

# 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 displays the Yandex Cloud console interface for a specific virtual machine. On the left, a sidebar contains navigation links: '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). Below these is a 'Документация' (Documentation) section with links to 'Создать виртуальную машину' (Create virtual machine), 'Тарифы Compute Cloud' (Compute Cloud rates), and 'Настройка синхронизации времени NTP' (NTP time synchronization settings).

The main panel is titled 'Обзор' (Overview) and shows the following details for the VM 'compute-vm-2-1-10-hdd-1768992849890':

- Идентификатор (Identifier): epdrvthtisgg39dovbue
- Статус (Status): Running
- Имя (Name): compute-vm-2-1-10-hdd-1768992849890
- Дата создания (Creation date): 21.01.2026, в 1:58 дня
- Внутренний FQDN (Internal FQDN): compute-vm-2-1-10-hdd-1768992849890.ru-central1.internal
- Зона доступности (Availability zone): ru-central1-b

Below the overview, the 'Доступ' (Access) section shows 'Доступ по OS Login' (OS Login access) is 'Выключен' (Disabled). There are two expandable sections for connection instructions:

- Подключиться с помощью SSH-клиента** (Connect using SSH client):  
Для подключения к машине с помощью SSH используйте следующую команду:  

```
ssh -l user 178.154.199.103
```

  
В случае утери SSH-ключа есть [инструкция по восстановлению доступа к ВМ](#).
- Подключиться с помощью CLI Yandex Cloud** (Connect using Yandex Cloud CLI)

Server IP: 178.154.199.103

Client IP (started locally): 192.168.3.10

Server sends to a client

Client sends to a server

tcp.port == 10000									
Source	Destination	Protocol	Length	Info					
178.154.199.103	192.168.3.10	TCP	66	10000 → 55143 [ACK] Seq=1 Ack=1					
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=1 A					
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=1 Ack=21					
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=1 A					
178.154.199.103	192.168.3.10	TCP	66	10000 → 55143 [ACK] Seq=21 Ack=1					
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=21					
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=1209 Ack					
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=120					
178.154.199.103	192.168.3.10	TCP	66	10000 → 55143 [ACK] Seq=41 Ack=2					
178.154.199.103	192.168.3.10	TCP	86	10000 → 55143 [PSH, ACK] Seq=41					
192.168.3.10	178.154.199.103	TCP	66	55143 → 10000 [ACK] Seq=2417 Ack					
192.168.3.10	178.154.199.103	TCP	1274	55143 → 10000 [PSH, ACK] Seq=241					
> Frame 6: Packet, 1274 bytes on wire (10192 b				0040	59 fd b2 58 81 78 1c 5f	03 d5 05 8d cc			
> Ethernet II, Src: Apple_6f:5c:4c (5c:9b:a6:6				0050	e7 1b 92 fd de dd d3 65	eb f3 c0 5c 40			
> Internet Protocol Version 4, Src: 192.168.3.				0060	dc d5 2a c2 f0 d2 b0 12	2f 07 8c 6c 9f			
> Transmission Control Protocol, Src Port: 551				0070	ca 90 00 56 0f 6a 97 cc	65 ed c1 88 de			
Data (1208 bytes)				0080	c9 a2 d5 72 b3 f7 e3 dd	54 e9 49 e1 94			
Data [...]: b25881781c5f03d5058dced7d0ee71k				0090	e0 97 e6 5f ac bf 55 70	c4 56 84 88 f8			
[Length: 1208]				00a0	b6 ba 6d e5 24 dc 48 d4	13 70 6b 7c e9			
				00b0	a8 2c 6d 74 05 a2 29 87	6e dc f7 9d 41			
				00c0	50 47 af 86 ea a9 27 ad	a6 d6 c3 3a 22			
				00d0	5b 48 07 7b d2 32 26 f2	d0 6b f0 0c 5e			
				00e0	c7 01 db 71 16 ac f8 05	09 92 48 d5 a8			
				00f0	d1 00 e5 ed 75 7b ce 27	85 1a a5 f3 23			
				0100	ea 76 37 3d 35 32 ec 74	92 e9 93 a1 71			
				0110	ce 1e 7f ee ca 45 9c ee	7a 82 34 b3 ee			
				0120	8e 65 8c d6 98 75 64 6b	f1 1f 64 02 4d			

The process is not very heavy (using htop)

0.7% Tasks: 33, 42 thr, 70 kthr; 1 running

0.0% Load average: 0.00 0.00 0.00

Uptime: 00:47:17

Mem[|||||] 172M/960M

Swp[||] 224K/2.00G

[Main]

[I/O]

