a card plugged into an expansion slot inside your computer, but typically these days the network interface is built into the computer's motherboard. For most computers, the network interface will be the thing in the back of your computer into which you plug an RJ-45 connector.

The picture above shows an old acoustic coupler, of the type I used when I was in high school in the 1970s. This was an early way of connecting computers together. Data was sent over a phone line as a series of tones, and these were decoded by devices like the on above and turned into digital signals.

Network Interfaces:

To get a list of your computer's network interfaces, use the “ifconfig” command. Normally, you'll see at least two interfaces:

```
[root@demo ~]# ifconfig
eth0  Link encap:Ethernet  HWaddr 00:1A:A0:BF:6B:5F
      inet addr:192.168.100.2  Bcast:192.168.255.255  Mask:255.255.0.0
      inet6 addr: fe80::21a:a0ff:febf:6b5f/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:78617868 errors:0 dropped:0 overruns:0 frame:0
      TX packets:25924911 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:725202911 (691.6 MiB)  TX bytes:1837757879 (1.7 GiB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:10774315 errors:0 dropped:0 overruns:0 frame:0
      TX packets:10774315 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:4152450841 (3.8 GiB)  TX bytes:4152450841 (3.8 GiB)
```

The interface “eth0” is an ethernet interface. Ifconfig shows its MAC address and IP address. The interface called “lo” is the “loopback” interface. It's used when a network-aware program wants to talk to the same computer.

Programs could talk to the local computer through eth0, too, but there are some disadvantages:

- There's more overhead involved. To talk on eth0 the system needs to go through the whole process of composing a message on the network, sending it out on the network, then hearing it and interpreting it.

- There may be firewall rules applied to eth0 that you don't want to apply to purely internal communications.

Configuring a Network Interface:

You can also use ifconfig to configure network interfaces. For example, to assign the IP address "192.168.100.2" to eth0, you could use a command like:

```
ifconfig eth0 192.168.100.2 netmask 255.255.0.0 broadcast 192.168.0.0
```

Interface IP address Network mask Broadcast address

- The "network mask" specifies what part of the IP address specifies the local network. It's used to determine whether a given IP address is part of the local network or a remote network. Traffic to remote networks may need to be sent to a gateway computer that knows how to deliver it.

- The "broadcast address" specifies the address to which broadcast messages should be sent. Broadcast messages are messages that should be heard by all hosts on the local network.

You can actually have more than one IP address on a single interface. You can use ifconfig to add extra addresses to an interface by specifying "aliases" for the interface. These have names like "eth0:0", "eth0:1", "eth0:2", etc. Once you've configured the interface with its first address, as above, you can add others like this:

```
ifconfig eth0:0 192.168.100.50
ifconfig eth0:1 192.168.100.38
ifconfig eth0:2 192.168.100.5
```

The “route” Command:

Each computer maintains a “routing table” containing information about where to send packets destined for various networks. You can view or manipulate the routing table with the “route” command:

```
[root@demo ~]# route
Kernel IP routing table
Destination Gateway      Genmask     Flags Metric Ref Use Iface
192.168.0.0 *            255.255.0.0 U     0      0   0   eth0
default     gw.mydom.org 0.0.0.0     UG    0      0   0   eth0
```

The two routing rules above say:

1. Traffic for addresses beginning with 192.168 should be transmitted to the recipient directly on interface eth0.

2. Traffic for other addresses should be sent to the gateway “gw.mydom.org” through the interface eth0.

This table could be created by issuing the following commands:

```
[root@demo ~]# route add -net 192.168.0.0 netmask 255.255.0.0 dev eth0
[root@demo ~]# route add default gw gw.mydom.org
```

Static versus Dynamically-assigned IP Addresses:

How do you know what IP address to assign to your network interface? Your ISP can provide you with an IP address in two ways:

1. Static IP Addresses:
If your computer always needs to have the same IP address, your ISP may assign a "fixed" or "static" IP address to you. The ISP will keep a list of static IP address assignments, so that they can be sure they don't assign the same address more than once. The ISP will also probably assign the computer a name, and associate this name with the computer's static address in their DNS servers.

2. Dynamic IP Addresses:
If your computer doesn't need a fixed address, you can obtain a randomly-assigned address through a process called "Dynamic Host Configuration Protocol" (DHCP). Your ISP probably maintains a DHCP server that will, on demand, provide your computer with an IP address and other configuration information, such as a list of DNS servers.

Static addresses are usually appropriate for servers. Dynamic addresses are appropriate for most personal computers.

There are many tools for querying DHCP servers and obtaining an IP address. Some of the common ones are "dhclient" (used in Red Hat-derived distributions and Ubuntu), "pump" (used by KNOPPIX), and "dhcpcd" (not to be confused with "dhcpd" -- one's a client and the other's a server!). All of these will query the DHCP server, obtain an IP address and configure the interface for you.

The syntax for dhclient is just "dhclient eth0".

Network Configuration Files:

You can use the ifconfig and route commands directly, but usually Linux distributions provide scripts, configuration files and graphical tools for configuring your network interfaces. The details will depend on your Linux distribution.

Red Hat, Fedora, CentOS:

Configuration files for each interface live in the directory /etc/sysconfig/network-scripts, with file names like "ifcfg-eth0". The files look like this:

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp
```

These distributions also provide a graphical too for configuring network interfaces. It can be invoked by typing the command "system-config-network".

Ubuntu:

The configuration file for all interfaces is "/etc/network/interfaces". It looks like this:

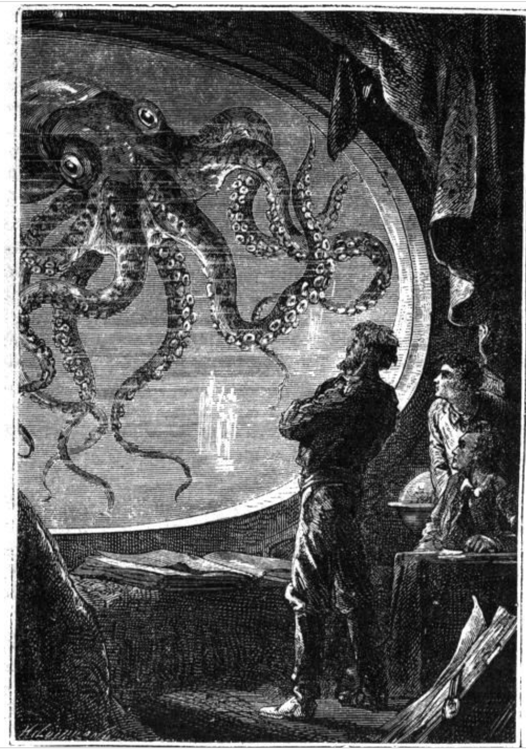
```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
```

There's also a graphical tool for manually configuring network interfaces. It can be invoked by typing "network-admin".

BUT WAIT! By default, Ubuntu uses a tool called NetworkManager that attempts to dynamically and automatically configure all of your network interfaces. You should only fiddle with the network configuration by hand if NetworkManager fails.

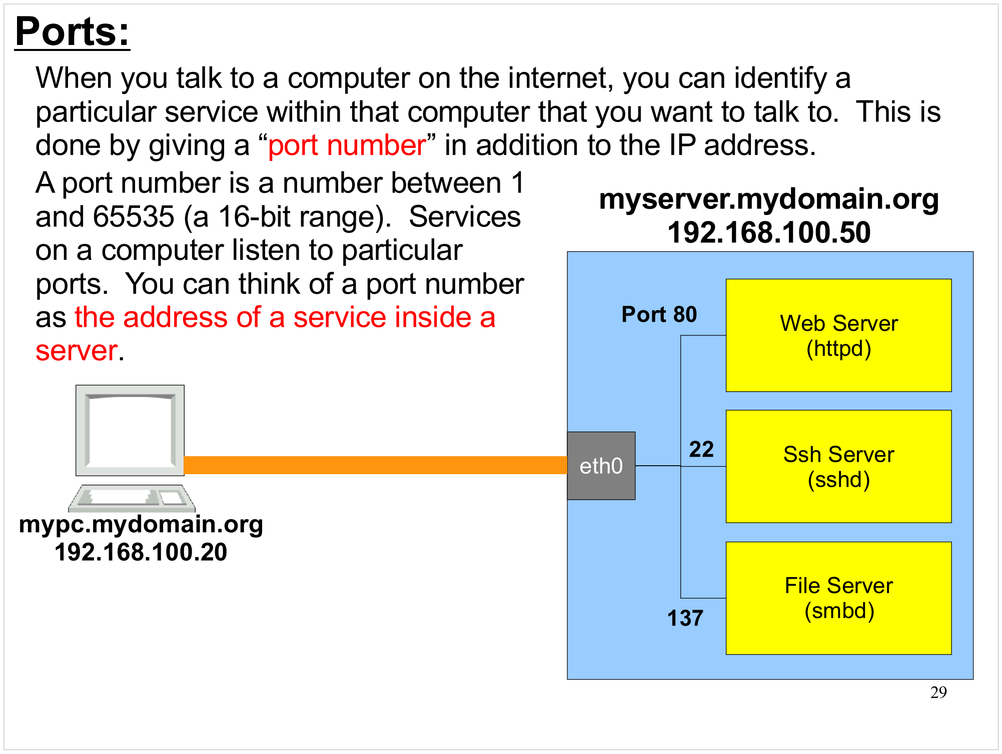
Most distributions are moving toward using NetworkManager to manage network connections.

Part 5: Ports



28

We know how to contact a remote computer, using its IP address, but what about contacting a particular service running on that remote computer? For that, we'll need to introduce an internal address called a “port number”. Sometimes people are confused by the name “port”. It sounds like some physical thing that you would plug a cable into. In the following, remember that ports are just addresses. They're not anything physical.



A port number is actually used at both ends of the connection. In the illustration above, the traffic leaving "mypc.mydomain.com" would originate from a particular source port on that computer. If the computer were browsing the web, for example, the traffic would originate from some randomly-assigned port on mypc, and would be address to destination port 80, on the remote computer, where the web server would be listening.

When an application requests a port on the local computer, it can ask for port "0". In this special case, the computer picks a port at random, from a pool of ports available for that purpose, and gives it to the application.

Port Number Assignment:

The IANA maintains an official list of standard port numbers for various services. This list is purely advisory, but software authors seldom use ports in non-standard ways. The list of ports is divided into three sections:

• **Well-Known Ports (aka Privileged Ports):**
Ports 1-1023 are called "Well-Known" ports, and many of them have been in common use since the beginning of the Internet. These are the ports used by familiar services such as web, ftp, ssh, telnet and smb. Only processes owned by the root user are allowed to bind to these ports.

• **Registered Ports:**
Ports 1024-49151 are "Registered" ports. These are associated with applications that have registered with IANA and been assigned an official port number. Registering a port makes it less likely that someone else will use that port for another purpose.

• **Dynamic/Private/Ephemeral ports:**
Ports 49152-65535 are available for temporary use, or for private use.

Some Common Ports:

Here are a few commonly-used ports:

- 22: Ssh
- 80: Http
- 443: Https
- 25: Smtp
- 20/21: Ftp
- 53: DNS
- 110: Pop3
- 143: Imap
- 389: Ldap
- 993: Imaps

You don't need to remember the numbers. You can usually refer to them by name. Most applications will look up ports by name, using the file "/etc/services". This file contains a list of port names, their associated numbers, and other information.

The TCP and UDP Protocols:

IP packets carry no information about port numbers. To use ports, we need to introduce new protocols that can be layered on top of IP. The two most common ones are "Transmission Control Protocol" (TCP) and "User Datagram Protocol" (UDP).

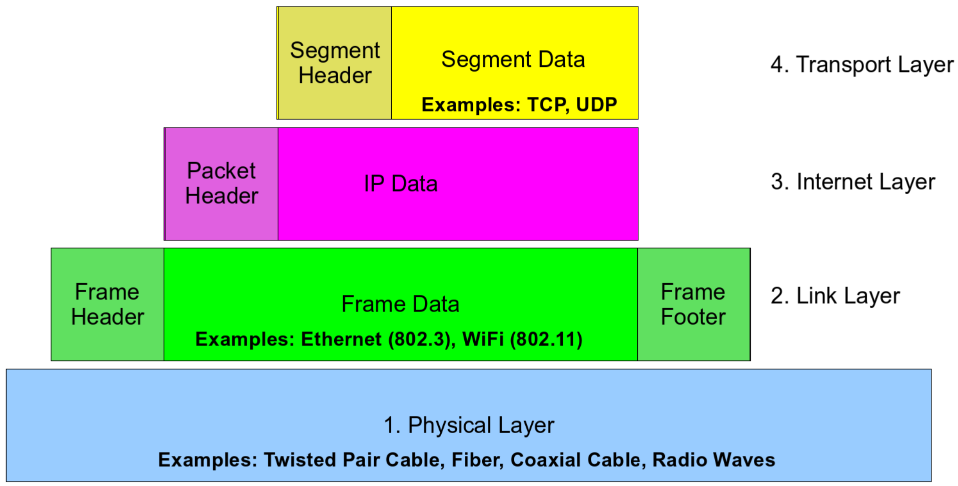
Segment/Packet Header:	Segment/Packet Data:
• Source Port • Destination Port ...etc.	

Units of TCP traffic are called "segments" and units of UDP data are called "packets". UDP is a very simple protocol, but TCP is quite complex. TCP tries very hard to maintain a reliable connection between two hosts. It keeps track of which segments are delivered, allowing a host to request re-transmission of missed segments, and it can can reassemble sequences of packets (which may be scrambled during transmission) into their original order.

Because it's so much more complex, TCP also has a lot more overhead. In situations where dropped packets can be tolerated, or where the order of packets doesn't matter, UDP is used instead.

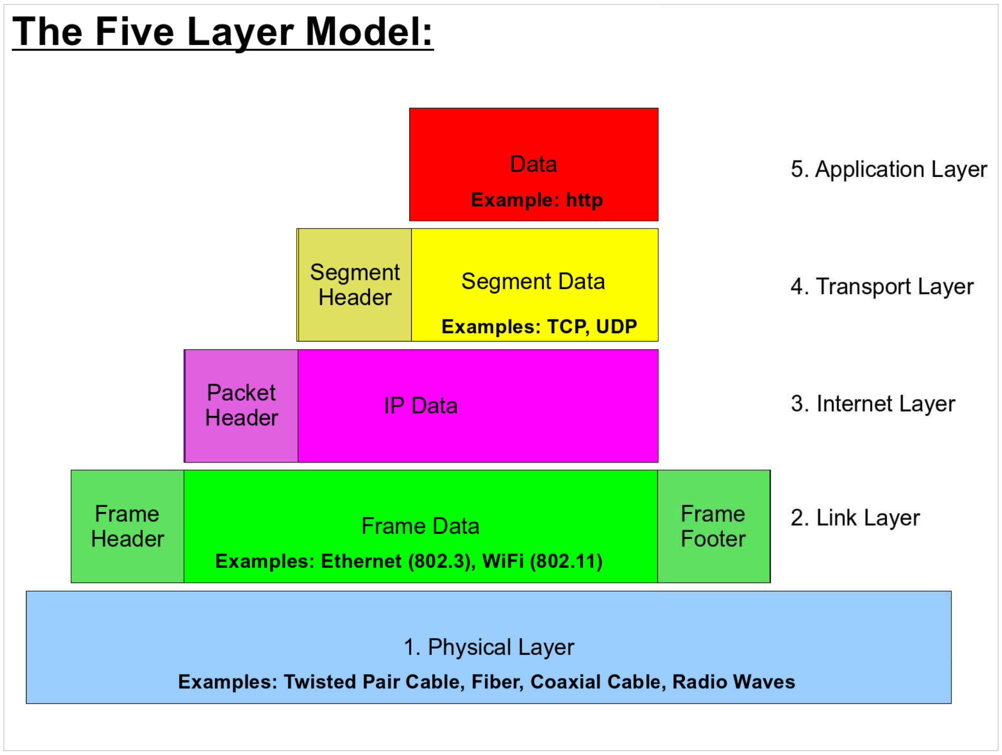
The Transport Layer:

Protocols like TCP and UDP form yet another layer (the “Transport” layer) on top of the Internet layer.



So now we have four layers in our layer-cake. On top of the physical layer, Ethernet frames carry data from computer to computer, guided by MAC addresses. Within those frames are IP packets, guided by IP addresses. Inside the IP packets are TCP or UDP segments/packets, addressed by port number.

And to finish our cake.....



...we add one more layer: the application layer, where protocols like http live. These layers make up what's called the TCP/IP Five Layer Model of networking.

The Five Layer Model is a simplified version of a more general model of networking called the OSI Seven Layer Model, which you may have heard of. The Five layer model gives a more intuitive picture of how the most common type of networking works.

Part 6: Monitoring Traffic

Now let's look at some tools for monitoring network activity.

The “netstat” Command:

The “netstat” command shows information about network connections to your computer. It shows the source and destination address and port for each connection, and it can be made to show the process ID and process name of the processes that are bound to these ports.

