

Computer Networks Task1

Frolov Ivan

Note: the file was automatically converted to a PDF. Sorry if the formatting is not perfect

Here are some screenshots on running a VM

Running a Yandex VM

The screenshot shows the Compute Cloud interface for managing virtual machines. On the left, a sidebar lists options like 'Обзор' (Overview), 'Диски и хранилища' (Disks and storage), 'Резервные копии' (Backups), 'Права доступа' (Access rights), 'Операции' (Operations), 'Мониторинг' (Monitoring), 'Серийная консоль' (Serial console), and 'Последовательный порт' (Serial port). The 'Обзор' option is selected. The main panel displays details for a virtual machine named 'compute-vm-2-1-10-hdd-1768992849890'. The status is shown as 'Running'. Other details include its identifier ('epdrvthtisgg39dovbue'), name, creation date ('21.01.2026, в 1:58 дня'), internal FQDN ('compute-vm-2-1-10-hdd-1768992849890.ru-central1.internal'), and availability zone ('ru-central1-b'). Below this, under 'Доступ' (Access), it shows 'Доступ по OS Login' (Access via OS Login) as 'Выключен' (Disabled). It provides instructions for SSH access: 'Подключиться с помощью SSH-клиента' (Connect via SSH client) with the command 'ssh -l user 178.154.199.103'. It also mentions that there is an 'инструкция по восстановлению доступа к ВМ' (instructions for recovering access to the VM) available if the SSH key is lost. Another section for connecting via the 'CLI Yandex Cloud' is partially visible.

Server IP: 178.154.199.103

Client IP (started locally): 192.168.3.10

Server sends to a client

tcp.port == 10000

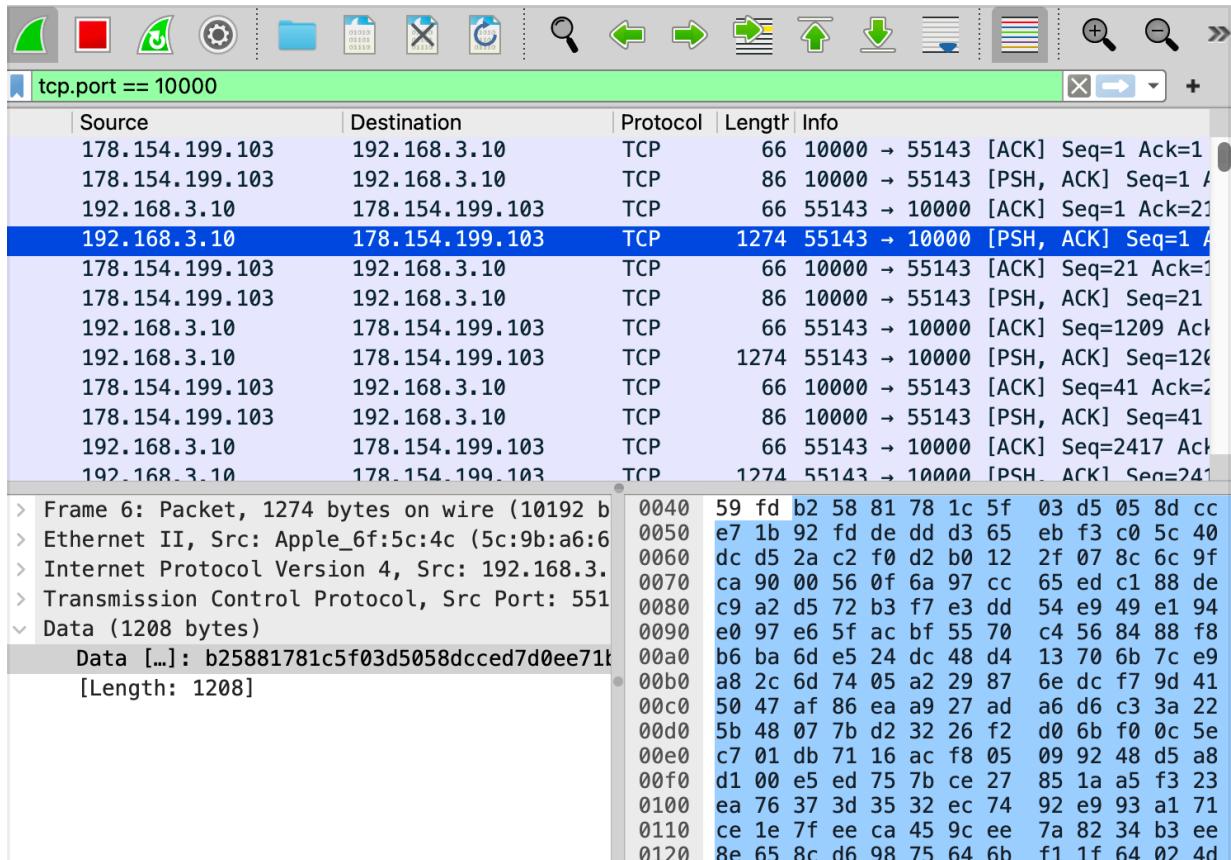
Source	Destination	Protocol	Length	Info
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=1 Ack=
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=1 Ack=21 Wi
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=1 Ack=
178.154.199.103	192.168.3.10	TCP	66	10000 → 55143 [ACK] Seq=21 Ack=1209
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=21 Ack
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=1209 Ack=41
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=1209 A
178.154.199.103	192.168.3.10	TCP	66	10000 → 55143 [ACK] Seq=41 Ack=2417
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=41 Ack
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=2417 Ack=61
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=2417 A

