Hacking 10Base–T Ethernet for Underwater Optical Communication

UC the Fish

April 10, 2016

1 Introduction

One of the funtional requirements for UC the Fish was modular optic communication, or the communication system should be easy to integrate with devices that need to communicate; a plug–and–play solution.

Nearly every modern computer has at least 10Base–T Ethernet and USB 2.0 connectivity, so any "modular" communication attachement should use one of these protocols–at least at its endpoints. We chose to 10Base–T ethernet, not just at the interfaces but to send the ethernet signal itself through water in the form of intensity of blue light. This allowed us to focus on building blue light transmit and receive hardware instead of attempting to develop light transmit/receive hardware *and* a modulation protocol + logic giving it USB endpoints.

Ethernet allows multiple stations to share a single medium, which is appropriate for water because light spreads in every direction and data cannot be multiplexed over different wavelegth channels. It includes hardware support for cyclic redundancy bit checking, up to 16 retransmission attempts in the case of bit errors or disruption of the medium, and at 10 MHz, ethernet is fast enough for live streaming video.

Unlike USB, a temporary dissruption in the data link does not require renegotiation time and application level link management. Software applications can use the operating system to handle TCP protocol when data integrity and delivery aknowledgement are necessary, instead of writing custom code to ensure control commands reach the sub intact.

2 Getting the Standard

IEEE Std 802.3 is the standard for Ethernet communication. It is freely available online but is so large it is split into multiple sections. The first section (only 555 pages!) is available at `https://standards.ieee.org/getieee802/download/802.3-2012_section1.pdf`.

3 Ethernet 101 [1]

Ethernet is a time–domain communication protocol for connecting any number of machines via a single, shared medium. Each *station* is given a unique address when it is manufactured and data is sent serially–1 bit at a time–in *packets* between stations.

PREAMBLE 1010....1010	SYNCH 11	DA	SA	TYPE	DATA	FCS
62 BITS	2 BITS	6 BYTES	6 BYTES	2 BYTES	46-1500 BYTES	4 BYTES

Figure 1: Format of an Ethernet Packet.

Time sharing of the medium is done by all stations following *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) protocol. Only one station may transmit at a time and all stations must constantly monitor the line, looking for packets addressed to them and seeing when the line is open for transmission (CSMA). After completion of a packet, all stations must wait an *interpacket gap* before transmitting a packet of their own. If two stations transmit at the same time, both must detect a collision and backoff for a period of time (/CD). A pseudorandom exponential backoff algorithm is used to make repeat collisions unlikely. Retransmission is attempted up to 16 times when collisions occur.

4 More Details from the Standard

5 Influence on Transmitter/Receiver Design

1. The first component in the light transmitter input and last component of the receiver output should be a 1:1, ethernet approved transformer and standard RJ45 connector.
2. The input impedance of light transmitter must be $100\,\Omega$.
3. The bandwidth of both light transmitter and receiver must exceed $10\,MHz$ by several harmonics, and
4. their phase shift between $5\,MHz$ and $10\,MHz$ should be negligible.
5. The magnitude response of light transmitter + receiver in series, to a $100\,ns$, $585\,mV$ step input, should be $\geq 585\,mV$ when loaded by $100\,\Omega$.

[1] Or in other words, an Ethernet Preamble. Hehe.

6 Limitations for Underwater Communication

There are two limitations associated with sending 10Base–T ethernet signals, unmodified, through water by modulating light intensity. Both limitations are related to the separation of transmit and receive signals in 10Base–T networks how the stations detect collisions.

First, is the isolation of transmit and receive signals in twisted–pair media.

