LAB 2, Part 1+2

1 Files, Datagram Sockets

1.1 Datagram Sockets

Done!

1.2 Files

Average size of I, P, B frames is shown in Table 1, and the code is "part1-2.c".

2 Traffic Generators

2.1 Poisson Traffic

The traffic generator code is "uclient.c". It takes the following options from input:

- **-a *output address*** : This is the IP address (or name) of the destination, where the UDP packets go. Default is "127.0.0.1".
- **-o *output port*** : This is the port number. Default is 4444.
- **-f *file-name*** : Input file containing the trace. It is assumed that there are 4 columns in the file: Sequence Number, Packet Time, Packet Length, Cumulative packet size. The last column is not used here, but it is later used to draw plots.

2.2 Traffic Sink

The traffic sink code is "userver.c". It takes the following options from input:

- **-i *input port*** : This is the input port number. Default is 4445.

Average Size of I Frames (Bytes)	183776.39
Average Size of P Frames (Bytes)	111412.36
Average Size of B Frames (Bytes)	36093.04

Table 1: Video Frame Statistics

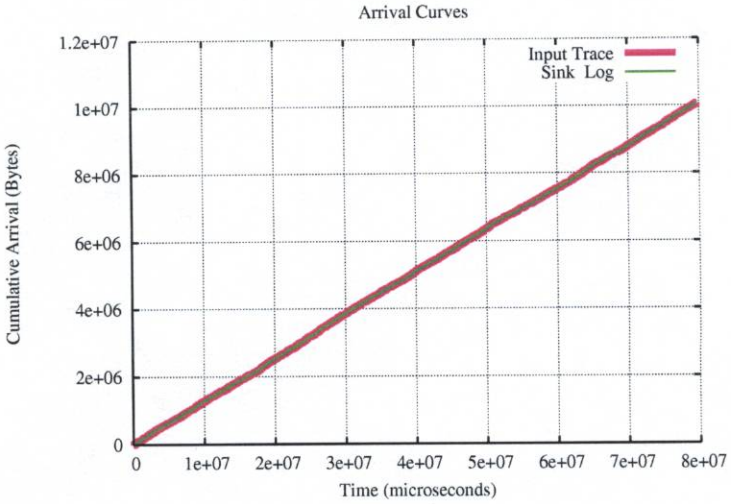

Figure 1: Comparison of trace (input to generator) and the captured traffic (logged by the sink)

- **-f *file-name*** : Output file containing the trace. It has 4 columns: sequence number (integer), packet arrival time since the beginning of the arrivals in microseconds (float), packet size in bytes (integer), accumulative arrivals in bytes (long integer).

2.3 Evaluation

For evaluation we plot the arrivals of the trace and the arrivals captured in the sink. Using the script "run" which is provided this can be done as follow:

```
./run build-trace 10000 # 10000 is the number of samples
./run no-shaping
```

Figure 1 shows the comparison.

