

Unix System Administration

Chris Schenk

Lecture 06 – Tuesday Feb 05

CSCI 4113, Spring 2007

Important Notes on Symlinks

- Work across partitions
 - Have their own INODE that points to another directory entry
 - Hard links directly reference INODEs specific to that partition
- Can point to directories
 - Hard links only link files
- Are less efficient than hard links
 - Extra de-reference to reach the actual file
- Can point to a non-existent file

SETUID and SETGID

- SETUID – program runs as user who owns file
- SETGID – program runs as group who owns file
 - One or the other (or both) can be set
- Only works on the actual executable running
 - Doesn't work in scripts!
- Real vs Effective user/group
 - Real remains the same
 - Effective is what permissions you have

Devices

- /dev directory describes all available devices
 - /dev/cdrom, /dev/sda*
- Different device types
 - block 'b', character stream 'c', pipe 'p'
 - TTYs, sound card, hard drives, cd drives
- Filesystems use block devices
 - /dev/hda, /dev/sda, /dev/scd0
- All devices are exposed as files!
 - Makes interacting with them much easier

Disk Devices

- Labeling differs for IDE and SCSI devices
- IDE uses `/dev/hd*`
 - Primary IDE channel 1 - `/dev/hda`
 - Primary IDE channel 2 - `/dev/hdc`
 - Many times this is your CDROM
 - `lrwxrwxrwx 1 root root 3 2007-01-21 16:18 /dev/cdrom -> hdc`
- SCSI uses `/dev/sd*`
 - Based on SCSI channel ID
 - SCSI channel 1 - `/dev/sda`
 - SCSI channel 2 - `/dev/sdb`

Partitions

- A device is divided into physical extents
- Physical extents are raw
 - Filesystems built on top of partitions!
 - Use block devices under /dev
- Numbers are placed after device IDs
 - First partition on IDE device /dev/hda is /dev/hda1
 - Second partition on SCSI device /dev/sdb is /dev/sda2

Partition Table Editors

- Most installations have automatic partitioners
 - And even if you ask to manually partition, there's a gui involved
- The old-school command line tool is `fdisk`
- usage: `fdisk <device>`
 - `fdisk /dev/sda`
 - 'p' – print partition table
 - 'n' – add new partition
 - 't' – change partition's system ID
 - 'w' – write changes to disk

Partition Table

- Same sector as the Master Boot Record (MBR)
- 64-byte data structure, each entry 16 bytes long
- Maximum of 4 partitions!
 - Extra partitions handled by the 'extended' type
 - Created as a solution for the 4 partition limit
- 10 fields in each entry in the table
 - Bootable flag
 - starting/ending head/sector/cylinder
 - System ID

System ID

- Describes the filesystem on each partition
 - independent to the OS, standard format
 - **one** bytes long, 256 total entries available
 - 0x06 (FAT16), 0x07 (NTFS), 0x05 (Extended), 0x82 (linux)
- Read by the OS at boot time
 - need to know which modules to load

Filesystems

- Formatting
 - Partitions must be formatted to a specific filesystem
 - `mkfs` utility
 - different versions, `mkfs.ext3`, `mkfs.reiser`, etc
- Each filesystem requires a module in the kernel
 - OS needs to know how to read data from a device
 - `lsmod` (list modules) utility
 - 'used by' number of devices

Filesystem Types

- More than simply ext3, reiser
 - These exist mainly on physical disks
 - Sometimes in 'ramdisks' as well
- Two big examples are /dev and /proc
 - /dev exposes physical devices to the user
 - Not always a separate filesystem, however
 - /proc exposes process information to the user
 - `ps` and `top` read this data
 - Also read CPU and RAM data (`cpuinfo`, `meminfo`)

Mounts

- Filesystem starts at / (root)
 - not to be confused with root user
 - tree structure starting a /, branching with directories
- A partition may be mounted to any point on tree
 - partitions always mounted to directories:
 - /dev/sda1 - /
 - /dev/sda2 - /usr
 - /dev/sda5 - /var
 - Can override existing files!
 - Doesn't mean they are gone...

