

# CS 655 Final Project Report

Hanyu Chen <[chenhy1@bu.edu](mailto:chenhy1@bu.edu)>  
Yichen Mu <[ycmu@bu.edu](mailto:ycmu@bu.edu)>  
Zheng Jiang <[zhejiang@bu.edu](mailto:zhejiang@bu.edu)>  
Zixiong Cheng <[chouchou@bu.edu](mailto:chouchou@bu.edu)>

GitHub link: <https://github.com/momoxiaocaiji/CS655-final>

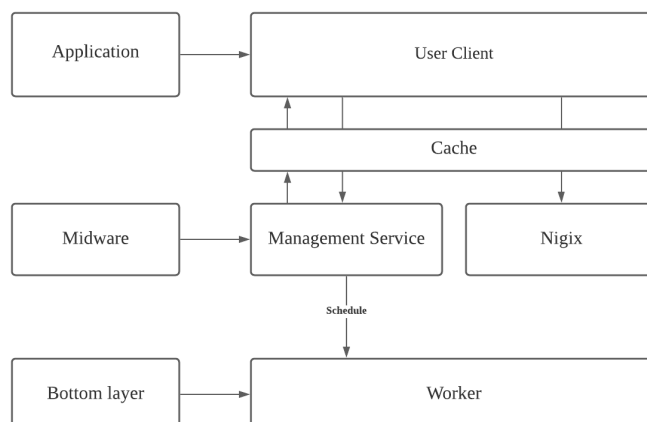
Project on GENI and any public link:

[https://portal.geni.net/secure/slice.php?project\\_id=4457203f-0732-4def-9197-3b35125798cd&slice\\_id=aece176f-36df-401b-8a55-efe2c9a1167d&member\\_id=](https://portal.geni.net/secure/slice.php?project_id=4457203f-0732-4def-9197-3b35125798cd&slice_id=aece176f-36df-401b-8a55-efe2c9a1167d&member_id=)  
GENI Slice name: [final-ZXC](#)

## 1. Introduction

Our system is a md5 password cracker, where a user submits the md5 hash of a 5-character password (a-z, A-Z) to the system using a web interface. The web interface with the help of our 10 worker nodes cracks the password by a brute force approach. The system is also scalable, that is, users can specify the number of workers dedicated to cracking when assigning a task.

## 2. Experimental Methodology



Logically, there are mainly three components in our product: frontend, management service and workers. Due to evaluation focus and limited scale node cluster, we assume that our system can dedicate to handling one request at a time. However, users can specify how many nodes they want to use to crack the password.

Frontend: Designs and implements frontend for users to upload their encoded password. In order to deploy the website onto the server, we use Nginx as our web server. After rewriting the nginx configure file, if there is a request to visit the website, nginx would send the website resources back to the users. The static resources are stored in the directory 'var/www/html'. What's more, we choose Vue as the frontend framework of this project, since it perfectly combines html, css and javascript into one system, and this property makes us much easier to develop a clear frontend interface.

Management service: Provides an interface for the front end to start deciphering the password, "<http://ip:port/start>". According to the parameters passed by the front end, it is decided to start the corresponding number of Workers to start working. Then for every 0.5 seconds, the manager asks each worker whether they completed the task. If one of the workers returned the decoded password, the manager will send the finish signal to all invoked workers, which means this work has been done and they should reset their statements for the next job.

Workers can only work passively, which means that they can't communicate with other nodes on their own. A worker node starts to crack with a separate thread once it receives a request from the manager. Meanwhile, the main thread also keeps replying with cracking status (found or not) every time when it receives a query from the management service. If they received the found status from the management, they will stop cracking right now and be ready for the next task.

As for physical resources, we reserved 10 worker nodes and 3 controller nodes on GENI with a public routable ip assigned to each VM. Each worker node runs as a cracking worker and controller nodes runs as management service and also holds frontend files.