> Frame 2: Packet, 86 bytes on wire (688 bits)
 > Ethernet II, Src: HuaweiDevice_45:a8:49 (c8:
 > Internet Protocol Version 4, Src: 178.154.19
 > Transmission Control Protocol, Src Port: 100
 > Data (20 bytes)

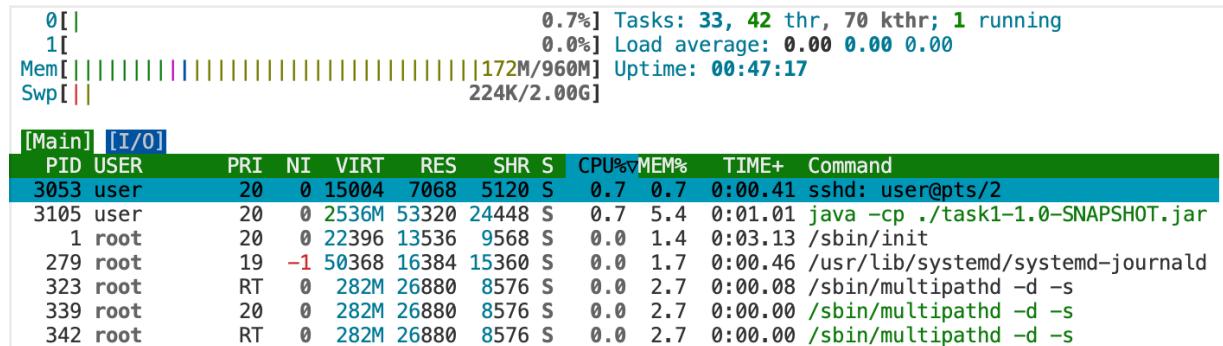
```
  Data: 323032362e30312e32312031313a35373a30
  [Length: 20]
```

0000	5c	9b	a6	6f	5c	4c	c8	bc	9c	45	a8	49	08	00
0010	00	48	b1	fa	40	00	39	06	51	ff	b2	9a	c7	00
0020	03	0a	27	10	d7	67	10	67	b2	78	fb	8f	e4	00
0030	26	fc	f5	4e	00	00	01	01	08	0a	e6	0d	59	00
0040	16	0b	32	30	32	36	2e	30	31	2e	32	31	20	00
0050	35	37	3a	30	32	0a								

Client sends to a server



The process is not very heavy (using htop)



Note: at first my server was breaking

That's what I mean:

It started reading bytes from client normally

Then the messages started arriving very very fast

And finally the server just locked and didn't receive anything

Just hanging like this

The way I discovered to fix it was to send to the server not only the bytes array

with data but also the size of that buffer

```
ByteBuffer bb = ByteBuffer.allocate(4);
    bb.putInt(dataLength);
    out.write(bb.array());
    out.write(dataToSend);
```

