California Polytechnic State University
Department of Computer Science and Software Engineering
Dr. John M. Bellardo
Fall Quarter 2017
Week 01 - 1

1 Course Introduction

- Welcome to networks
- syllabus review
  - Contact Info
  - Course Text. Online http://intronetworks.cs.luc.edu/current/html
  - Grade breakdown

Programs 1-3	30%
Final Program	20%
Midterm 1	25%
Midterm 2	25%
Labs	CR/NC
Total	100%

  - Exam ground rules (no aids, notes, closed book, etc).
  - Assigned homework problems not collected but will appear on exams.
  - Laptops / electronics not allow in lecture
  - Lecture notes posted on line prior to class
  - all information available on the course website, https://users.csc.calpoly.edu/~bellardo/courses/464
  - Labs
    * Required to complete all labs to pass the class
    * Will not require much extra time outside lab, but may sometimes run long
    * Showing up more than 10 minutes late reduces your course letter grade
    * If retaking you may not need to repeat the labs. See me after class.
  - Programs
    * Heavy programming component in this course
    * Programs will take a lot of time
    * Lots of self-learning in the programs, for example I will not be detailing new APIs.

- Don't Cheat!
  - Do not copy online code
  - Do not look at another student's code
  - Do not let other students look at your code
- I don't write your programs for you! But I will help with:
  - Debugging tips
  - Answers to specific questions
  - Pointers to reference material
- Don't ask me for help after 5 minutes, but don't wait 10 hours either!
- Grading Programs
  - I expect your program to work, and the policy grading reflects that
  - Programs are scored out of a fixed number of points
  - You earn the first points (typically around 30%) by meeting the basic functionality of the program
  - Basic functionality will typically include a large percentage of the full functionality
  - Additional points are earned by successfully handling additional functionality and corner cases
  - A score of 0 means your program didnt meet the basic functionality, and is considered failing the assignment.
  - Each assignment you fail reduces your maximum potential grade in the course.
  - The program is tested for functionality (I/O or unit test based)
  - I do not read the code and form an opinion as to how close you were to completing the assignment
  - Late submissions are accepted until the handin closes date listed on the assignment writeup and are subject to the following penalty – 5% per 24 hour period, prorated continuously
  - A passing but late program will always get at least 1 point
  - You will submit the required files for you program via handin
  - Program grading is handled via an automated cron job
  - There is no penalty for submitting multiple times, except a potential late penalty
  - If you have multiple submissions only the highest score will be used
  - Your submissions will only be tested once every 8 hours
  - Program Requirements

- Must be written in C or C++
- Must compile cleanly with -Wall and -Werror
- Must exit with a value of 0 when successful, 1 if it detects a command line error, and non-zero on error
- All required output must go to standard out, and exactly match the output description.
- Stderr is ignored
- Your program can not leak memory – smartalloc
- All programs must work correctly on unix4
- You must submit all code required to compile your program (e.g., smartalloc.c)

2 Program 1

- What are packets? Conceptually and concretely?
- Bits sent over the network have very specific formats so both ends can interpret them correctly
- IPv4 Header shown in Figure 1
- UDP Header shown in Figure 2
- Different "layers" get encapsulated inside each other like an onion.
- Example with wireshark
- Program 1
  - Read packets from input files using libpcap (refer to API docs to figure out how to actually do this)
  - Remove a few levels of encapsulation
  - Print selected fields from each level to stdout
  - Must meet course programming guidelines!

3 Packed Structures

1. Compiler adds padding between fields in C structures to keep fields aligned on "native" boundaries. This speeds up execution and is generally a good thing.
2. Sometimes padding is at the end of the structure to make any potential array elements aligned correctly.

