TCP/IP performance test

y.ma@riken.jp

20170407

Outline

- Motivation:
  - Measure the real performance and prepared as input for more integrated study
- Setup: (more details in slides)
  - Software: mini-DAQ
  - Hardware: 2 sets of 40 core machine with direct 10gbps connection
- Results
- Summary & todo

Setup: Hardware

- Test performance with "*iperf*" tools
- Test performance with "*scp*": 1.6gbps, limited by hard drive throughput?

oper@e50_server1:~

```
meson ~ $  e50_server1_login
oper@192.168.10.202's password:
Last login: Wed Apr  5 15:17:25 2017 from gateway
[oper@e50_server1 ~]$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.0.0.13 port 5001 connected with 10.0.0.3 port 43422
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 1.05 MByte (default)
------------------------------------------------------------
[  6] local 10.0.0.13 port 48888 connected with 10.0.0.3 port 5001
Waiting for server threads to complete. Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  6]  0.0-10.0 sec  10.8 GBytes  9.32 Gbits/sec
[  4]  0.0-10.0 sec  10.9 GBytes  9.34 Gbits/sec
[oper@e50_server1 ~]$
```

oper@e50_server0:~

```
meson ~ $  e50_server0_login
oper@192.168.10.201's password:
Last login: Fri Apr  7 12:30:40 2017 from gateway
[oper@e50_server0 ~]$ iperf -c 10.0.0.13 -d
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
------------------------------------------------------------
Client connecting to 10.0.0.13, TCP port 5001
TCP window size: 1.14 MByte (default)
------------------------------------------------------------
[  5] local 10.0.0.3 port 43422 connected with 10.0.0.13 port 5001
[  4] local 10.0.0.3 port 5001 connected with 10.0.0.13 port 48888
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec  10.9 GBytes  9.35 Gbits/sec
[  4]  0.0-10.0 sec  10.8 GBytes  9.31 Gbits/sec
[oper@e50_server0 ~]$
```

Setup: Software

- Default CentOS 7.0 x86_64:
- [oper@e50_server0 ~]$ cat /proc/sys/net/ipv4/tcp_rmem
- 4096 (min [Byte]) 87380(default [Byte]) 6291456(max [Byte])
- [oper@e50_server0 ~]$ cat /proc/sys/net/ipv4/tcp_wmem
- 4096(min [Byte]) 16384(default [Byte]) 4194304(max [Byte])

- mini-DAQ:
- minimum DAQ functionality from scratch in C
- data source (ring buffer) —>*TCP/IP socket*—> data sink (ring buffer)

default C library function,
no customization

Setup: Software

oper@e50_server1:~/ethernet_test_tools/data_sink

```
[oper@e50_server1 data_sink]$ pwd
/home/oper/ethernet_test_tools/data_sink
[oper@e50_server1 data_sink]$ ./data_sink root_file_name.root 8602
```

Data sink

```
pthread_t          threads[THRD_MAX];
pthread_attr_t     attr;
int                status = 1;

pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
void *thrd_args = (void *)&server_buffer;

pthread_create(&threads[MNGR_THRD_ID], &attr, manager_thrd, thrd_args);
pthread_create(&threads[PARS_THRD_ID], &attr, parser_thrd, thrd_args);
pthread_create(&threads[RECV_THRD_ID], &attr, receiver_thrd, thrd_args);
```

