CS4513 Project 4 Report

author: Mengwen Li (mli2)

Design:

a) How you instrumented/measured your system:

To measure the network data rates, the following log code was used at “send” and “receive” method to record the time and size of packet sent and received during game play:

```
Role &role = Role::getInstance();
if (role.isHost()) {
    log_manager.writeLog("df::NetworkManager::send(): Host sending %d bytes to client", bytes);
}
else {
    log_manager.writeLog("df::NetworkManager::send(): Client sending %d bytes to host", bytes);
}
```

Code in send mehtod

```
if (!peek) {
    if (role.isHost()) {
        log_manager.writeLog("df::NetworkManager::receive(): Host pull %d bytes out from socket", rRtn);
    }
    else {
        log_manager.writeLog("df::NetworkManager::receive(): Client pull %d bytes out socket", rRtn);
    }
}
```

Code in receive method

From these logs, we can record the packet size, packet rate and bit rate.

To measure the in-game round trip time, the following log code was uesed in the “Client” class to record the time a key stroke was detected and the time an object update message was received:

```
if (p_e->getType() == df::KEYBOARD_EVENT) {
    SYSTEMTIME st;
    GetSystemTime(&st);
    char ct[84] = "";
    sprintf(ct, "%d/%d/%d  %d:%d:%d %d", st.wDay, st.wMonth, st.wYear, st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);
    log_manager.writeLog(ct);
    log_manager.writeLog("Client key stroke detected.");
    const df::EventKeyboard *p_keyboard_event = dynamic_cast<const df::EventKeyboard *>(p_e);
    handleKeyboard(p_keyboard_event);
    return 1;
}
```

Code to record when a key stroke is detected

```
// If message is update object.
if (msg_type == UPDATE_OBJECT) {
    df::WorldManager &world_manager = df::WorldManager::getInstance();
    df::LogManager &log_manager = df::LogManager::getInstance();
    // Get the id of the object.
    std::string msg_str = std::string(buffer + 3 * sizeof(int));
    int id;
    memcpy(&id, buffer + 2 * sizeof(int), sizeof(int));
    // Get the object with id.
    Object *p_obj = world_manager.objectWithId(id);
    if (p_obj == NULL) {
        log_manager.writeLog("No such object found.");
    }
    else {
        SYSTEMTIME st;
        GetSystemTime(&st);
        char ct[84] = "";
        sprintf(ct, "%d/%d/%d  %d:%d:%d %d", st.wDay, st.wMonth, st.wYear, st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);
        log_manager.writeLog(ct);
        log_manager.writeLog("Client Object update received");
        // Update the object.
        p_obj->deserialize(msg_str);
    }
}
```

Code to record when an update message is received.

Using these two times, we can calculate the in-game round trip time.

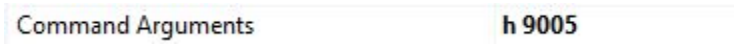
b) How many runs your performed:

For measuring packet size and packet rate, 10 samples are taken and the average is calculated. For measuring bitrate both from client to host and from host to client, 6 samples are taken. For measuing in-game round trip time, 5 samples are taken.

c) What the system conditions were like:

The program was tested across two computers in my apartment. These two computers are both connected to Ethernet from the same router. Both computers starts the program from Visual Studio. The command line argument setting for client is as following:

and the command line argument setting for host is as following:

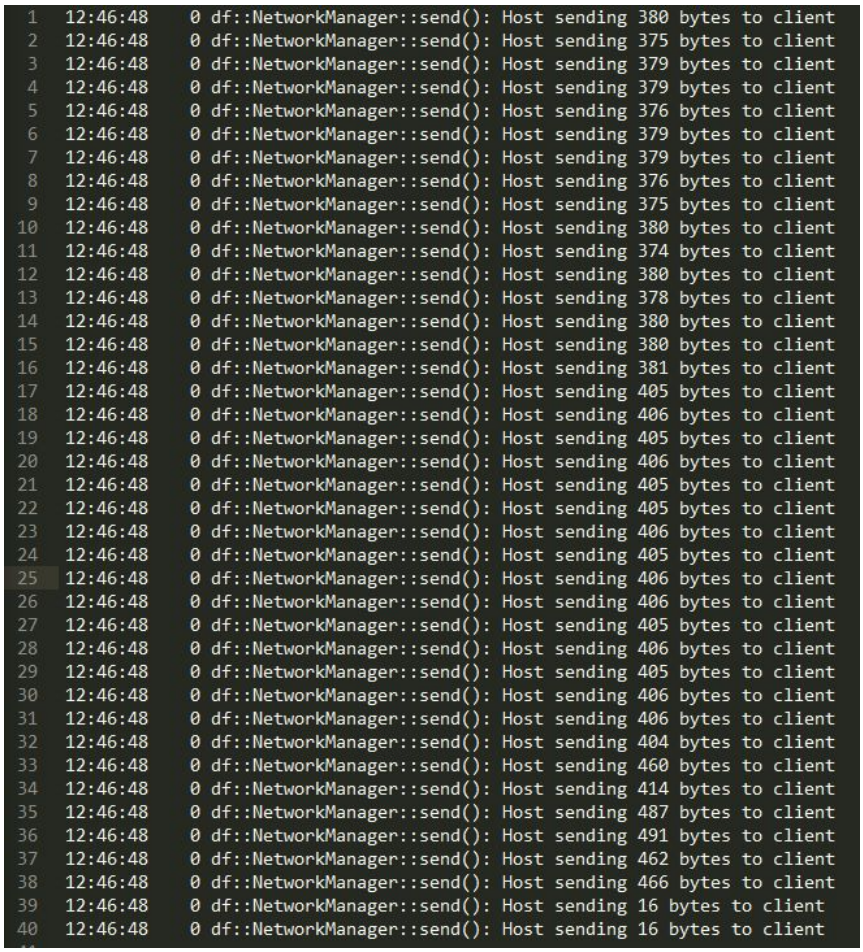


Results and Analysis:

a) Results and analysis for measuring the network data rates:

Packet size:

At start, all the objects are sent to client, the messages sent from host was shown as following:



```
12:46:48    0 df::NetworkManager::send(): Host sending 380 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 375 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 379 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 379 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 376 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 379 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 379 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 376 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 375 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 380 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 374 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 380 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 378 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 380 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 380 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 381 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 405 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 406 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 404 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 460 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 414 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 487 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 491 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 462 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 466 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 16 bytes to client
12:46:48    0 df::NetworkManager::send(): Host sending 16 bytes to client
```

The packets received at the client side was shown as following:

```
12:47:24    2 df::NetworkManager::isData(): Client has 8889 bytes in socket
12:47:24    2 df::NetworkManager::receive(): Client pull 380 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 375 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 379 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 379 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 376 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 379 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 379 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 376 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 375 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 380 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 374 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 380 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 378 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 380 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 380 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 381 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    2 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::isData(): Client has 5010 bytes in socket
12:47:24    3 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 405 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 406 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 404 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 460 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 414 bytes out socket
12:47:24    3 df::NetworkManager::receive(): Client pull 487 bytes out socket
12:47:24    4 df::NetworkManager::isData(): Client has 1451 bytes in socket
12:47:24    4 df::NetworkManager::receive(): Client pull 491 bytes out socket
12:47:24    4 df::NetworkManager::receive(): Client pull 462 bytes out socket
12:47:24    4 df::NetworkManager::receive(): Client pull 466 bytes out socket
12:47:24    4 df::NetworkManager::receive(): Client pull 16 bytes out socket
12:47:24    4 df::NetworkManager::receive(): Client pull 16 bytes out socket
```

From the log of “NetworkManager::isData()”, we can tell that three packets are received at the client size, the first one is 8889 bytes, the second one is 5010 bytes and the third one is 1451 bytes.

Then, after the game starts, some package sizes were shown in the following table:

Time	Package size (bytes)
12:47:26 57	150
12:47:26 62	182
12:47:27 68	150
12:47:27 77	150
12:47:27 83	150
12:47:27 88	150
12:47:30 167	403

12:47:31 206	402
12:47:31 216	434
12:47:35 315	230
Average	240.1
Standard deviation	122.2

From the above table, we can tell that the average package size after the game starts is 240 bytes. When there are more operations, the package size is bigger. Also, most of the package is in the range between 100 bytes and 500 bytes after start.

Packet rates:

Time of receiving a packet	Time difference between two packets (ms)
12:47:24 2	not apply
12:47:24 3	1
12:47:24 4	1
12:47:25 32	1028
12:47:26 57	1025
12:47:26 62	5
12:47:27 68	1006
12:47:27 77	9
12:47:27 83	6
12:47:27 88	5
12:47:27 92	4
Average	309

From the above table, we can tell that the average time a package arrives is 309 ms, the time difference is bigger when no object needs to be synced over a long period of time and the time difference is smaller when a lot of objects and operations needs to be synced.
From the average, we can calculate the packet rate as 1/(309 * 10 ^ -3) = 3.2 packets/second.
As a result, the packet rate is around 4 packets/second.