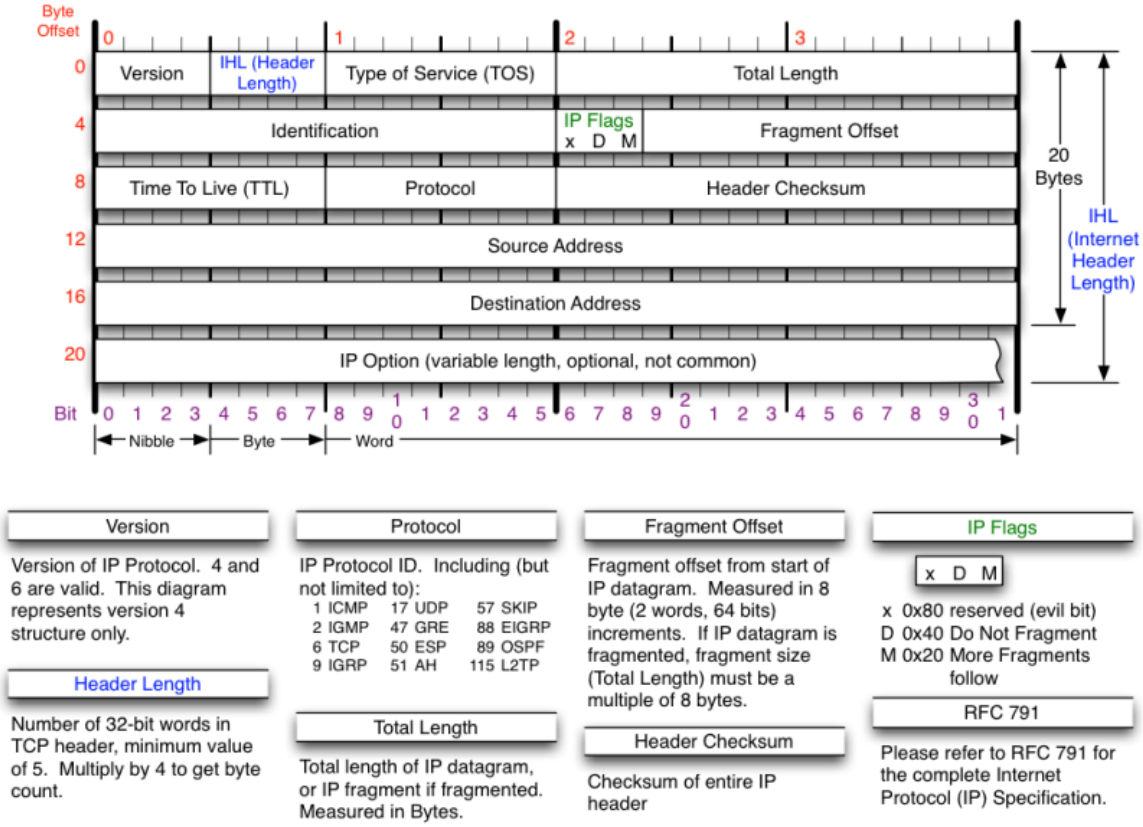

Figure 1: IPv4 Header

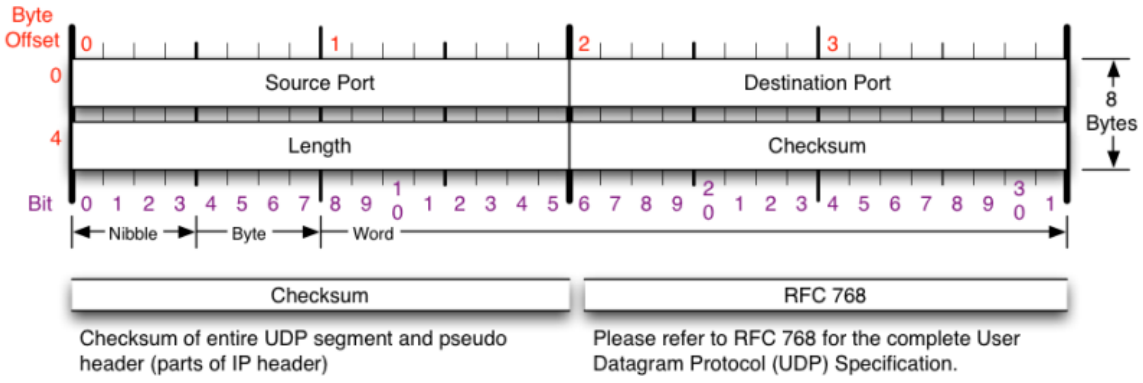

Figure 2: UDP Header

3. Use the gcc "__attribute__((packed))" attribute to force compiler to remove padding from structures.

4. Test program:

```
struct ex1
{
   char char1;
   short short1;
   int int1;
   char char2;
};

struct ex2
{
   char char1;
   char char2;
   short short1;
   int int1;
};

struct ex3
{
   char char1;
   short short1;
   int int1;
   char char2;
} __attribute__((packed));

   printf("sizeof(struct ex1) == %d\n", sizeof(struct ex1));
      printf("\toffset of char1:  %d\n", (int)&pex1->char1);
      printf("\toffset of short1: %d\n", (int)&pex1->short1);
      printf("\toffset of int1:   %d\n", (int)&pex1->int1);
      printf("\toffset of char2: %d\n\n", (int)&pex1->char2);

   printf("sizeof(struct ex2) == %d\n", sizeof(struct ex2));
      printf("\toffset of char1:  %d\n", (int)&pex2->char1);
      printf("\toffset of char2: %d\n", (int)&pex2->char2);
      printf("\toffset of short1: %d\n", (int)&pex2->short1);
      printf("\toffset of int1:   %d\n\n", (int)&pex2->int1);
```

```
printf("sizeof(struct ex3) == %d\n", sizeof(struct ex3));
   printf("\toffset of char1:  %d\n", (int)&pex3->char1);
   printf("\toffset of short1: %d\n", (int)&pex3->short1);
   printf("\toffset of int1:   %d\n", (int)&pex3->int1);
   printf("\toffset of char2: %d\n\n", (int)&pex3->char2);
```

5. Example output:

```
sizeof(struct ex1) == 12
offset of char1:  0
offset of short1: 2
offset of int1:   4
offset of char2: 8

sizeof(struct ex2) == 8
offset of char1:  0
offset of char2: 1
offset of short1: 2
offset of int1:   4

sizeof(struct ex3) == 8
offset of char1:  0
offset of short1: 1
offset of int1:   3
offset of char2: 7
```

4 Lab 1 Background

- Local Area Networks (LANs)
  - Scope
    - All devices attached to the same physical medium
    - WiFi channel
    - Ethernet cable
    - LTE cell
  - Ethernet addresses
    - All Ethernet network interface cards (NICs) are assigned a 6 byte MAC address
    - 12:34:56:78:90:AB

* Included in all ethernet packets. Ethernet devices *only* look at these MAC addresses
* Ethernet header is Figure 3