Mounts (cont)

- How do you see your current mounts?
 - `mount` command
 - `df` command (with `-h` parameter)
- `mount` shows options on the filesystem
 - `[no]exec`, `[no]atime`, `defaults`, `nosuid`, `ro/rw`
- Partitions mounted at boot time read from a config file
 - `/etc/fstab`
 - Manually mounted partitions looked up in `fstab`
 - manual override of options

/etc/fstab

- Six fields for each line
 - device or label, mount point, filesystem type, options, dump enable, fsck priority
- Some devices are marked 'none'
 - filesystems used by the kernel for special tasks
 - process information, device addresses
- CDRoms list 'auto' for filesystem type
 - three types of cdrom filesystems: IS9660, Joilet, HFS
 - Data is the same, but the filesystem metadata is different

More on mount

- Manual mounting
 - `mount [-o <options>] <device> <mount point>`
- For entries in `/etc/fstab` only the device or mount point is needed
 - `mount /windows`
- Shows more information than simple devices?
 - NFS mounts
 - `dev/proc` filesystems (depending on distribution)

Mount Options

- `defaults` contains a number of options
 - `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, `async`
- `rw/ro` – Mount read/write, readonly
- `[no]exec` – Allow/disallow execution of binaries
- `[no]suid` – Allow/disallow setuid/setgid bits to work
- `[no]user` – Allow/disallow a user to mount the filesystem

umount

- Used to unmount a filesystem
- Almost identical in usage
 - `umount <device | mount point>`
- Problems may arise
 - `umount: /home: device is busy`
 - no file handles can be open if the device is to be unmounted
 - sitting in a directory on a device means files are open!
 - Your shell has handles open (check with `ls -o`)

Bash Scripting

- Like nothing you may have done before
 - Even Perl, Python, PHP are more organized
- Bash == Bourne Again SHell
 - Rewritten (but backwards compatible) from Bourne Shell (sh)
- Support for looping, functions, variables, arrays
 - And a host of other strange things
- Used all over your machine
 - Every init.d script is a Bash script

Shell Variables

- We've already taken a look at these
 - `$PS1` holds your prompt
 - `$PATH` holds your path directories
 - Shell parses out directories and searches for command
- We can use variables in scripts
 - And they remain local to scripts
 - Although I get confused with 'exported' variables
- Set a variable in bash:
 - `% myvar="some string of stuff"`

If Statements

- One of the more cracked out ways I've seen
 - “IF”s are always terminated with “FI”s
- The conditionals are even stranger
 - `[-f file] || exit 1`
 - What does it mean?! -- return values!
 - `test` is an alternative
- `if ["$UID" -ne "0"]; then echo "You are not root"; fi`
- There are **MANY** types of conditional tests
- They are really picky about spaces and quotes
 - Not like your average language

Case Statements

- Should be familiar from any C programming
 - ```
case "$VAR" in
 start)
 <cmd>
 <cmd>
 ;;
 *)
 <cmd>
 ;;
esac
```
- Used in every single `/etc/init.d` script
  - `*` is equivalent to 'default'

# Init Script Rules

- There are guidelines for writing init.d scripts
  - Good examples exist in `/etc/init.d` everywhere!
- Start, stop, restart, status, usage (default)
  - These are the basics for startup and shutdown
- Also useful to use the logging library
  - Just another script that has functions
  - `/lib/lsb/init-functions` on Ubuntu
  - Logs things to syslog and has nice output on screen

# An IPTables Init Script

- IPTables has no configuration on bootup
  - Why not simply have it load rules from file on boot?
  - Allows us to add configurations in a file instead of with command-line tools
    - Can be cumbersome to input rules by hand
- IPTables does not clear rules with a single command
  - Default policies
  - Extra user-defined chains

# Networking and Routing

- Computers have to talk to each other
  - How do I get from here to there?
    - Routes!
- Networks define regions
  - Different layers within networks for communication
  - Described within definitions of subnets
- Routes define paths to networks
  - Many different protocols for routes
  - BGP, OSPF, RIP, etc

# Network Layers

- Two ways to describe the layers on a network
  - OSI vs TCP/IP
- Different layers have different parts of a packet
  - MAC, IP, TCP, RPC, etc
- Users mainly deal with things at highest level
  - Application level – SSH, Firefox, iTunes, etc
- Sysadmins get the whole shabang
  - Top to Bottom, we need to know it all (or at least most of it)

