Networking, USB, and Threads
CSEE W4840

Prof. Stephen A. Edwards

Columbia University

Fall 2016
Ethernet and the Internet
Ethernet

Started in about 1976 at Xerox PARC

IEEE Standard 802.3

Carrier-sense multiple access/carrier detect protocol:

1. Listen to the cable
2. If nobody’s there, start talking
3. If someone interrupts, stop, and retry after a random time
10Base-5 “Thicknet”

Shared coax bus with “vampire tap” tranceivers

Yellow color suggested by the 802.3 standard

From http://www.turkcenet.org/yerel_htm/10base5.htm
10Base-2 “Thinnet”

50-Ohm coax segments with BNC “T” connectors

Coax invariably black

From http://www.answers.com/topic/10base2
10Base-T and 100Base-T

Put the shared medium in a hub: a star topology. Everybody uses it now.

100Base-TX wiring (CAT 5)

Pair of twisted pairs, one pair for each direction.

Uplink or MDI port

Straight-through twisted pair cable

Normal or MDI-X port

Hub-to-computer cable is straight-through.
Computer-to-computer cable is a “crossover.”

From the Netgear EN104TP 4-port hub manual off of Amazon.com
An Ethernet Frame

<table>
<thead>
<tr>
<th>7 bytes</th>
<th>1</th>
<th>6</th>
<th>6</th>
<th>2</th>
<th>46–1500</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preamble</td>
<td>SOF</td>
<td>Dest.</td>
<td>Src.</td>
<td>Type</td>
<td>Payload</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

- **SOF**  Start of Frame
- **Dest.** Destination address
- **Src.** Source address
- **Type** Type of packet or length of data field
  - 0x0800 for IP, 0x0806 for ARP, etc.

Bytes sent LSB first

Minimum packet length: 64 (6 + 6 + 2 + 46 + 4)

Lengths > 1500 indicate packet type
Ethernet (MAC) addresses

48 bits \( \approx 281 \) trillion (world population: 6.5 billion)

Bits 48–24: Vendor code

Bit 41: 0=ordinary, 1=group (broadcast) address

Bits 23–0: Serial number

On my desktop:

```
$ ifconfig eth0
eth0 Ethernet HWaddr 00:18:f3:ef:2b:36
```

OUI (Organizationally Unique Identifier):

00:18:f3 is ASUS (my machine’s motherboard manufacturer)

Address FF:FF:FF:FF:FF:FF is broadcast
An Ethernet Packet

00d006269c00  Destination MAC address (router)
00087423ccab  Source MAC address (my desktop)
0800          Type = IP packet
45            IPv4, 5 word (20-byte) header
00            Normal service
0028          Total length = 40 bytes
c31c          Identification (unique)
4000          “Don’t Fragment”
40            64 hops to live
06            TCP protocol
3ff1          Header checksum (one’s complement)
803b1372      Source IP 128.59.19.114 (desktop)
40ec6329      Destination IP 64.236.99.41

deac 0050 bf49 9ba6 a1a4 8bed 5010 ffff 1093 0000
# IP Header

<table>
<thead>
<tr>
<th>31</th>
<th>28</th>
<th>27</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>13</th>
<th>12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version = 4</td>
<td>Words in Header</td>
<td>Type of Service (typically 0)</td>
<td>Total number of bytes in the IP packet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identification Number (which packet)</td>
<td>Flags - DF MF</td>
<td>Fragment Offset (which fragment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time-to-Live (hops left)</td>
<td>Protocol 6=TCP, 17=UDP</td>
<td>Header checksum (one’s complement sum)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination IP Address</td>
</tr>
</tbody>
</table>

| Options and padding |
IP Addresses

32 bits \( \approx \) 4 billion (world population: 6.5 billion)

First \( n \) bits indicate network \((n = 8, 16, 24)\)

For example, columbia.edu
owns 128.59.0.0 – 128.59.255.255

Magical addresses:

