Network Administration

ComS 252 — Iowa State University

Barry Britt and Andrew Miner

Overview of this lecture

This lecture covers assorted topics in “network administration”

- Tools to manage a home network
- Tools to manage a huge network

Overview of this lecture

This lecture covers assorted topics in “network administration”

- Tools to manage a home network
- Tools to manage a huge network

- Large networks can be complicated
  - So necessarily, these topics are complicated
  - That is the price of flexibility
- There is a reason you get these network topics last

What is DHCP?

- Dynamic Host Configuration Protocol
- A network protocol for configuring network devices
- Information that can be sent to a client includes:
  - An IP address to use
  - The subnet mask to use
  - IP addresses for gateway machines
  - IP addresses for DNS servers
- Is an open protocol
  - Described in RFC 2131 (1997) for IPv4
  - Described in RFC 3315 (2003) for IPv6

Why use DHCP? Why not static IP addresses?

- A DHCP server is easy to set up
- DHCP clients are trivial to set up
- Easy to move clients to other networks
- Network changes are centralized
- Fewer IP addresses may be required

Why use DHCP? Why not static IP addresses?

- A DHCP server is easy to set up
- DHCP clients are trivial to set up
- Easy to move clients to other networks
- Network changes are centralized
- Fewer IP addresses may be required

But what if I need fixed IP addresses for my server machines?

Why use DHCP? Why not static IP addresses?

- A DHCP server is easy to set up
- DHCP clients are trivial to set up
- Easy to move clients to other networks
- Network changes are centralized
- Fewer IP addresses may be required

But what if I need fixed IP addresses for my server machines?

- DHCP can handle this
  - Use a static IP for your server machine; or
  - DHCP can always give the same address to the server machine

How does DHCP work?

- How can the client contact a server when it knows nothing?

How does DHCP work?

- How can the client contact a server when it knows nothing?
- UDP broadcast packets are used
  - Go to all machines on the physical subnet
  - Servers listen for broadcasts on port 67
  - Clients listen for broadcasts on port 68

How does DHCP work?

- How can the client contact a server when it knows nothing?
- UDP broadcast packets are used
  - Go to all machines on the physical subnet
  - Servers listen for broadcasts on port 67
  - Clients listen for broadcasts on port 68
- Can we use one DHCP server for multiple physical subnets?

How does DHCP work?

- How can the client contact a server when it knows nothing?
- UDP broadcast packets are used
    - Go to all machines on the physical subnet
    - Servers listen for broadcasts on port 67
    - Clients listen for broadcasts on port 68
- Can we use one DHCP server for multiple physical subnets?
    - Yes, but will need a relay server on each subnet
    - The relay servers forward messages to the DHCP server

The actual (simplified) DHCP protocol

1. Discovery:
   - Broadcast message sent by client
   - Will include client's MAC address
   - May request to keep its last–used IP address
2. Offer:
   - Broadcast message sent by server, including
     - IP address
     - Subnet mask
     - Server's IP address
     - How long the address may be used (lease duration)
     - Clients must renew the address before the lease expires
3. Request:
   - Client accepts one offer (may receive several)
4. Acknowledgement:
   - Server acknowledges; IP address is assigned

DHCP server

- Service `dhcpd`
  - Manage as usual with `systemctl`
- Configuration file: path depends on distribution
  - `/etc/dhcpd.conf`
  - `/etc/dhcp/dhcpd.conf`
- Several useful `man` pages:
  - `dhcpd`: overview of the server
  - `dhcpd.conf`: configuration file
  - `dhcp-options`: configuration options
  - Others!

Format of `dhcpd.conf`

- Lines starting with "#" are ignored
- Files contain statements:
  - Settings for the server
  - DHCP options to send to the client
  - "`subnet`" block
    - A block of statements that apply to a subnet
  - "`host`" block
    - A block of statements that apply to a host
  - "`group`" block
    - A way to group things with common settings
  - There are others
- Let's look at these...

Settings and options in `dhcpd.conf`

Useful server setting statements

Specify the default and maximum lease times:

- `default-lease-time <sec>;`
- `max-lease-time <sec>;`

Settings and options in `dhcpd.conf`

Useful server setting statements

Specify the default and maximum lease times:

- `default-lease-time <sec>;`
- `max-lease-time <sec>;`

Option statements

- Have the form `option <optname> <value>;`
- Will cause the specified information to be sent to the client
- Some useful option names:

`subnet-mask`:	Good old subnet mask
`broadcast-address`:	IP address for broadcast
`routers`:	IP address of gateway
`domain-name-servers`:	IP addresses for DNS servers

Statements that apply to subnets or hosts

`subnet <IP-addr> netmask <mask> { <stmts> }`

- May include settings discussed earler
- Should include one or more "`range`" statements:
  - `range <low-IP-addr> <high-IP-addr>;`
  - Specifies a range of addresses to pull from

Statements that apply to subnets or hosts

`subnet <IP-addr> netmask <mask> { <stmts> }`

- May include settings discussed earlier
- Should include one or more "`range`" statements:
  - `range <low-IP-addr> <high-IP-addr>;`
  - Specifies a range of addresses to pull from

`host <name> { <stmts> }`

- May include settings discussed earlier
- Should specify a hardware address:
  - `hardware <HW-type> <HW-addr>;`
- Can specify a fixed address:
  - `fixed-address <IP-addr>;`