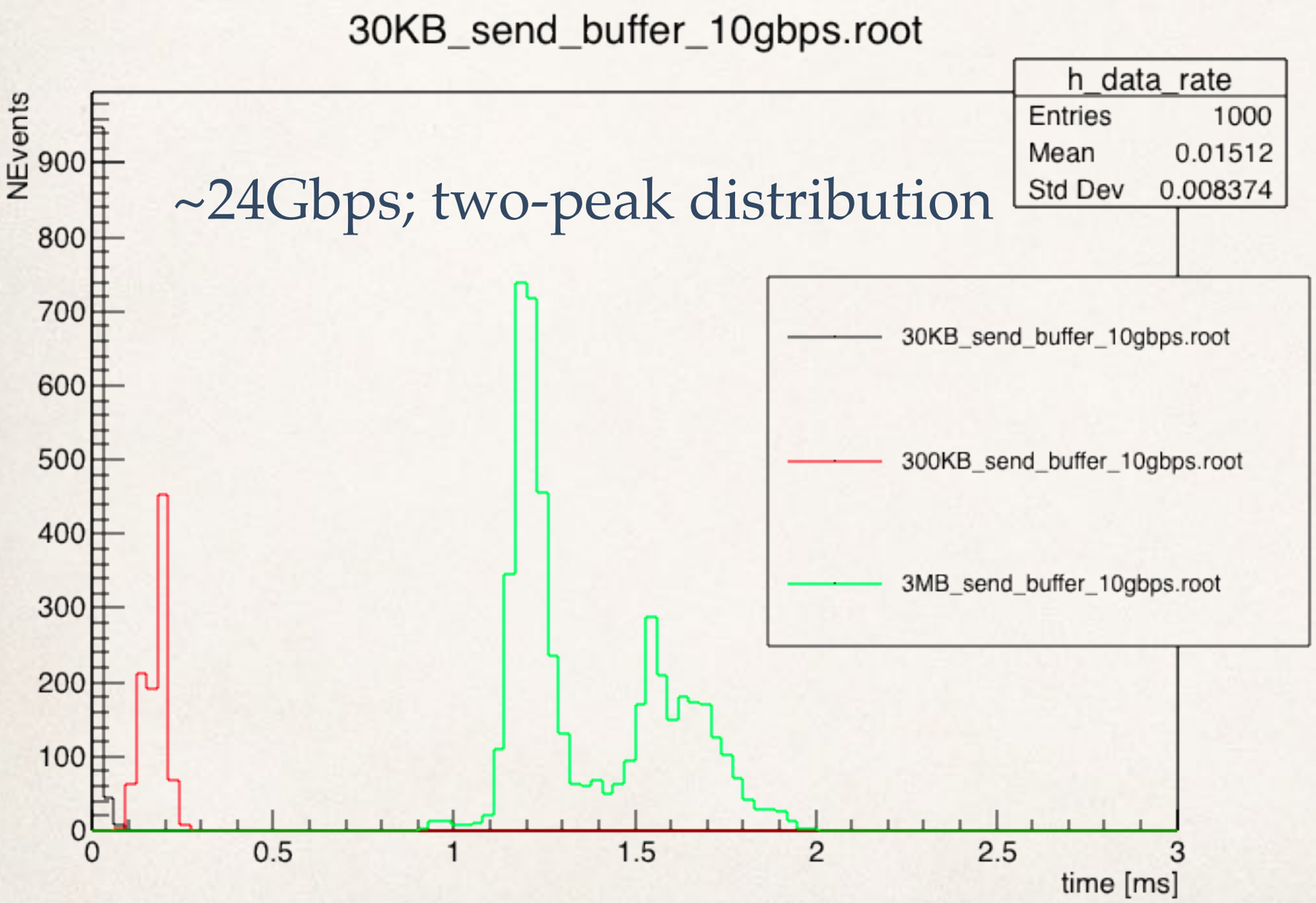
```
int main(int argc, char *argv[])
{
    client_buffer_t    client_buffer;
    pthread_t          threads[THRD_MAX];
    pthread_attr_t     attr;
    int                status = 1;

    client_buffer_init(&client_buffer, CLIENT_BUFFER_SIZE);

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
    void *thrd_args = (void *)&client_buffer;
    pthread_create(&threads[COLL_THRD_ID], &attr, collector_thrd, thrd_args);
    pthread_create(&threads[TRAN_THRD_ID], &attr, transporter_thrd, thrd_args);
    pthread_join(threads[COLL_THRD_ID], NULL);
    pthread_join(threads[TRAN_THRD_ID], NULL);
```

oper@e50_server0:~/ethernet_test_tools/data_source

```
[oper@e50_server0 data_source]$ pwd
/home/oper/ethernet_test_tools/data_source
[oper@e50_server0 data_source]$ ./data_source
```