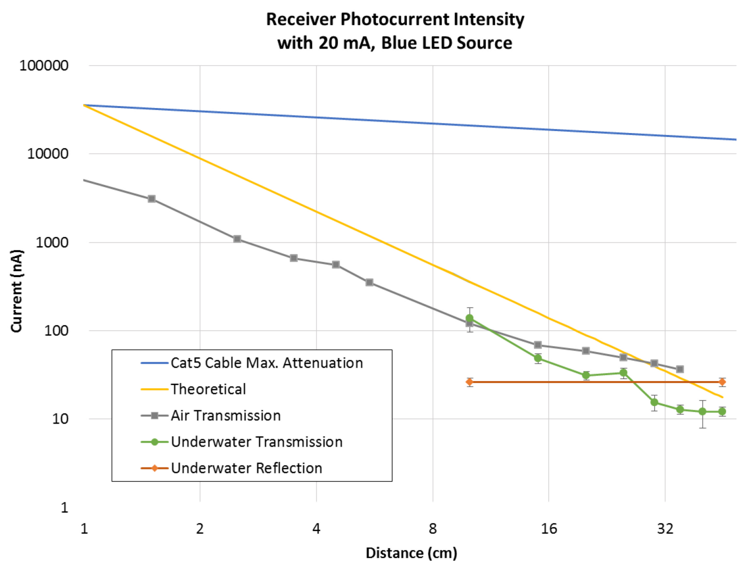

Figure 2: The Crossover Point: distance at which reflected light from station A's own transmitter is more intense than light arriving from far–away station B's.

Second, the *link integrity test* specified in the standard for twisted–pair media.

