Ethernet and the Internet

Sockets

USB: The Universal Serial Bus

libusb 1.0

POSIX Threads (pthreads)
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  802.3 std. suggests yellow
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: star topology

Everybody uses it now

Choice of colors

100Base-TX wiring (CAT 5)

Pair of twisted pairs, one pair for each direction.

- **Uplink or MDI port**
- **Normal or MDI-X port**

Hub-to-computer cable is straight-through.
Computer-to-computer cable is a “crossover;” most hardware can now adapt.

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 ≈ 281 trillion (world population: 7.9 billion)

- Bits 48–24  Vendor code (OUI)
- Bit 41  0=ordinary, 1=group (broadcast) address
- Bits 23–0  Serial number

On one of my machines:

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

OUI (Organizationally Unique Identifier):

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

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

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00d0629c00</td>
<td>Destination MAC address (router)</td>
</tr>
<tr>
<td>00087423ccab</td>
<td>Source MAC address (desktop)</td>
</tr>
<tr>
<td>0800</td>
<td>Type = IP packet</td>
</tr>
<tr>
<td>45</td>
<td>IPv4, 5 word (20-byte) header</td>
</tr>
<tr>
<td>00</td>
<td>Normal service</td>
</tr>
<tr>
<td>0028</td>
<td>Total length = 40 bytes</td>
</tr>
<tr>
<td>c31c</td>
<td>Identification (unique)</td>
</tr>
<tr>
<td>4000</td>
<td>“Don’t Fragment”</td>
</tr>
<tr>
<td>40</td>
<td>64 hops to live</td>
</tr>
<tr>
<td>06</td>
<td>TCP protocol</td>
</tr>
<tr>
<td>3ff1</td>
<td>Header checksum (one’s complement)</td>
</tr>
<tr>
<td>803b1372</td>
<td>Source IP 128.59.19.114 (desktop)</td>
</tr>
<tr>
<td>40ec6329</td>
<td>Destination IP 64.236.99.41</td>
</tr>
<tr>
<td>deac 0050 bf49 9ba6 a1a4 8bed 5010 ffff 1093 0000</td>
<td>(payload)</td>
</tr>
</tbody>
</table>
# 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><strong>Version</strong> = 4</td>
<td><strong>Words in Header</strong></td>
<td><strong>Type of Service</strong> (typically 0)</td>
<td><strong>Total number of bytes in the IP packet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Identification Number</strong> (which packet)</td>
<td><strong>Flags</strong> - DF MF</td>
<td><strong>Fragment Offset</strong> (which fragment)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Time-to-Live</strong> (hops left)</td>
<td><strong>Protocol</strong> 6=TCP, 17=UDP</td>
<td><strong>Header checksum</strong> (one’s complement sum)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Source IP Address</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Destination IP Address</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Options and padding</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
IP Addresses

32 bits ≈ 4 billion (world population: 7.9 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
Sockets

// Create an Internet socket (SOCK_STREAM = TCP)
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

#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) // host to network byte order short

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

// Connect to the server
connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr));

// Write to the socket
write(sockfd, "Hello_World!\n", 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; two additional pairs)

Point-to-point, differential, twisted pair

3–5m maximum cable length
<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 <strong>Host System</strong></td>
<td>✦ Series &quot;B&quot; plugs are always oriented <strong>downstream</strong> towards the <strong>USB Device</strong></td>
</tr>
<tr>
<td><img src="image" alt="&quot;A&quot; Plugs (From the USB Device)" /></td>
<td><img src="image" alt="&quot;B&quot; Plugs (From the Host System)" /></td>
</tr>
<tr>
<td><img src="image" alt="&quot;A&quot; Receptacles (Downstream Output from the USB Host or Hub)" /></td>
<td><img src="image" alt="&quot;B&quot; Receptacles (Upstream Input to the USB Device or Hub)" /></td>
</tr>
</tbody>
</table>
USB signaling

NRZI: 0 = toggle, 1 = no change

Bit stuffing: 0 automatically inserted after six consecutive 1s

Each packet prefixed by a SYNC field: 0 0 0 1 1

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

Always start with SYNC

4-bit type; 4-bit type complemented

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

Data, depending on packet type

Data checked using a CRC

Addresses (1-128) assigned by bus master, each with 16 possible endpoints
Polled bus (USB 1.0, 2.0): host initiates all transfers. USB 3.0 is full-duplex.

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 Bus Topology

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

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

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

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

/: 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 4, 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

<table>
<thead>
<tr>
<th>Devices</th>
<th>Keyboards, Mice: physical object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td>usually one</td>
</tr>
<tr>
<td>Interfaces</td>
<td>“logical device”: usually one; my webcam has 4</td>
</tr>
<tr>
<td>Endpoints</td>
<td>one per input/output stream</td>
</tr>
</tbody>
</table>
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
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:
  bLength 7
  bDescriptorType 5
  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

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;
                }
            }
    }
}
#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

| Code | Meaning                          | 00 00 00 00 00 00...00 | 00 00 04 00 00...00 | 02 00 04 00 00...00 | 03 00 04 00 00...00 | 02 00 04 05 00...00 | 02 00 04 05 00...00 |
|------|---------------------------------|------------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 0    | No event                        |                        |                    |                    |                    |                    |                    |                    |
| 4    | a or A                          |                        |                    |                    |                    |                    |                    |                    |
| 5    | b or B                          |                        |                    |                    |                    |                    |                    |                    |
| 29   | z or Z                          |                        |                    |                    |                    |                    |                    |                    |
| 30   | 1 or !                          |                        |                    |                    |                    |                    |                    |                    |
| 38   | 9 or (                           |                        |                    |                    |                    |                    |                    |                    |
| 39   | 0 or )                           |                        |                    |                    |                    |                    |                    |                    |
| 40   | Enter                           |                        |                    |                    |                    |                    |                    |                    |
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: Ensuring atomic access

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

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
tint 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: Notifying waiters

```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; }
```