127.0.0.1         “Me”
192.168.x.x      Never assigned worldwide
10.x.x.x          Never assigned worldwide
255.255.255.255   Broadcast
Sockets
// Create an Internet socket (SOCK_STREAM = TCP)
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Connect to the server
#define IPADDR(a,b,c,d) (htonl(((a)<<24)|((b)<<16)|((c)<<8)|(d)))
#define SERVER_HOST IPADDR(192,168,1,1)
#define SERVER_PORT htons(42000)

struct sockaddr_in serv_addr = { AF_INET,
                                 SERVER_PORT,
                                 { SERVER_HOST } }

connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));

// Write to the socket
write(sockfd, "Hello World!
", 13);

// Read from the socket: block until data arrives
#define BUFFER_SIZE 128
char recvBuf[BUFFER_SIZE];
read(sockfd, &recvBuf, BUFFER_SIZE - 1);
USB: The Universal Serial Bus
USB: Universal Serial Bus

1.5 Mbps, 12 Mbps, 480 Mbps (USB 2.0), 5 Gbps (USB 3.0)

Point-to-point, differential, twisted pair

3–5m maximum cable length
## USB Connectors

<table>
<thead>
<tr>
<th>Series &quot;A&quot; Connectors</th>
<th>Series &quot;B&quot; Connectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series &quot;A&quot; plugs are always oriented <strong>upstream</strong> towards the <em>Host System</em></td>
<td>Series &quot;B&quot; plugs are always oriented <strong>downstream</strong> towards the <em>USB Device</em></td>
</tr>
</tbody>
</table>

"A" Plugs  
(From the *USB Device*)

"B" Plugs 
(From the *Host System*)

"A" Receptacles 
(Downstream Output from the *USB Host or Hub*)

"B" Receptacles 
(Upstream Input to the *USB Device or Hub*)
## USB signaling

NRZI: 0 = toggle, 1 = no change

Bit stuffing: 0 automatically inserted after six consecutive 1s

Each packet prefixed by a SYNC field: 3 0s followed by two 1s

Low- vs. full-speed devices identified by different pull-ups on D+/D- lines
USB Packets

Always start with SYNC

Then 4-bit type, 4-bit type complemented

2 bits distinguish Token, Data, Handshake, and Special, other two bits select sub-types

Then data, depending on packet type

Data checked using a CRC

Addresses (1-128) assigned by bus master, each with 16 possible endpoints
Polled bus: host initiates all transfers.

Most transactions involve three packets:

- "Token" packet from host requesting data
- Data packet from target
- Acknowledge from host

Supports both streams of bytes and structured messages (e.g., control changes).
USB Data Flow Types

- Control
  For configuration, etc.

- Bulk Data
  Arbitrary data stream: bursty

- Interrupt Data
  Timely, reliable delivery of data. Usually events.

- Isochronous Data
  For streaming real-time transfer: prenegotiated bandwidth and latency
USB Physical Bus Topology

Source: http://www.usblyzer.com/usb-topology.htm
**lsusb output**

Front: USB keyboard  
Back: IR receiver  
Back: Monitor w/ webcam, microphone (internal hub)  
Back: 7-port hub w/  
  SD card reader  
  Bluetooth dongle  
  SoCKit board (USB Blaster/JTAG)  
  SoCKit board (Serial debugging)