## 3. Results

### 3.1 Usage Instructions

Geni nodes public ips:

Worker nodes:

{1: '171.67.92.157', 2: '171.67.92.183', 3: '171.67.92.155', 4: '171.67.92.146', 5: '171.67.92.182', 6: '171.67.92.162', 7: '171.67.92.186', 8: '171.67.92.185', 9: '171.67.92.163', 10: '171.67.92.184'}

Controller nodes:

UChicago: 192.170.230.108

UCLA: 164.67.126.46

NYU: 192.86.139.70

Two automated scripts were created: run\_manager.sh and run\_worker.sh. Both scripts will achieve downloading, unzipping, extracting useful code, removing extra parts and starting running components on a node.

To start a worker node, first upload run\_worker.sh onto the worker node using scp command:

```
(base) joey@dhcp-wifi-8021x-155-41-68-242 scripts % scp -i ~/.ssh/headnode.key run_worker.sh chouchou@171.67.92.157:.  
run_worker.sh 100% 315 3.4KB/s 00:00
```

Then using ssh to log into the worker node:


```
(base) joey@dhcp-wifi-8021x-168-122-203-222 ~ % ssh chouchou@171.67.92.157 -i ~/.ssh/headnode.key  
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-75-generic x86_64)  
  
* Documentation: https://help.ubuntu.com  
* Management: https://landscape.canonical.com  
* Support: https://ubuntu.com/advantage  
New release '18.04.6 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Wed Dec 8 00:42:37 2021 from 155.41.68.242  
chouchou@node-1:~$
```

Finally, run `bash run_worker.sh` on worker node:

```
chouchou@node-1:~$ bash run_worker.sh  
--2021-12-08 11:11:40-- https://github.com/momoxiaocaiji/CS655-final/archive/refs/heads/worker.zip  
Resolving github.com (github.com)... 140.82.114.4  
Connecting to github.com (github.com)[140.82.114.4]:443... connected.  
HTTP request sent, awaiting response... 302 Found  
Location: https://codeload.github.com/momoxiaocaiji/CS655-final/zip/refs/heads/worker [following]  
--2021-12-08 11:11:40-- https://codeload.github.com/momoxiaocaiji/CS655-final/zip/refs/heads/worker  
Resolving codeload.github.com (codeload.github.com)... 140.82.113.10  
Connecting to codeload.github.com (codeload.github.com)[140.82.113.10]:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: unspecified [application/zip]  
Saving to: 'worker.zip'  
  
worker.zip [ <=> ] 218.32K 911KB/s in 0.2s  
  
2021-12-08 11:11:41 (911 KB/s) - 'worker.zip' saved [223558]  
  
Worker is running at 9000
```

Do the same for the controller node, except using run\_manager.sh this time.

After starting worker node and controller node, enter the ip address of the controller node directly in the browser. This will show the frontend. Enter the encryption and the number of workers you want to use, click start cracking. The result will show in the area below once finished.



### Md5 Cracker

\* Encryption2ecdde3959051d913f6\*

\* Worker number4

☐ Use cache

Start cracking!!

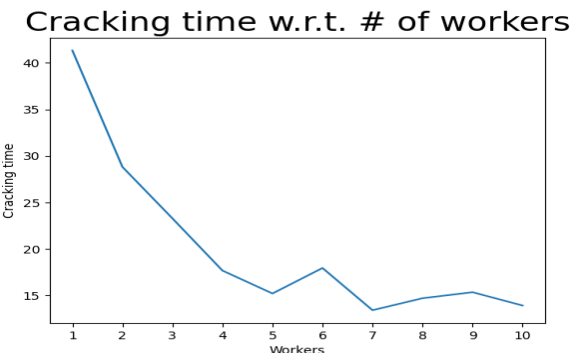
Decoding result of your encryption is: **ABCDE**

Cracking time is: **19793ms.**

### 3.2 Evaluations and Analysis

#### 3.2.1 Cracking time w.r.t. # of workers

In this experiment, we test a random md5 hash value and use worker number from 1 to 10 and compare different cracking time when using different numbers of workers.