A Ethernet Test Scripts

```
# Makefile for Ethernet Broadcast and Listen Test Programs
all: broadcast_packet ethernet_listen
#
broadcast_packet: broadcast_packet.o
        gcc broadcast_packet.o -o broadcast_packet.exe
#
broadcast_packet.o: broadcast_packet.c
        gcc -c broadcast_packet.c
#
ethernet_listen: ethernet_listen.o
        gcc ethernet_listen.o -o ethernet_listen.exe
#
ethernet_listen.o: ethernet_listen.c
        gcc -c ethernet_listen.c
#
clean:
        rm ethernet_listen.o broadcast_packet.o ethernet_listen.exe broadcast_pa
#
# end of Makefile
```

```c
/* broadcast_packet.c
 *
 * Builds an 802.3 ethernet frame and then uses Linux sockets to broadcast
 * this packet to all devices on the network. The user passes the data to be
 * sent and the name of the ethernet interface which should be used.
 *
 * Use a packet analyzer like Wireshark to confirm packets are leaving the OS.
 *
 * Code adapted from two webpages:
 * hacked10bits.blogspot.com/2011/12/sending-raw-ethernet-frames-in-6-easy.html
 * aschauf.landshut.org/fh/linux/udp_vs_raw/
 * and from Encyclopedia of Telecommunications Volume 9 by Froelich and Kent.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h> /* htons () */
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
#include <sys/ioctl.h>

union eth_frame
{
  struct
  {
    unsigned char dest_addr[6]; /* ETH_ALEN = 6 */
    unsigned char src_addr[6];
    unsigned char length[2];
    unsigned char data[1518 - 6 - 6 - 2 - 4];
    unsigned char fcs[4]; /* see note 1 */
  } field;
  unsigned char buffer[1518]; /* 1518 is maximum allowed Eth frame length */
};

/* Note 1:
 * An ethernet "packet" is a 5 MHz preamble + synchronization field of
 * 64 bits, followed by an ethernet "frame". The higher level software
 * (this program or in other cases the OS's TCP/IP stack) is responsible
 * for passing a frame with the fields dest_addr, src_addr, length, and data
 * complete. The ethernet hardware is then responsible for wrapping the
 * supplied frame with preamble+synchronization and a frame check sequence
 * that is computed in hardware.
 * The frame check sequence immediately follows the last byte of data,
```

```
 * however long the data may be. In the struct above, it is only located
 * after a full data array for human readability.
 */

const char BROADCAST_ADDRESS[6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

int main (int argc, char *argv[])
{
  char interface_name[1000];
  int interface_index;
  union eth_frame frame1;
  int data_length;
  memset (&frame1, 0, sizeof (frame1));

  /* Get Ethernet Interface Name and Data to Broadcast from User */
  if (argc != 3)
  {
    printf ("Usage: %s <ethernet interface> <data>\n", argv[0]);
    printf ("  where <ethernet interface> is the human readable name of the");
    printf (" sytems' ethernet.\n");
    printf ("  Get the name with command \"$ ifconfig | more\".\n");
    printf ("  <data> is a character array to be broadcast.\n");
    printf ("  Example:\n");
    printf ("  $ %s eth0 \"Hello World!\"\n", argv[0]);
    exit (EXIT_FAILURE);
  }
  else
  {
    /* Copy the <ethernet interface> name provided by User */
    strcpy (interface_name, argv[1]);
    /* Copy the <data> provided by user into the Ethernet Frame */
    strcpy (frame1.field.data, argv[2]);
    data_length = strlen (argv[2]);
    frame1.field.length[0] = data_length / 256; /* in Network Byte Order */
    frame1.field.length[1] = data_length % 256;
  }

  /* Set Destination Address to Broadcast */
  memcpy (frame1.field.dest_addr, BROADCAST_ADDRESS, 6);

  /* Open an Ethernet Socket */
  int sfd;
  sfd = socket (AF_PACKET, SOCK_RAW, htons (ETH_P_ALL));
  if (sfd < 0)
  {
    fprintf (stderr, "socket() error %d\n", sfd);
```

```
    fprintf (stderr, "(try running with sudo)\n");
    exit (EXIT_FAILURE);
  }

  /* Look Up the Ethernet Interface Index */
  struct ifreq interface_req;
  memset (&interface_req, 0x00, sizeof (interface_req));
  strcpy (interface_req.ifr_name, interface_name);
  if (ioctl (sfd, SIOCGIFINDEX, &interface_req) < 0)
  {
    fprintf (stderr, "ioctl(SIOCGIFINDEX) error.\n");
    exit (EXIT_FAILURE);
  }
  interface_index = interface_req.ifr_ifindex;
/*printf ("Index of interface \"%s\": %d\n", interface_name, interface_index);
*/

  /* Look Up the Source MAC Address */
  unsigned char mac[ETH_ALEN];
  if (ioctl (sfd, SIOCGIFHWADDR, &interface_req) < 0)
  {
    fprintf (stderr, "ioctl (SIOCGIFHWADDR) error.\n");
    exit (EXIT_FAILURE);
  }
  memcpy ( (void*)mac, (void*)(interface_req.ifr_hwaddr.sa_data), ETH_ALEN);
/*printf ("Source MAC Address %d:%d:", (int) mac[0], (int) mac[1]);
  printf ("%d:%d:%d:%d\n", (int)mac[2], (int)mac[3], (int)mac[4], (int)mac[5]);
*/

  /* Set the Source Address */
  memcpy (frame1.field.src_addr, mac, 6);

  /* Prepare sockaddr_ll Structure for sendto() */
  struct sockaddr_ll saddr;
  saddr.sll_family = PF_PACKET; /* repitition that this is a packet socket */
  saddr.sll_ifindex = interface_index;
  saddr.sll_halen = ETH_ALEN; /* confirms that ethernet address are 6 bytes */
  memcpy ( (void*)(saddr.sll_addr) , (void*)BROADCAST_ADDRESS, ETH_ALEN);

  /* Send Packet */
