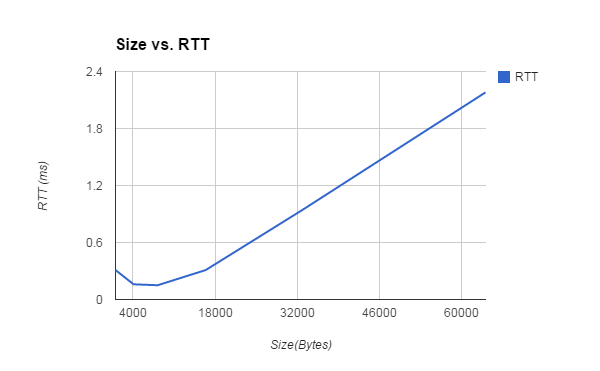
TCP Report

The purpose of TCP experiments to investigate the relationship between payload size, RTT, and throughput in the TCP protocol. This is achieved by sending payloads of various sizes from one a client to a server.

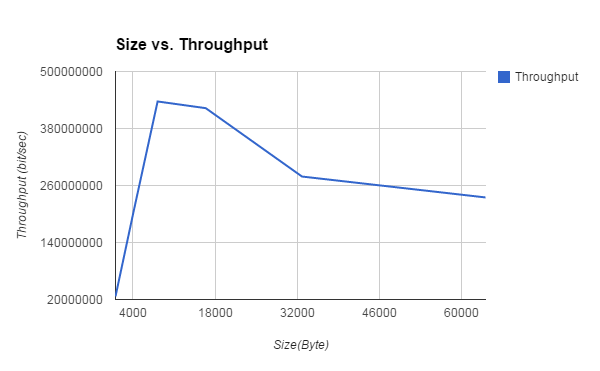
The first experiment to find out how the size of a payload will affect throughput and RTT. We sent payloads of 1KB, 4KB, 8 KB, 16KB, 32KB, and 64KB from a client to a server (both behind same router). The results are presented below.

|  |  |
| --- | --- |
| Size(byte) | RTT(ms) |
| 1024 | 0.31 |
| 4096 | 0.16 |
| 8192 | 0.15 |
| 16384 | 0.31 |
| 32768 | 0.94 |
| 64000 | 2.18 |



From the graph, it is clearly that RTT is proportionally to payload size. This is because the MTU of the Ethernet is 1500 bytes. Any package that is greater than 1500 bytes have to be broken into segments. Thus, more packets have to be sent, and the overall RTT increases.

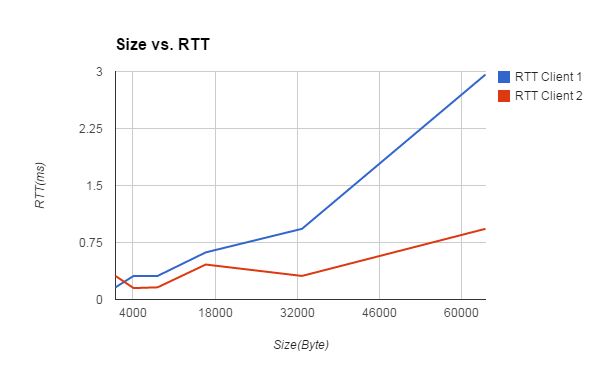
|  |  |
| --- | --- |
| Size(byte) | Throughput (bit/sec) |
| 1024 | 26425806 |
| 4096 | 204800000 |
| 8192 | 436906666 |
| 16384 | 422812903 |
| 32768 | 278876595 |
| 64000 | 234862385 |



Throughput also increases with payload size .However, if the packet become way too big and have to be broken down into segments, and the overall throughput will decrease due to more packets have to be sent in order to transmit a complete message. The graph shows that throughput only increases for a certain range and fall after payloads become too big.

