CSCD433/533 Advanced Networks Winter 2017

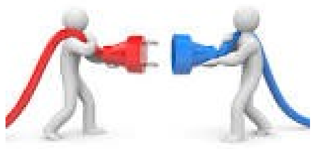
Lecture 13

Raw vs. Cooked Sockets

Introduction

- Better Understand the Protocol Stack
  - Use Raw Sockets
  - So far, sockets in Java either
    - TCP or UDP based
  - In fact, Java does not have built-in support for Raw Sockets!!!
  - To program Raw Sockets in Java - **Use Libraries**
  - C and Python have native support for Raw Sockets

Motivation for Raw Sockets



- Standard Java Sockets do not fit all our needs
- Normal sockets lack some functionality

1. We cannot read/write ICMP or IGMP protocols with normal sockets

   Ping tool cannot be written using normal sockets

2. Some Operating Systems do not process Ipv4 protocols other than ICMP, IGMP, TCP or UDP

   What if we have a proprietary protocol that we want to handle?

How do we send/receive data using that protocol?

Answer: Raw Sockets!!!

Raw Socket Defined

- Raw sockets allow a program or application to provide custom headers for a protocol which are otherwise provided by kernel/os network stack
- Raw sockets allow adding custom headers instead of headers provided by underlying operating system
- Bypasseses network stack and allows an application to also process packet headers

What can raw sockets do?

- Bypass TCP/UDP layers
- Read and write ICMP and IGMP packets
  - ping, traceroute, multicast daemon
- Read and write IP datagrams with an IP protocol field not processed by the kernel
  - OSPF – sits directly on top of IP
  - User process versus kernel
- Send and receive your own IP packets with your own IP header using the IP_HDRINCL socket option
  - Can build and send TCP and UDP packets
  - Testing, hacking
  - Only superuser can create raw socket though
- You need to do all protocol processing at user-level

Normal Sockets - Cooked

- Normal sockets, use OS kernel and built-in network stack to add headers like IP header and TCP header
- So an application only needs to take care of what data it is sending and what reply it is expecting
- Headers are added and removed without your application having to worry about this

More Motivation for Raw Sockets

- Recall, CSCD330, can we send true ICMP packets in Java?
  - Not exactly.
  - There is this work-around

```
InetAddress.getByName("192.168.1.1").isReachable(4000);
```

  - What does this do?

What does this do?

- InetAddress.getByName("192.168.1.1").isReachable(4000);
  - Does several things depending on OS and user permissions
    - Linux/MacOS
    - Linux/MacOS environment, **no Superuser** rights, JVM tries to establish TCP connection on port 7
      - Function returns true if TCP handshake is successful
    - With **Superuser** rights, Correct ICMP request is sent and function returns true if an ICMP reply is received
    - Windows XP
    - Windows XP environment, TCP handshake is used to test if machine is up, no matter if program has admin rights or not

Solution Using Raw Sockets

- There is a way in java, using various libraries
  - Using an older library jpcap, it is possible to assemble and send ICMP Packets
  - The library is here

