Cs553 cloud computing-program assignment1

1. overall program design

CPU MyCPUBench.c:

Program design: For cpu benchmark. Since we are going to benchmark the quarter precision , half precision, single precision and double precision speed of cpu, I use char compare as one quarter precision operation.

```
char ch1='a';
char ch2='b';
while(circle<WORKLOAD){
    while(current_process<WORKLOAD)
    {
        if(ch1<ch2) current_process++;
        else current_process++;
    }
    circle++;
    current_process=0;

}
```

use two short, int, double value compare as HP SP DP operation.

tradeoffs considered: for this program, when you do the loop, there is also a loop condition compare, circle<WORKLOAD and current_process<WORKLOAD compare, and also the ++ operation. which also use CPU operation that is not calculated in the my program, since this beyond my knowledge to accurately calculate but they should be considered.

Also, for accurate calculating, when I set the circle++, I should use mutex_lock for this operation, otherwise the circle++ will be not accurate since in multi thread environment, but add lock may slow the speed which is much more than the over added circle, so lock is not used.

Memory MyMemoryBenchmark.c

program design: Since we are going to benchmark the thoughput in sequencial and random access pattern in different block size. I use global variable bsize and access_pattern to set the value, and thread will execute deferent pattern by reading these global values, and also set the global value count with lock to totally execute 10G data.

```
pthread_mutex_lock(&mutex);
memcpy(temp,vx,bsize);
```

```
memset(vx, 'b', bsize);
vx=vx+bsize;
current_process++;
if(current_process>read_times-1){
    current_circle++;
    current_process=0;
    vx=VA;
}
```

tradeoffs considered: since there is cache used for the memory I/O, cache should be disabled or flushed.

DISK MyDiskBench.c

Program design: Use four function, write_random(), read_random(), write_sequential() and read_sequential() to execute RR, RS, WS, WR operation, also use global variable current_process to count the operation and bsize to set the block size, for latency test, use the same four function, just set the bsize to 1KB.

Also, to avoid the memory cache of the disk I/O, use

```
f = fopen("new_file.bin", "r");
setbuf(f, NULL);
```

to set the buffer to zero.

Implementation WS of the four functions:

```
void *write_sequence(void *args){
    char * buffer = (char *)malloc(bsize * sizeof(char));
    memset(buffer, '0', bsize);

    while(current_process<operate_times){
        // write to disk sequentially
        fwrite(buffer, 1, bsize, f);
        current_process++;
    }

    free(buffer);
    pthread_exit(NULL);
}
```

tradeoffs considered: Pretty straight forward, no tradeoffs.

NETWORK MyNETBench-server.c MyNETBench-client.c

Program design: two program, MyNETBench-server.c run in server and

MyNETBench-client.c run in client.
Use two function to handle TCP protocol and UDP protocol separately.
Same as memory and disk, use global variable bsize and current_process, and noperatons to tell threads how to execute. For latency test, reuse same function just to set bsize to 1KB and noperations to 1million times.
Implementation of udp_client sending data function:

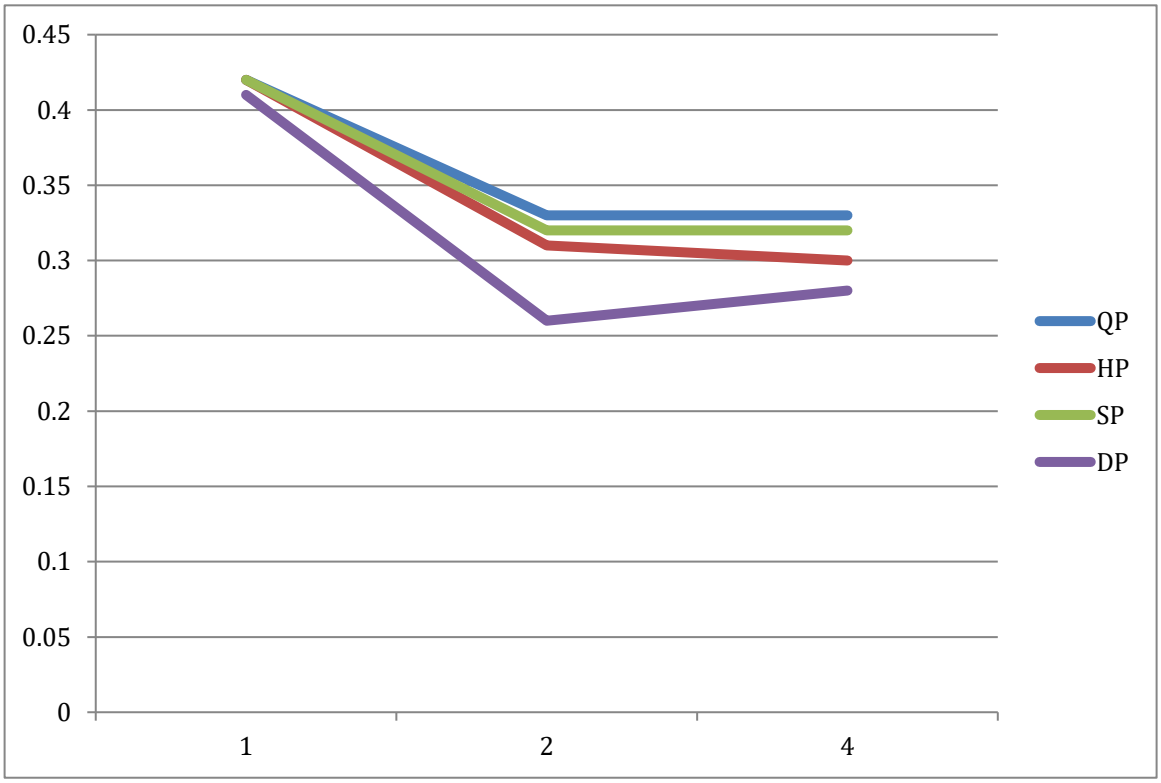
```
void *udp_client(void *args){
	char * buffer = (char *)malloc(bsize * sizeof(char));
	memset(buffer, '0', bsize);
	int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

	while(current_process<operate_times){
		sendto(sockfd, buffer, bsize, 0, (struct sockaddr *)&ser_add, sizeof(struct sockaddr));
		if(strcmp(bm_type,"LTC_UDP")==0){
			recvfrom(sockfd, buffer, bsize, 0, (struct sockaddr *)&ser_add, &sockaddr_in_size);
		}
		current_process++;

	}
	close(sockfd);
	free(buffer);
	pthread_exit(NULL);
}
```

tradeoffs considered: No tradeoffs

2.Benchmark result

CPU Benchmark

Workload	Concurrency	MyCPUBench Measured Ops/Sec (GigaOPS)	HPL Measured Ops/Sec (GigaOPS)	Theoretical Ops/Sec (GigaOPS)	MyCPUBench Efficiency (%)	HPL Efficiency (%)
QP	1	0.42	N/A	2355	0.018	N/A
QP	2	0.33	N/A	2355	0.014	N/A
QP	4	0.33	N/A	2355	0.036	N/A
HP	1	0.42	N/A	1177	0.026	N/A
HP	2	0.31	N/A	1177	0.025	N/A
HP	4	0.30	N/A	1177	0.071	N/A
SP	1	0.42	N/A	588	0.054	N/A
SP	2	0.32	N/A	588	0.054	N/A
SP	4	0.32	N/A	588	0.139	N/A
DP	1	0.41	64.8613	294	0.088	0.22
DP	2	0.26	59.1145	294	0.085	0.20
DP	4	0.28	46.4090	294	0.095	0.16