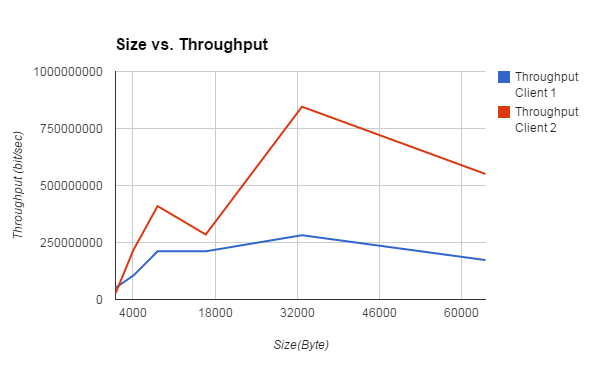
The second experiment is the repeat of the first experiment. Except this time we simultaneously send packets to a server from two different clients. The experiment is conducted so that both clients and the server are behind same router. Below is the outcome of the experiment

|  |  |  |
| --- | --- | --- |
| Size(Byte) | RTT Client 1 (ms) | RTT Client 2 (ms) |
| 1024 | 0.16 | 0.31 |
| 4096 | 0.31 | 0.15 |
| 8192 | 0.31 | 0.16 |
| 16384 | 0.62 | 0.46 |
| 32768 | 0.93 | 0.31 |
| 64000 | 2.96 | 0.93 |



From the graph, both clients’ RTT are proportional to payload size as expected. While the result of client1 (same machine as the first experiment) doesn’t deviate too much from the first experiment, the result of client2 have overall lower RRT. One explanation is the router gives client2 better QoS. According to Wireshark, there are no interweaving requests . This means the server serves the clients in FCFS basis, and no clients can interrupt the server while the server is in the middle of sending a complete message (i.e message that is bigger than MTU).

|  |  |  |
| --- | --- | --- |
| Size | Throughput Client 1(bit/sec) | Throughput Client 2 (bit/sec) |
| 1024 | 51200000 | 26425806 |
| 4096 | 105703225 | 218453333 |
| 8192 | 211406451 | 409600000 |
| 16384 | 211406451 | 284939130 |
| 32768 | 281875268 | 845625806 |
| 64000 | 172972972 | 550537634 |



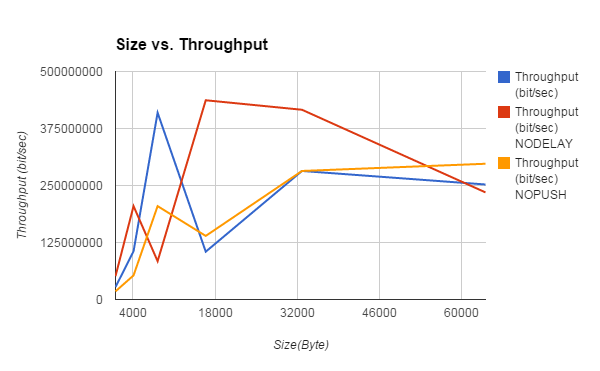
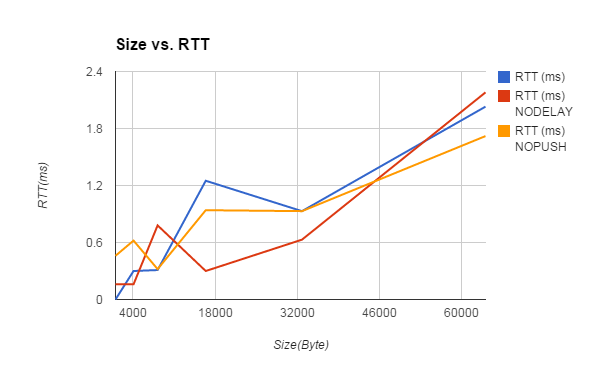
The throughput also behave like the first experiment. It increases proportionally at first but decreases if payload size becomes too big. Client2 have overall higher throughput than client1 due to lower RTT.

We also send 1 byte of data from a client to a server 1000 times. The outcome doesn’t seem to have any significance except it has very low RTT and low throughputs

TCP OPTIONS

We tested TCP\_NODELAY and TCP\_NOPUSH options that are available in setsockopt() method.

TCP\_NODELAY disable Nagle’s algorithm that buffer sent data. This means that packet will be sent immediately without delay without waiting for its buffer to fill. TCP\_NOPUSH does exact opposite of TCP\_NODELAY. It delays TCP data until its buffer is full. The result of experiment is presented below.



We expected that TCP\_NODELAY option will give us much lower RTT and high throughput, and TCP\_NOPUSH will give us higher RTT and low throughput. However, both option’s data doesn’t seem to have any significant change from data taken without any options.