<table>
<thead>
<tr>
<th>Bus</th>
<th>Device</th>
<th>ID</th>
<th>Manufacturer/Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>002</td>
<td>001</td>
<td>1d6b:0001</td>
<td>Linux Foundation 1.1 root hub</td>
</tr>
<tr>
<td>002</td>
<td>002</td>
<td>0471:0815</td>
<td>Philips (or NXP) eHome Infrared Receiver</td>
</tr>
<tr>
<td>002</td>
<td>006</td>
<td>04d9:1203</td>
<td>Holtek Semiconductor, Inc. Keyboard</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>1d6b:0002</td>
<td>Linux Foundation 2.0 root hub</td>
</tr>
<tr>
<td>001</td>
<td>002</td>
<td>0409:005a</td>
<td>NEC Corp. HighSpeed Hub</td>
</tr>
<tr>
<td>001</td>
<td>039</td>
<td>03f0:b116</td>
<td>Hewlett-Packard Webcam</td>
</tr>
<tr>
<td>001</td>
<td>005</td>
<td>0409:005a</td>
<td>NEC Corp. HighSpeed Hub</td>
</tr>
<tr>
<td>001</td>
<td>041</td>
<td>03f0:3724</td>
<td>Hewlett-Packard Webcam</td>
</tr>
<tr>
<td>001</td>
<td>004</td>
<td>04cc:1521</td>
<td>ST-Ericsson USB 2.0 Hub</td>
</tr>
<tr>
<td>001</td>
<td>006</td>
<td>0bda:0119</td>
<td>Realtek Semiconductor Corp. Storage Device (SD card reader)</td>
</tr>
<tr>
<td>001</td>
<td>007</td>
<td>0a5c:2101</td>
<td>Broadcom Corp. BCM2045 Bluetooth</td>
</tr>
<tr>
<td>001</td>
<td>042</td>
<td>09fb:6810</td>
<td>Altera</td>
</tr>
<tr>
<td>001</td>
<td>043</td>
<td>0403:6001</td>
<td>Future Technology Devices International, Ltd FT232 USB-Serial</td>
</tr>
</tbody>
</table>
Isusb -t output

Front: USB keyboard
Back: IR receiver
Back: Monitor w/ webcam, microphone (internal hub)
Back: 7-port hub w/
   SD card reader
   Bluetooth dongle
   SoCKit board (USB Blaster/JTAG)
   SoCKit board (Serial debugging)

/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ohci-pci/10p, 12M
   |_ Port 3: Dev 2, If 0, Class=Vendor Specific Class, Driver=mceusb, 12M
   |_ Port 5: Dev 6, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
   |_ Port 5: Dev 6, If 1, Class=Human Interface Device, Driver=usbhid, 1.5M
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/10p, 480M
   |_ Port 2: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
      |_ Port 3: Dev 39, If 0, Class=Video, Driver=uvcvideo, 480M
      |_ Port 3: Dev 39, If 1, Class=Video, Driver=uvcvideo, 480M
      |_ Port 3: Dev 39, If 2, Class=Audio, Driver=snd-usb-audio, 480M
      |_ Port 3: Dev 39, If 3, Class=Audio, Driver=snd-usb-audio, 480M
      |_ Port 4: Dev 5, If 0, Class=Hub, Driver=hub/2p, 480M
         |_ Port 2: Dev 41, If 0, Class=Mass Storage, Driver=usb-storage, 480M
      |_ Port 4: Dev 4, If 0, Class=Hub, Driver=hub/7p, 480M
      |_ Port 2: Dev 6, If 0, Class=Mass Storage, Driver=usb-storage, 480M
      |_ Port 3: Dev 7, If 0, Class=Wireless, Driver=btusb, 12M
      |_ Port 3: Dev 7, If 1, Class=Wireless, Driver=btusb, 12M
      |_ Port 3: Dev 7, If 2, Class=Vendor Specific Class, Driver=, 12M
      |_ Port 3: Dev 7, If 3, Class=Application Specific Interface, Driver=, 12M
      |_ Port 5: Dev 42, If 0, Class=Vendor Specific Class, Driver=, 480M
      |_ Port 6: Dev 43, If 0, Class=Vendor Specific Class, Driver=ftdi_sio, 12M
Devices, Configurations, Interfaces, and Endpoints

- **Devices**: (Keyboards, Mice: physical object)
- **Configurations**: (usually one)
- **Interfaces**: ("logical device": usually one; my webcam has 4)
- **Endpoints**: (one per input/output stream)
USB Addresses and Endpoints

Source: http://www.beyondlogic.org/usbnutshell/usb3.shtml
USB Keyboard: lsusb (highlights)