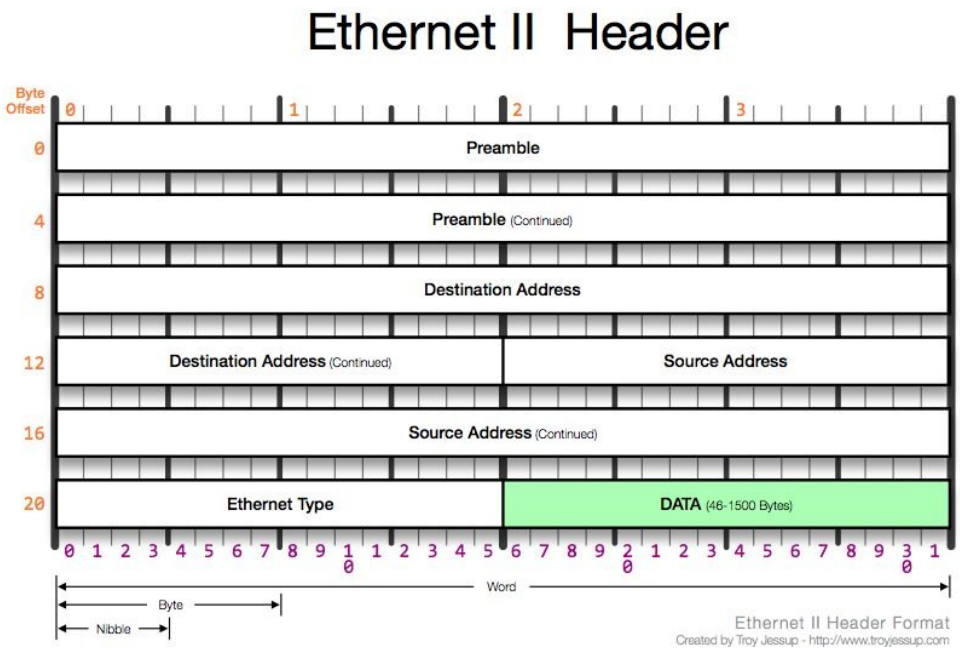

Figure 3: The ethernet 2 header

– Broadcast address
  * Special address indicating all nodes on LAN should process the packet
  * FF:FF:FF:FF:FF:FF
– Address resolution protocol (ARP)
  * Ethernet devices only pay attention to the ethernet addresses, ignoring all others (e.g., IP)
  * To send packets to another node on the LAN you must first know it's MAC address
  * Most higher-level communication uses IP addresses (and DNS names), not MAC addresses
  * ARP maps from IP addresses into MAC addresses
    · ARP Request is broadcast on the LAN ("Who has 1.2.3.4?")
    · If the IP address is on the LAN, a response is unicast back to the requestor

    * ARP cache
        · ARP lookups add an extra round trip time (RTT) to every packet
        · Cache responses to amortize the extra RTT
        · Entries removed after a timeout
        · Example ARP table

IP	MAC	Timeout
1.2.3.4	11:22:33:44:55:66	180

- Hubs
  - Hubs enable expansion of network to more that two machines. See firgure 4.
  - Packets that are received in one port are broadcast out all others

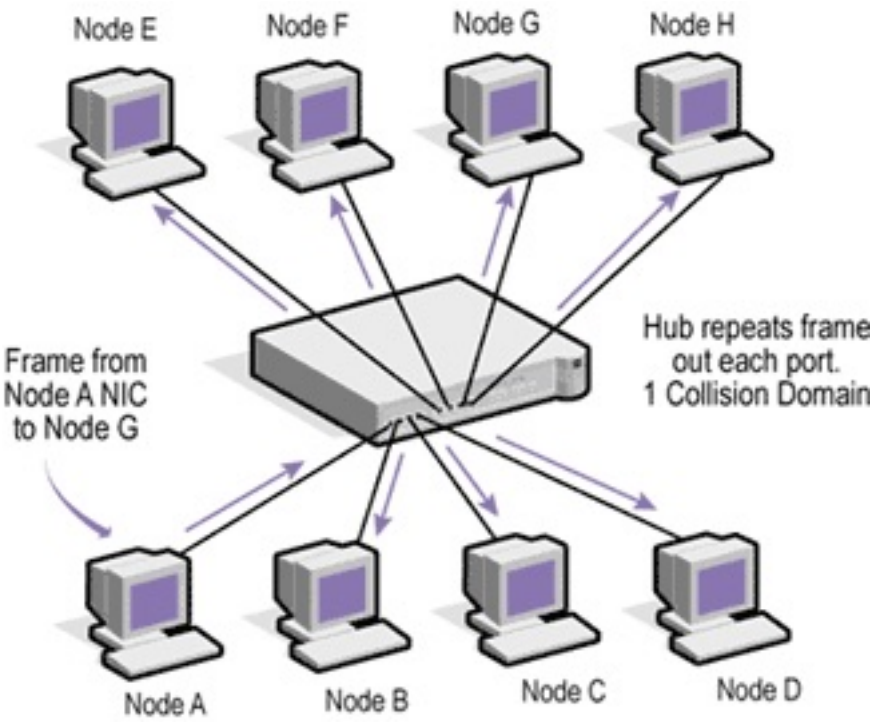

Figure 4: Ethernet Hub

- Ethernet Cabling
  - TX pair from one node needs to be connected to the RX pair on the other node.
  - Straight-through cables have TX connected to TX and RX connected to RX
  - Crossover cables have TX connected to RX and RX connected to TX

- Type of cable to use depends on which devices you are connecting

Network Equipment Group	**Device Group**
Hub	Everything Else
Switch	

- Use straight-through to connect nodes in the same group
- User crossover to connect nodes in different groups