Data source

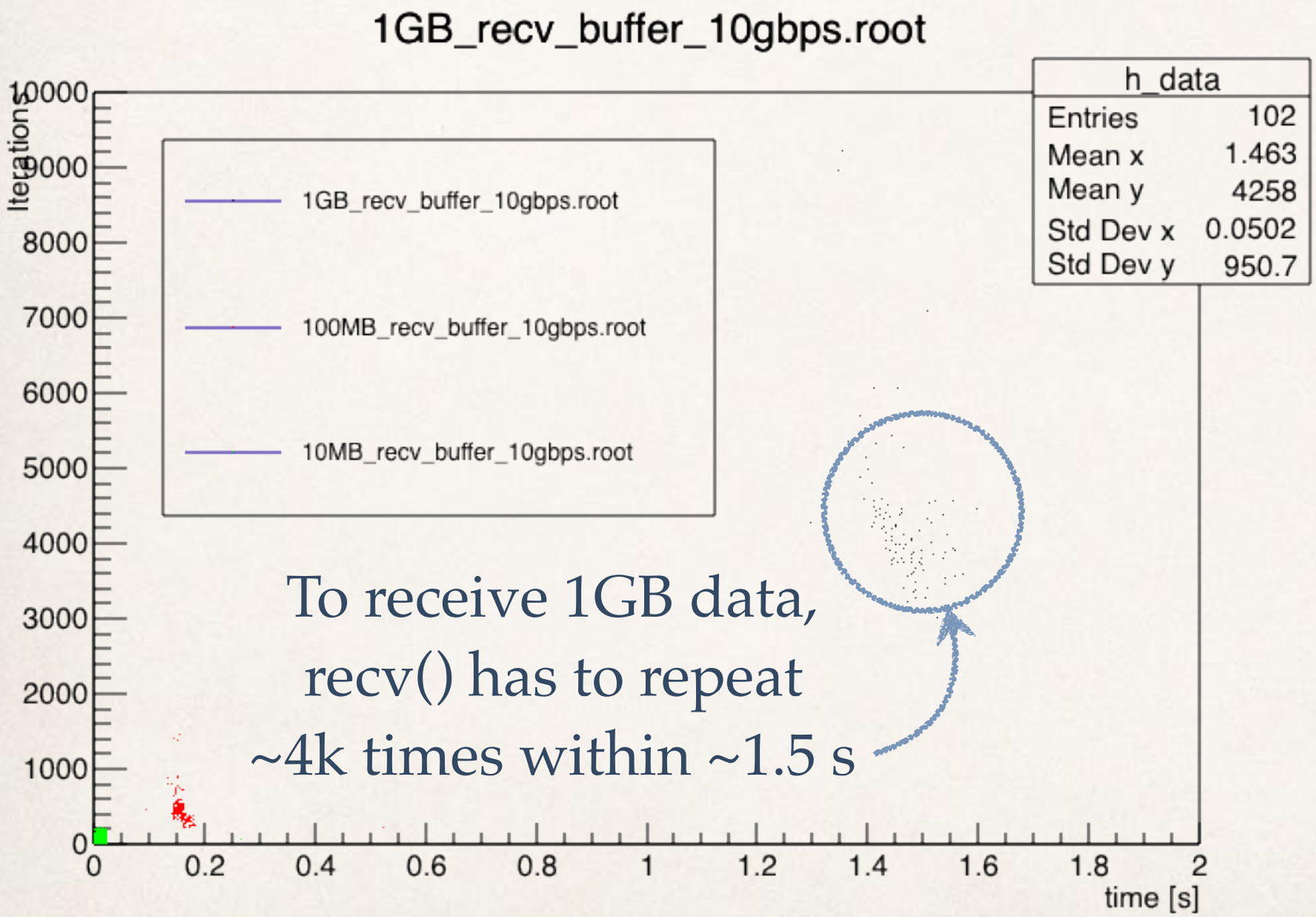
Results: data source *send()* throughput



~24Gbps; two-peak distribution

Method:
At data source,
change send()
buffer size
and measure the
time consumption;
Up to 30MB,
the send() function
works at 24Gbps;
Larger than 30MB,
process hangup…