2.4 Packet Losses

The potential packet losses can be identified by comparison of the sent and received sequence numbers. Local losses (e.g. packets dropped out of the socket queue), can be prevented (unless the bottleneck is the processing power, or the available memory) by having a dedicated thread to read the socket before it is overflowed. The default unix datagram socket queue length in my system is 10 packets, which can also be increased, for instance, by writing the desired queue length into

```
/proc/sys/net/unix/max_dgram_qlen
```

```
#include <stdio.h>
#include <stdlib.h>

int main() {

  // open files
  FILE * f_in;
  FILE * f_out;
  FILE * f_tex;
  if ((f_in=fopen("movietrace.data","r"))==NULL){
    fprintf(stderr, "opening input file failed\n");
    exit(1);
  }
  if ((f_out=fopen("movietrace-out.data","w"))==NULL){
    fprintf(stderr, "opening output file failed\n");
    exit(1);
  }
  if ((f_tex=fopen("h3-table1-2.tex","w"))==NULL){
    fprintf(stderr, "opening tex output file failed\n");
    exit(1);
  }
  //----------/

  // main loop
  int seq_number, packet_length;
  float dummy1, dummy2, dummy3, packet_time;
  float I_volume=0, P_volume=0, B_volume=0;
  char packet_type;
  long int I_frames=0, P_frames=0, B_frames=0;

  while(fscanf(f_in, "%d %f %c %d %f %f %f", &seq_number, &packet_time, &packet_type, &packet_len
gth, &dummy1, &dummy2, &dummy3 )>0)
    {
      switch(packet_type){
      case 'I':
        I_frames++;
        I_volume += packet_length;
        break;
      case 'P':
        P_frames++;
        P_volume += packet_length;
        break;
      case 'B':
        B_frames++;
        B_volume += packet_length;
        break;
      }

      fprintf(f_out, "SeqNo: %d  Frame time: %f  Frame type: %c  Frame size: %d \n",
              seq_number,
              packet_time,
              packet_type,
              packet_length);
    }
  //----------/

  // print out the average sizes of I, P, and B frames
  fprintf(f_tex,"Average Size of I Frames (Bytes) &  %.2f \\\\\n", I_volume/I_frames);
  fprintf(f_tex,"Average Size of P Frames (Bytes) &  %.2f \\\\\n", P_volume/P_frames);
  fprintf(f_tex,"Average Size of B Frames (Bytes) &  %.2f \\\\\n", B_volume/B_frames);

  printf("+--------------------------+-----------------------+\n");
  printf("| Average Size of I Frames |     %10.2f (Bytes) |\n", I_volume/I_frames);
  printf("| Average Size of P Frames |     %10.2f (Bytes) |\n", P_volume/P_frames);
  printf("| Average Size of B Frames |     %10.2f (Bytes) |\n", B_volume/B_frames);
  printf("+--------------------------+-----------------------+\n");
  //----------/

  fclose(f_tex);
  fclose(f_out);
  fclose(f_in);
  return 0;
```

```
}
```

```c
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define BUFLEN 16384

void diep(char *s)
{
  perror(s);
  exit(1);
}

int main(int argc, char * argv[])
{
  char * output_address  = "127.0.0.1";
  char * input_filename = "trace.dat";
  int output_port = 4444;
  int c;

  while ((c = getopt (argc, argv, "a:f:o:")) != -1)
    switch (c)
      {
      case 'a':
        output_address = optarg;
        break;
      case 'f':
        input_filename = optarg;
        break;
      case 'o':
        output_port = atoi(optarg);
        break;
      case '?':
        if (isprint (optopt))
          fprintf (stderr, "Unknown option `-%c'.\n", optopt);
        else
          fprintf (stderr,
                   "Unknown option character `\\x%x'.\n",
                   optopt);
        return 1;
      default:
        ;
      }

  struct sockaddr_in sock_server;
  int s, i;
  socklen_t socklen = sizeof(struct sockaddr_in);

  if ((s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
    diep("socket");
  }

  sock_server.sin_family = AF_INET;
  sock_server.sin_port = htons(output_port);

  if (inet_aton(output_address, &(sock_server.sin_addr)) == 0) {
    fprintf(stderr, "inet_aton() failed\n");