Bus 002 Device 007: ID 413c:2003 Dell Computer Corp. Keyboard

Device Descriptor:
  bDeviceClass 0 (Defined at Interface level)
  idVendor 0x413c Dell Computer Corp.
  idProduct 0x2003 Keyboard
  bNumConfigurations 1

Configuration Descriptor:
  bNumInterfaces 1

Interface Descriptor:
  bInterfaceNumber 0
  bNumEndpoints 1
  bInterfaceClass 3 Human Interface Device
  bInterfaceSubClass 1 Boot Interface Subclass
  bInterfaceProtocol 1 Keyboard
  iInterface 0

HID Device Descriptor:
  bcdHID 1.10
  bNumDescriptors 1
  bDescriptorType 34 Report
  wDescriptorLength 65

Endpoint Descriptor:
  bEndpointAddress 0x81 EP 1 IN
  bmAttributes 3
    Transfer Type Interrupt
    Synch Type None
    Usage Type Data
  wMaxPacketSize 0x0008 1x 8 bytes
USB SD Card Reader: lsusb (highlights)

Bus 001 Device 006: ID 0bda:0119 Realtek Semiconductor Corp. Storage Device (SD card reader)

Device Descriptor:
- bDeviceClass: 0 (Defined at Interface level)
- idVendor: 0x0bda Realtek Semiconductor Corp.
- idProduct: 0x0119 Storage Device (SD card reader)
- bNumConfigurations: 1

Configuration Descriptor:
- bNumInterfaces: 1
- bConfigurationValue: 1
- iConfiguration: 4 CARD READER
- bmAttributes: 0x80 (Bus Powered)
- MaxPower: 500mA

Interface Descriptor:
- bNumEndpoints: 2
- bInterfaceClass: 8 Mass Storage
- bInterfaceSubClass: 6 SCSI
- bInterfaceProtocol: 80 Bulk-Only

Endpoint Descriptor:
- bEndpointAddress: 0x01 EP 1 OUT
  - bmAttributes: 2
    - Transfer Type: Bulk
    - Synch Type: None
    - Usage Type: Data
  - wMaxPacketSize: 0x0200 1x 512 bytes

Endpoint Descriptor:
- bEndpointAddress: 0x82 EP 2 IN
  - bmAttributes: 2
    - Transfer Type: Bulk
    - Synch Type: None
    - Usage Type: Data
  - wMaxPacketSize: 0x0200 1x 512 bytes
libusb 1.0
Libusb 1.0

User-level C library for USB device access. lsusb built on it.

www.libusb.org

1.0 API supplants earlier libusb 0.1

Nice tutorial: http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
Using libusb

1. Initialize the library with `libusb_init()`
2. Select your device from the list returned by `libusb_get_device_list()`. Later, free the list with `libusb_free_device_list()`.
3. Initiate contact with `libusb_open()`
4. Claim the interface with `libusb_claim_interface()`
5. Communicate using the various `libusb_. . . _transfer()` functions
6. Release the interface with `libusb_release_interface()`
7. Close the device with `libusb_close()`
8. Close the library with `libusb_exit()`
libusb: Finding a Keyboard

```c
libusb_device **devs;
struct libusb_device_descriptor desc;
struct libusb_device_handle *keyboard = NULL;
ssize_t num_devs, d; uint8_t i, k;
uint8_t *endpoint_address;

num_devs = libusb_get_device_list(NULL, &devs);
for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    libusb_get_device_descriptor(dev, &desc);

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
        struct libusb_config_descriptor *config;
        libusb_get_config_descriptor(dev, 0, &config);
        for (i = 0 ; i < config->bNumInterfaces ; i++)
            for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
                const struct libusb_interface_descriptor *inter =
                    config->interface[i].altsetting + k;
                if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
                    inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
                    libusb_open(dev, &keyboard);
                    *endpoint_address = inter->endpoint[0].bEndpointAddress;
                    libusb_claim_interface(keyboard, i);
                    libusb_free_device_list(devs, 1);
                    return keyboard;
                }
            }
    }
}
```
libusb: Reading from a Keyboard

```c
#define USB_LCTRL (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT (1 << 2)
#define USB_LGUI (1 << 3)
#define USB_RCTRL (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT (1 << 6)
#define USB_RGUI (1 << 7)

struct usb_keyboard_packet {
    uint8_t modifiers;
    uint8_t reserved;
    uint8_t keycode[6];
};

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

libusb_interrupt_transfer(keyboard, endpoint_address,
                          (unsigned char *) &packet,
                          sizeof(packet),
                          &transferred, 0);

if (transferred == sizeof(packet))
    // Got a new keyboard event
```
<table>
<thead>
<tr>
<th>Byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Modifier keys</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Keycode 1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Keycode 6</td>
</tr>
</tbody>
</table>
## USB HID Keycodes