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3053	user	20	0	15004	7068	5120	S	0.7	0.7	0:00.41	sshd: user@pts/2
3105	user	20	0	2536M	53320	24448	S	0.7	5.4	0:01.01	java -cp ./task1-1.0-SNAPSHOT.jar
1	root	20	0	22396	13536	9568	S	0.0	1.4	0:03.13	/sbin/init
279	root	19	-1	50368	16384	15360	S	0.0	1.7	0:00.46	/usr/lib/systemd/systemd-journald
323	root	RT	0	282M	26880	8576	S	0.0	2.7	0:00.08	/sbin/multipathd -d -s
339	root	20	0	282M	26880	8576	S	0.0	2.7	0:00.00	/sbin/multipathd -d -s
342	root	RT	0	282M	26880	8576	S	0.0	2.7	0:00.00	/sbin/multipathd -d -s

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

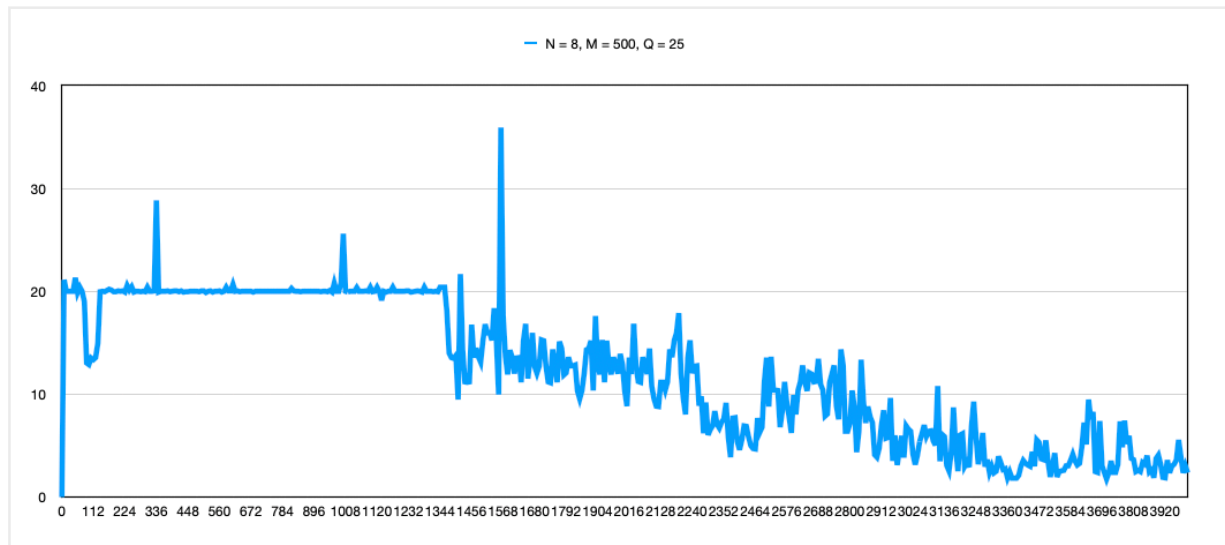
---

Local test

No sure it it is relevant

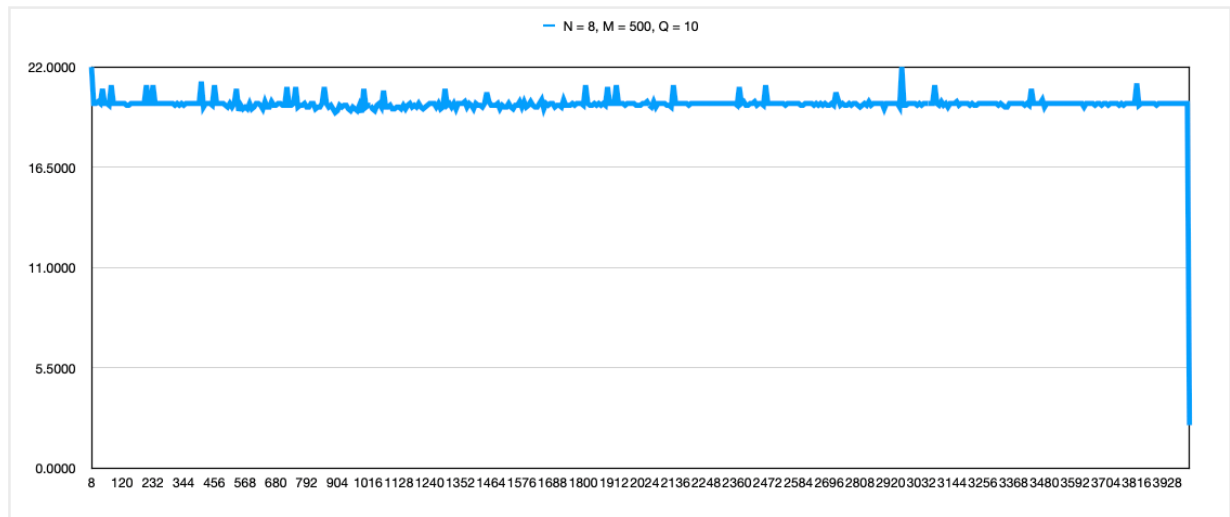
But the time depends on the array size almost linearly