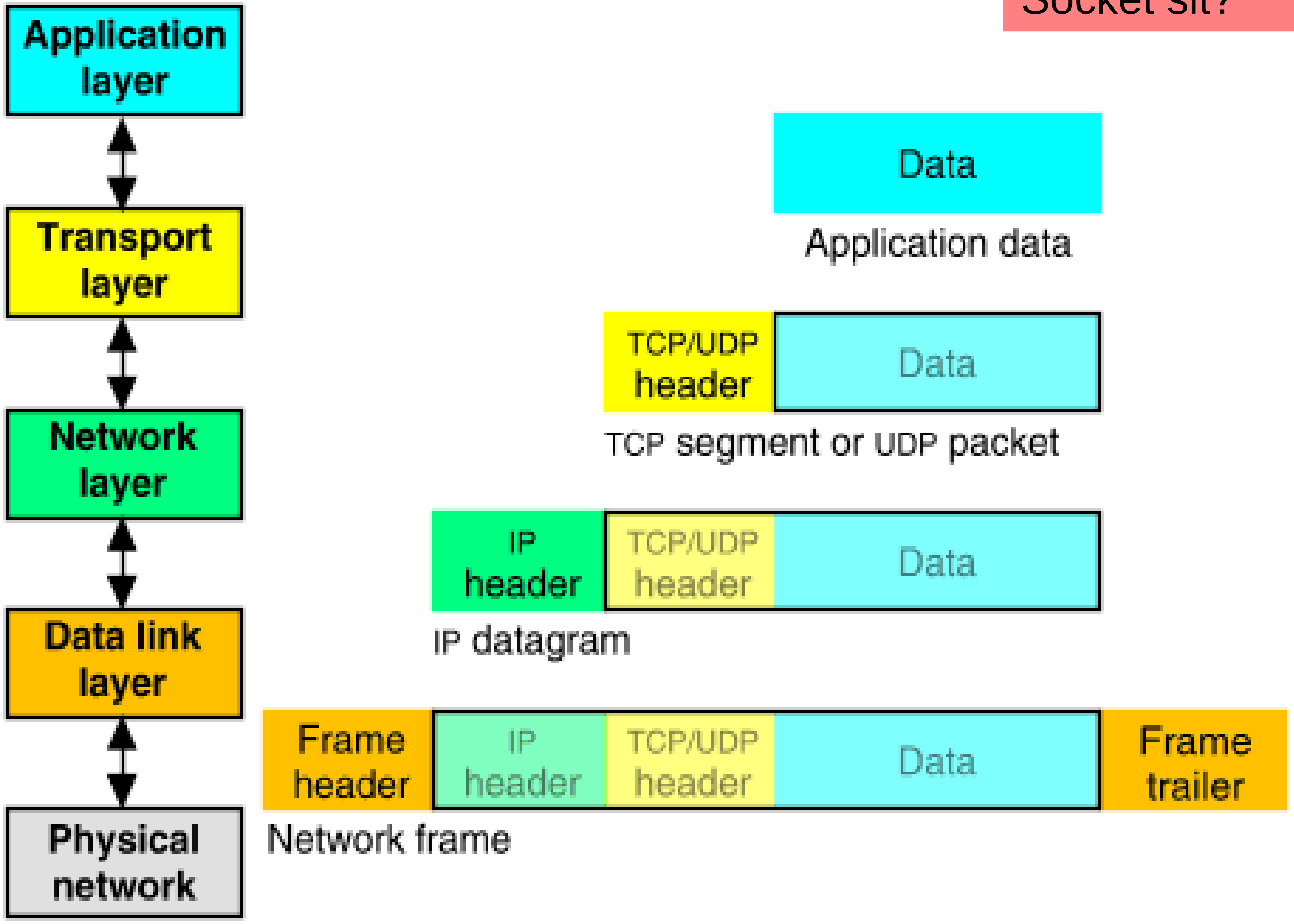
    http://www.sf.net/projects/jpcap

- Newer Library
  - jNetPcap

  http://jnetpcap.org/

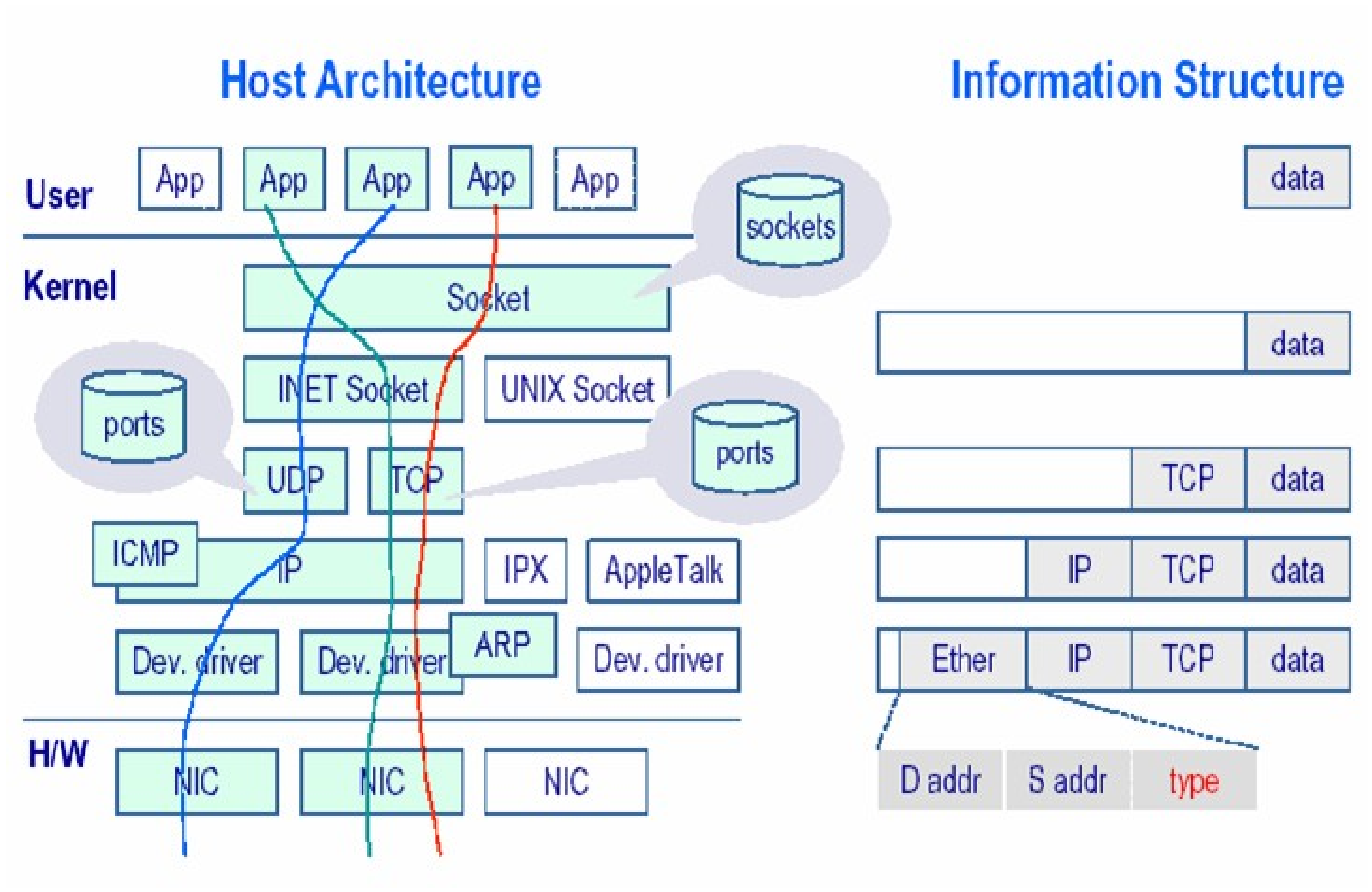
Recall Network Packets

Where does the Socket sit?



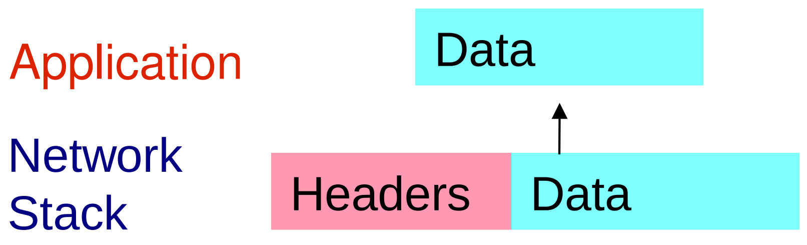
Some slides courtesy of Vivek Ramachandran

The gory details …..