Example dhcpd.conf for a server controlling 2 subnets

```
# Globals
default-lease-time 600;
max-lease-time 7200;
option subnet-mask 255.255.255.0;

subnet 192.168.123.0 netmask 255.255.255.0 {
  option routers 192.168.123.254;
  range 192.168.123.50 192.168.123.249;
}

subnet 192.168.4.0 netmask 255.255.255.0 {
  option routers 192.168.4.1;
  range 192.168.4.100 192.168.4.249;
}
```

Another example dhcpd.conf

```
# Globals
default-lease-time 3600;
max-lease-time 86400;
option subnet-mask 255.255.255.0;
option routers 192.168.0.1;
option domain-name-servers 192.168.0.1, 192.168.0.2;

subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.10  192.168.0.199;
}

host webserver {
  hardware ethernet 00:11:22:33:44:55; # MAC address
  fixed-address 192.168.0.8;
}
```

What is DNS?

- Domain Name System
- IP protocol is based *entirely* on IP addresses
  - 32 bits for IPv4
  - 128 bits for IPv6
  - Packets use these for source and destination addresses
- But IP addresses are annoying to remember
  - Will be worse with IPv6
- Host *names* are easier (e.g., `google.com`, `www.redhat.com`)
- DNS finds IP addresses for host names
  - The Internet's "phone book"
- Is described by dozens of RFCs
  - Basic specs are RFC 1034 and RFC 1035 (1987)

General overview of DNS

- Conceptually, DNS is a huge database
  - A directory that gives IP addresses for a host name
  - A *reverse* directory that gives host name(s) for IP address
- The database is

  Hierarchical for quick responses

  Distributed for fast access
  - There are *lots* of DNS servers worldwide
  - 2011 estimate: 18.5 million DNS servers

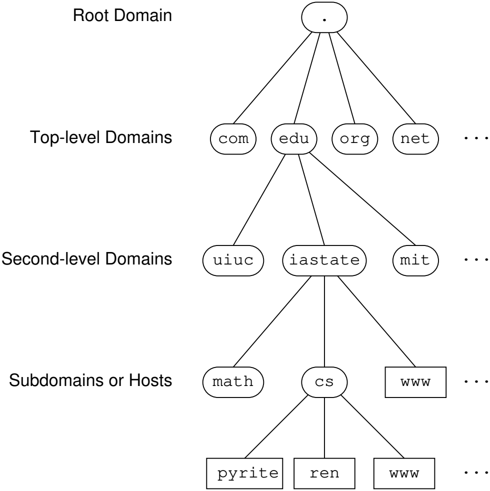
  Replicated for reliability
- Today, DNS implementation is

  Secure using authentication

  Fast using caching

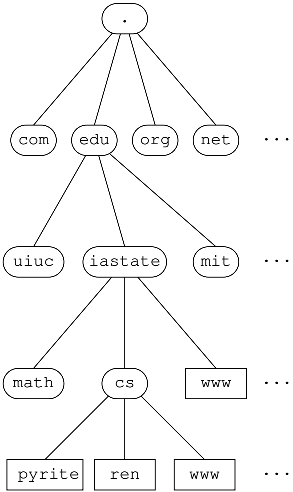
Hierarchical domain structure

- Each node is a domain
- Top node: root domain
- Node labels: 63 char max
- A subdomain may have its own subdomains
- Max. depth: 127
- Fully Qualified Domain Name (FQDN): a path from a node to the root
  - Should include root
  - E.g., `iastate.edu.`
- FQDN max length: 255



The Resolver

- FQDNs are like absolute paths
  - Include the final "."
- Unqualified domain names:
  - Do not include the final "."
  - Are like relative paths
- The resolver (on the client):
  - Tries to complete unqualified names
  - Appends domains to produce FQDNs
  - FQDNs are passed to DNS, one at a time
  - "." is always tried first
- File `/etc/resolv.conf` in Linux:
  - Configures the resolver
  - Use `search dom1 dom2 ...` to specify domains to append, in order
  - Is usually written by DHCP



Zones

- A DNS zone is a portion of a domain name space
- Each zone has at least one authoritative DNS server
  - Holds all information about the zone
  - Knows the DNS servers for any zones "below"
    - Called "delegation of authority"
    - Higher–level servers delegate authority to lower–level servers
- There are 13 authoritative DNS servers for the root domain
  - Operation is regulated by an ICANN committee
  - Names are `A.root-servers.net`, ..., `M.root-servers.net`
  - Many of these have multiple physical servers
  - Are distributed worldwide
- ICANN delegates authority for top–level domains
  - E.g., there is an authoritative DNS server for `edu` domain

The three types of DNS queries

1. Iterative queries
   - Client sends a domain name
   - The server does not query other servers
   - The server returns *either*
     - The IP address for the domain
     - The name of another DNS server
       (either the authoritative one for the domain, or an ancestor)

The three types of DNS queries

1. Iterative queries
   - Client sends a domain name
   - The server does not query other servers
   - The server returns *either*
     - The IP address for the domain
     - The name of another DNS server
       (either the authoritative one for the domain, or an ancestor)
2. Recursive queries
   - Client sends a domain name
   - The server returns the IP address for the domain
   - The server may query other servers to get this information

The three types of DNS queries

1. Iterative queries
   - Client sends a domain name
   - The server does not query other servers
   - The server returns *either*
     - The IP address for the domain
     - The name of another DNS server (either the authoritative one for the domain, or an ancestor)
2. Recursive queries
   - Client sends a domain name
   - The server returns the IP address for the domain
   - The server may query other servers to get this information
3. Inverse queries
   - Client sends an IP address
   - The server returns a domain name
   - Queries use a special domain named `in-addr.arpa`
     - E.g., to get the domain name for address `192.168.1.2`, send a request for `2.1.168.192.in-addr.arpa`.