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



Now let's do the VM

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

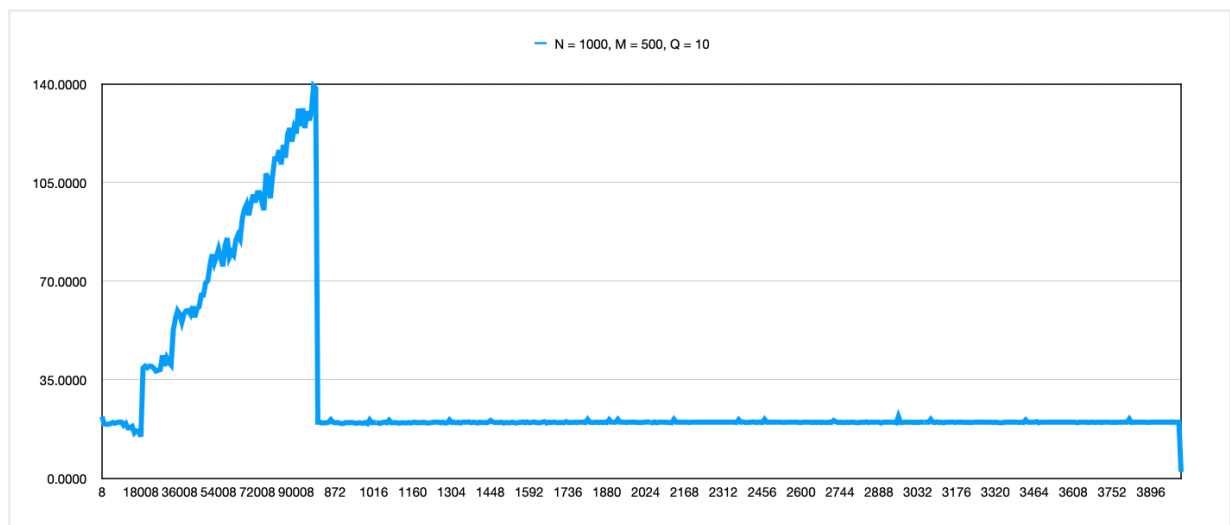


Our first test

With low N the speed pretty much stays the same

The increases in array sizes have almost no influence on speed

$N = 1000, M = 500, Q = 10$



When using  $N = 1000$  it got much slower

So using  $M = 100$  iterations here

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

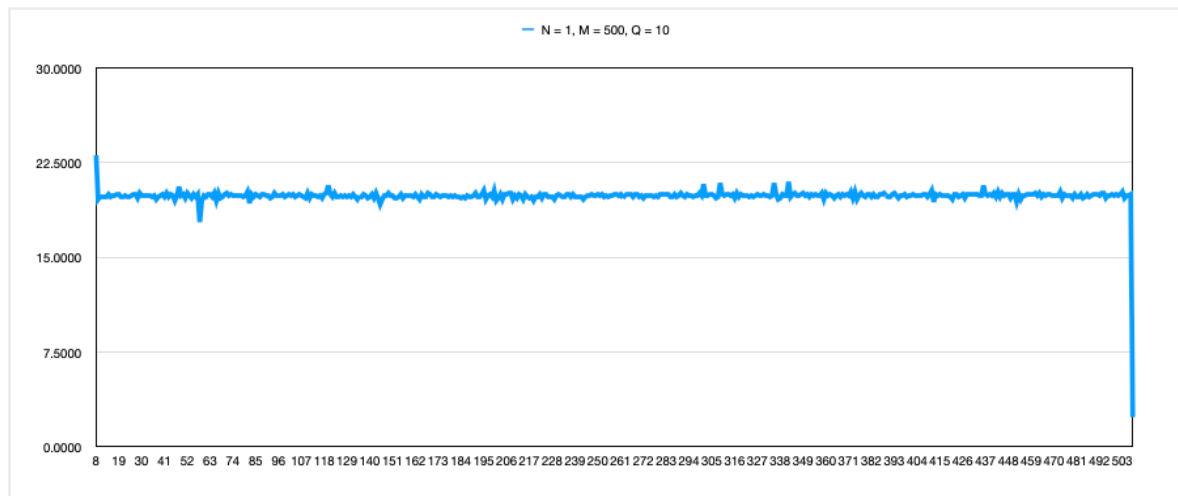
And then it hits a threshold and drops, staying low