# OSI Layers

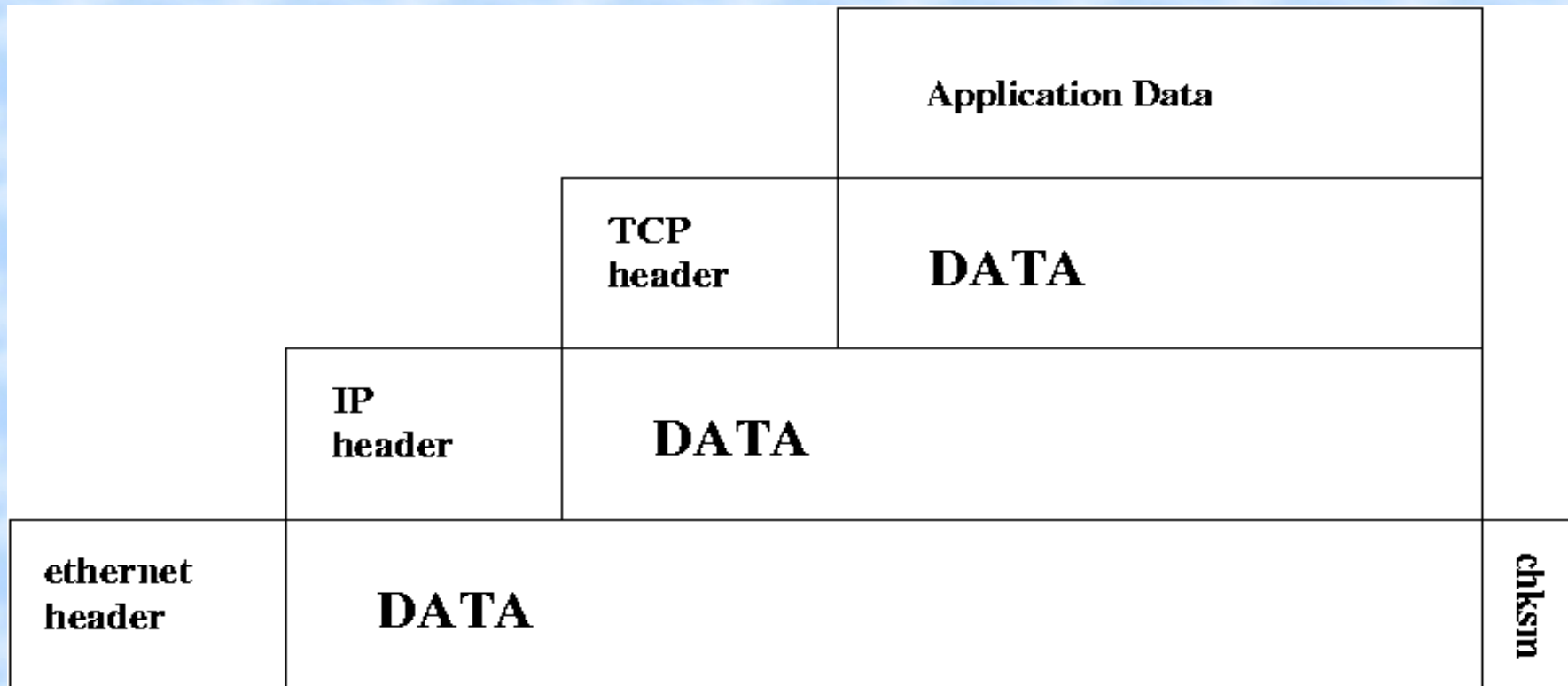
- Open System Interconnect model – 7 Layers
  - Layer 1 – Physical: Actual wires connecting hardware
  - Layer 2 – Data Link: Format of data, MTU, MAC addresses
  - Layer 3 – Network: IP addresses
  - Layer 4 – Transport: TCP, UDP, stateful packet transmission
  - Layer 5 – Session: RPC, HTTP
  - Layer 6 – Presentation: Translation (endianness), SSL
  - Layer 7 – Application: dig, mail, telnet, etc

# TCP/IP Layers

- 4 layers total – We will use this model
  - Layer 4 – Application
    - End user apps: DNS, arp, ftp, http, SSH, etc
  - Layer 3 – Transport: Communication among programs
    - TCP, UDP
  - Layer 2 – Network: Basic communication, addressing, routing
    - IP and ICMP
  - Layer 1 – Link: Defines network hardware and device drivers
    - ethernet layer, MAC addresses