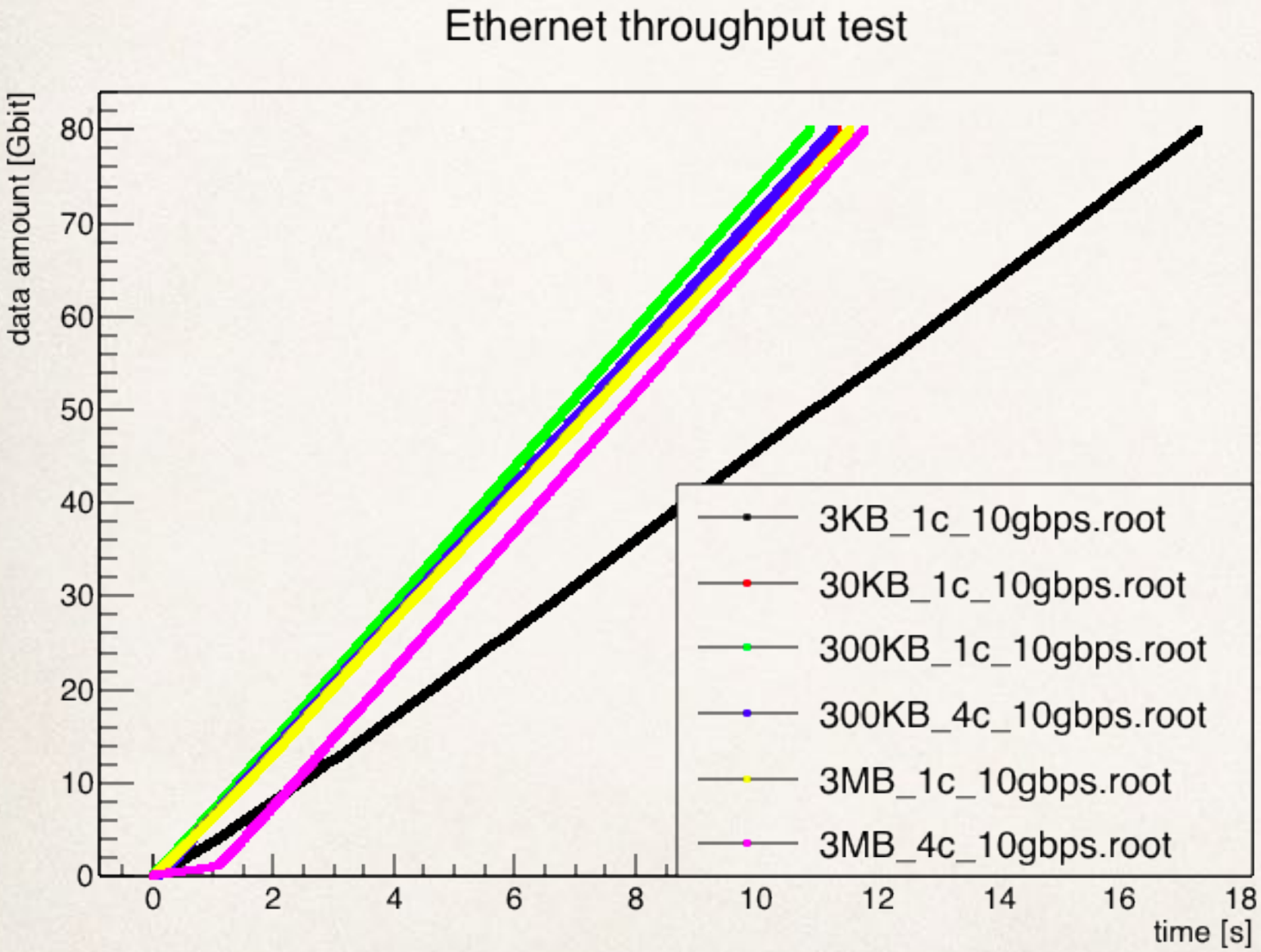
Results: data sink *receive()* throughput



To receive 1GB data,
recv() has to repeat
~4k times within ~1.5 s

Method:
At data sink,
change recv()
buffer size
and measure the
receive cycles and
time consumption;