Types of DNS servers

1. Primary (Master) server
   - Is authoritative
   - Holds the “master copy” of the zone data
   - Queries answered based on the zone file (a local file)
   - Should be iterative only
     - Except maybe for small systems on a trusted private network

Types of DNS servers

1. Primary (Master) server
   - Is authoritative
   - Holds the “master copy” of the zone data
   - Queries answered based on the zone file (a local file)
   - Should be iterative only
     - Except maybe for small systems on a trusted private network
2. Secondary (Slave) server
   - Is authoritative
   - Obtains zone data from another server (primary or secondary)
   - Can be iterative or recursive

Types of DNS servers

1. Primary (Master) server
   - Is authoritative
   - Holds the "master copy" of the zone data
   - Queries answered based on the zone file (a local file)
   - Should be iterative only
     - Except maybe for small systems on a trusted private network
2. Secondary (Slave) server
   - Is authoritative
   - Obtains zone data from another server (primary or secondary)
   - Can be iterative or recursive
3. Caching server
   - Is not authoritative
   - Store answers to previous queries in cache (memory)
   - Answers requests from cached queries, if it can
   - Otherwise, forwards the query

Resolving a query

- This is handled transparently by the resolver
- How does a query to `www.iastate.edu` get resolved?
  1. If the IP address is in the resolver's cache, return it
  2. The resolver contacts the DNS server
     - Usually, supplied by your ISP
  3. If address is in that server's cache, return it
  4. If this is a caching server, forward the query to a "real" server
  5. If address is in that server's cache, return it
  6. When nobody knows, start a query at one of the root servers
  7. Root server sends request to the authoritative server for `edu.`
  8. Request is sent to the authoritative server for `iastate.edu.`
  9. `iastate.edu.` server checks zone file for an entry `www`
- The `dig` utility can help with debugging DNS queries

Data in a zone file

- The basic data element in the DNS database is called a resource record
- These are defined in a zone file
- A resource record contains the following fields:

  Name of the node the record refers to

  TTL Time to live
  - This determines how long entries remain cached

  Class which should be `IN` (for Internet)

  Type of the record
  - There are many different types of records
  - Some of these will be discussed next

  Data which varies with the record type

Some types of resource records

`A` (IPv4 Address)
- Specifies the IPv4 address of a host:
  `host    IN   A         192.168.0.200`

`CNAME` (Canonical Name)
- Map an alias to a domain name; e.g.:
  `ftp     IN   CNAME     www.evilcorp.net.`
  maps host `ftp` to `www.evilcorp.net.`

`MX` (Mail exchange)

`NS` (Name server)
- Specify the name server for a domain

`PTR` (Pointer)
- Used for inverse name resolution, e.g.:
  `200     IN   PTR       host`

`TXT` Associates a character string with a domain

One last record type: SOA (Start of Authority)

- Every zone must have exactly one SOA record
- Must be the first resource record of the zone file
- Specifies a nameserver (with an `A` record)
- Specifies email address for mail regarding the domain
- Specifies the following, in order, in parentheses

  serial number
  - Used to indicate a change in zone data
  - By convention, this is `yyyymmddnn`: date of change, and change number

  refresh time (how often to update secondary servers)

  retry time (if a refresh fails, when do we try again)

  expire time (when do secondary servers expire)

  minimum time (how long to cache this zone data)

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL   3D
@       IN   SOA    namesrv.bogus.com.   email.bogus.com.    (
                    2015061101           ; serial number
                    8H                   ; refresh time
                    2H                   ; retry time
                    4W                   ; expire time
                    1D)                  ; minimum time
;
        IN   NS     namesrv.bogus.com.
;
namesrv      IN A   192.168.111.1
stan         IN A   192.168.111.15
kyle         IN A   192.168.111.17
cartman      IN A   192.168.111.18
kenny        IN A   192.168.111.22
```

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL   3D
@      IN   SOA    namesrv.bogus.com.   email.bogus.com.    (
                   2015061101           ; serial number
                   8H                   ; refresh time
                   2H                   ; retry time
                   4W                   ; expire time
                   1D)                  ; minimum time
;
       IN   NS     namesrv.bogus.com.
;
namesrv     IN A   192.168.111.1
stan        IN A   192.168.111.15
kyle        IN A   192.168.111.17
cartman     IN A   192.168.111.18
kenny       IN A   192.168.111.22
```

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL   3D
@      IN   SOA    namesrv.bogus.com.   email.bogus.com.    (
                   2015061101           ; serial number
                   8H                   ; refresh time
                   2H                   ; retry time
                   4W                   ; expire time
                   1D)                  ; minimum time
;
       IN   NS     namesrv.bogus.com.
;
namesrv     IN A   192.168.111.1
stan        IN A   192.168.111.15
kyle        IN A   192.168.111.17
cartman     IN A   192.168.111.18
kenny       IN A   192.168.111.22
```

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL  3D
@     IN   SOA   namesrv.bogus.com.   email.bogus.com.   (
                 2015061101           ; serial number
                 8H                   ; refresh time
                 2H                   ; retry time
                 4W                   ; expire time
                 1D)                  ; minimum time
;
      IN   NS    namesrv.bogus.com.
;
namesrv    IN A  192.168.111.1
stan       IN A  192.168.111.15
kyle       IN A  192.168.111.17
cartman    IN A  192.168.111.18
kenny      IN A  192.168.111.22
```