```
[root@demo ~]# netstat -anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address     Foreign Address      State       PID/Program
name
tcp        0      0 0.0.0.0:750       0.0.0.0:*            LISTEN      354/rpc.statd
tcp        0      0 0.0.0.0:111       0.0.0.0:*            LISTEN      3507/portmap
tcp        0      0 0.0.0.0:6000      0.0.0.0:*            LISTEN      13034/X
tcp        0      0 127.0.0.1:631     0.0.0.0:*            LISTEN      5858/cupsd
tcp        0      0 0.0.0.0:3551      0.0.0.0:*            LISTEN      8305/apcupsd
tcp        0      0 10.2.1.43:37218   10.2.2.108:22        ESTABLISHED 7491/ssh
tcp        0      0 10.2.1.43:25      10.9.3.3:50071       TIME_WAIT   -
tcp        0      0 10.2.1.43:38860   10.2.1.159:22        ESTABLISHED 5581/ssh
tcp        0      0 10.2.1.43:54874   10.2.2.107:22        ESTABLISHED 25409/ssh
tcp        0      0 10.2.1.43:57525   10.2.1.57:2200       ESTABLISHED 27818/ssh
tcp        0      0 127.0.0.1:39788   127.0.0.1:783        TIME_WAIT   -
tcp        0      0 10.2.1.43:47548   128.143.100.51:22    ESTABLISHED 11350/ssh
tcp        0      0 10.2.1.43:42177   10.2.1.44:22         ESTABLISHED 15294/ssh
tcp        0      0 10.2.1.43:25      10.2.1.105:53651     TIME_WAIT   -
tcp        0      0 127.0.0.1:49912   127.0.0.1:22         ESTABLISHED 28866/ssh
tcp        0      0 10.2.1.43:37956   10.2.1.114:22        ESTABLISHED 7362/ssh
tcp        0      0 10.2.1.43:47173   10.2.1.113:22        ESTABLISHED 7405/ssh
tcp        0      0 127.0.0.1:60554   127.0.0.1:22         ESTABLISHED 26185/ssh
...etc.
```


Netstat will also show you information about “Unix domain sockets”. Ignore this for now. These are local connections between processes running on your computer.

The switches shown above are:

-a Show all connections, including servers that are just listening.

-n Don't resolve host or port names. Just show the numbers.

-p Show the process IDs and process names.

Viewing Connections with "lsof":

You can also view network connections with "lsof". The "-i" switch will show all network connections or, if followed by a port name or a port number, will show only the connections to or from that port.