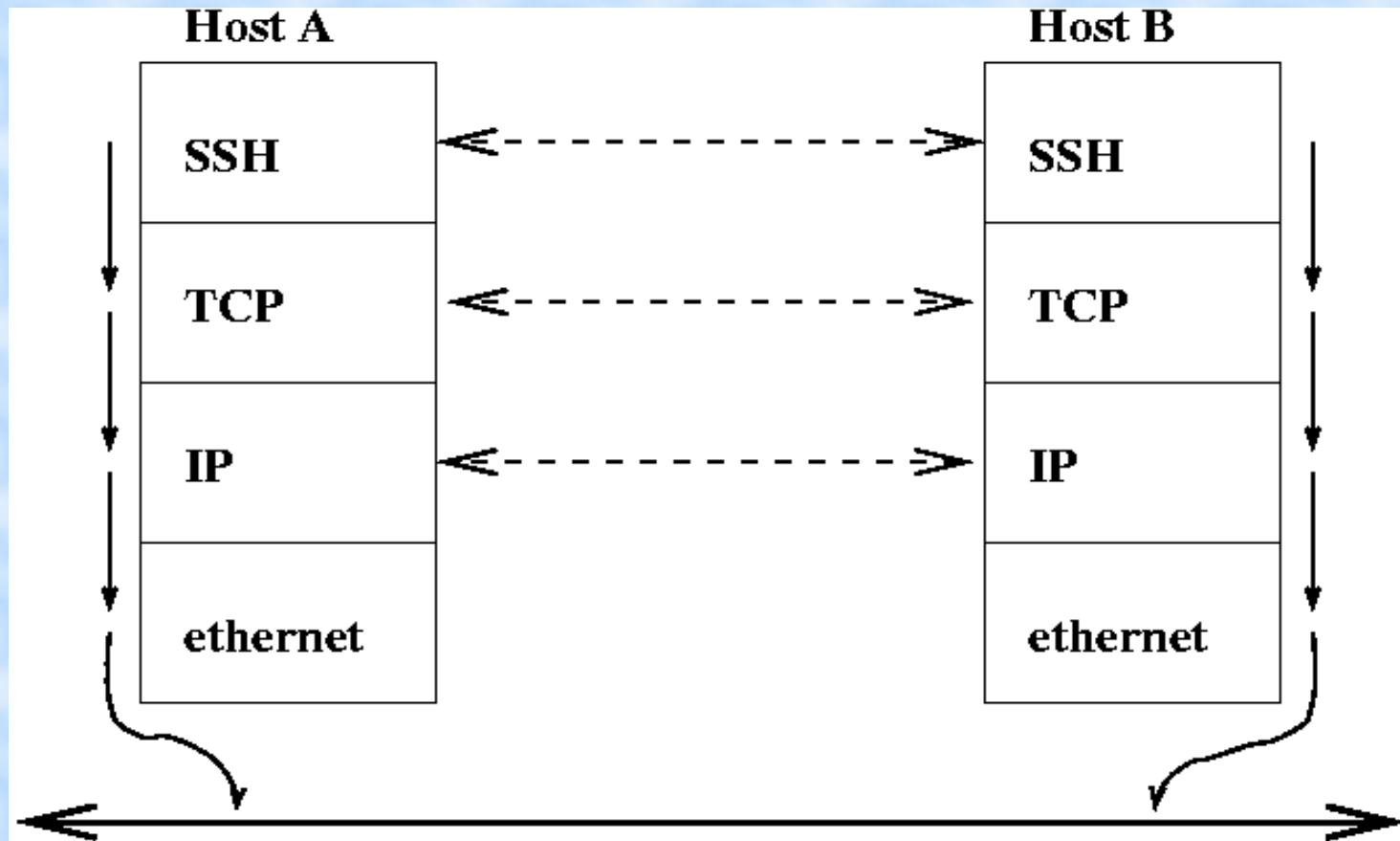
# Packets

- Encapsulate data for each layer



# Layer Independence

- Each layer doesn't care about the other



# Link Layer (1)

- Examples are:
  - Ethernet
    - Broadcast medium – many hosts
    - ubiquitous
  - **PPP – Point to Point Protocol**
    - unicast – two hosts
    - Old dial-up methods
- Media refers to:
  - Physical type – copper, fiber, etc
  - Protocol – ethernet, PPP, ARP, etc