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL   3D
@      IN   SOA    namesrv.bogus.com.   email.bogus.com.   (
                   2015061101           ; serial number
                   8H                   ; refresh time
                   2H                   ; retry time
                   4W                   ; expire time
                   1D)                  ; minimum time
;
       IN   NS     namesrv.bogus.com.
;
namesrv     IN A   192.168.111.1
stan        IN A   192.168.111.15
kyle        IN A   192.168.111.17
cartman     IN A   192.168.111.18
kenny       IN A   192.168.111.22
```

Simple example zone file

```
; All text after a semi-colon is ignored on a line
$TTL   3D
@      IN   SOA    namesrv.bogus.com.   email.bogus.com.    (
                   2015061101           ; serial number
                   8H                   ; refresh time
                   2H                   ; retry time
                   4W                   ; expire time
                   1D)                  ; minimum time
;
       IN   NS     namesrv.bogus.com.
;
namesrv     IN A   192.168.111.1
stan        IN A   192.168.111.15
kyle        IN A   192.168.111.17
cartman     IN A   192.168.111.18
kenny       IN A   192.168.111.22
```

Simple example "inverse" zone file

```
; The inverse zone data for our example domain
$TTL  3D
@      IN   SOA   namesrv.bogus.com.    email.bogus.com.    (
                  2015061101            ; serial number
                  8H                    ; refresh time
                  2H                    ; retry time
                  4W                    ; expire time
                  1D)                   ; minimum time
;
       IN   NS    namesrv.bogus.com.
;
1      IN   PTR   namesrv.bogus.com.
15     IN   PTR   stan.bogus.com.
17     IN   PTR   kyle.bogus.com.
18     IN   PTR   cartman.bogus.com.
22     IN   PTR   kenny.bogus.com.
```

Simple example "inverse" zone file

```
; The inverse zone data for our example domain
$TTL   3D
@      IN   SOA   namesrv.bogus.com.   email.bogus.com.    (
                  2015061101           ; serial number
                  8H                   ; refresh time
                  2H                   ; retry time
                  4W                   ; expire time
                  1D)                  ; minimum time
;
       IN   NS    namesrv.bogus.com.
;
1      IN   PTR   namesrv.bogus.com.
15     IN   PTR   stan.bogus.com.
17     IN   PTR   kyle.bogus.com.
18     IN   PTR   cartman.bogus.com.
22     IN   PTR   kenny.bogus.com.
```

Setting up a DNS server in Linux

1. Find a decent HOWTO
   - I like `http://www.tldp.org/HOWTO/DNS-HOWTO.html`
2. Install BIND
   - Berkeley Internet Name Domain
   - Is open source (BSD license)
   - Version 9 is current
3. Edit the `named` configuration file (more on this next...)
   - In Fedora: `/etc/named.conf`
   - Make a backup copy first
   - There are lots of options here; some are important to keep
   - Zone files will be specified here
4. Create the zone files (if you want an authoritative server)
5. Manage the `named` service as usual

Configuring a DNS server in Linux

Format of `/etc/named.conf`:

- Text file
- Sequence of statements of various types
- Statement syntax:

```
<statement> <arguments> {
  <option> ;
  <option> ;
};
```

- Statements may be nested
- Errors in `/etc/named.conf` may prevent `named` from starting
  - Utility `named-checkconf` can help you here
- Errors in zone files may prevent `named` from starting
  - Utility `named-checkzone` can help with these

Configuring a DNS server in Linux (2)

Useful statement types:

`options`

- Used for global options (at top level)
- Some useful options:
  - `listen-on`: Specify subnets where `named` listens for queries
  - `allow-query`: Specify hosts or subnets allowed to query
  - `forwarders`: Specify IP addresses to forward queries to
  - `forward first`: Ask nameservers listed in `forwarders` first

Configuring a DNS server in Linux (3)

Useful statement types:

`zone zone-name zone-class`

- Used to define a zone with given name and (optional) class
- Some useful options:
  - `file`: absolute or relative pathname for zone file
    - Relative to `/var/named`
    - `/etc/named` also a reasonable location
  - `notify (yes/no)`: Notify secondary servers on update?
  - `type`: Type for the zone
    - `master`: This server is authoritative and primary
    - `slave`: This server is secondary
    - Others . . .

Tiny example named.conf