```
[root@demo ~]# lsof -i :ssh

COMMAND   PID  USER FD   TYPE   DEVICE NODE NAME
ssh      5581 bkw1a  3u  IPv4 16201308 TCP mypc.mydom.org:38860->print.mydom.org:ssh
sshd     5872  root  3u  IPv6    13485 TCP *:ssh (LISTEN)
ssh      7362 bkw1a  3u  IPv4 15108847 TCP mypc.mydom.org:37956->data.mydom.org:ssh
ssh      7405 bkw1a  3u  IPv4 15109181 TCP mypc.mydom.org:47173->tracking.mydom.org:ssh
ssh      7491 bkw1a  3u  IPv4 15109863 TCP mypc.mydom.org:37218->memory.mydom.org:ssh
ssh     11350 bkw1a  3u  IPv4 17186056 TCP mypc.mydom.org:47548->test.mydom.org:ssh
ssh     15294 bkw1a  3u  IPv4 15137397 TCP mypc.mydom.org:42177->test2.mydom.org:ssh
ssh     25409 bkw1a  3u  IPv4 15883849 TCP mypc.mydom.org:54874->blarg.mydom.org:ssh
ssh     26185    nx  7u  IPv4  6782492 TCP localhost.localdomain:60554->localhost:ssh
sshd    26190  root  3u  IPv6  6782493 TCP localhost:ssh->localhost.localdomain:60554
sshd    26192 elvis  3u  IPv6  6782493 TCP localhost.localdomain:ssh->localhost:60554
```

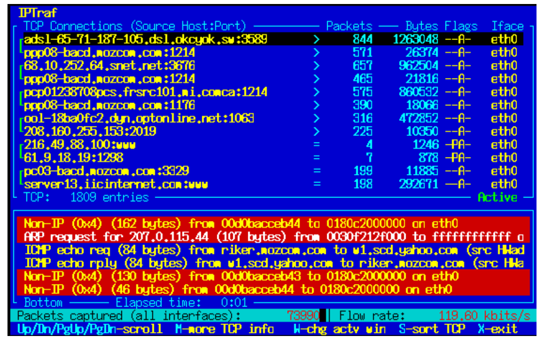

lsof is a tool for seeing which files were currently being used by various processes. It can also show us network connections being used by processes.

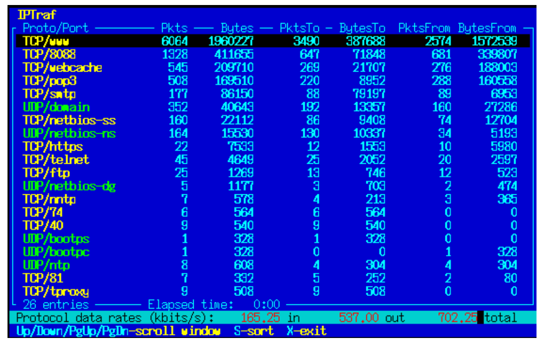
Using “iptraf” to Monitor Traffic in Real Time:

You can use the “iptraf” command to monitor network traffic in real time. Iptraf is menu-driven, and has several modes.

In the mode shown at top, it will show information about each new connection in the top pane, and a running stream of information about incoming packets in the bottom pane.

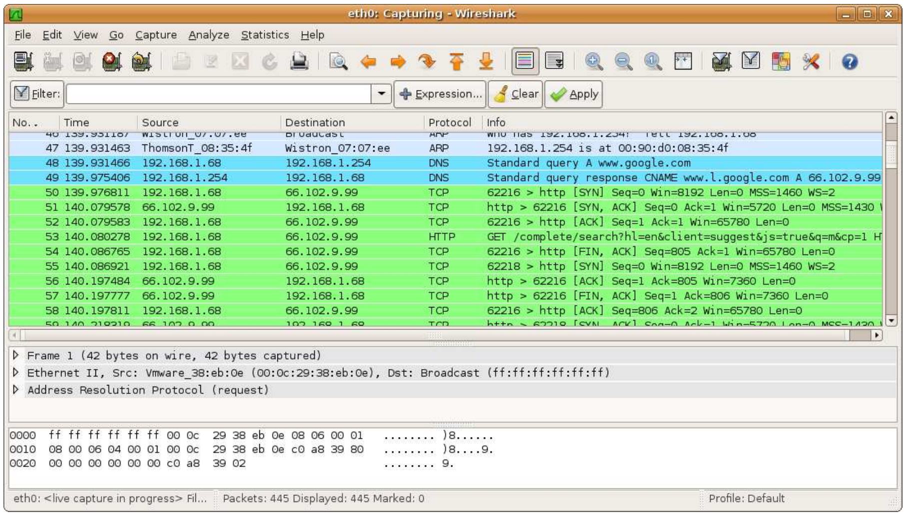
In the mode shown below, iptraf gathers statistics about traffic on each port.



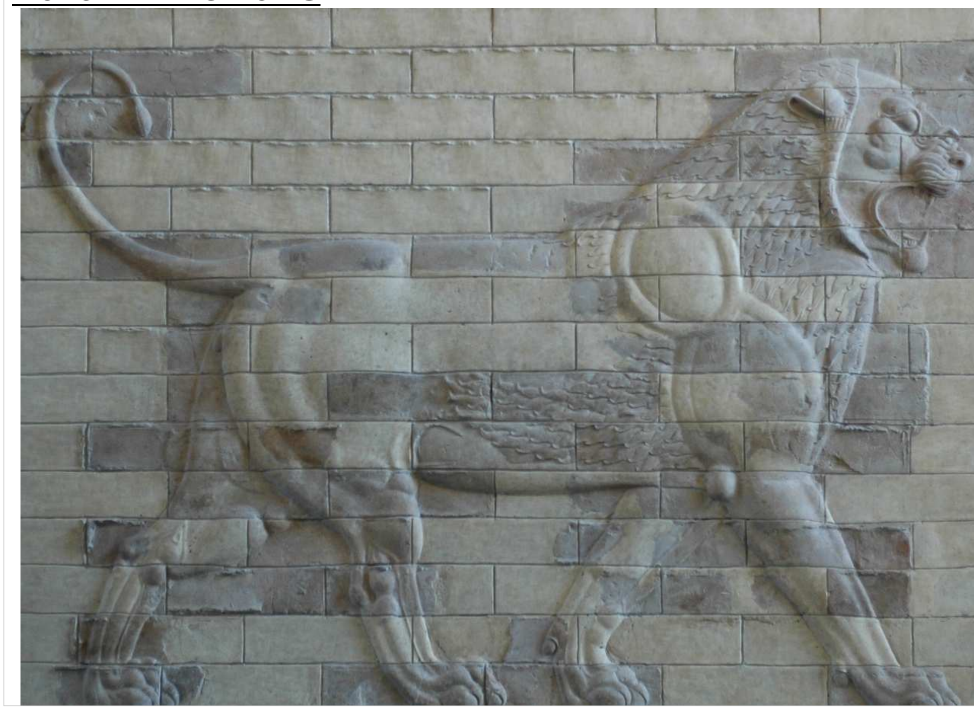


Using “wireshark” to Monitor Traffic in Real Time:

Wireshark is an invaluable tool for analyzing network traffic. It allows you to capture (optionally filtered) traffic, dissect it, do offline filtering, and produce graphs and statistics.



Part 7: Firewalls



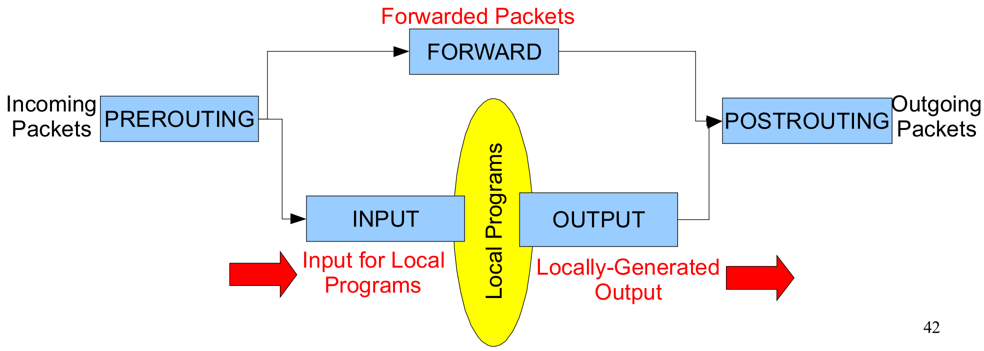
In the terminology we'll use today, a firewall is anything that blocks or modifies network traffic. Most desktop computers today have some sort of firewall capability. They can, for example, selectively block incoming IP packets.

Even if your computer is behind a department firewall, or is running other security software, it's very important to have a properly-configured local firewall on your computer. This reflects a security philosophy called “defense-in-depth”, which says that you need multiple layers of defense. Multiple layers provide redundancy, in case one layer fails, and they tend to fill in the gaps in each other's coverage.

Netfilter:

Built into the Linux kernel is a system called “Netfilter” that allows for monitoring, modifying or blocking IP packets as they pass through the kernel, based on packet header information. Netfilter associates user-defined functions with pre-defined “hooks” at various points along a packet's path through the kernel. These functions are managed by programs like “iptables”.

The diagram below shows some of the available hooks, in blue:



Netfilter is just a framework within the kernel. To use it, you need a program like iptables.

The input and output hooks let you filter traffic going to or coming out of local program. The forward hook allows you to filter network traffic that's just passing through your computer. The prerouting and postrouting hooks allow you to do things like re-writing the address on incoming/outgoing packets.