Server Code

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <sys/types.h>

#include <time.h>

#define MSize 65000

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

{

if (argc < 2) {

printf("Error, port argument is missing\n");

return 0;

}

int server = 0, client = 0;

struct sockaddr\_in server\_addr;

struct sockaddr\_in client\_addr;

int len,sin\_size ;

char buf[MSize];

//create TCP socket

server = socket(AF\_INET, SOCK\_STREAM, 0);

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(atoi(argv[1]));

sin\_size = sizeof(struct sockaddr\_in);

//bind socket

bind(server, (struct sockaddr\*)&server\_addr, sizeof(server\_addr));

//listening for connection

listen(server, 65000);

puts("accepting connection\n");

while(1){

//accept connection

client = accept(server, (struct sockaddr\*)& client\_addr, &sin\_size);

if (client < 0) {

perror("error");

exit(1);

}

//keep communicating with client

while(len=recv(client, buf, sizeof(buf) , 0)>0){

//buf[len]='\0';

//printf("%s\n", buf);

char \*msg = "Hello!";

int size;

size = strlen(msg);

send(client, msg, sizeof(msg), 0);

//c++;

//printf("%d\n", c);

//bzero(buf, MSize);

}

close(client);

}

}

Client Code

#include<stdio.h> //printf

#include<string.h> //strlen

#include<sys/socket.h> //socket

#include<arpa/inet.h> //inet\_addr

#include <sys/time.h>

#define MSize 64000

void sendBytes(const char \*port, const char \*hostname, const int BYTE\_SIZE, const int LOOP\_AMOUNT);

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

{

if (argc < 3) {

printf("Error, port/ip address argument is missing\n");

return 0;

}

sendBytes(argv[1], argv[2], 1, 1000);

sendBytes(argv[1], argv[2], 1024, 1000);

sendBytes(argv[1], argv[2], 1024 \* 4, 1000);

sendBytes(argv[1], argv[2], 1024 \* 8, 1000);

sendBytes(argv[1], argv[2], 1024 \* 16, 1000);

sendBytes(argv[1], argv[2], 1024 \* 32, 1000);

sendBytes(argv[1], argv[2], 64000, 1000);

return 0;

}

void sendBytes(const char \*port, const char \*hostname, const int BYTE\_SIZE, const int LOOP\_AMOUNT) {

clock\_t startClock, endClock;

double elapsed=0.0f ;

double totalRTT=0.0f;

int sock;

struct sockaddr\_in server;

char server\_reply[MSize];

char Msg\_size[BYTE\_SIZE];

int x=0;

for(x = 0; x < BYTE\_SIZE; x++) {

Msg\_size[x] = 'a';

}

//Create socket

sock = socket(AF\_INET , SOCK\_STREAM , 0);

if (sock == -1)

{

printf("Could not create socket");

}

puts("Socket created");

//get server information

server.sin\_addr.s\_addr = inet\_addr(hostname);

server.sin\_family = AF\_INET;

server.sin\_port = htons( atoi(port) );

//Connect to server

if (connect(sock , (struct sockaddr \*)&server , sizeof(server)) < 0)

{

perror("connect failed. Error");

return;

}

puts("Connected\n");

//keep communicating with server

int y=0;

for(y=0;y<LOOP\_AMOUNT;y++){

startClock = clock();

//Send some data

if( send(sock , Msg\_size , BYTE\_SIZE, 0) < 0)

{

puts("Send failed");

return;

}

//Receive a reply from the server

int len=0;

if(len = recv(sock , server\_reply , sizeof(server\_reply) , 0) < 0)

{

puts("recv failed");

}

endClock = clock();

double x=(double)CLOCKS\_PER\_SEC;

elapsed = ((double)endClock - (double)startClock)/x;

totalRTT = totalRTT+elapsed;

}

printf("\n");

printf("Stats on sending %d bytes %d times\n", BYTE\_SIZE, LOOP\_AMOUNT);

double delay = totalRTT / (LOOP\_AMOUNT + 0.0f);

printf("Average RTT (delay): %fms\n", delay\*1000);

double throughput = BYTE\_SIZE\*8 / delay;

printf("Average throughput: %fbps\n", throughput);

printf("\n");

close(sock);

return ;

}