```
options {
    listen-on port 53 { 127.0.0.1; 192.168.111.0/24; };
    :
    allow-query { localhost; 192.168.111.0/24; };
    forward first;
    forwarders { 192.168.50.1; };
};
zone "bogus.com" {
    type master;
    notify no;
    file "/etc/named/bogus.com.zone";
};
zone "111.168.192.in-addr.arpa" {
    type master;
    notify no;
    file "/etc/named/inverse.bogus.com.zone";
};
```

Kernel Routing Tables

- A Routing Table is a list of rules for sending packets
- For each outgoing IP packet, determines how to send it
- The same mechanism for both simple and complex networks
- Linux supports multiple routing tables
  - Allows routing based on policy
  - Routing table may be selected based on the packet attributes
  - Uses a routing policy database
- We will discuss the main routing table
  and assume that all packets are routed the same way

Destination classes

A machine may send packets to

1. Itself
   - Often through a special loopback interface
   - These IP addresses are called local IPs
2. Locally–connected machines
   - Machines connected to this machine on some LAN
   - These IP addresses are called directly reachable IPs
3. Everything else
   - These addresses are reached through one or more gateways

Structure of a routing table

Each routing table entry contains:

- A destination "subnet"
  - An IP address
  - Number of significant bits to match
    - As an IP mask, usually denoted as `Genmask`
- The gateway to use
  - `0.0.0.0` for none (directly reachable)
  - Otherwise, IP address of a directly reachable gateway
- The interface to use

Deciding how to route a packet:

1. Find the rule with the longest matching destination "subnet"
2. Send the packet according to that rule

The kernel does this for every outgoing IP packet

Viewing a routing table

`netstat` utility

- Shows network stuff

  `-n` : numerical (shows IP addresses)

  `-r` : shows kernel routing tables
- Is obsolete but still widely used
- Will display the following flags:

  `U` : route is up

  `G` : through a gateway

  ... possibly others

Example routing table

```
prompt$ █
```

Example routing table

```
prompt$ netstat -nr
```

Example routing table

```
prompt$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags ... Iface
192.168.42.0    0.0.0.0         255.255.255.0   U     ... eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     ... lo
0.0.0.0         192.168.42.1    0.0.0.0         UG    ... eth0
prompt$
```

Reading this table:

Example routing table

```
prompt$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags ... Iface
192.168.42.0    0.0.0.0         255.255.255.0   U     ... eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     ... lo
0.0.0.0         192.168.42.1    0.0.0.0         UG    ... eth0
prompt$
```

Reading this table:

- Packets for address `192.168.42.x` are sent via `eth0`
  - This machine must be part of the `192.168.42.0/24` subnet

Example routing table

```
prompt$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags ... Iface
192.168.42.0    0.0.0.0         255.255.255.0   U     ... eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     ... lo
0.0.0.0         192.168.42.1    0.0.0.0         UG    ... eth0
prompt$
```

Reading this table:

- Packets for address `192.168.42.x` are sent via `eth0`
  - This machine must be part of the `192.168.42.0/24` subnet
- Packets for local address `127.x.x.x` are sent via `lo`

Example routing table