More Details Raw sockets

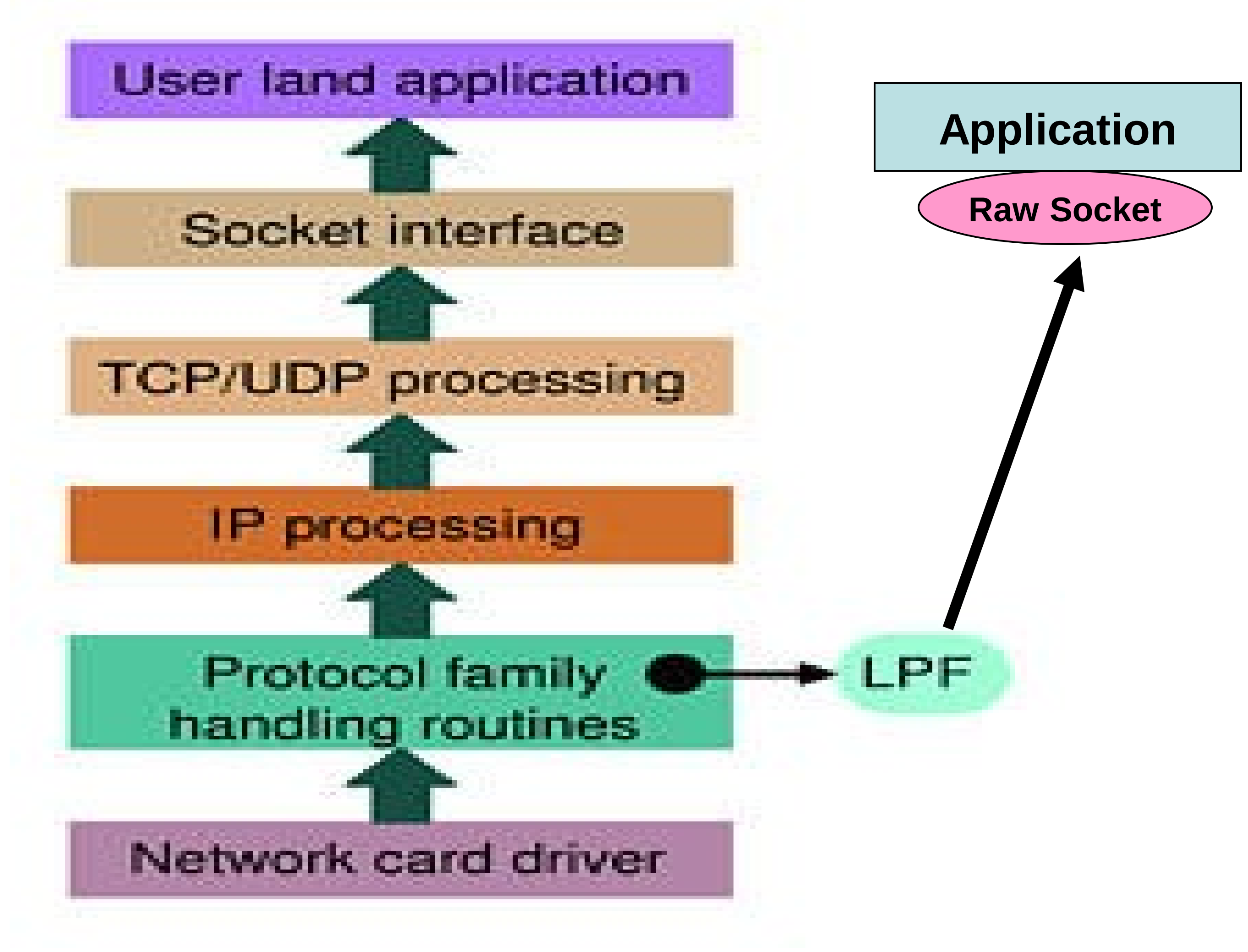
– All Headers i.e. Ethernet, IP, TCP etc are stripped by network stack and only data is shipped to application layer
– We cannot modify packet headers of packets when they are sent out from our host
– Is this a good thing in general?