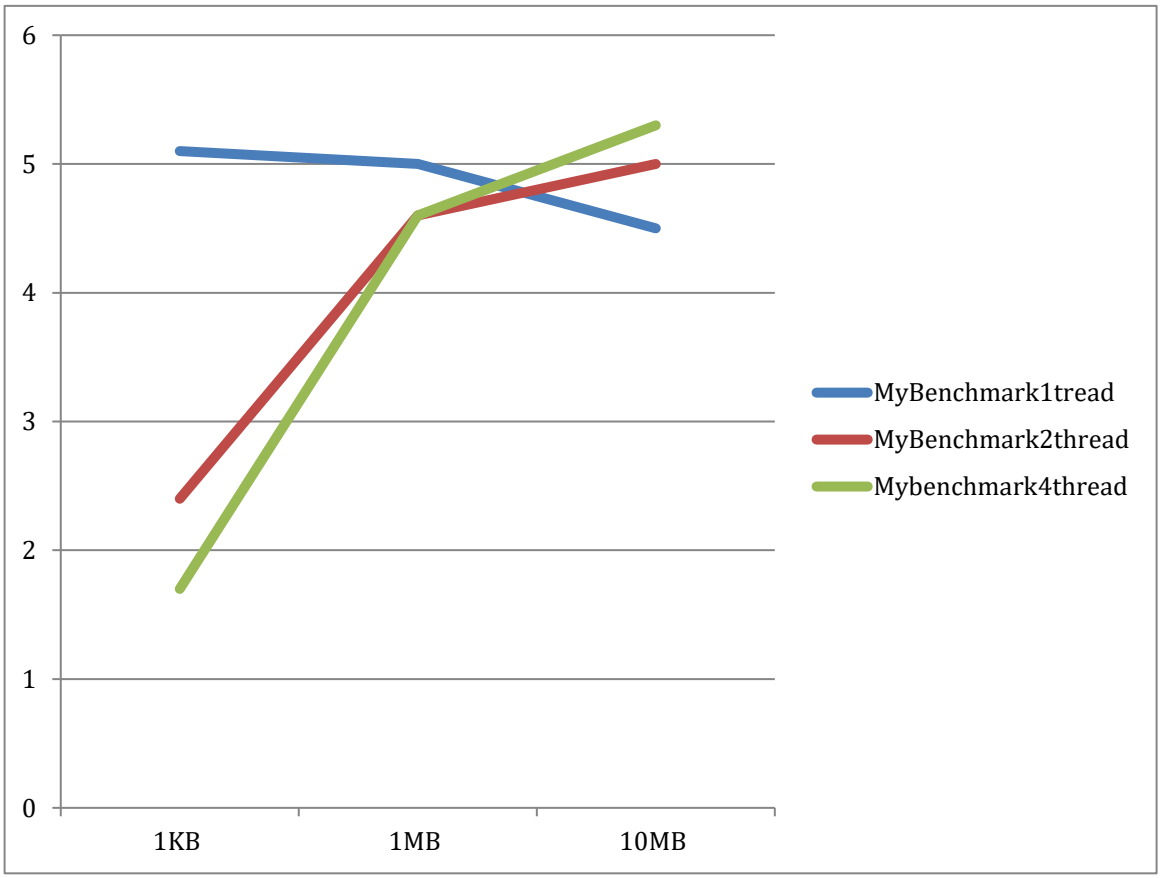
Conclusion: we can see that as the thread number grows from 1 2 4
the CPU speed slow down from about 400MHz to 250MHz
Quarter precision has fastest speed, double precision has lowest speed when operates with same times.
Because of my program doesn't take the loop operation into count, my result is absolutely much lower than the tool benchmark and theoretical value.

Memory Throughput

Work-load	Con-currency	Block Size	MyRAMBench Measured Throughput (GB/sec)	pmbw Measured Throughput (GB/sec)	Theoretical Throughput (GB/sec)	MyRAMBench Efficiency (%)	pmbw Efficiency (%)
RWS	1	1KB	5.1	40.1	54.6	9.3	73.4
RWS	1	1MB	5.0	44.7	54.6	9.2	81.8

RWS	1	10MB	4.5	47.4	54.6	8.2	86.8
RWS	2	1KB	2.4	25.6	54.6	4.4	46.8
RWS	2	1MB	4.6	22.8	54.6	8.4	41.7
RWS	2	10MB	5.0	23.5	54.6	9.2	43.0
RWS	4	1KB	1.7	20.2	54.6	3.1	36.9
RWS	4	1MB	4.6	22.3	54.6	8.4	40.8
RWS	4	10MB	5.3	23.9	54.6	9.7	43.7
RWR	1	1KB	3.1	39.6	54.6	5.7	72.5
RWR	1	1MB	5.7	45.4	54.6	10.4	83.1
RWR	1	10MB	4.5	50.3	54.6	8.2	92.1
RWR	2	1KB	1.9	19.5	54.6	3.5	35.7
RWR RWR	2	1MB	4.7	24.7	54.6	8.6	45.2
RWR	2	10MB	4.8	24.1	54.6	8.8	44.1
RWR	4	1KB	0.8	17.7	54.6	1.5	32.4
RWR	4	1M	5.2	21.1	54.6	9.5	38.6

		B					
RWR	4	10MB	4.8	24.6	54.6	8.8	45.1



Conclusion: with thread number increase, the thought put drops quickly from about 5G/sec to 2G/sec in block size 1KB. But vary little with 1MB and 10MB. In 1MB and 10BM size, the though put get best performance.