# Link Layer Ethernet Communication

- Hosts have to talk on a shared physical line
  - Common analogy: A polite dinner party
- ethernet protocol uses CSMA/CD
  - Carrier Sense – detect when others are talking
    - Wait for a random amount of time to see if line is clear
  - Multiple Access – hosts use line at any time
    - Every host sees every packet (not always true!)– a **broadcast** medium
  - Collision Detection – detect when you are talking while another is
    - Host backs off and stops communication for a random amount of time

# Link Layer Packet Handling

- PPP networks don't need or have broadcasts
- Broadcast media means each host 'sees' each packet
  - Not always true if switching hardware is being used
- Packets seen by host are only processed when:
  - Destination MAC address matches 'this' host's interface
  - Destination MAC address is the broadcast MAC address
  - The interface is in PROMISCUOUS mode

# Packet MTU – Maximum Transmission Unit

- Largest size that data can be transmitted
  - depends on protocol and media
  - 10bt ethernet (and higher) use 1500 byte MTU
- Path MTU
  - Smallest unit a packet is fragmented between source and dest
  - Computed to avoid fragmentation along the way
    - for efficiency in multiple ways

# Link Layer – Ethernet Frames

|                            |                       |             |                        |                   |
|----------------------------|-----------------------|-------------|------------------------|-------------------|
| <b>destination address</b> | <b>source address</b> | <b>type</b> | <b>DATA</b>            | <b>check sum</b>  |
| <b>six bytes</b>           | <b>six bytes</b>      | <b>two</b>  | <b>46 ~ 1500 bytes</b> | <b>four bytes</b> |

- Addresses are SIX bytes long
  - Also known as MAC (machine) addresses
  - First three bytes denote manufacturer
  - MAC broadcast address is FF:FF:FF:FF:FF:FF
- Type defines the payload
  - IP, IPv6, ARP, RARP

# Link Layer – ARP and RARP

- ARP – Address Resolution Protocol
  - Used to resolve IP address -> MAC address mappings
    - Used in all switching hardware!
  - Local broadcast segments only (LANs)
    - This is how packets are routed on a LAN!
- RARP – Reverse Address Resolution Protocol
  - Used (sometimes) to obtain an IP address at boot time
  - **rarpd** server with MAC address -> IP address mappings

# Network Layer (2)

- Internet Protocol (IP) is defined as:
  - Unreliable
    - no guarantee that a packet reaches the destination
  - Stateless (connectionless)
    - No notion of a multi-packet IP session
    - See a packet, route it, and forget about it
- How are packets handled in terms of MTU splitting?
  - information duplicated in each packet

# Network Layer – IP Header

BYTE

|    |                               |                        |                          |                                 |                           |
|----|-------------------------------|------------------------|--------------------------|---------------------------------|---------------------------|
| 0  | 4-bit<br>version              | 4-bit<br>hdr<br>length | 8-bit<br>type-of-service | 16-bit<br>total length in bytes |                           |
| 5  | 16-bit<br>ID number           |                        |                          | 3-bit<br>flags                  | 13-bit<br>fragment offset |
| 9  | 8-bit<br>time-to-live         |                        | 8-bit<br>protocol        | 16-bit<br>header checksum       |                           |
| 13 | 32-bit Source IP address      |                        |                          |                                 |                           |
| 17 | 32-bit Destination IP address |                        |                          |                                 |                           |
|    | // IP Options (if any) //     |                        |                          |                                 |                           |
|    | // IP DATA //                 |                        |                          |                                 |                           |