And then the server will read exactly the size of the buffer

That was crazy to fix so hopefully it works correctly

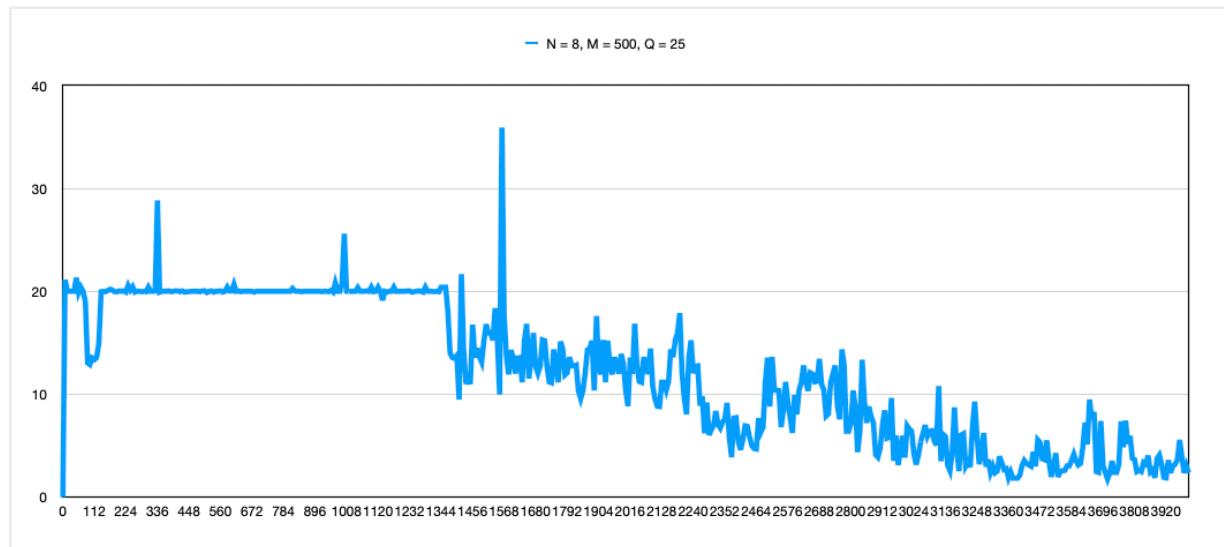
Note: I'll be using $N = 500$ or less
 $N = 5000$ is incredibly slow
I just runs forever

Local test

No sure it it is relevant

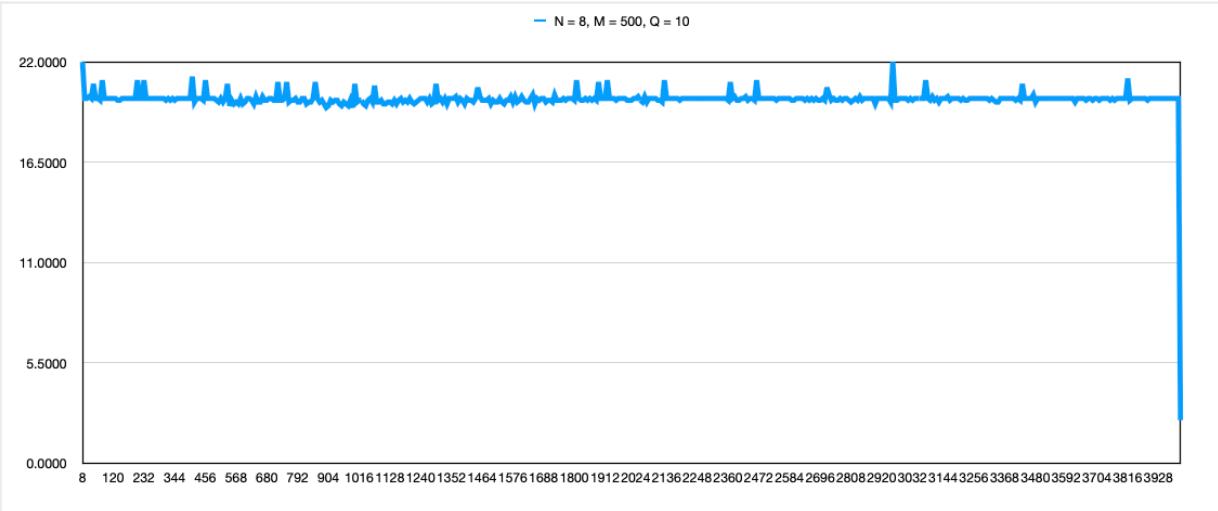
But the time depends on the array size almost linearly

