ICS 351: Today's plan

- routing protocols
- linux commands

Routing protocols: overview

- maintaining the routing tables is very labor-intensive if done manually
  - so routing tables are maintained automatically:
  - each router knows what networks it is connected to
  - and communicates that to other routers
  - a routing protocol defines how routing information is communicated among routers
  - popular routing protocols are OSPF, BGP, RIP
- with the information from the routing protocol, each router can build and maintain its routing tables

Routing protocols: properties

- routing is not a perfect process: sometimes the routing tables are inconsistent, because it takes time for a router to discover changes and it takes time to communicate the new information
  - if a router doesn't have a route to a packet's destination, it will drop the packet, i.e., not forward it
- most of the time routing protocols maintain the routing tables correctly
  - much faster than with manual updates

Router hardware and software

- big, expensive Cisco routers
- inexpensive Linux boxes with multiple network interfaces
- a Linux general-purpose computer can:
  - use routing software to route packets
  - just as the expensive Cisco router can
  - but not as fast
  - and perhaps not on the same media
- an expensive router should have hardware acceleration for
  - looking up routes in a routing table, and
  - forwarding packets from one interface to another
- the software to run the routing protocol might be very similar on a generic box and on an expensive specialized router

Command line

- the part of an operating system (or an application) that interfaces with a user is the **user interface**, sometimes called a **shell**
- most user interfaces are graphical: significant parts of the functionality are accessible through the mouse and windows
- many system functions use a simpler user interface, which is text based
- the user gets a prompt whenever the system is ready to handle new commands
- the user can type commands, which the system then executes
- the commands may print output on the screen or, less commonly, request input from the users

Unix/Linux shell

- on Unix systems, including Linux, commands are interpreted by an application program called the **shell**
- there are many possible Linux shells, but this class will use the default, which is `bash` (Bourne-Again SHell). Another notable shell is `tcsh`.
- some commands are built-in to the shell, but usually a command entered on the shell results in executing an application
- the typical shell command syntax is:
- `command -switch .. --switch parameter .. parameter`
- the most important command to remember is `man`, short for *manual*, which gives information about other commands
  - e.g. `man ls` gives the manual "page" for the `ls` command

Useful Unix commands and concepts

- shells on Linux usually implement commands from the Unix family of operating systems
- `ls` lists files and directories (`ls -a` also lists files beginning with ".")
- `pwd` displays the name of the current directory, and `cd` changes the current directory
- `mkdir name` creates the directory `name`, and `rmdir name` removes it if it is empty
- `rm name` removes the file (permanently!)
- `rm -i name` asks first
- `cp name1 name2` copies the file `name1` to another file (or directory) `name2`
- `mv name1 name2` moves/renames the file `name1` to another file (or directory) `name2`
- in Unix, the root of the file system is "/", and "/" is also used as a separator at the end of directory names, e.g. `/etc/hosts` is the name of a file in the directory `/etc` (or `/etc/`)
- `mount /dev/sdb1 /mnt/mydisk` makes the file system on the device `/dev/sdb1` accessible as `/mnt/mydisk`, assuming such a device is connected and such file system exists
- to safely remove the device, simply `umount /mnt/mydisk`

Unix/Linux file commands

`gedit file` runs the `gedit` text editor on the file

`more file` displays the contents of the file, one screenful at a time

`cat file` does the same, without stopping

the output of a command can be sent ("piped") to the input of another command, or directly to a file

`cat file > file2` is another way of copying `file` to `file2`

`cat file >> file2` appends `file` to the end of `file2`

`cat file | tee file2` shows the contents of `file` and also writes them to `file2`

`command x > file & tail -f file` puts the output of the command into `file`, and also shows the growing contents of `file`

the & at the end of the command puts its execution into the background, so the shell prompts again while the command is still running

a command in the background can be brought to the foreground with `fg`

Ctrl-Z (^Z) can be used to suspend a running foreground command, and `bg` will send it to the background

Ctrl-C (^C) will normally kill a foreground command

`jobs` will show the running background command, `kill %3` will kill background command 3, and `pkill abcd` will kill command `abcd`

Special shell constructs

- a pipe is a sequence of commands separated by |
  - the output of the first command is the input to the second command
  - the output of the (n-1)th command is the input to the nth command
  - an input redirect ‘<’ is usually only found on the first command in a pipe
  - an output redirect ‘>’ is usually only found for the last command in a pipe

    example: `tr '\n' ' ' < file.html | sed 's/<[^>]*>/ /g' > file.txt`
- an ‘&’ starts the previous command in the background, then executes the next command (if any)

  example: `sleep 100 &`
  - ‘jobs’ lists the commands that are running in the background
  - ‘fg’ puts a background command in the foreground
  - Control-Z stops the currently executing command
  - control-C kills the currently executing command
- part of a command in `backticks` is executed and replaced with its output

  `more `grep -l foo *`` runs “more” on the file names printed by grep
- ‘single quotes’ and “double quotes” are used for any parameter that contains spaces or other special chars
- the backslash \ is used as an escape
- the star * matches anything, e.g. ba* matches bar and baz

Unix/Linux networking commands

`telnet host port` connects to the given port on the given host:

if there is a telnet server on that port of the given host, then allows entering commands remotely (but very insecurely)

`ftp host` opens a **File Transfer Protocol** session to the given host (assuming there is an FTP server running there)

ftp supports simple commands to transfer files, including `ls`, `cd`, `lcd`, `binary`, `ascii`, `get`, `mget`, `put`, `mput`, `quit`

both telnet and ftp transfer everything in the clear

anyone with access to the network can see what is transferred

modern systems use encrypted transfers, particularly based on the **secure shell** with the two commands `ssh` and `scp`

`ssh host` or `ssh -p port host`

`scp file host:remote/path`, or `scp host:remote/path local/path`, or `scp -P port host:remote/path local/path`

`ping host` sends packets to a host that are likely to elicit replies, and prints any replies it gets

`traceroute host` sends packets that are likely to be dropped enroute to a host, and prints the error messages

nc/netcat

on one computer, or in one window, run:

```
nc -l 12345
```

on the other side, run:

```
nc <IP address> 12345
```

now, anything typed in one (window, computer) appears in the other