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

From http://www.turkcenet.org/yerel_htm/10base5.htm
10Base-2 “Thinnet”
50-Ohm coax segments with BNC “T” connectors

From Computer Desktop Encyclopedia
10Base-T and 100Base-T

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

Star topology

Choice of colors

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

Tx

1

2

Rx

3

6

1

2

3

Tx

6
An Ethernet Frame

<table>
<thead>
<tr>
<th></th>
<th>Preamble</th>
<th>SOF</th>
<th>Dest.</th>
<th>Src.</th>
<th>Type</th>
<th>Payload</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bytes</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>46–1500</td>
<td>4</td>
<td></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: 6.5 billion)
Bits 48–24: Vendor code
Bit 41: 0=ordinary, 1=group (broadcast) address
Bits 23–0: Serial number

On my desktop:

```bash
$ 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>Bit Position</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-28</td>
<td>Version</td>
<td>Version of IP protocol (typically 4)</td>
</tr>
<tr>
<td>27-24</td>
<td>Words in Header</td>
<td>Number of 32-bit words in the IP packet</td>
</tr>
<tr>
<td>23</td>
<td>Type of Service</td>
<td>Type of transport protocol (typically 0)</td>
</tr>
<tr>
<td>16-15</td>
<td>Total number of bytes</td>
<td>Total number of bytes in the IP packet</td>
</tr>
<tr>
<td>13-12</td>
<td>Identification Number</td>
<td>Identification number of packet</td>
</tr>
<tr>
<td>12</td>
<td>Flags</td>
<td>Flags: DF (Don't Fragment), MF (More Fragments)</td>
</tr>
<tr>
<td>11-8</td>
<td>Fragment Offset</td>
<td>Fragment offset (which fragment)</td>
</tr>
<tr>
<td>7-4</td>
<td>Time-to-Live</td>
<td>Time-to-Live (hops left)</td>
</tr>
<tr>
<td>3-0</td>
<td>Protocol</td>
<td>Protocol: 6 (TCP), 17 (UDP)</td>
</tr>
<tr>
<td></td>
<td>Header checksum</td>
<td>One's complement sum of the header</td>
</tr>
<tr>
<td>Source IP Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination IP Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Options and padding
IP Addresses

32 bits \approx 4 \text{ billion} \ (\text{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!\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)

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 <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 Host 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
Layered Architecture

- **Host**
  - Client SW
  - USB System SW
  - USB Host Controller

- **Interconnect**
  - Function
  - USB Logical Device
  - USB Bus Interface

- **Physical Device**
  - Function Layer
  - USB Device Layer
  - USB Bus Interface Layer

- Actual communications flow
- Logical communications flow
- Implementation Focus Area
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 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 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
## Devices, Configurations, Interfaces, and Endpoints

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices</td>
<td>(Keyboards, Mice: physical object)</td>
</tr>
<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
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:
  bLength 7
  bDescriptorType 5
  bEndpointAddress 0x82 EP 2 IN
  bmAttributes 2
    Transfer Type Bulk
    Synch Type None
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;
                }
            }
    }
}
```


```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>
<tr>
<td>Code</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
</tr>
<tr>
<td>0</td>
<td>No event</td>
</tr>
<tr>
<td>4</td>
<td>a or A</td>
</tr>
<tr>
<td>5</td>
<td>b or B</td>
</tr>
<tr>
<td>29</td>
<td>z or Z</td>
</tr>
<tr>
<td>30</td>
<td>1 or !</td>
</tr>
<tr>
<td>38</td>
<td>9 or (</td>
</tr>
<tr>
<td>39</td>
<td>0 or )</td>
</tr>
<tr>
<td>40</td>
<td>Enter</td>
</tr>
</tbody>
</table>

00 00 00 00 00...00          Nothing pressed
00 00 04 00 00...00          "A" pressed
02 00 04 00 00...00          Shift+A
03 00 04 00 00...00          Shift+Ctrl+A
02 00 04 00 00...00          Shift+A
02 00 04 05 00...00          Shift+A+B
POSIX Threads (pthreads)
Creation and Termination

```c
#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;
}
```

http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
#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; }
```