Honestly that's quite interesting

I've read it is related to header overhead

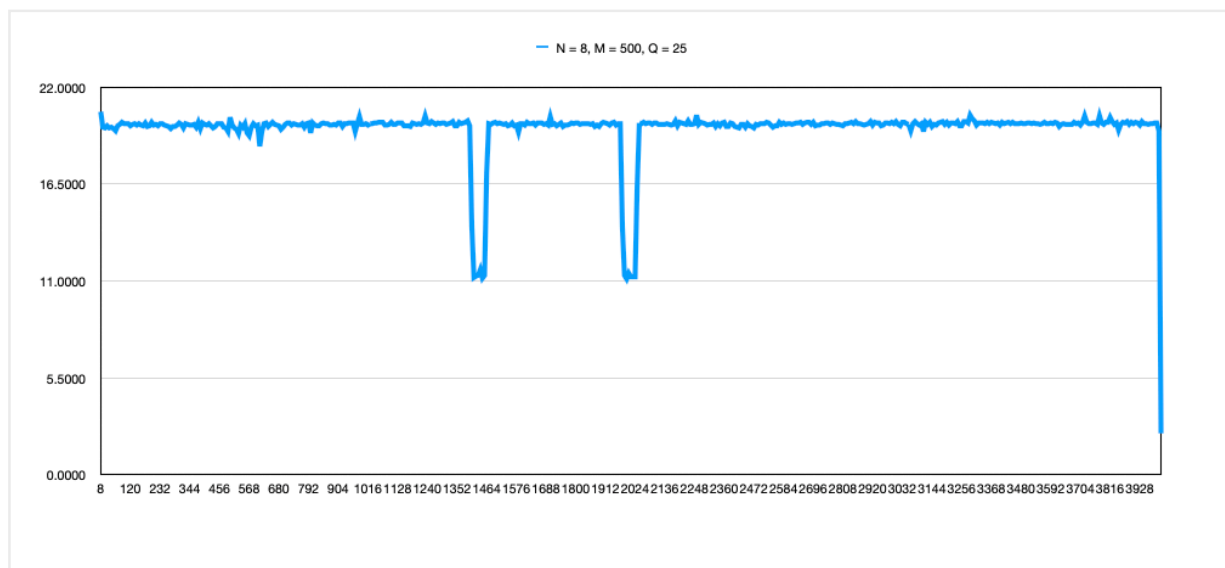
So as the array size gets large enough we can send them as full packets which is more efficient

$N=1, M=500, Q=10$



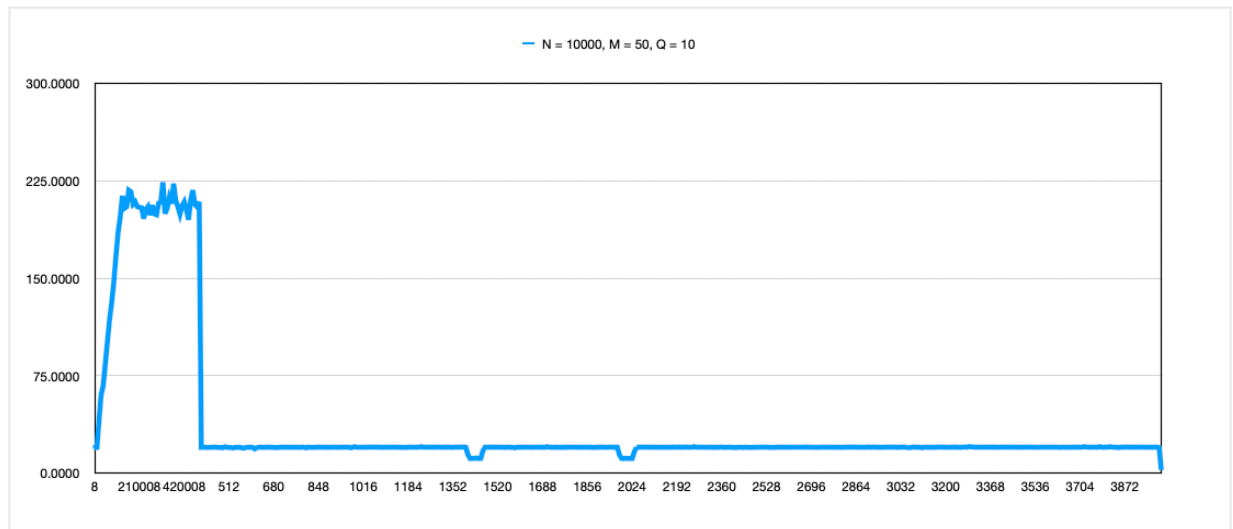
Here we will see if making  $N = 1$  (small sizes) improves the speed  
It is actually pretty much the same as  $N = 8$

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



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