Results: data *transportation* speed

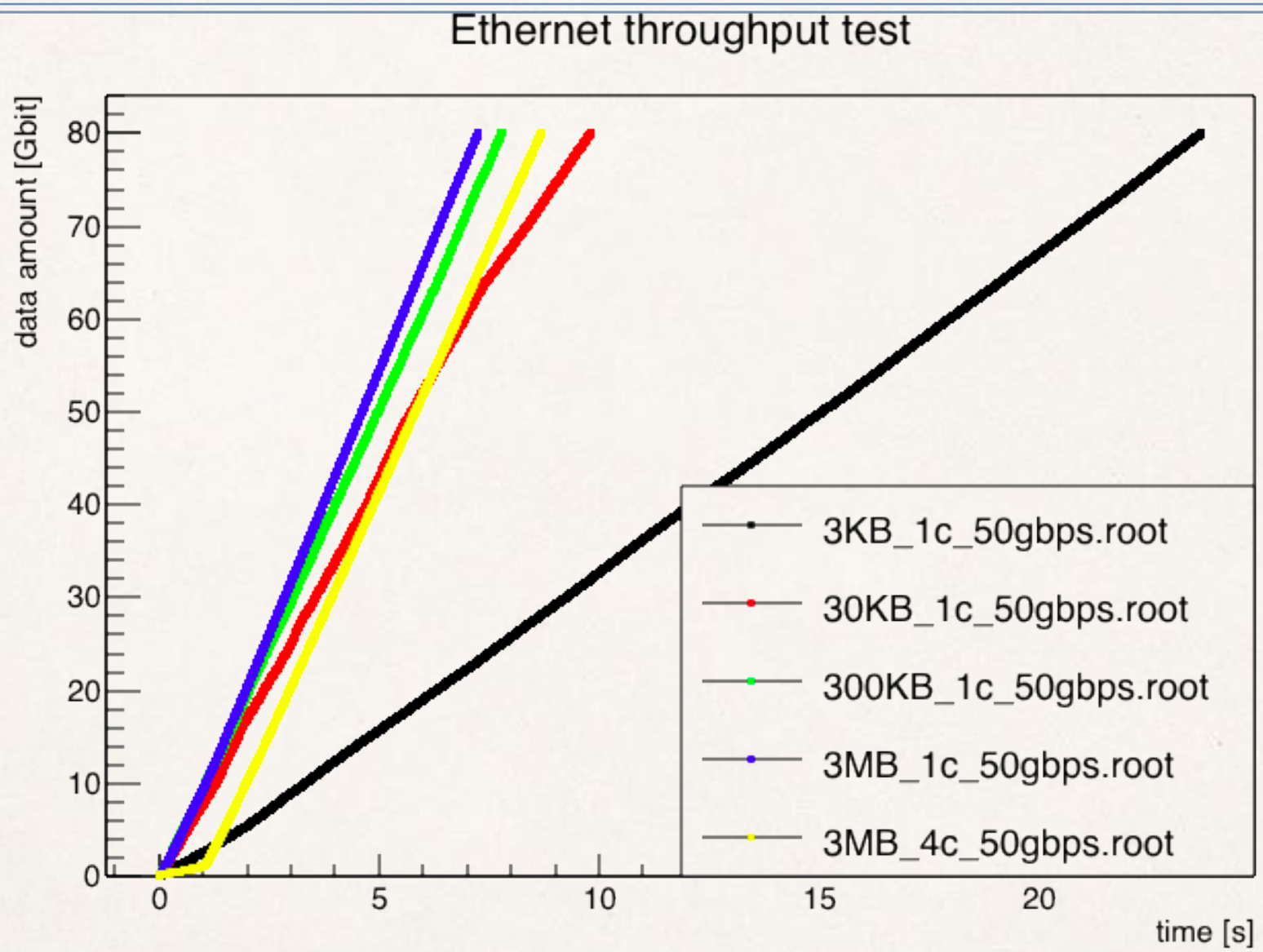


Method:
Adjust buffer size
on both recv() and send(),
measure data transfer speed;
Too small buffer size (3KB)
results in low performance;
multi clients (data source)
have similar speed,
though it takes
some time to
reach the optimized point.

Summary & todo

- Single, direct link 10Gbps ethernet has reasonable performance;
- The jittering is not negligible and a *safety factor of two* seems necessary for a reliable performance estimation
- Use 10Gbps switch to test data collision effect(Cisco UCS 6120xp);
- At the moment, 50Gbps seems have a performance bottle neck issue…

- Backup


Ethernet throughput test
data amount [Gbit]
0
10
20
30
40
50
60
70
80
0
5
10
15
20
time [s]
3KB_1c_50gbps.root
30KB_1c_50gbps.root
300KB_1c_50gbps.root
3MB_1c_50gbps.root
3MB_4c_50gbps.root