```
prompt$ netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags ... Iface
192.168.42.0    0.0.0.0         255.255.255.0   U     ... eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     ... lo
0.0.0.0         192.168.42.1    0.0.0.0         UG    ... eth0
prompt$
```

Reading this table:

- Packets for address `192.168.42.x` are sent via `eth0`
  - This machine must be part of the `192.168.42.0/24` subnet
- Packets for local address `127.x.x.x` are sent via `lo`
- Default route: all other packets are sent via `eth0`
  - And will go through the gateway `192.168.42.1`

Changing a routing table

- Linux is pretty clever about correctly initializing a routing table
- If your network interfaces are configured correctly
  - The routing table should be initialized properly
  - Usually it is unnecessary to change this

Changing a routing table

- Linux is pretty clever about correctly initializing a routing table
- If your network interfaces are configured correctly
  - The routing table should be initialized properly
  - Usually it is unnecessary to change this
- But let's see how to change it, anyway, "by hand"

Changing a routing table

- Linux is pretty clever about correctly initializing a routing table
- If your network interfaces are configured correctly
  - The routing table should be initialized properly
  - Usually it is unnecessary to change this
- But let's see how to change it, anyway, "by hand"
- Old utility: route
  - `route add ...` to add a route
  - `route del ...` to delete a route
  - Can also display the routing table

Changing a routing table

- Linux is pretty clever about correctly initializing a routing table
- If your network interfaces are configured correctly
  - The routing table should be initialized properly
  - Usually it is unnecessary to change this
- But let's see how to change it, anyway, "by hand"
- Old utility: route
  - `route add ...` to add a route
  - `route del ...` to delete a route
  - Can also display the routing table
- New utility: ip
  - `ip route add ...` to add a route
  - `ip route del ...` to delete a route
  - Can also display the routing table
- Check your `man` pages for details

Properties of a network router

- A router has more than one network interface
  - The kernel routing table will be more interesting
  - Might have different gateways for different destinations
  - How do we update routing tables in a complex network?
    - We will discuss this in a minute...

Properties of a network router

- A router has more than one network interface
  - The kernel routing table will be more interesting
  - Might have different gateways for different destinations
  - How do we update routing tables in a complex network?
    - We will discuss this in a minute...
- A router will forward packets
  - From the last example, `192.168.42.1` was the gateway for the `192.168.42.0/24` subnet
  - Suppose on the gateway machine:
    - Interface `eth0` is on the `192.168.42.0/24` subnet
    - Interface `eth1` is "connected to the Internet"
  - The gateway must forward packets arriving on `eth0` to `eth1`
    - We will discuss this in a minute...

Properties of a network router

- A router has more than one network interface
  - The kernel routing table will be more interesting
  - Might have different gateways for different destinations
  - How do we update routing tables in a complex network?
    - We will discuss this in a minute. . .
- A router will forward packets
  - From the last example, `192.168.42.1` was the gateway for the `192.168.42.0/24` subnet
  - Suppose on the gateway machine:
    - Interface `eth0` is on the `192.168.42.0/24` subnet
    - Interface `eth1` is “connected to the Internet”
  - The gateway must forward packets arriving on `eth0` to `eth1`
    - We will discuss this in a minute. . .
- A router may use NAT (Network Address Translation)
  - This is done in the firewall. . .

Forwarding packets

How to enable packet forwarding in Linux?

Forwarding packets

How to enable packet forwarding in Linux?

- Remember the `/proc` virtual filesystem?
  - Dynamically updated, kernel information
- File `/proc/sys/net/ipv4/ip_forward`
  - If this contains "1", the kernel is forwarding packets
  - If this contains "0", the kernel is not forwarding packets

Forwarding packets

How to enable packet forwarding in Linux?

- Remember the `/proc` virtual filesystem?
  - Dynamically updated, kernel information
- File `/proc/sys/net/ipv4/ip_forward`
  - If this contains "1", the kernel is forwarding packets
  - If this contains "0", the kernel is not forwarding packets
- Extreme magic: we can start or stop forwarding using:
  - `echo 1 > /proc/sys/net/ipv4/ip_forward` (starts)
  - `echo 0 > /proc/sys/net/ipv4/ip_forward` (stops)
  - But this will not persist across reboots

Forwarding packets

How to enable packet forwarding in Linux?

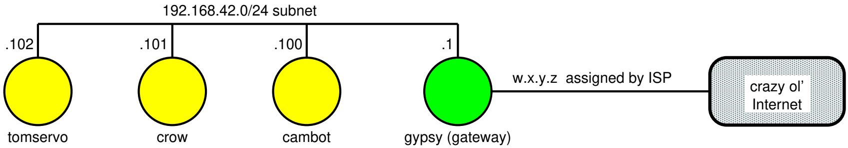
- Remember the `/proc` virtual filesystem?
  - Dynamically updated, kernel information
- File `/proc/sys/net/ipv4/ip_forward`
  - If this contains "1", the kernel is forwarding packets
  - If this contains "0", the kernel is not forwarding packets
- Extreme magic: we can start or stop forwarding using:
  - `echo 1 > /proc/sys/net/ipv4/ip_forward` (starts)
  - `echo 0 > /proc/sys/net/ipv4/ip_forward` (stops)
  - But this will not persist across reboots
- To set this value at boot time:
  - Edit text in file `/etc/sysctl.conf`:
    `net.ipv4.ip-forward=1`

Forwarding packets

How to enable packet forwarding in Linux?

- Remember the `/proc` virtual filesystem?
  - Dynamically updated, kernel information
- File `/proc/sys/net/ipv4/ip_forward`
  - If this contains "1", the kernel is forwarding packets
  - If this contains "0", the kernel is not forwarding packets
- Extreme magic: we can start or stop forwarding using:
  - `echo 1 > /proc/sys/net/ipv4/ip_forward` (starts)
  - `echo 0 > /proc/sys/net/ipv4/ip_forward` (stops)
  - But this will not persist across reboots
- To set this value at boot time:
  - Edit text in file `/etc/sysctl.conf`:
    `net.ipv4.ip-forward=1`
- I have also seen these instructions:
  - Edit text in file `/etc/sysconfig/network`:
    `FORWARD_IPV4 = YES`

A LAN gateway



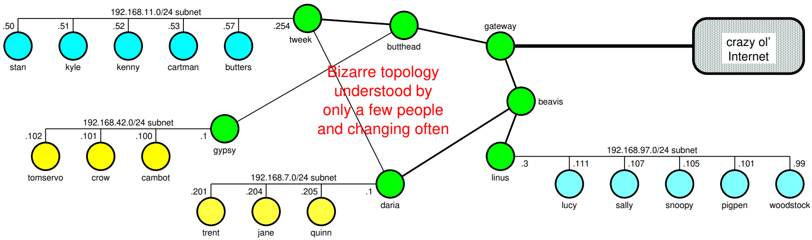
Setting up the kernel routing tables for LAN machines is trivial

- We already saw an example of this
- OS usually gets it right with no intervention

Setting up the kernel routing tables for `gypsy` is trivial

- Send packets for `192.168.42.0/24` subnet to that interface
- Send all other packets to Internet interface
- Use the default gateway given by ISP for the default route
- OS usually gets this one right with no intervention

What about a collection of LANs?



- Suppose we have dozens or hundreds of interconnected LANs
- E.g., the network on Iowa State campus
- The topology of routers is complicated
- Not feasible to set routing tables by hand
- Not possible for OS to “guess” routing tables
- Need to do something clever . . .

Routing protocols

- Routers communicate with each other to determine topology:
  - Each router knows about networks it is directly attached to
  - Routers share information with their immediate neighbors
  - Routers gradually gain knowledge of the network topology
- Routing algorithms can then choose routes based on topology
- The kernel routing tables are updated automatically
- There are different types of protocols
  - Based on the underlying algorithm
  - For different sizes and types of networks
- Basic division of protocols:
  1. Interior Gateway Protocols