Tables, Chains and Rules in iptables:

Iptables binds a set of functions to the Netfilter hooks. These functions use lists of rules (called "chains") to decide what to do with a packet as it passed through each hook. The chains are organized in tables, such as:

"filter" Table:
- INPUT Chain
- OUTPUT Chain
- FORWARD Chain

...etc.

"nat" Table:
- PREROUTING
- OUTPUT
- POSTROUTING

...etc.

"mangle" Table:
- PREROUTING
- INPUT
- OUTPUT
- POSTROUTING

...etc.

"raw" Table:
- PREROUTING
- OUTPUT
- POSTROUTING

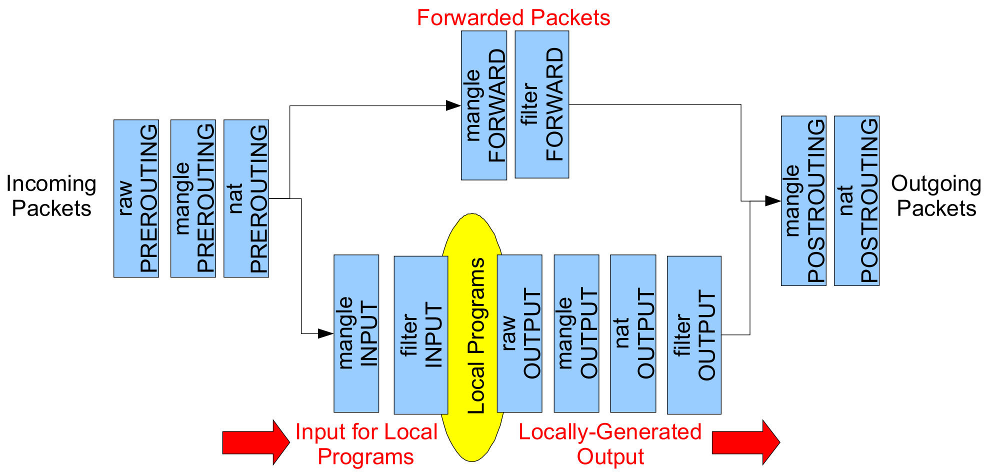
...etc.

The list of tables is hard-coded into iptables, but chains can be added by the user, through the "iptables" command. Each table starts with a set of built-in, empty, chains. The built-in chains are used directly by the functions iptables binds to the Netfilter hooks.

Note that the names of tables and chains are case-sensitive.

How iptables Chains are Bound to Netfilter Hooks:

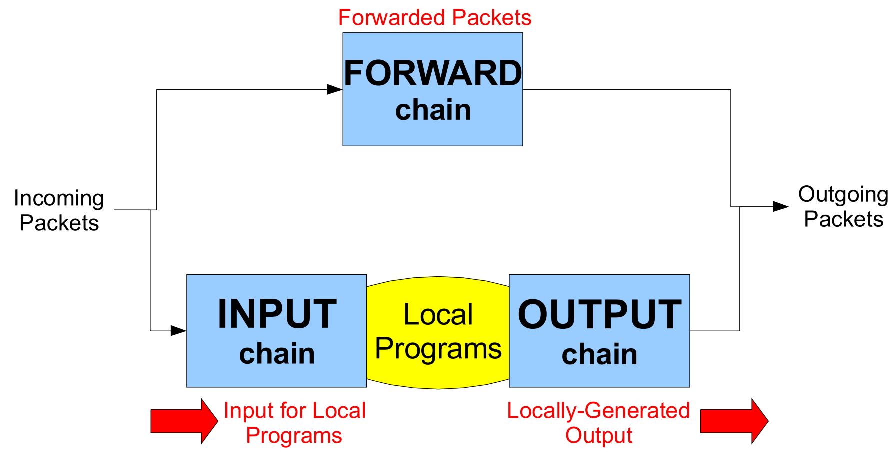
Here's how iptables binds various built-in chains to the Netfilter hooks:



This shows where the iptables chains from the previous slide plug into the hooks provided by Netfilter.

The “filter” Table:

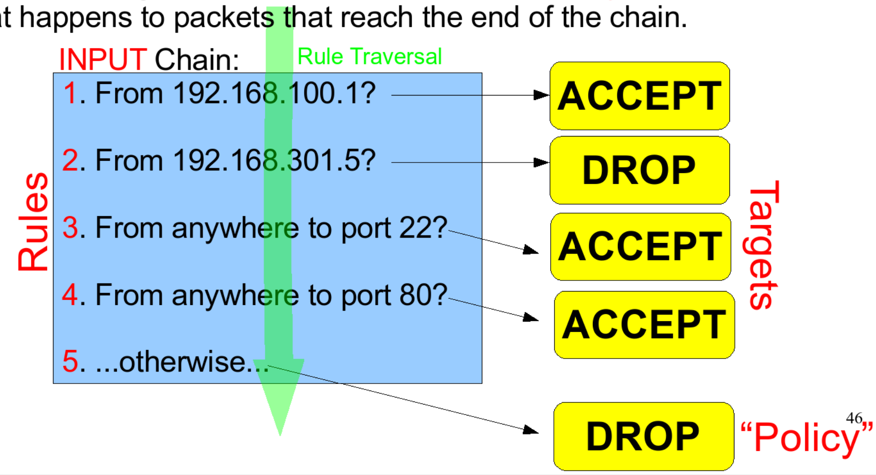
The most often-used table is the “filter” table, which initially contains built-in chains called “INPUT”, “OUTPUT” and “FILTER”. These chains of rules are used by functions plugged into the Netfilter hooks shown below:



These built-in chains are directly connected to the Netfilter hooks. As we'll see, you can also create user-defined chains, but they're used indirectly.

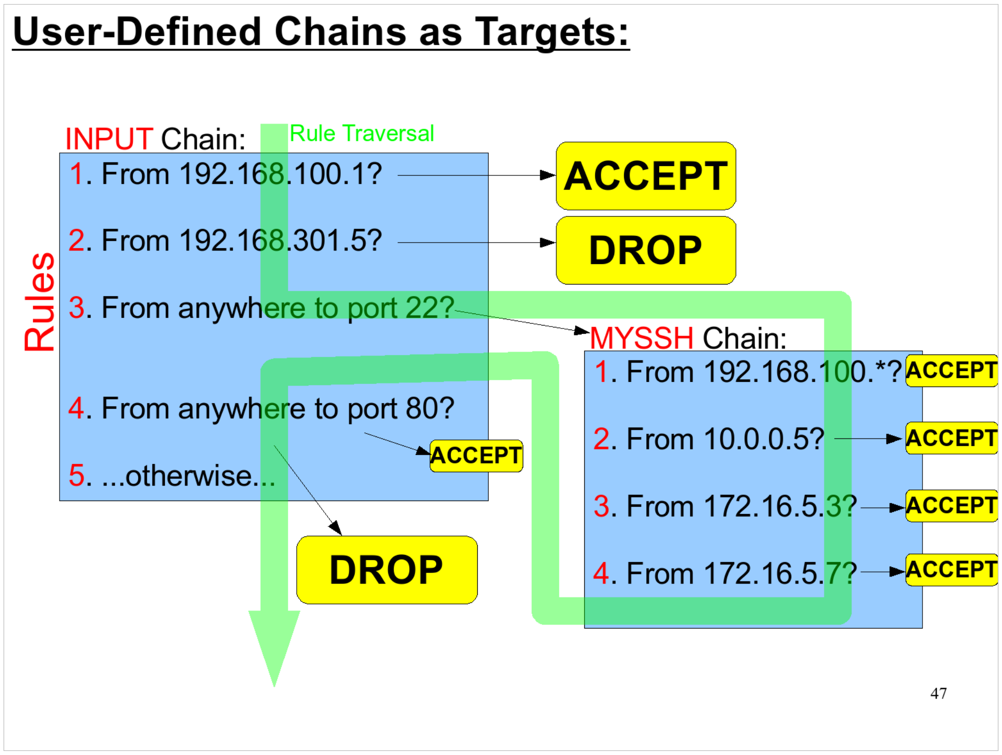
Iptables Chains:

Each iptables chain is a list of rules. Each rule consists of a test and a "target". The target can either be selected from a list of built-in targets (e.g., "ACCEPT" or "DROP"), or it can point to a different, user-defined, chain of other rules. The rules in the chain are processed ("traversed") from top to bottom, like a program. Some targets (like "DROP") will cause the "program" to halt. Others (like "LOG") will allow the "program" to keep running. Built-in chains each have a "policy" that determines what happens to packets that reach the end of the chain.



Targets that cause rule traversal to stop are called "terminating" targets. Those that don't are called "non-terminating" targets.

Only built-in chains have policies. The built-in chains are the ones that are directly attached to Netfilter's hooks. User-defined chains are always called by one of the built-in chains.



Here, the INPUT chain is a built-in chain, and the MYSSH chain is user-defined.

Some iptables Targets:

Here are some examples of built-in iptables targets:

ACCEPT	Stop traversal, allow the packet to continue.
DROP	Stop traversal, ignore the packet.
REJECT	Stop traversal, ignore the packet, but notify the sender.
LOG	Log the packet, then continue traversal.
TARPIT	Wait forever without responding to sender (TCP only).

...etc.

Viewing Chains:

You can look at the the current chains by using the "iptables -L -v" command. By default, this will show you the chains in the "filter" table. You can look at other tables by adding the "-t" switch (e.g., "-t nat"). This is what the "filter" table looks like by default. Three built-in chains are defined, but the chains are empty of rules:

```
[root@demo ~]# iptables -L -v
Chain INPUT (policy ACCEPT 16 packets, 1274 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 8 packets, 1088 bytes)
 pkts bytes target     prot opt in     out     source    destination
```


You can also just use "iptables -L", but if you have non-trivial firewall rules you'll find that the output is misleading. For one thing, "iptables -L" doesn't tell you which network interfaces a given rule applies to.

Adding Rules to a Chain:

- Accept all incoming traffic destined for port 80 on the local computer:

```
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

Add a rule... | to this chain. | Packets using this protocol... | and destined for this port... | jump to this target.

- Adding a rule to match packets from a given host, destined for port 22 on the local computer:

```
iptables -A INPUT -s 1.2.3.4 -p tcp --dport 22 -j ACCEPT
```

Add a rule... | to this chain. | Packets from this source... | using this protocol... | and destined for this port... | jump to this target.

- Ignore all incoming traffic from a particular computer:

```
iptables -A INPUT -s 4.3.2.1 -j DROP
```

Add a rule... | to this chain. | Packets from this source... | jump to this target.

A few simple examples. Later, we'll see how to define firewall rules automatically at boot time.

More Rule Examples:

- Setting a default policy:

```
iptables -P INPUT DROP
```

Apply this Policy... to this chain. The policy.

- Adding a rule that only applies to one network interface:

```
iptables -A INPUT -i eth1 -p tcp --dport 22 -j ACCEPT
```

Add a rule... to this chain. Packets from this interface... using this protocol... and destined for this port... jump to this target.

- Adding a rule that allows incoming traffic that is associated with an already-established outgoing connection:

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

Add a rule... to this chain. Load the “state” module. If the packet is related to an established connection... jump to this target.

You'll find many more examples here: http://danieldegraaf.afraid.org/info/iptables/examples

As you can see from the last example, iptables can be extended through “modules”. Many of these modules are already installed in most Linux distributions. These make iptables very powerful. You can, for example, do rate limiting, or limit the number of connections from a given host. You can filter by MAC address. You can select every n^th^ packet (!). You can assign tags to packets for use in later rules. You can filter packets based on their length. You can even match strings within packets.

Minimal Firewall Rules:

Here's a set of minimal firewall rules. They allow anything to go out, but only allow incoming packets that are associated with an already-established outgoing connection. Everything else is dropped.

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP
```

```
[root@demo ~]# iptables -L -v
Chain INPUT (policy DROP 36 packets, 5000 bytes)
 pkts bytes target     prot opt in     out     source    destination
   60  3644 ACCEPT     all  --  any    any     anywhere  anywhere  state RELATED,ESTABLISHED
    0     0 ACCEPT     all  --  lo     any     anywhere  anywhere

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

Chain OUTPUT (policy ACCEPT 33 packets, 4564 bytes)
 pkts bytes target     prot opt in     out     source    destination
```


This is similar to the default firewall rules you'll find under Red Hat/Fedora/CentOS, or in any home internet router/firewall.

The “iptables-save” and “iptables-restore” Tools:

The firewall rules you create with iptables are volatile. They won't automatically be restored the next time you restart your computer, unless you take steps to restore them. One mechanism for doing this is the “iptables-save” and “iptables-restore” commands. If you've configured a set of firewall rules and want to save that configuration, issue a command like:

```
[ root@demo ~]# iptables-save > myfirewall.conf
```

Then you can restore these rules later by typing:

```
[ root@demo ~]# iptables-restore < myfirewall.conf
```

The output of iptables-save is just text, and can be edited with any text editor. It looks like this:

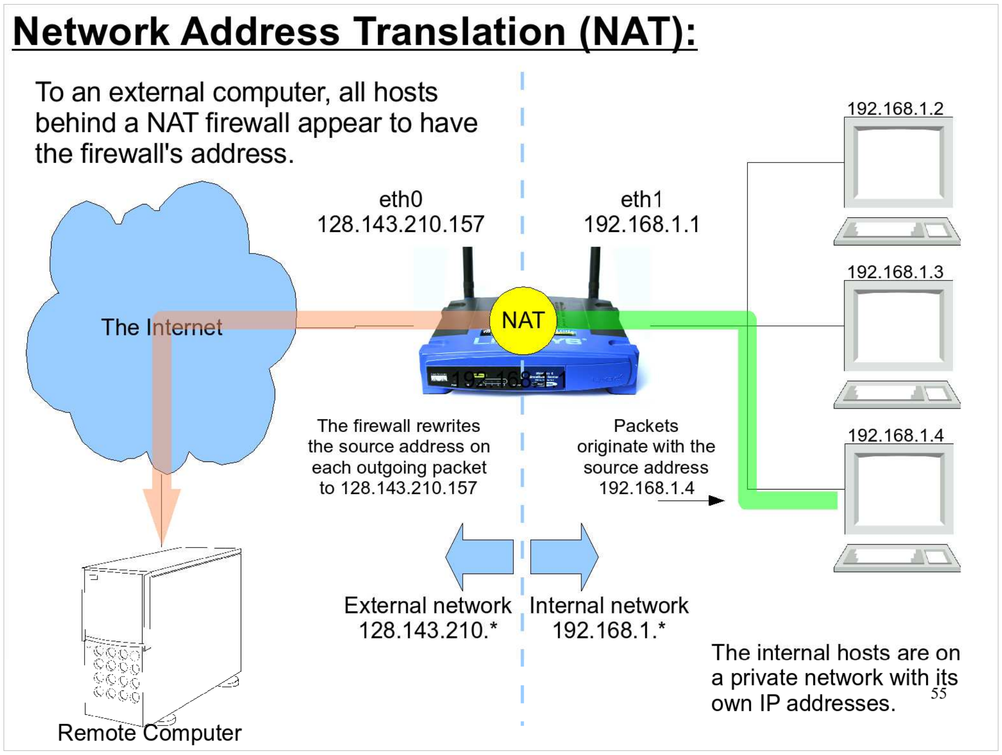
```
# Generated by iptables-save v1.3.5 on Tue Mar  3 14:38:46
2009
*filter
:INPUT DROP [18:2119]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [28:2832]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
COMMIT
# Completed on Tue Mar  3 14:38:46 2009
```

Iptables Configuration Files:

On Red Hat/Fedora/CentOS distributions a minimal set of firewall rules is enabled by default. These rules are in the same format that iptables-save produces, and are stored in the file /etc/sysconfig/iptables. At boot time, this file is automatically read by iptables-restore to set up the firewall rules.

Ubuntu distributions don't have firewall rules enabled by default, and don't use /etc/sysconfig/iptables, but recent versions of the Ubuntu distribution include a front-end to iptables called “ufw”. The ufw program stores its configuration in /var/lib/ufw/user.rules.

See https://wiki.ubuntu.com/UbuntuFirewall for more information about ufw.



Inexpensive home routers use NAT to connect computers in your home to the Internet. Many of these routers are actually running Linux, and use iptables, just as you'd use it on your desktop computer or a Linux server.

Setting up NAT Using iptables:

You can use iptables to configure a Linux computer with two network interfaces to perform network address translation. (Indeed, many home routers are small Linux computers configured in this way.) Here's a set of iptables commands to do that. In this example, eth0 is on the external (public) network and eth1 is on the internal (private) network:

```
iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth1 -j ACCEPT

iptables -A INPUT -i eth1 -j ACCEPT

iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

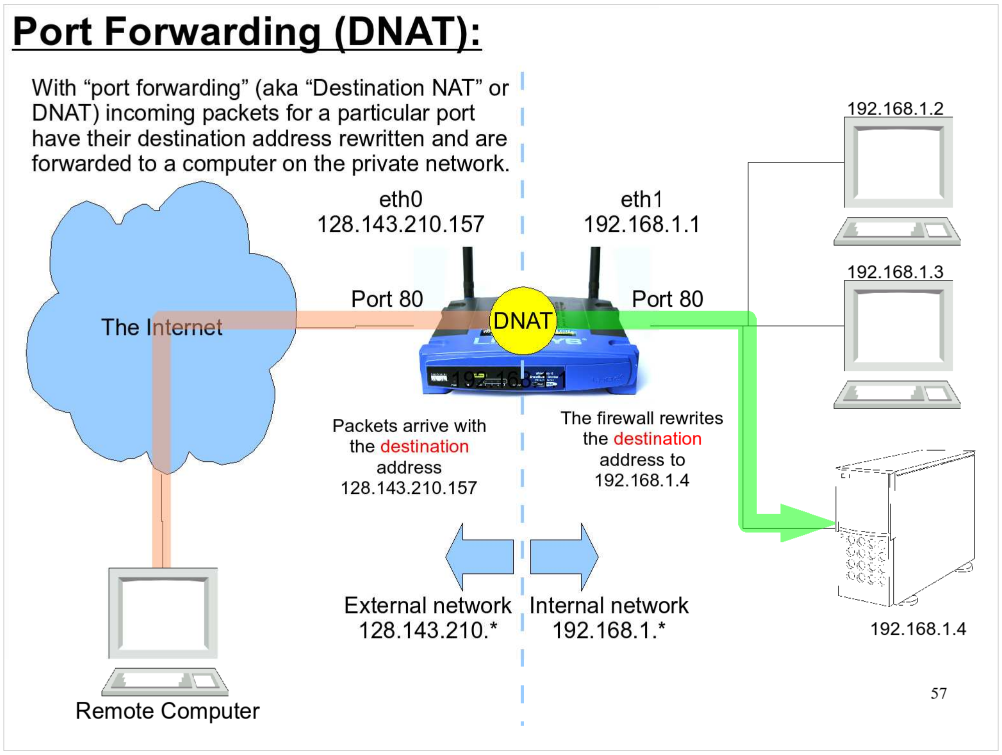
You can use the “netstat-nat” command to monitor NATed connections:

```
Proto NATed Address         Foreign Address                State
tcp   192.168.1.3:53094   balrog-e.psi.ch:ssh            ESTABLISHED
tcp   192.168.1.7:56063   lm4.license.Virginia.EDU:16286 TIME_WAIT
tcp   192.168.1.4:56065   lm4.license.Virginia.EDU:16286 TIME_WAIT
udp   192.168.1.4:ntp     dns1.unix.Virginia.EDU:ntp     UNREPLIED
```


The netstat-nat command is similar to the netstat command we looked at earlier, except that it shows you information about NATed connections passing through your computer.

This type of NAT is also called “source NAT”, or SNAT, since it re-writes the address of the source computer. As we'll see, there's also “destination NAT” or DNAT.

iptables actually has two possible targets for source NAT. The one shown above, MASQUERADE, is appropriate for devices that have variable IP addresses, supplied by a DHCP server. The other target is SNAT, which is more appropriate for hosts with fixed IP addresses. See the iptables man page for more information about the differences between the two.



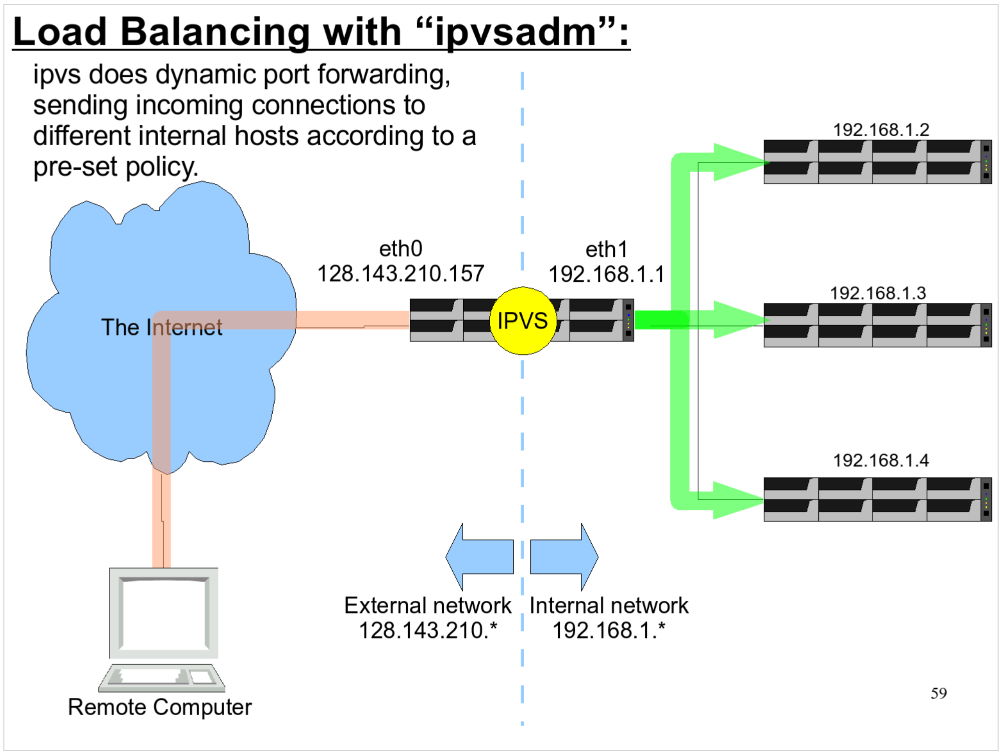
You could use port forwarding to connect a home web server to the Internet, for example. The details of how to do this will depend on the particular network hardware you have at home. In general, you'll need to connect to your router or DSL modem (or both) through these devices' web interfaces and configure NAT appropriately. If you have a DSL modem and a router, you may need to tell the DSL modem to forward packets to the router, and then tell the router to forward packets to your internal server. Documentation for most of these devices can be found on the web.