transporter_thrd.c - emacs@e50_server0

File Edit Options Buffers Tools C Help

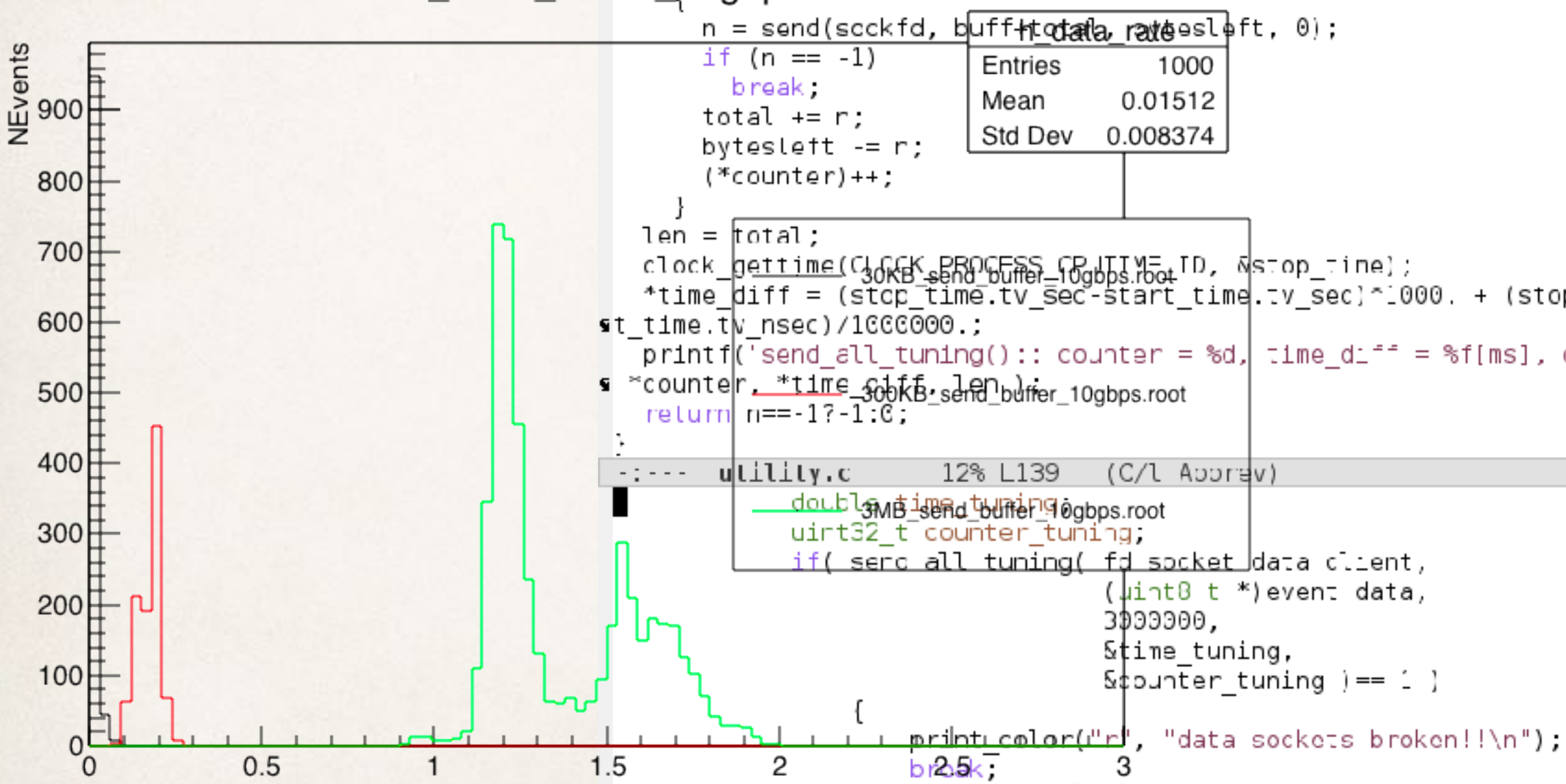
Save Unco

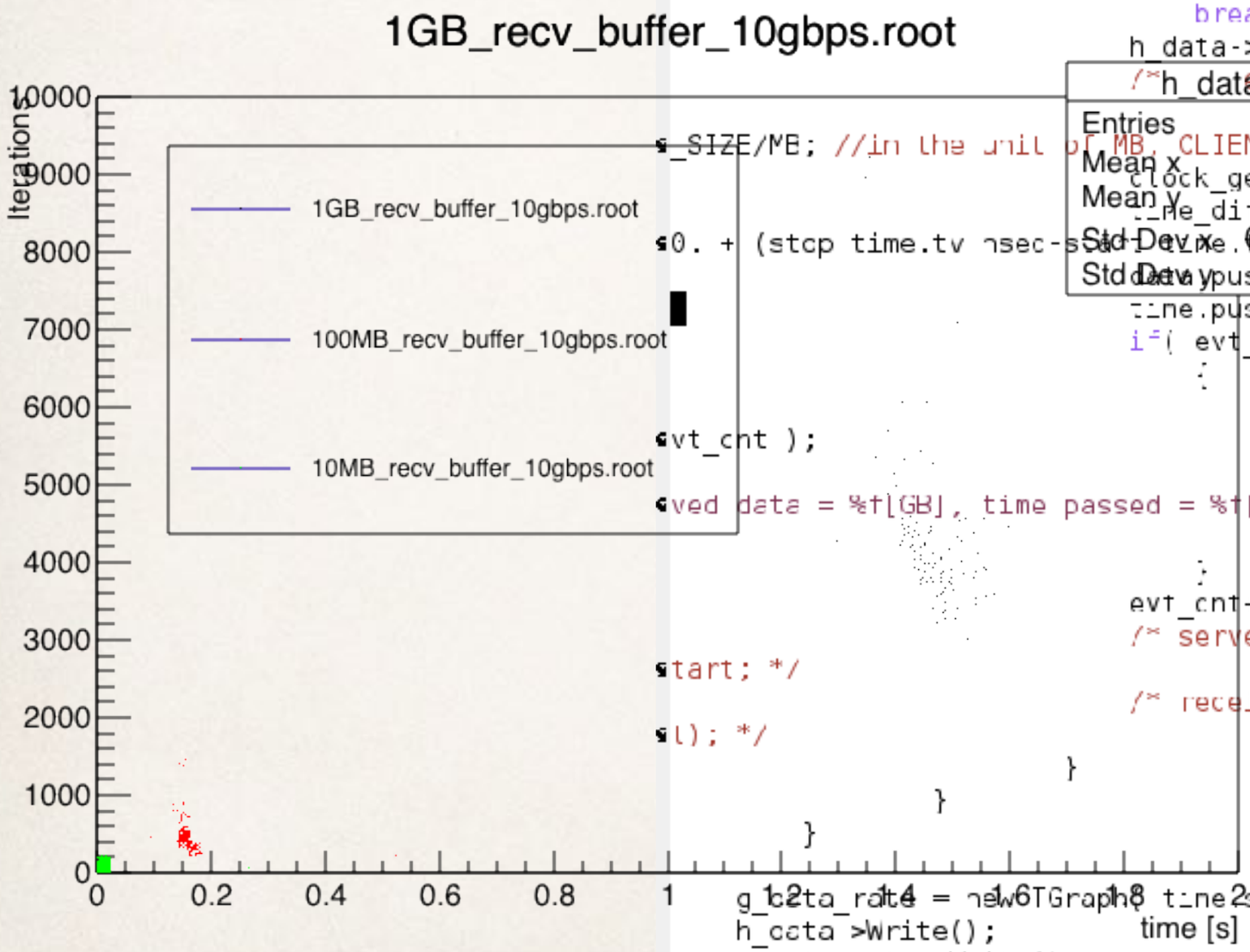
```
int send_all_tuning(int sockfd, void *buff, uint32_t len, double *time_diff, uint32
_t *counter)
{
  uint32_t total = 0;
  uint32_t bytesleft = len;
  uint32_t n;
  timespec start_time, stop_time;
  *counter = 0;
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start_time);
  while(total < len)
    {
      n = send(sockfd, buff+total, bytesleft, 0);
      if (n == -1)
        break;
      total += n;
      bytesleft -= n;
      (*counter)++;
    }
  len = total;
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop_time);
  *time_diff = (stop_time.tv_sec-start_time.tv_sec)*1000. + (stop_time.tv_nsec-star
t_time.tv_nsec)/1000000.;
  printf('send_all_tuning():: counter = %d, time_diff = %f[ms], data = %d[Byte]\n",
 *counter, *time_diff, len );
  return n==-1?-1:0;
}
```