# Layer 2 – IP Addresses

- IP version 4
  - 32-bit unsigned integer
  - written in decimal as a “dotted quad” - 128.138.202.19
- Historical IP address classes

| <b>Class</b> | <b>1st Byte</b>  | <b>Net bits</b>            |                |
|--------------|------------------|----------------------------|----------------|
| <b>A</b>     | <b>1 – 126</b>   | <b>8</b>                   | <b>N.h.h.h</b> |
| <b>B</b>     | <b>128 – 191</b> | <b>16</b>                  | <b>N.N.h.h</b> |
| <b>C</b>     | <b>192 – 223</b> | <b>24</b>                  | <b>N.N.N.h</b> |
| <b>D</b>     | <b>224 – 239</b> | <b>multicast</b>           |                |
| <b>E</b>     | <b>240 – 254</b> | <b>research / reserved</b> |                |

# Network Classes

- 32-bit internet addresses originally divided simplistically
  - Class A – 8 net bits, 24 host bits – 16777214 hosts
  - Class B – 16 net bits, 16 host bits – 65534 hosts
  - Class C – 24 net bits, 8 host bits – 254 hosts
    - Why do we lose two addresses per network?
- Companies originally given a classed network
  - IP space ownership similar to old phone #s
    - Companies owned phone numbers
- Large problems with classed networks
  - IP space limitations, routing nightmare

# IP Addresses – CIDR

- Classless Inter-Domain Routing
  - Solution to sucky 'classed' based splitting
  - Allowed for shorter routing tables on backbone routers
  - Net/Host boundary can be arbitrary
    - Not just on byte boundaries like the classed networks
    - Host portion still contiguous set of least-significant bits
  - Written as: <IP addr> / <net-bits>
    - 128.138.202.19 / 24
    - / **24** means 24 bits designate the NET portion of the address

# IP Addresses – Netmasks

- 32-bit quantity (same as IP address)
  - Consists of all ones (1) for the NET portion of the address
  - Necessary for CIDR to function
- Example
  - Host: `csel.cs.colorado.edu`
  - IP: `128.138.202.19`
  - Subnet: `128.138.202.0 / 24` (this is CIDR mask!)
  - Netmask: `255.255.255.0`

# IP Addresses – Subnets

- A **subnet** is a range of IP addresses
  - recall the CSEL subnet
    - Subnet: 128.138.202.0 / 24
  - the subnet has eight (8) bits for the HOST portion
  - a HOST portion of all zeros (0) designates the NET itself
  - a HOST portion of all ones (1) designates the broadcast address for the subnet
    - Subnet address: 128.138.202.0
    - Broadcast address: 128.138.202.255

# IP Addresses – CSEL Example

- Consider 128.138.202.0 and 128.138.202.255 in binary
  - 1000 0000 . 1000 1010 . 1100 1010 . 0000 0000
  - 128 . 138 . 202 . 0
  - 1000 0000 . 1000 1010 . 1100 1010 . 1111 1111
  - 128 . 138 . 202 . 255
- A CIDR address of 128.138.202.0/24 means
  - the NET portion is 24 bits:
  - 1000 0000 . 1000 1010 . 1100 1010 . 0000 0000
  - and the HOST portion is 8 bits:
  - 1000 0000 . 1000 1010 . 1100 1010 . 0000 0000

# IP Addresses – CSEL (cont)

- **/24 (8 HOST bits) subnet – 128.138.202.0/24**
  - Subnet: 128.138.202.0/24
  - Netmask: 255.255.255.0
    - 1111 1111 . 1111 1111 . 1111 1111 . 0000 0000
  - IP Broadcast: 128.138.202.255
- **Another /24 subnet – 128.138.237.0/24**
  - Engineering Wireless!
  - Subnet: 128.138.237.0/24
  - Netmask: 255.255.255.0
  - IP Broadcast: 128.138.237.255

# IP Addresses – Another Example

- University of Colorado given 128.138.0.0/16
  - That exact address and CIDR mask used in internet routers
- CS department given some of this space
  - 128.138.242.0 – 128.138.243.255 (512 addresses)
  - subnetted into six / **26** networks (6 HOST bits, 64 addresses) and one / **25** network (7 HOST bits, 128 addresses)
  - One 128.138.242.0 / 23 route used by CU routers
    - They have no idea about our internal subnets
    - More efficient to have a single route