Setting up Port Forwarding Using iptables:

Port forwarding can also be done with the rules in the “nat” table. Again, eth0 is on the external (public) network. The host 192.168.1.4 is a web server on the internal (private) network. The rule below forwards incoming traffic bound for port 80 (the standard port for web traffic) to the internal host.

```
iptables -t nat -A PREROUTING -i eth0 -p tcp \
--dport 80 -j DNAT --to-destination 192.168.1.4
```

Here we see an iptables target (DNAT) that requires an argument. In this case, we need to specify the address of the internal computer to which we want to send the packets.



If you were running an Internet business and you expected a lot of traffic on your web servers, you might want to be able to spread the traffic around, so that the load is handled by several web servers. IPVS is one way of doing this.

IPVS doesn't use iptables, but they both use the underlying Netfilter framework.

Using ipvsadm to Set Up Load Balancing:

Ipvsadm is different from iptables, although they both use Netfilter as a backend. Ipvsadm lets you create a “Virtual Server” that actually corresponds to a cluster of many real computers. Incoming connections for this virtual server will be forwarded to one of the real computers based on a predetermined policy (“scheduling method”). Here's an ipvsadm configuration for a cluster of six web servers:

First, add the service:

```
ipvsadm -A -t 128.143.210.157:http -s wlc
```

Add this service. TCP Host Port Scheduling method: “weighted least-connections”

Then, add the servers:

```
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.2:http -m -w 1
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.3:http -m -w 1
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.4:http -m -w 1
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.5:http -m -w 1
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.6:http -m -w 1
ipvsadm -a -t 128.143.210.157:http -r 192.168.1.7:http -m -w 1
```

Add this server... to this service. Real server. Use Masquerading. Weight=1

Like iptables, there are ipvsadm-save and ipvsadm-restore commands to save and restore an ipvsadm configuration. In the Red Hat/Fedora/CentOS world, the file /etc/sysconfig/ipvsadm will automatically be used to configure ipvsadm at boot time if the ipvsadm service is turned on.

Available scheduling methods include round-robin, fixed target based on source address, and many others in addition to the wlc method shown above.

Using “fail2ban”:

One of the most common types of malicious activity on the Internet is the “brute-force ssh attack”. In these attacks, Bad Guys use automated tools to try logging into your computer by ssh. They use a dictionary of common usernames and passwords, and they may make thousands of login attempts. In the best case, this uses some of your computer's resources. In the worst case, they stumble upon a valid username/password combination and gain access to your computer.

One of the best tools for dealing with these attacks is “fail2ban”. Fail2ban looks for groups of unsuccessful login attempts and automatically blocks the attacking machine, using iptables firewall rules. Fail2ban remembers which hosts are blocked, and automatically unblocks them after some timeout period.

/var/log/fail2ban.log:

```
2009-03-03 10:28:31,776 fail2ban.actions: WARNING [ssh-iptables] Ban 85.233.64.178
2009-03-03 10:38:31,986 fail2ban.actions: WARNING [ssh-iptables] Unban 85.233.64.178
2009-03-03 13:31:18,984 fail2ban.actions: WARNING [ssh-iptables] Ban 195.14.29.12
2009-03-03 13:41:19,264 fail2ban.actions: WARNING [ssh-iptables] Unban 195.14.29.12
2009-03-03 13:45:47,325 fail2ban.actions: WARNING [ssh-iptables] Ban 195.14.29.12
2009-03-03 13:55:47,555 fail2ban.actions: WARNING [ssh-iptables] Unban 195.14.29.12
2009-03-04 06:49:17,178 fail2ban.actions: WARNING [ssh-iptables] Ban 116.7.255.86
2009-03-04 06:59:17,421 fail2ban.actions: WARNING [ssh-iptables] Unban 116.7.255.86
2009-03-04 08:35:42,481 fail2ban.actions: WARNING [ssh-iptables] Ban 122.9.63.150
2009-03-04 08:45:42,623 fail2ban.actions: WARNING [ssh-iptables] Unban 122.9.63.150
```

Arno's Iptables Firewall:

"Arno's Iptables Firewall" (AIF) is a script to help automate the creation of a complex set of firewall rules. The script reads a configuration file that describes, at a high level, the layout of the desired firewall. The configuration is usually "/etc/arno-iptables-firewall/firewall.conf".

AIF can be used for even trivial firewalls, but it's invaluable for setting up complex firewalls with multiple network interfaces, NAT, forwarding, etc.

Here's a tiny section of the firewall rules produced by AIF for a non-trivial configuration:

```
...
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,PSH,URG -m limit --limit 3/min -j LOG --log-prefix "Stealth XMAS scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,ACK,URG -m limit --limit 3/min -j LOG --log-prefix "Stealth XMAS-PSH scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,PSH,ACK,URG -m limit --limit 3/min -j LOG --log-prefix "Stealth XMAS-ALL scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN -m limit --limit 3/min -j LOG --log-prefix "Stealth FIN scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags SYN,RST SYN,RST -m limit --limit 3/min -j LOG --log-prefix "Stealth SYN/RST scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN FIN,SYN -m limit --limit 3/min -j LOG --log-prefix "Stealth SYN/FIN scan(?): " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -m limit --limit 3/min -j LOG --log-prefix "Stealth Null scan: " --log-level 7
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,PSH,URG -j DROP
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,ACK,URG -j DROP
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN,SYN,RST,PSH,ACK,URG -j DROP
-A VALID_CHK -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG FIN -j DROP
...
```


AIF can be downloaded here: http://rocky.eld.leidenuniv.nl/

TCP Wrappers:

Before firewall rules, we had “tcp_wrappers”. Tcp_wrappers is a library of functions that helps programs decide on their own whether they will allow a network connection from a particular remote computer. The library, called “libwrap”, provides routines for parsing rules stored in the files /etc/hosts.deny and /etc/hosts.allow, and applying those rules to incoming network connections.

Each line in these files specifies a service and a list of clients (i.e., computers to be allowed or denied access to that service). As a special case, the word “ALL” can be used for either service or client.

The files are processed in this order:

- Access will be granted when a (service,client) pair matches an entry in the /etc/hosts.allow file.
- Otherwise, access will be denied when a (service,client) pair matches an entry in the /etc/hosts.deny file.
- Otherwise, access will be granted.

For example, here are files that allow web server access to everybody, and allow computers at UVa to have access to all services, but deny all other computers access to anything:

hosts.allow

```
httpd: ALL
ALL:  .virginia.edu
```

hosts.deny

```
ALL: ALL
```

The End

64

Thanks!