-:--- utility.c 12% L139 (C/l Abbrev)

```
    double time_tuning;
    uint32_t counter_tuning;
    if( send_all_tuning( fd_socket_data_client,
                         (uint8_t *)event_data,
                         3000000,
                         &time_tuning,
                         &counter_tuning )== -1 )
      {
        print_color("r", "data sockets broken!!\n");
        break;
      }
    h_data_rate->Fill( time_tuning );
    evt_cnt++;
```

-:--- transporter_thrd.c 86% L104 (C/l Abbrev)





receiver_thrd.c - emacs@e50_server1

File Edit Options Buffers Tools C Help

Save Undo

```
/*  break; */

double time_tuning;
uint32_t counter_tuning;
if( recv_all_tuning( fd_accept_data_server[i],
                     (void *)dummy,
                     10000000,
                     &time_tuning,
                     &counter_tuning )==-1 )
    break;
h_data->Fill( time_tuning/1000., counter_tuning );
/*h_data                    event_data->evt_id_start); */
                    data_received = data_received - CLIENT_EVENT
_SIZE/MB; //in the unit of MB, CLIENT_EVENT_SIZE=30003byte
                clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop_time);
                time_diff = (stop_time.tv_sec-start_time.tv_sec)*1000
0. + (stop time.tv_nsec-start time.tv_nsec)/1000000.;
                data_push_back( data_received );
                time.push_back( time_diff );
                if( evt_cnt%100 == 0 )
                    {
                     printf("receiver_thrd():: evt_cnt = %d\n', c
evt_cnt );
                     printf("receiver_thrd() on socket[%d], recei
ved data = %f[GB], time passed = %f[ms]\n",
                            i, data_received/1000, time_diff);
                    }
                evt_cnt++;
                /* server_evert->evt_id_start = event_data->evt_id_s
start; */
                /* receiver_buffer_fill(receiver_buffer, server_even
l(); */
            }
        }
    }
g_data_rate = new TGraph( time.size(), &(time[0]), &(data[0]) );
h_data->Write();
g_data_rate->Write();
root_file->Close();
```

-:--- receiver_thrd.c 81% L188 (C/l Abbrev)