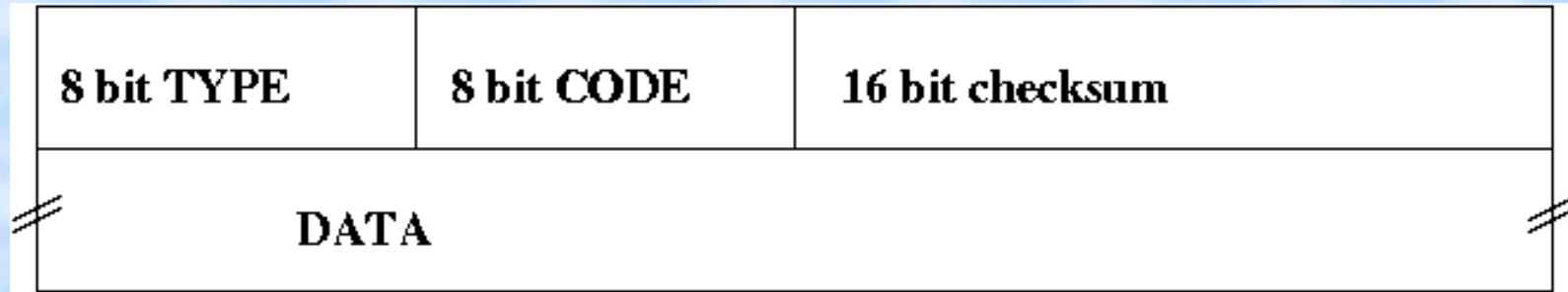
# IP Addresses – Example (cont)

- Consider 128.138.242.0 and 128.138.243.255 in binary
  - 1000 0000 . 1000 1010 . 1111 0010 . 0000 0000
  - 128 . 138 . 242 . 0
  - 1000 0000 . 1000 1010 . 1111 0011 . 1111 1111
  - 128 . 138 . 243 . 255
- A CIDR address of 128.138.242.0 / 23 means
  - the NET portion is 23 bits:
  - 1000 0000 . 1000 1010 . 1111 0010 . 0000 0000
  - and the HOST portion is 9 bits:
  - 1000 0000 . 1000 1010 . 1111 0010 . 0000 0000

# IP Addresses – Example (cont)

- A / **26** (six HOST bits) subnet – 128.138.243.0 / 26
  - Subnet: 128.138.243.0 / 26
  - Netmask: 255.255.255.192
    - 1111 1111 . 1111 1111 . 1111 1111 . 1100 0000
  - IP Broadcast: 128.138.243.63
- Another / 26 subnet – 128.138.243.64 / 26
  - Subnet: 128.138.243.64 / 26
  - Netmask: 255.255.255.192
  - IP Broadcast: 128.138.243.127

# Network Layer (2) – ICMP



- Internet Control Message Protocol
  - Part of network layer but relies on IP for delivery
- Possible types are:
  - Echo reply (0), Dest Unreachable (3)
  - Redirect (5), Echo request (8)
- For some types, CODE gives more info
  - try pinging the CU gateway 128.138.240.1

# Network Layer (2) – Summary

- Contains IP Addresses
  - Used for general internet routing
- Unreliable protocols at this layer
  - IP and ICMP
- Protocols listed in a file under /etc
  - /etc/protocols
  - Same values used in the 8-bit 'protocol' field

# Transport Layer (3)

- Port numbers are 16-bit unsigned int values
  - Used to differentiate applications and services
    - IP address is the **host**, Port is the **process**
  - Transport layer protocols must use both source and dest ports
  - Important services use well-known port numbers
    - Usually found within /etc/services
    - Very long list full of ports you've never heard of
  - Restricted ports (ports < 1024) require root
    - *Very* mild security

# Transport Layer - UDP

- User Datagram Protocol
  - **Unreliable** and **stateless** like IP
    - Applications must write-in reliability if needed
  - Good for short, one-time things like
    - NTP
    - DNS queries
    - Streaming video

|                                          |                                       |
|------------------------------------------|---------------------------------------|
| <b>16 bit SOURCE port number</b>         | <b>16 bit DESTINATION port number</b> |
| <b>16 bit length (incl hdr) in bytes</b> | <b>16 bit hdr chksm – unused</b>      |

# Traceroute

- Used to 'map' routes between you and the destination
- Traceroute sends arbitrary UDP packets to dest
  - Dest port is set to an unlikely value
  - IP time-to-live (TTL) field is incremented at each hop
  - ICMP time exceeded (11) error returned from routers
  - ICMP port unreachable (code 3)