Sending arbitrary packets – Packet Injection

- We "manufacture" our own packets and send it out on the network.
  - Absolute power!!!
- Total network stack bypass
- Most active network monitoring tools and hacking tools use this.
- Dos attacks ? Syn Floods ? IP Spoofs ?
- Plus, network tools like Wireshark

Raw Sockets – a closer look



Getting All headers - Sniffing

- Note: Way it has worked in past
- Once we set NIC interface to promiscuous mode we can get "**full packets**" with all the headers.
  - We can process these packets and extract data from it
  - Note, we are receiving packets meant for all hosts

Promiscuous Mode of NIC Card

- It is the “See All, Hear All” mode
  - Tells network driver to accept all packets irrespective
    - Used for Network Monitoring – both legal and illegal monitoring
    - We can do this by programmatically setting the IFF_PROMISC flag or
    - Using the ifconfig utility (ifconfig eth0 promisc)

Possible to Inject Packets

- If we could receive frames for all computers connected to our broadcast domain ….
- And, If we could get all the headers
  - Ethernet , TCP, IP etc from the network and analyze them
- Then, we could inject packets with custom headers and data into the network directly

More on Promiscuous Mode

- Questions
- Under what circumstances can we see all packets on a LAN segment?
- Is promiscuous mode truly magic?

More on Promiscuous Mode

- Under what circumstances can we see all packets on a LAN segment?
- Is promiscuous mode truly magic?
- Answer: NO
  - Can see broadcast traffic
  - Can see all traffic if hosts are on a hub
  - Can see all traffic if one switch port is a mirror or spanning port
  - Can see all traffic, if card is able to go into promiscuous mode and LAN is wireless
    - Recall, data should be encrypted these days !!!

Library Support for Raw Sockets

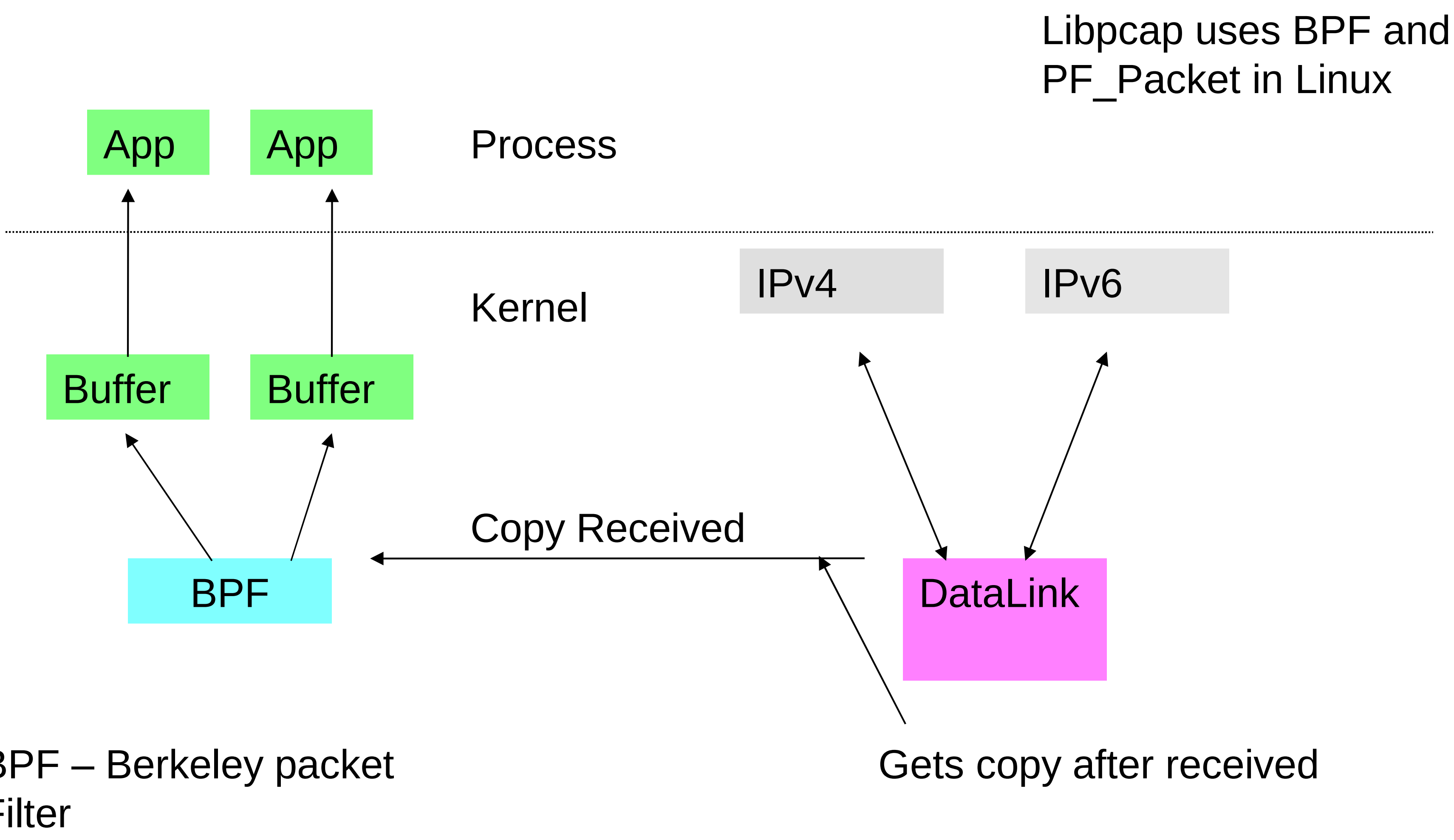
- In Java, no way to get to RAW interfaces in OS kernels with standard Java libraries
  - Two C libraries support standard way of interfacing to network cards
  - Libpcap – Linux/MAC/Unix
  - Winpcap – Windows