Page 53: [http://www.usb.org/developers/devclass_docs/Hut1_12v2.pdf](http://www.usb.org/developers/devclass_docs/Hut1_12v2.pdf)

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Hex Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No event</td>
<td>00 00 00 00 00...00</td>
<td>Nothing pressed</td>
</tr>
<tr>
<td>4</td>
<td>a or A</td>
<td>00 00 04 00 00...00</td>
<td>“A” pressed</td>
</tr>
<tr>
<td>5</td>
<td>b or B</td>
<td>02 00 04 00 00...00</td>
<td>Shift+A</td>
</tr>
<tr>
<td>29</td>
<td>z or Z</td>
<td>03 00 04 00 00...00</td>
<td>Shift+Ctrl+A</td>
</tr>
<tr>
<td>30</td>
<td>1 or !</td>
<td>02 00 04 00 00...00</td>
<td>Shift+A</td>
</tr>
<tr>
<td>38</td>
<td>9 or (</td>
<td>02 00 04 05 00...00</td>
<td>Shift+A+B</td>
</tr>
<tr>
<td>39</td>
<td>0 or )</td>
<td>02 00 04 05 00...00</td>
<td>Shift+A+B</td>
</tr>
</tbody>
</table>
POSIX Threads (pthreads)
#include <stdio.h>
#include <pthread.h>

void *mythread(void *ptr)
{
    printf("%s\n", (char *)ptr);
    return NULL;
}

int main()
{
    pthread_t thread1, thread2;
    const char *message1 = "Thread_1", *message2 = "Thread_2";
    pthread_create( &thread1, NULL, mythread, (void *)message1);
    pthread_create( &thread2, NULL, mythread, (void *)message2);
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
    return 0;
}
# Mutexes

```c
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0; /* Caution: shared variable */

void *incCounter() {
    int tmp;
    pthread_mutex_lock(&mutex1); /* Grab the lock */
    tmp = counter; /* Needlessly complicated to make a point */
    tmp = tmp + 1;
    counter = tmp;
    pthread_mutex_unlock(&mutex1); /* Release the lock */
    return NULL;
}

int main() {
    pthread_t thread1, thread2;
    pthread_create( &thread1, NULL, &incCounter, NULL);
    pthread_create( &thread2, NULL, &incCounter, NULL);
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
    return 0;
}
```
Condition Variables

```c
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond1 = PTHREAD_COND_INITIALIZER;
int count; int valid = 0;

void *writeCounter() {
    int i;
    for (i = 0 ; i < 10 ; i++) {
        pthread_mutex_lock(&mutex1);
        while (valid) pthread_cond_wait(&cond1, &mutex1);
        count = i; valid = 1;
        pthread_cond_signal(&cond1);
        pthread_mutex_unlock(&mutex1);
    }
    return NULL; }

void *readCounter() {
    int done = 0;
    do {
        pthread_mutex_lock(&mutex1);
        while (!valid) pthread_cond_wait(&cond1, &mutex1);
        printf("%d\n", count);
        valid = 0; done = count == 9;
        pthread_cond_signal(&cond1);
        pthread_mutex_unlock(&mutex1);
    } while (!done);
    return NULL; }
```