Memory Latency

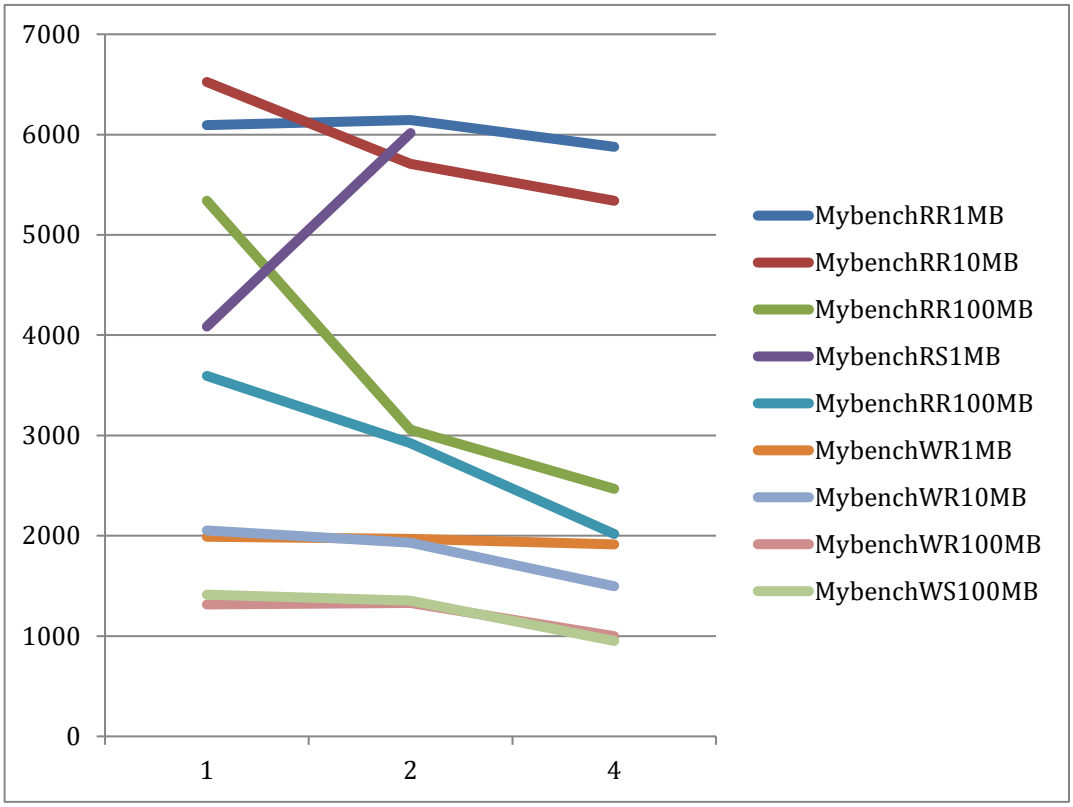
Work-load	**Con-currency**	**Block Size**	**MyRAMBench Measured** Latency (us)	**pmbw Measured Latency**	**Theoretical Latency (us)**	**MyRAMBench Efficiency (%)**	**pmbw Efficiency (%)**
RWS	1	1B	0.2	0.06	0.014	7	23
RWS	2	1B	0.4	0.15	0.014	3.5	9.3
RWS	4	1B	0.7	0. 34	0.014	2	3
RWR	1	1B	0.3	0.27	0.014	4.7	5.2
RWR	2	1B	0.5	0.24	0.014	2.8	5.8
RWR	4	1B	1.0	0.42	0.014	1.4	3.3

Conclusion : As we can see from the table, as thread number increases, the latency also increase.