  Both use something called the Berkeley Packet Filter - BPF

Link Layer Packet Capture

- Stevens in his classic book, makes a distinction between **Raw Sockets** and capturing packets at the **link layer with a packet filter**
  - See, Unix Network Programming by R. Stevens, B. Fenner and A. Rudoff for details
- For our purposes, since Java doesn't have true RAW socket interface, only way we can capture raw traffic is through link layer packet capture
- Jpcap or jNetPcap running on top of libpcap library is one example of Java special purpose library

Picture of this



Berkeley Packet Filter

- Berkeley Packet Filter (BPF) provides
  - Raw interface to data link layers, permitting raw link-layer packets to be sent and received
  - It is available on most Unix-like operating systems
  - BPF provides pseudo-devices that can be bound to a network interface
    - Reads from device reads buffers full of packets received on network interface, and
    - Writes to device will inject packets on network interface

https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

jNetPcap As an Example

- jNetPcap
  - Open source library for capturing and sending network packets from Java applications
  - Runs on both Linux and Windows

It contains:

- * A Java wrapper for nearly all libpcap library native calls
- * Decodes captured packets in real-time
- * Provides library of network protocols (core protocols)
- * Users can add own protocol definitions using java SDK
- * jNetPcap uses mixture of native and java implementation for optimum packet decoding performance

http://jnetpcap.org/

jNetPcap and Other Java Raw Libraries

- C Language allows native access to Raw Socket interface
- More details are left to you, the programmer
- Must build the headers for IP, TCP, UDP or any new protocol you might create
- More work, but gives you more control
- Look at C example next time ….

Java jNetPcap

- **For Most Programs written Using jNetPcap**
- **First,** you need to acquire a list of available network interfaces for working with the live network
    - Pcap.findAllDevs()
- **Second**, you need to use one of static open calls found in
    - Pcap class, Pcap.openXXXX()
- **Third**, after open call succeeds, do something through return Pcap class instance such as read packets, write packets or acquire some information about network interface
    - Pcap.sendPacket(), Pcap.loop(), Pcap.dispatch()

Java jNetPcap

- Example program
  - **Classic Example**

    Provide a list of devices

    Presents a simple menu

    We will select one for the user

    Using a packet handler, it loops to catch few packets, say 10.