$N=8, M=500, Q=25$



Now let's do the VM

$N = 8, M = 500, Q = 10$

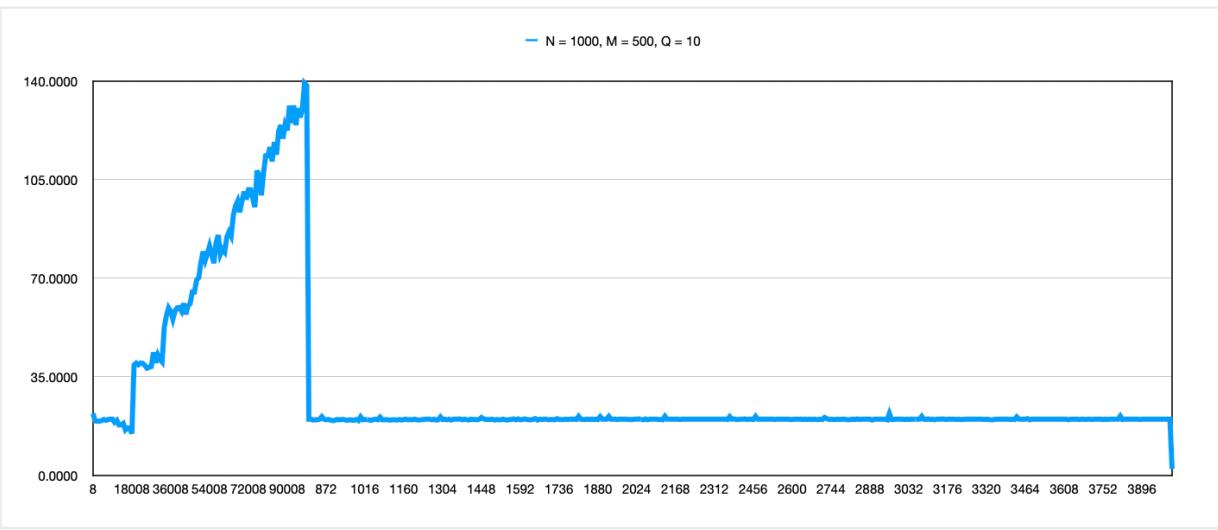


Our first test

With low N the speed pretty much stays the same

The increases in array sizes have almost no influence on speed

N = 1000, M=100, Q = **10**



When using N = 1000 it got much slower

So using M = 100 iterations here (I misclicked on the picture and wrote M=500; it is actually M=100)

We can see that the time really increases as the array size grows

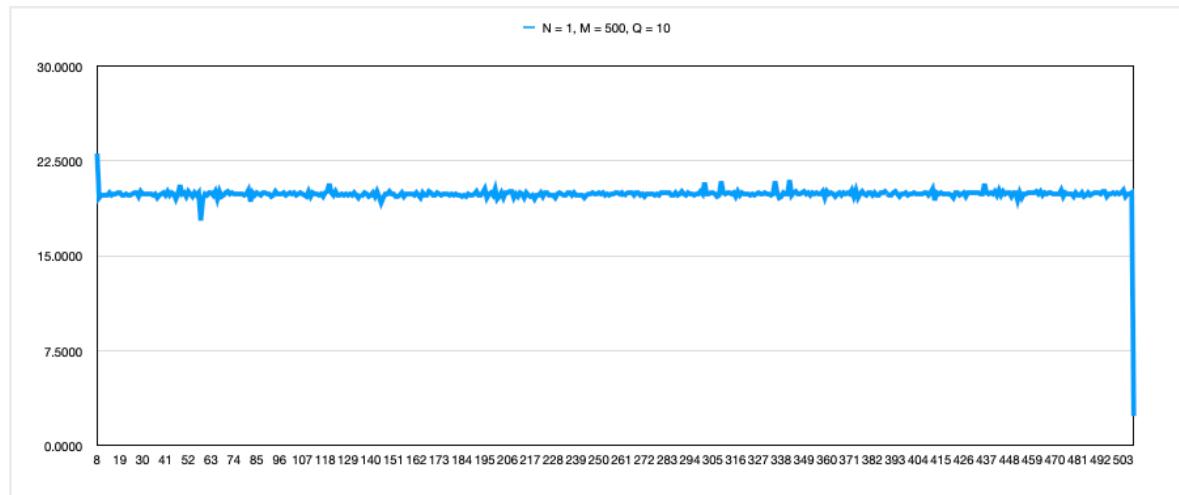
And then it hits a threshold and drops, staying low