  int sent, f_len;
  f_len = data_length + ETH_HLEN;
  sent = sendto (sfd, frame1.buffer, f_len, 0, (struct sockaddr*)&saddr, sizeof(
  if (sent <= 0)
  {
    fprintf (stderr, "sendto() fails %d\n", sent);
```

```
    exit (EXIT_FAILURE);
  }
  if (sent != f_len)
  {
    fprintf (stderr, "incomplete transmission; %d of %d bytes\n", sent, f_len);
    exit (EXIT_FAILURE);
  }

  /* Close the Socket */
  close (sfd);

  return EXIT_SUCCESS;
}
```

```c
/* ethernet_listen.c
 *
 * Builds an 802.3 ethernet frame and then uses Linux sockets to broadcast
 * this packet to all devices on the network. The user passes the data to be
 * sent and the name of the ethernet interface which should be used.
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h> /* htons() */
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
#include <sys/ioctl.h>

union eth_frame
{
  struct
  {
    unsigned char dest_addr[6]; /* ETH_ALEN = 6 */
    unsigned char src_addr[6];
    unsigned char length[2];
    unsigned char data[1518 - 6 - 6 - 2 - 4];
    unsigned char fcs[4]; /* see note 1 */
  } field;
  unsigned char buffer[1518]; /* 1518 is maximum allowed Eth frame length */
};

/* Note 1:
 * An ethernet "packet" is a 5 MHz preamble + synchronization field of
 * 64 bits, followed by an ethernet "frame". The higher level software
 * (this program or in other cases the OS's TCP/IP stack) is responsible
 * for passing a frame with the fields dest_addr, src_addr, length, and data
 * complete. The ethernet hardware is then responsible for wrapping the
 * supplied frame with preamble+synchronization and a frame check sequence
 * that is computed in hardware.
 * The frame check sequence immediately follows the last byte of data,
 * however long the data may be. In the struct above, it is only located
 * after a full data array for human readability.
 */

char* printsafe_cpy(char*, const union eth_frame*, int);
```

```
const unsigned char BROADCAST_ADDRESS[6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

unsigned char host_address[ETH_ALEN]; /* ETH_ALEN is 6 */
char interface_name[1000];
int interface_index;
union eth_frame frame1;
int frame_length;
int data_length;

int main (int argc, char *argv[])
{
  memset (&frame1, 0, sizeof (frame1));

  /* Get Ethernet Interface Name from User */
  if (argc != 2)
  {
    printf ("Usage: %s <ethernet interface>\n", argv[0]);
    printf ("  where <ethernet interface> is the human readable name of the");
    printf (" sytems' ethernet.\n");
    printf ("  Get the name with command \"$ ifconfig | more\".\n");
    printf ("  Example:\n");
    printf ("  $ %s eth0\n", argv[0]);
    exit (EXIT_FAILURE);
  }

  /* Copy the <ethernet interface> name provided by User */
  strcpy (interface_name, argv[1]);

  /* Open an Ethernet Socket */
  int sfd;
  sfd = socket (AF_PACKET, SOCK_RAW, htons (ETH_P_ALL));
  if (sfd < 0)
  {
    fprintf (stderr, "socket() error %d\n", sfd);
    fprintf (stderr, "(try running with sudo)\n");
    exit (EXIT_FAILURE);
  }

  /* Look Up the Ethernet Interface Index */
  struct ifreq interface_req;
  memset (&interface_req, 0x00, sizeof (interface_req));
  strcpy (interface_req.ifr_name, interface_name);
  if (ioctl (sfd, SIOCGIFINDEX, &interface_req) < 0)
  {
    fprintf (stderr, "ioctl(SIOCGIFINDEX) error.\n");
    exit (EXIT_FAILURE);
```

```
  }
  interface_index = interface_req.ifr_ifindex;
/*printf ("Index of interface \"%s\": %d\n", interface_name, interface_index);
*/

  /* Look Up the Source MAC Address */
  if (ioctl (sfd, SIOCGIFHWADDR, &interface_req) < 0)
  {
    fprintf (stderr, "ioctl (SIOCGIFHWADDR) error.\n");
    exit (EXIT_FAILURE);
  }
  memcpy ( (void*)host_address, (void*)(interface_req.ifr_hwaddr.sa_data), ETH_A
/*printf ("Source MAC Address %d:%d:", (int) mac[0], (int) mac[1]);
  printf ("%d:%d:%d:%d\n", (int)mac[2], (int)mac[3], (int)mac[4], (int)mac[5]);
*/

  int count = 0;
  while (1)
  {
    int received = 0;
    received = recvfrom (sfd, (void*)&frame1, ETH_FRAME_LEN, 0, NULL, NULL);
    if (received < 0)
    {
      fprintf (stderr, "recvfrom() error %d\n", received);
      exit (EXIT_FAILURE);
    }
    if (received == 0)
      continue;
    /* If received > 0, then print out the first X chars of data */
    char temp[1500];
    printf ("Packet (%d): %s\n", count++, printsafe_cpy (temp, &frame1, 70));
  }

  /* Close the Socket */
  close (sfd);

  return EXIT_SUCCESS;
}

char* printsafe_cpy (char *dest, const union eth_frame *frame, int max_length)
{
  /* Determine Data Length to Print */
  int length;
  length = 256 * frame->field.length[0] + frame->field.length[1];
  length = (length < 1500) ? length: 1500; /* Max 802.3 data length */
  length = (length < max_length) ? length: max_length;
```

```
  /* Convert all special characters to spaces */
  int i;
  const unsigned char *data;
  data = frame->field.data;
  for (i=0; i<length; i++)
  {
    if (0 <= data[i] && data[i] <= 32)
      dest[i] = ' ';
    else if (33 <= data[i] && data[i] <=126)
      dest[i] = data[i];
    else
      dest[i] = ' ';
  }
  /* Null Terminate and Return ptr for fPrinting */
  dest[length] = 0;
  return dest;
}
```