Disk Throughput

Work-load	Con-currency	Block Size	MyDiskBench Measured Throughput (MB/sec)	IOZone Measured Throughput (MB/sec)	Theoretical Throughput (MB/sec)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RS	1	1MB	1682	244	540	311	39
RS	1	10MB	5767	528	540	1067	97
RS	1	100 MB	3594	356	540	665	85
RS	2	1MB	4086	212	540	756	65
RS	2	10MB	5570	462	540	1031	45
RS	2	100 MB	2923	244	540	541	45
RS	4	1MB	6015	528	540	1113	39
RS	4	10MB	5378	356	540	995	87
RS	4	100 MB	2018	212	540	373	39
WS	1	1MB	1335	462	420	317	97
WS	1	10MB	1397	244	420	332	86
WS	1	100 MB	1413	528	420	336	65
WS	2	1MB	1382	356	420	329	65
WS	2	10MB	1382	212	420	321	45
WS	2	100 MB	1352	462	420	389	39
WS	4	1MB	1635	244	420	366	54
WS	4	10M	1539	528	420	226	97

		B					
WS	4	100 MB	951	356	420	1128	86
RR	1	1MB	6094	212	540	1208	58
RR	1	10MB	6524	462	540	988	85
RR	1	100 MB	5340	244	540	1137	45
RR	2	1MB	6145	528	540	1057	39
RR	2	10MB	5708	356	540	566	45
RR	2	100 MB	3058	212	540	1088	97
RR	4	1MB	5878	462	540	988	84
RR	4	10MB	5340	244	540	457	58
RR	4	100 MB	2468	528	540	473	50
WR	1	1MB	1990	356	420	473	50
WR	1	10MB	2053	212	420	488	125
WR	1	100 MB	1315	462	420	313	110
WR	2	1MB	1970	244	420	468	110
WR	2	10MB	1932	528	420	460	84
WR	2	100 MB	1330	356	420	316	50
WR	4	1MB	1914	212	420	455	58
WR	4	10MB	1497	462	420	356	125
WR	4	100 MB	999	210	420	237	110



Conclusion: As we can see from the figure, disk thoughput in read mode is obviously faster than write mode, 1MB and 10MB block size is faster than100 MB size.
As thread increase, the thoughtput has a slightly decrease.

Disk Latency

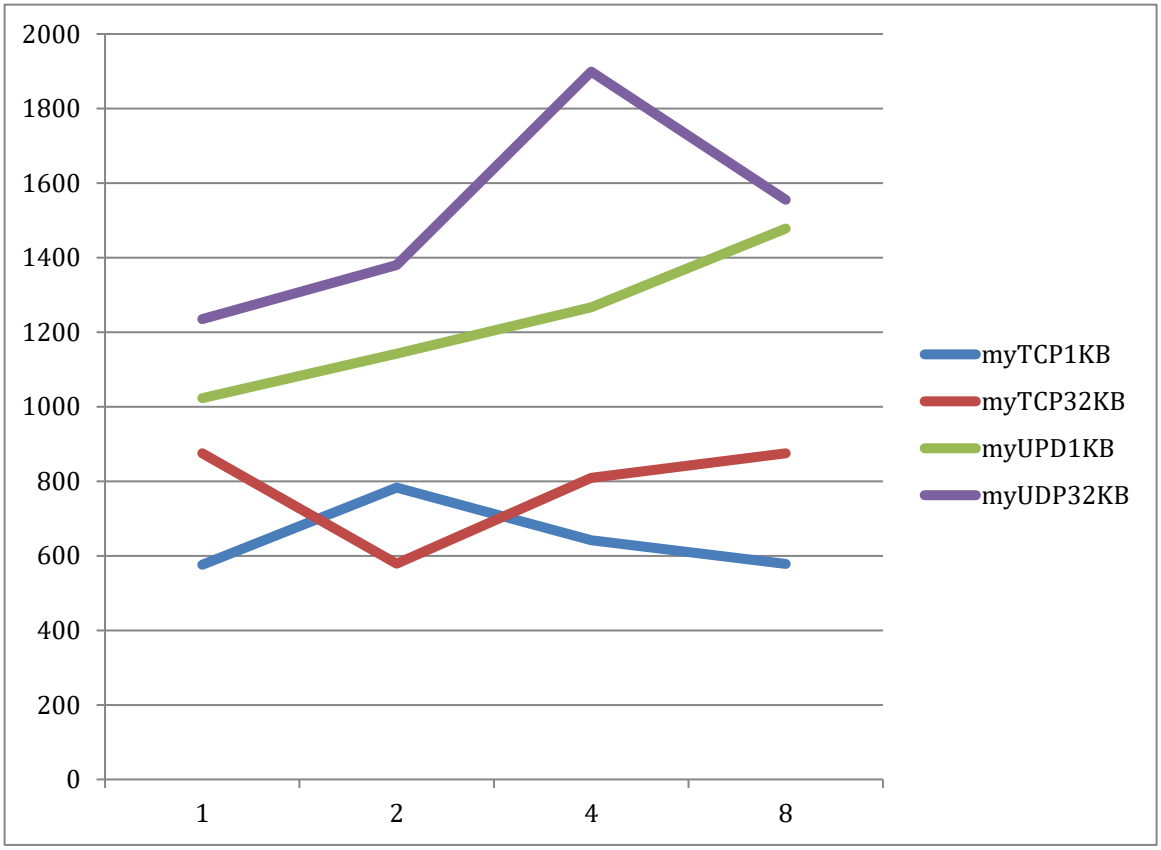
Work-load	Con-currency	Block Size	MyDiskBench Measured Latency (ms)	IOZone Measured Latency (ms)	Theoretical Measured Latency (ms)	MyDiskBench Efficiency (%)	IOZone Efficiency (%)
RR	1	1KB	0.05	0.03	0.013	26	43
RR	2	1KB	0.03	0.02	0.013	43	65
RR	4	1KB	0.02	0.02	0.013	65	65
RR	8	1KB	0.02	0.02	0.013	65	65
RR	16	1KB	0.03	0.04	0.013	43	32
RR	32	1KB	0.04	0.06	0.013	32	21
RR	64	1KB	0.06	0.06	0.013	21	21