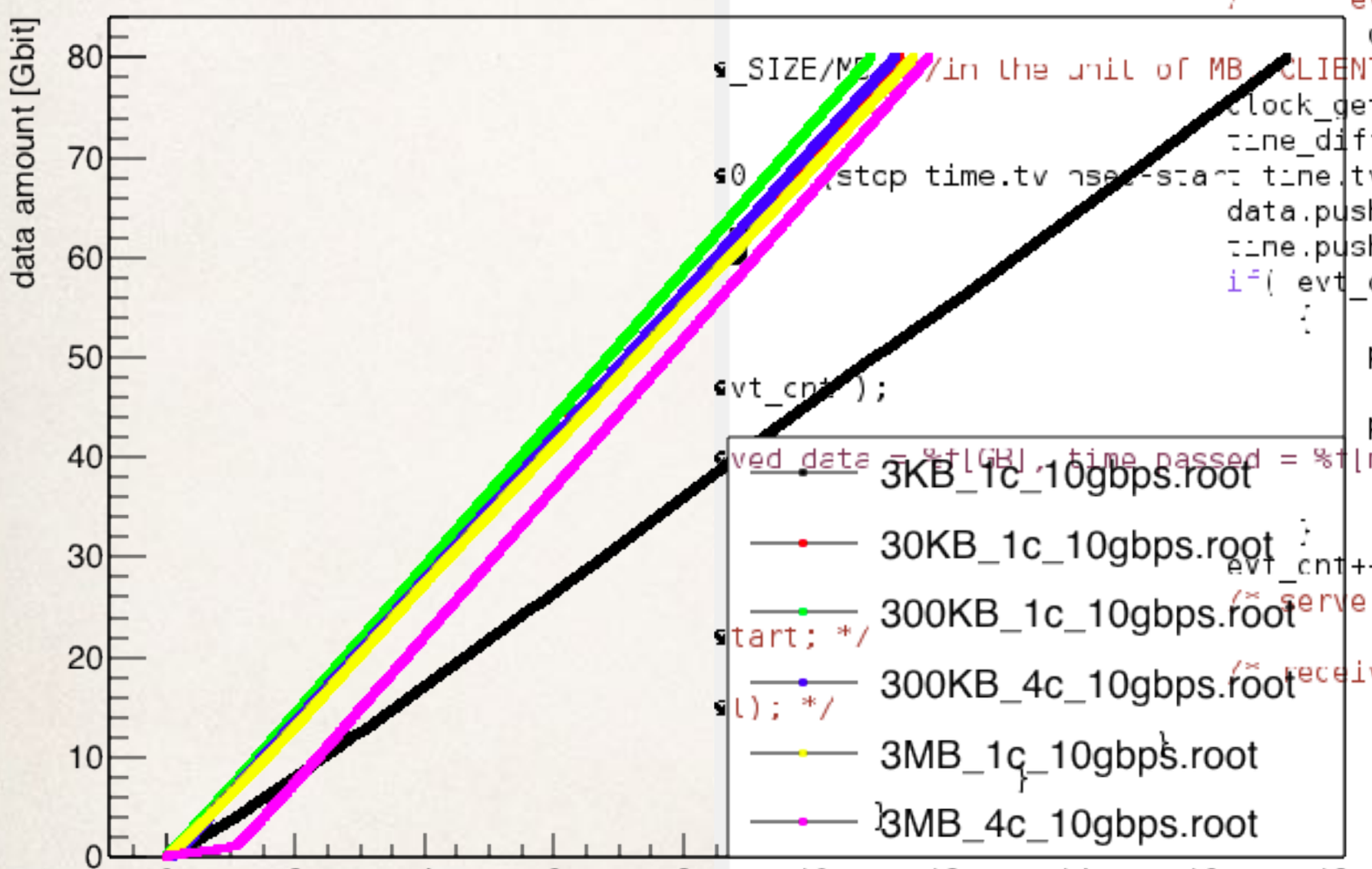
receiver_thrd.c - emacs@e50_server1

File Edit Options Buffers Tools C Help

Save Undo

```
/*  break; */

double time_tuning;
uint32_t counter_tuning;
if( recv_all_tuning( fd_accept_data_server[i],
                     (void *)dummy,
                     10000000,
                     &time_tuning,
                     &counter_tuning )==-1 )
    break;
h_data->Fill( time_tuning/1000., counter_tuning );
/*     event_data->evt_id_start); */
                 data_received = data_received - CLIENT_EVENT
_SIZE/M      //in the unit of MB  CLIENT_EVENT_SIZE=30003yte
                clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop_time);
0      (stop time.tv nsec-start time.tv nsec)/1000000.;
                data.push back( data received );
                time.push_back( time_diff );
                if( evt_cnt%100 == 0 )
                    {
                     printf("receiver_thrd():: evt_cnt = %d\n', c
                     printf("receiver_thrd() on socket[%d], recei
                            i, data_received/1000, time_diff);
                    }
vt_cnt );
ved data = %f[GB], time passed = %f[ms]\n",
                evt_cnt++;
start; */       /* server_event->evt_id_start = event_data->evt_id_s
                /* receiver_buffer_fill(receiver_buffer, server_even
l); */
                }
                g_data_rate = new TGraph( time.size(), &(time[0]), &(data[0]) );
                h_data->Write();
                g_data_rate->Write();
                root_file->Close();
```

Ethernet throughput test



-:--- receiver_thrd.c 81% L188 (C/l Abbrev)