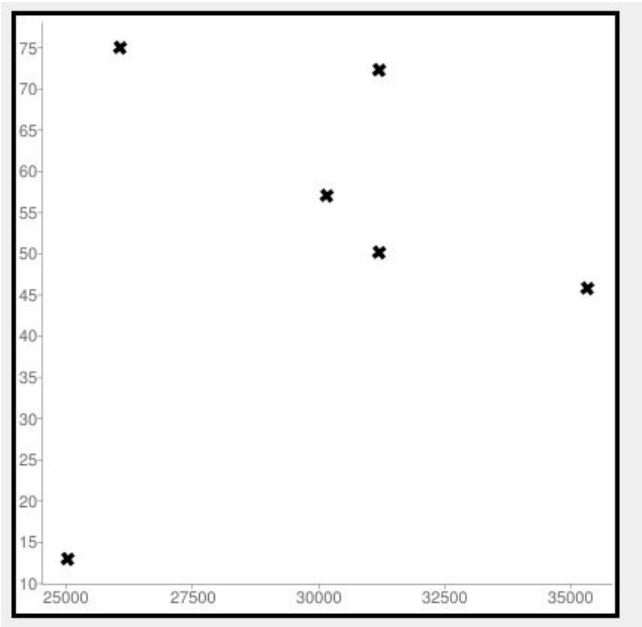
Bitrates:

Bitrate from host to client:

Host sending	Client receiving	Time difference (ms)	Bytes sent (bytes)	Bitrate (kb/s)
12:47:25 33	12:47:25 34	1	16	16
12:47:26 55	12:47:26 57	2	150	75
12:47:30 160	12:47:30 167	7	403	57
12:47:31 198	12:47:31 206	8	402	50.25
12:47:31 210	12:47:31 216	6	434	72.33
12:47:35 313	12:47:35 318	5	229	45.8
Average:				52.73
Standard diviation:				21.46

From the above table, we can tell that the average bit rate is about 52.73 kb/s.

A figure for network bitrate in kb/s over time is as following:



The x-axis is the time and the y-axis is bit rate in kb/s.

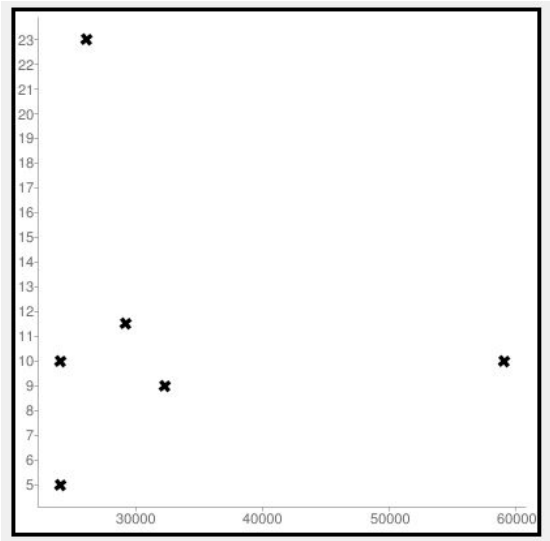
From the graph, we can tell that the bit rate from host to client is around 60 kb/s. The high variance may be caused by the instablity of the internet speed.

Bitrate from client to host:

Client sending	Host receiving	Time difference (ms)	Bytes sent (bytes)	Bitrate (kb/s)
15:03:59 22	15:03:59 23	1	10	10
15:03:24 19	15:03:24 21	2	20	10
15:03:24 29	15:03:24 31	2	10	5
15:03:26 82	15:03:26 83	1	23	23
15:03:29 194	15:03:29 196	2	23	11.5
15:03:32 275	15:03:32 276	1	9	9
Average:				11.42
Standard diviation:				6.09

From the above table, we can tell that the average bit rate is about 11.42 kb/s.

A figure for network bitrate in kb/s over time is as following:



The x-axis is the time and the y-axis is bit rate in kb/s.

From the figure, we can tell that the bit rate is between 5 and 30 kb/s. The over all data rate from client to server is relative smaller comparing to the data rate from server to client. This is probably because the packet sending from server to client is much larger than the packets from client to server.

b) Results and analysis for measuring in-game round trip time:

The measure of in-game round trip time is shown in the following table:

Time detecting key	Time receiving update msg	In-game round trip time (ms)
28/2/2016 20:33:45 162	28/2/2016 20:33:45 194	32
28/2/2016 20:35:51 998	28/2/2016 20:35:52 22	24
28/2/2016 20:36:47 581	28/2/2016 20:36:47 613	31
28/2/2016 20:37:47 647	28/2/2016 20:37:47 699	52
28/2/2016 20:38:42 353	28/2/2016 20:38:42 386	33
Average:		34.4
Standard deviation:		10.45

From the above table, we can tell that the average in-game round trip time is about 34.4 ms. The round trip time is much smaller than human can detected, as a reuslt, there's no noticiable legging in the game.

Scalabilty to more people: When multiple players are participated, the game will first send all the objects to all the players, as a result, at the start of game, there may be some small legging, but after the game started, there won't be such big packets since there won't be a lot of sychnorizations happening at the same time, so host don't need to send really big packets to clients. Also, since the packets client sending to host are really small, there won't be noticible latency when the game is started.

Playability over networks: According to the tables above, when the network speed is really low, the game may experience some delay at the start. When the normal game play starts, it is not likely to experiencing delay even if the network condition is not very good since there won't be too much data to sync.