Honestly that's quite interesting

I've read it is related to header overhead

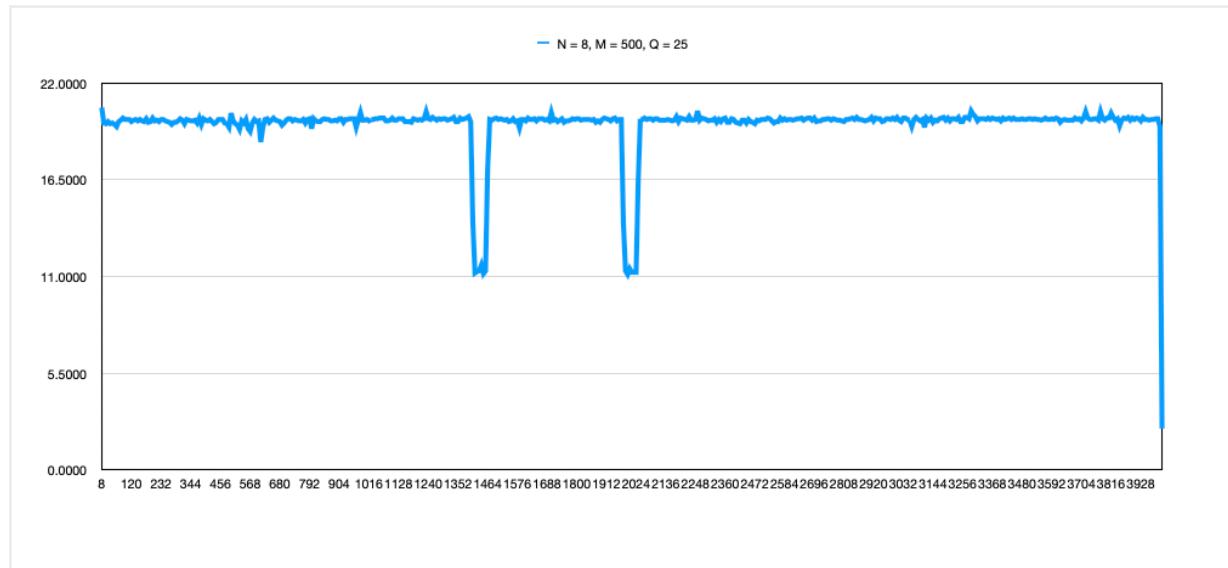
So as the array size gets large enough we can send them as full packets which is more efficient

N=1, M=500, Q=10



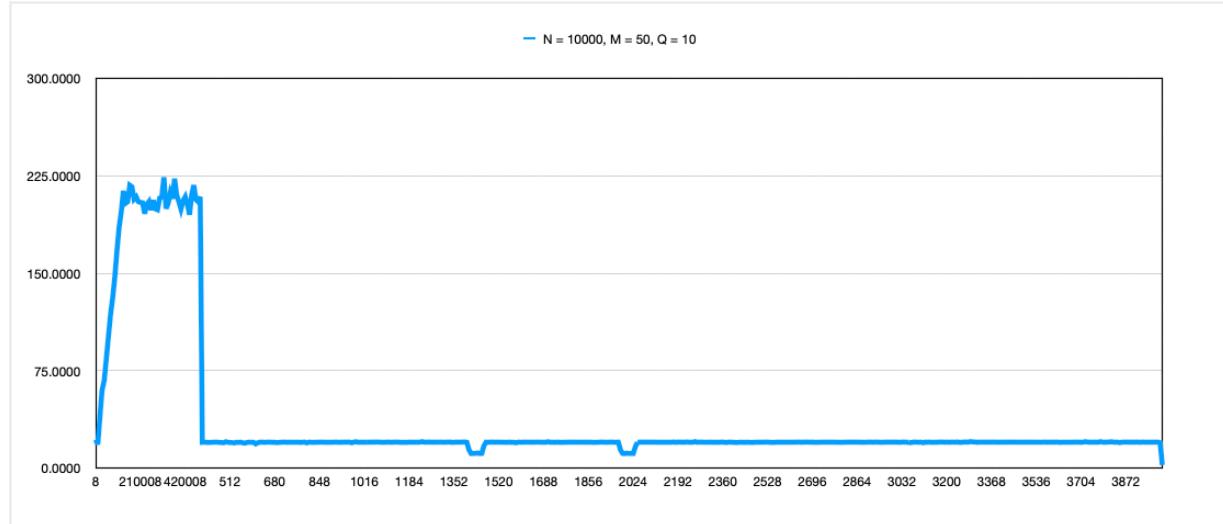
Here we will see if making N = 1 (small sizes) improves the speed
It is actually pretty much the same as N = 8

N=8, M=500, Q=25



Here we will see what difference increasing Q makes
It is similar to our first VM graph - stable time across the array sizes
And we see two drops - probably some Java / Network optimizations

N=10,000, M=50, Q=10



I was really interested to see if sending super big array sizes influences the speed
Making $N=10,000$

We are running just for about 50 iterations so that I don't wait for a week)
Looking at the logs in CLI, it gets much slower (at least feels like that)

Looking at the graph, we see similar picture to $N=1000$

Yet it seems like at first the speed is not optimized at all is consistently very high -
about 225,000

And then it gets optimized to the level of $N=1000$

Maybe it is because the longer the connection runs the more optimized it gets
So that's cool