    Prints some simple info about the packets

    Closes the pcap handle and exits

Java jNetPcap - Example

- You need to indicate in jNetPcap which network device you want to listen to
- API provides Pcap.findAllDevs() class

```
//  First get a list of devices on this system
        int r = Pcap.findAllDevs(alldevs, errbuf);
        if (r == Pcap.NOT_OK || alldevs.isEmpty()) {
              System.err.printf("Can't read list of devices, error is %s",
                     errbuf.toString());
               return;
             }

            System.out.println ("Network devices found:");
```

Java jNetPcap

```
//  Print the list of devices on this system
 int i = 0;
 for (PcapIf device : alldevs) {
      String description =
            (device.getDescription() != null) ? device.getDescription()
                : "No description available";
       System.out.printf("#%d: %s [%s]\n", i++, device.getName(),
             description);
  }
  PcapIf device = alldevs.get(0); // We know we have 1 or more device
  System.out.printf("\nChoosing '%s' on your behalf:\n",
              (device.getDescription() != null) ? device.getDescription()
              : device.getName());
```

Java jNetPcap

```
//Second we open up the selected device
      int snaplen = 64 * 1024;      // Capture all packets, no truncation
       int flags = Pcap.MODE_PROMISCUOUS; // capture all
       int timeout = 10 * 1000;                // 10 seconds in millis
       Pcap pcap =
          Pcap.openLive(device.getName(), snaplen, flags, timeout,
              errbuf);

     if (pcap == null) {
          System.err.printf("Error while opening device for capture: "
             + errbuf.toString());
          return;
       }
```

Java jNetPcap

```
// Third we create a packet handler which will receive packets from the libpcap loop.
  PcapPacketHandler<String> jpacketHandler = new PcapPacketHandler<String>() {

      public void nextPacket(PcapPacket packet, String user) {

          System.out.printf("Received packet at %s caplen=%-4d len=%-4d %s\n",
           new Date(packet.getCaptureHeader().timestampInMillis()),
           packet.getCaptureHeader().caplen(),  // Length actually captured
           packet.getCaptureHeader().wirelen(), // Original length
           user                                 // User supplied object
           );
       }
    };
 // Fourth we enter the loop and tell it to capture 10 packets.
   pcap.loop(10, jpacketHandler, "jNetPcap rocks!");
```

JPCAP Example

The packet output of executing the test class looks like this

```
Network devices found:
#0: lo [No description available]
#1: any [Pseudo-device that captures on all interfaces]
#2: wlan0 [No description available]
#3: eth0 [No description available]

Choosing 'wlan0' on your behalf:

Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=66   len=66   jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:19 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
Received packet at Sun Feb 26 20:48:20 PST 2017 caplen=1486 len=1486 jNetPcap rocks!
```

Summary

- Raw sockets through jNetPcap allows capability not built into Java
  - Raw sockets are possible
  - Can write programs that gain access lower level protocols
  - Gives power to you, the programmer!!!
  - Allows for fun in manipulating packets!

References

jNetPcap - Examples

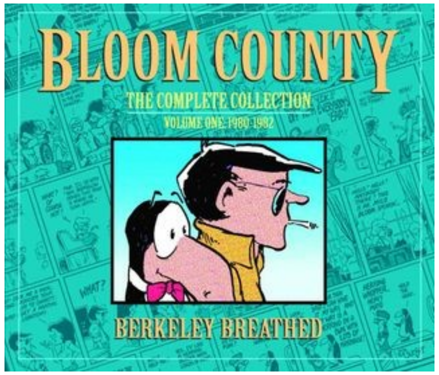
http://jnetpcap.org/examples/

jNetPcap Tutorial

http://jnetpcap.org/tutorial

Capturing network packets – Blog – Windows Example

https://compscipleslab.wordpress.com/2013/02/17/capturing-network-packets-through-java/



Midterm due today ….