  2. Exterior Gateway Protocols
- Before we talk about the Gateway Protocols. . .

Autonomous System (AS)

- A collection of connected IP routing prefixes (subnets)
- Under the control of one or more network operators
- Idea: the individual networks that formed the Internet
- See RFC 1930 for an in–depth discussion
- ISU's network is (probably) an Autonomous System

Interior Gateway Protocols (IGPs)

- An IGP is a routing protocol for use within an AS
- Two categories based on the algorithm used:
  1. Link–state routing protocol
  2. Distance–vector routing protocol

Link–state routing (an IGP)

- Each router knows the complete network topology
  - This is usually learned as part of the protocol
  - If links fail, the topology is updated

Link–state routing (an IGP)

- Each router knows the complete network topology
  - This is usually learned as part of the protocol
  - If links fail, the topology is updated
- Each router independently calculates routes
  - For every possible destination, determine "best next hop"
    - This is exactly the routing table
    - "Best next hop" is usually based on lowest latency
  - Dijkstra's "shortest path" algorithm will do this

Link–state routing (an IGP)

- Each router knows the complete network topology
  - This is usually learned as part of the protocol
  - If links fail, the topology is updated
- Each router independently calculates routes
  - For every possible destination, determine "best next hop"
    - This is exactly the routing table
    - "Best next hop" is usually based on lowest latency
  - Dijkstra's "shortest path" algorithm will do this
- Example protocols of this type:
  - Open Shortest Path First (OSPF)
    - Common in large enterprise networks
    - OSPF Version 2 (RFC 2328) is for IPv4
    - OSPF Version 3 (RFC 5340) is for IPv6
  - Intermediate System to Intermediate System (IS–IS)
    - Common in large service provider network backbones
    - Defined by ISO; republished in RFC 1142

Distance–vector routing protocol (an IGP)

- Each router does not know the complete topology
- Each router knows its "distance" to immediate neighbors
  - Distance is either hop count or latency
- Each router builds a vector of distances to all other routers
- Each router shares its vector with other routers, periodically
- Routers update their own vectors based on received ones
  - This is the Bellman–Ford algorithm
  - The vectors will eventually converge to actual distances
  - Routing tables are updated based on received vectors

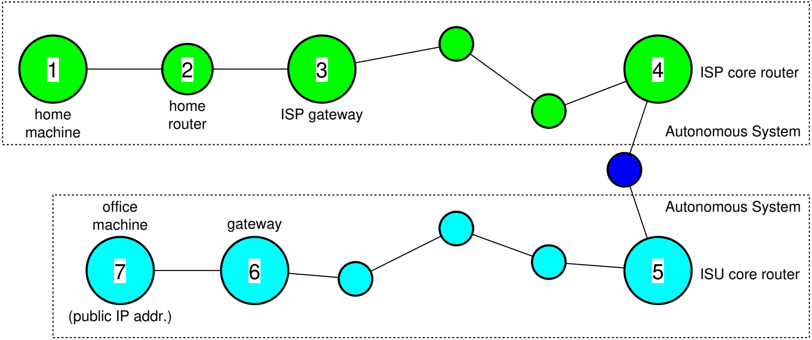
Distance–vector routing protocol (an IGP)

- Each router does not know the complete topology
- Each router knows its "distance" to immediate neighbors
  - Distance is either hop count or latency
- Each router builds a vector of distances to all other routers
- Each router shares its vector with other routers, periodically
- Routers update their own vectors based on received ones
  - This is the Bellman–Ford algorithm
  - The vectors will eventually converge to actual distances
  - Routing tables are updated based on received vectors
- Example protocols of this type:
  - Routing Information Protocol (RIP)
    - Maximum number of hops allowed is 15
    - Version 2 described in RFC 2453 (1998)
  - Enhanced Interior Gateway Routing Protocol (EIGRP)
    - Proprietary, by Cisco
    - Replaces now obsolete IGRP

Exterior Gateway Protocols

- Each Autonomous System (AS) has one or more Core Routers
  - Low–latency, high–throughput
- Each AS has an official number
- Core routers use an Exterior Gateway Protocol to determine routes between AS's
- In today's Internet, that protocol is the Border Gateway Protocol (BGP)
  - Described in RFC 4271

Example: connecting from home to campus



- Routes 1 $\rightarrow$ 2 and 2 $\rightarrow$ 3 are trivial
- Route 3 $\rightarrow$ 4 determined by ISP's Interior Gateway Protocol
- Route 4 $\rightarrow$ 5 determined by Border Gateway Protocol
- Route 5 $\rightarrow$ 6 determined by ISU's IGP
- Route 6 $\rightarrow$ 7 is trivial

Firewalld

- User-space system for dynamic firewall management
- Discussed basics in earlier lecture
- This lecture:
    - A bit more detail how it works
    - Zones
    - How to set up NAT

Firewalld architecture

- Firewalld is essentially a *front-end*
- Actual packet filtering happens in the kernel
- Kernel packet filtering framework is `nftables`
  - `nftables` replaced older `iptables` in 2014
- You can set up `nftables` "by hand"
  - User-space utility is `nft`
  - Gives more control than Firewalld
  - …but also more of a pain

Basics of `nftables`

- Based on (user-defined) tables
- Each table can have multiple (user-defined) chains
- A chain is a list of rules, applied in order
- A rule typically is
  - Criteria for matching a packet, such as
    - Port number
    - Protocol (e.g., TCP)
    - Status of a connection
    - IP address (source or destination)
    - Combinations of these
  - A "verdict": what to do if the packet matches, such as
    - Accept the packet
    - Drop the packet
    - Keep going in the chain
    - Jump to another chain

Firewalld services

- Can use `firewall-cmd` to allow or deny *services*
  - This sets up appropriate rule(s) in appropriate chain and table
  - Can define your own services
    - As `.xml` files
    - Pre-defined usually in `/usr/lib/firewalld/services`
    - User-defined usually in `/etc/firewalld/services`
- Can allow/deny services for different zones
  - If not specified, applies to the default zone
- But what exactly are *zones*?

Firewalld zones

- A zone defines a level of trust
  - Each connection or interface belongs to one zone
  - A zone may have many connections and interfaces
- Idea: zones with more trust can allow more services
- Examples:
  - Public WiFi conection vs. wired home network connection
  - Network device connected to “Internet” vs. private network
- There are pre-defined zones (may vary by distribution)
  - Typically configured in `/usr/lib/firewalld/zones`
- You can define your own zones
  - Typically under `/etc/firewalld/zones`

Some pre-defined zones

- drop: Nothing accepted
  - All incoming packets dropped; no reply
- block: Nothing accepted
  - All incoming connections rejected with "icmp-host-prohibited"
- public: Mostly untrusted
  - Only selected incoming connections are accepted
- external: for routers and NAT
- dmz
- work
- home
- internal: Mostly trusted
  - Only selected incoming connections are accepted
- trusted: All incoming connections are accepted

Firewall zone configuration

- Can be done using `firewall-cmd`
  - `--remove-interface=iface`: remove interface from a zone
  - `--add-interface=iface`: add interface to a zone
  - `--get-active-zones`
    - Show list of active zones
    - For each zone, show what is using the zone
  - `--info-zone=zone-name`
    - Show detailed information for a zone
    - Includes interfaces and allowed services
- Can be done "by hand"
  - Add text "`ZONE=zone-name`" in `ifcfg-iface` file
  - This assumes the interface always belongs to the zone

NAT with Firewalld

- Packet fowarding is enabled or disabled as discussed earlier
- NAT can be turned on for a zone
  - All forwarded packets that go out on the zone use NAT
  - Replies are sent back to the right place using NAT
- `--add-masquerade`: start using NAT on a zone
- `--remove-masquerade`: stop using NAT on a zone
- Usually you want to specify a zone with those

  (otherwise you get the default zone)
- Can specify `--permanent` or not as usual
- `--info-zone=zone-name` shows if masquerading is on/off

Quick example with `firewall-cmd`