    exit(1);
  }
  //

  FILE * f;

  if ((f=fopen(input_filename,"r"))==NULL){
    fprintf(stderr, "opening input file failed\n");
    exit(1);
  }
```

```
  char   first_time=1;
  long int    seq_number, packet_length, packet_time, arrival_bytes;
  char   buf[BUFLEN];
  struct timeval gt,gt_init;
  for(i=0; i<BUFLEN-1; i++) buf[i] = '.';
  buf[BUFLEN-1]=0;

  gettimeofday(&gt_init,NULL);
  while(fscanf(f, "%ld %ld %ld %ld", &seq_number, &packet_time, &packet_length, &arrival_bytes)>0
) {
    gettimeofday(&gt,NULL);
    if(first_time){
      gt_init.tv_sec = gt.tv_sec;
      gt_init.tv_sec = gt.tv_sec;
      first_time = 0;
    }
    while((gt.tv_usec - gt_init.tv_usec + (gt.tv_sec - gt_init.tv_sec) * 1000000) < packet_time)
      gettimeofday(&gt,NULL);
    if (packet_length < BUFLEN){
      buf[packet_length] = 0;
      if (sendto(s, buf, strlen(buf)+1, 0,
                 (struct sockaddr *) &sock_server, socklen) == -1) {
        diep("sendto()");
      }
      buf[packet_length] = '.';
    } else {
      fprintf(stderr, "packet too long, packet length : %d\n", packet_length);
    }
  }
  if (sendto(s, ";", 1, 0,
             (struct sockaddr *) &sock_server, socklen) == -1) {
    diep("sendto()");
  }

  fclose(f);
  close(s);
  return 0;
}
```

```c
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define BUFLEN 16384

void diep(char *s)
{
    perror(s);
    exit(1);
}

int main(int argc, char * argv[]) {

  char * log_filename = "trace.dat";
  int input_port = 4445;
  int c;

  while ((c = getopt (argc, argv, "f:i:")) != -1)
    switch (c)
      {
      case 'f':
        log_filename = optarg;
        break;
      case 'i':
        input_port = atoi(optarg);
        break;
      case '?':
        if (isprint (optopt))
          fprintf (stderr, "Unknown option `-%c'.\n", optopt);
        else
          fprintf (stderr,
                   "Unknown option character `\\x%x'.\n",
                   optopt);
        return 1;
      default:
        ;
      }

    struct sockaddr_in sock_server;
    struct sockaddr_in sock_client;
    int s;
    socklen_t socklen = sizeof(struct sockaddr_in);
    char buf[BUFLEN];

    if ((s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        diep("socket()");
    }
    sock_server.sin_family = AF_INET;
    sock_server.sin_port = htons(input_port);
    sock_server.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(s, (struct sockaddr *) &sock_server, socklen) == -1) {
        diep("bind()");
    }

    FILE * f;
    int seq_number = 1, packet_length;
    struct timeval timestamp, timestamp_old, * timestamp_old_p=NULL;
    double elapsed_time = 0;
    long int bytes_received_so_far = 0;

    if ((f=fopen(log_filename, "w"))==NULL){
      fprintf(stderr, "opening output file failed\n");
      exit(1);
    }
```

```
    for(;;) {
      if (recvfrom(s, buf, BUFLEN, 0,
                   (struct sockaddr *) &sock_client, &socklen) == -1)
        diep("recvfrom()");
      gettimeofday (&timestamp, NULL);

      if (buf[0]==';') break;
      if (timestamp_old_p)
        elapsed_time = (timestamp.tv_sec - timestamp_old_p->tv_sec)*1000000 + (timestamp.tv_usec-
timestamp_old_p->tv_usec);
      else {
        timestamp_old.tv_sec = timestamp.tv_sec;
        timestamp_old.tv_usec = timestamp.tv_usec;
        timestamp_old_p = &timestamp_old;
      }

      // comment for rel time, uncomment for diff-time
      //timestamp_old.tv_sec = timestamp.tv_sec;
      //timestamp_old.tv_usec = timestamp.tv_usec;


      // fprintf(f,"\"%s:%d\" %d %f %d\n", inet_ntoa(sock_client.sin_addr), ntohs(sock_client.sin
_port), seq_number, elapsed_time,  strlen(buf));
      packet_length = strlen(buf);
      bytes_received_so_far += packet_length;
      fprintf(f,"%d %f %d %ld\n", seq_number, elapsed_time,  packet_length, bytes_received_so_far
);
      seq_number ++;
    }

    fclose(f);
    close(s);
    return 0;
}
```