RR	128	1KB	0.07	0.08	0.013	18	16
WR	1	1KB	0.05	0.02	0.013	26	65
WR	2	1KB	0.00	0.02	0.013	0	43
WR	4	1KB	0.00	0.03	0.013	0	65
WR	8	1KB	0.00	0.04	0.013	0	65
WR	16	1KB	0.00	0.06	0.013	0	65
WR	32	1KB	0.00	0.07	0.013	0	32
WR	64	1KB	0.00	0.12	0.013	0	21
WR	128	1KB	0.00	0.14	0.013	0	21

Conclusion: As we can see from the table , disk latency are all in 0.0x latitude.

Network Throughput

Proto-col	Con-currency	Block Size	MyNETBench Measured Throughput	iperf Measured Throughput (Mb/sec)	Theoretical Throughput (Mb/sec)	MyNETBench Efficiency (%)	iperf Efficie (%)
TCP	1	1KB	576	915	1960	29	46
TCP	1	32KB	875	1090	1960	44	55
TCP	2	1KB	783	1080	1960	39	55
TCP	2	32KB	579	1100	1960	29	56
TCP	4	1KB	642	988	1960	32	50
TCP	4	32KB	809	895	1960	41	45
TCP	8	1KB	578	894	1960	29	45
TCP	8	32KB	875	1040	1960	44	53
UDP	1	1KB	1023	1030	1960	52	52
UDP	1	32KB	1235	1010	1960	63	51
UDP	2	1KB	1142	990	1960	58	50
UDP	2	32KB	1380	1050	1960	70	53
UDP	4	1KB	1267	1020	1960	64	52
UDP	4	32KB	1899	990	1960	96	50
UDP	8	1KB	1478	1020	1960	75	52
UDP	8	32KB	1555	1040	1960	79	53



Conclusion: From the figure we can see that UDP has a faster speed than TCP
Thread number from 1 2 4 8 doesn't significantly effect the speed.
32KB size has better speed than 1KB

Network Latency:

Proto-col	Con-currency	Block Size	MyNETBench Measured Latency (ms)	iperf Measured Latency (ms)	Theoretical Latency (ms)	MyNETBench Efficiency (%)	iperf Efficiency (%)
TCP	1	1B	0.08	0.036	0.0256	45	84
TCP	2	1B	0.09	0.045	0.0256	50	90
TCP	4	1B	0.17	0.034	0.0256	53	80
TCP	8	1B	0.21	0.022	0.0256	46	62
UDP	1	1B	0.03	0.026	0.0256	52	99
UDP	2	1B	0.05	0.032	0.0256	64	90
UDP	4	1B	0.07	0.025	0.0256	44	52
UDP	8	1B	0.11	0.034	0.0256	70	72