```
prompt$
```

Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --get-active-zones
```

Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --get-active-zones
home
  interfaces: eth1
external
  interfaces: eth0
prompt$ 
```

Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --get-active-zones
home
  interfaces: eth1
external
  interfaces: eth0
prompt$ firewall-cmd --info-zone=external --permanent
```

Quick example with `firewall-cmd`

```
prompt$ firewall-cmd --info-zone=external --permanent
external (active)
  target: default
  icmp-block-inversion: no
  interfaces: eth0
  sources:
  services: ssh
  ports:
  protocols:
  masquerade: yes
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

prompt$ 
```

Some utilities

`dig` : DNS lookup utility

`ip` : Update the kernel routing table

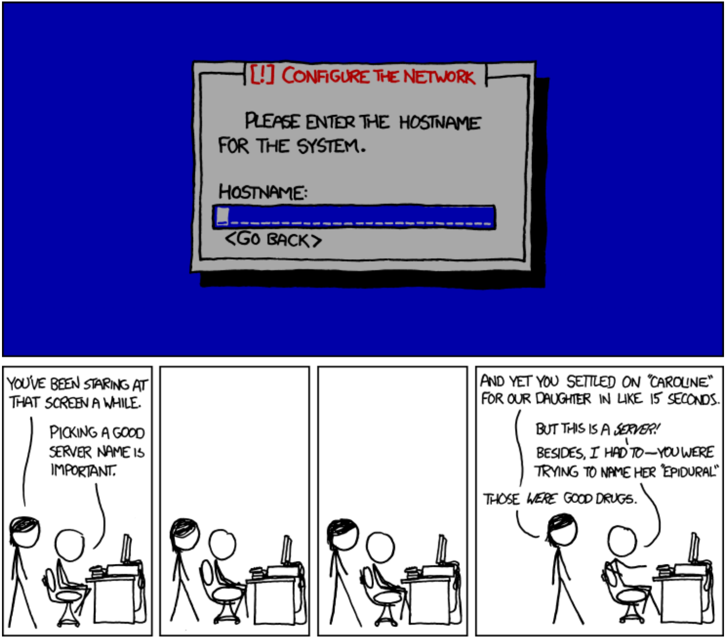
`firewall-cmd` : Update firewall settings

`named-checkconf` : Check `named` configuration file

`named-checkzone` : Check a `named` zone file

`netstat` : Show network status

`route` : Update the kernel routing table


[!] CONFIGURE THE NETWORK
PLEASE ENTER THE HOSTNAME FOR THE SYSTEM.
HOSTNAME:
<GO BACK>
YOU'VE BEEN STARING AT THAT SCREEN A WHILE.
PICKING A GOOD SERVER NAME IS IMPORTANT.
AND YET YOU SETTLED ON "CAROLINE" FOR OUR DAUGHTER IN LIKE 15 SECONDS.
BUT THIS IS A SERVER!
BESIDES, I HAD TO—YOU WERE TRYING TO NAME HER "EPIDURAL."
THOSE WERE GOOD DRUGS.

End of lecture