Analysis:

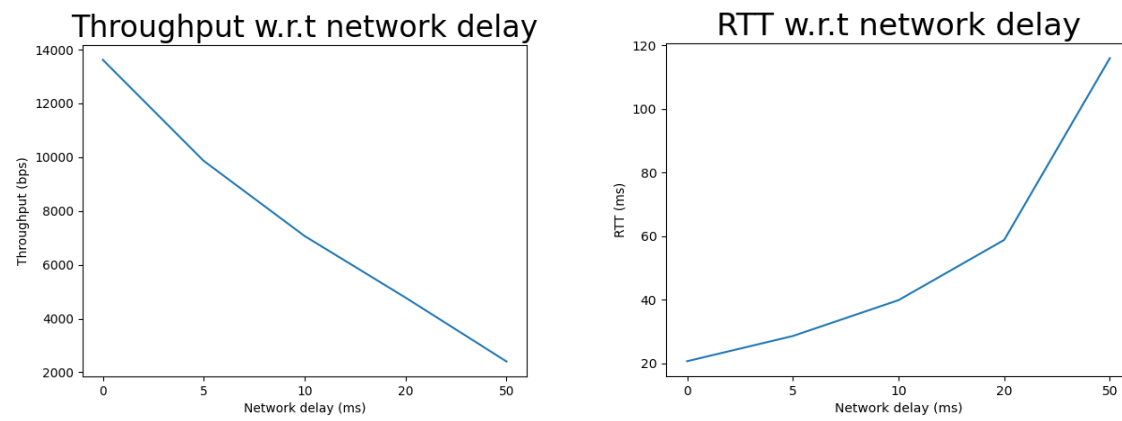
At the beginning, the number of workers has an obvious effect on the results. However, as the number increases, because the manager needs to poll the worker request results one by one, the downward trend gradually flattens out.

#### 3.2.2 RTT and Throughput w.r.t network delay

In this experiment, we simulate network delay on each worker node with tc command:

```
tc qdisc add dev eth0 root netem delay 10ms
```

And we fix the worker number to 4 and crack the same passcode. Meanwhile using network delay 0, 5ms, 10ms, 20 ms and 50 ms. Then calculate the RTT and throughput of the communication between the management server and cracking workers.



Analysis:

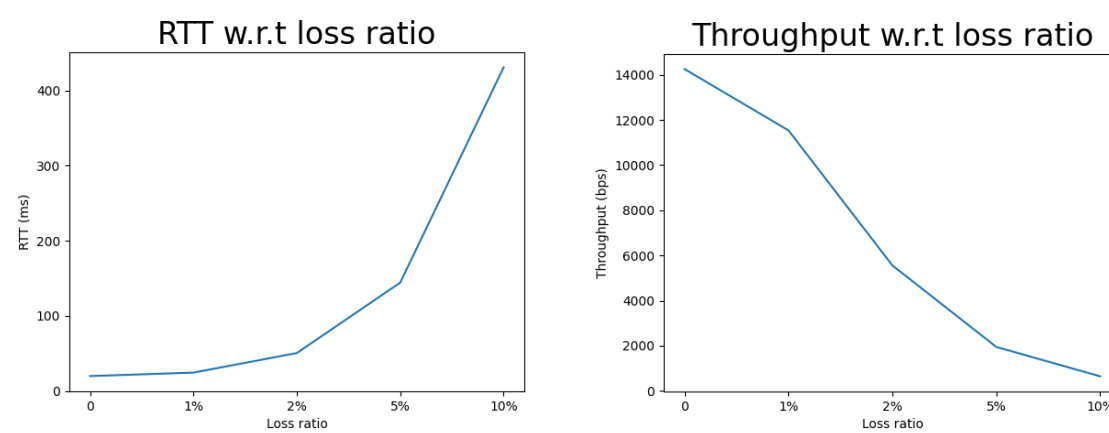
As the delay increases, the rtt becomes larger, which leads to a smaller throughput.

### 3.2.3 RTT and throughput w.r.t loss ratio

In this experiment, we simulate network delay on each worker node with tc command:

`tc qdisc add dev eth0 root netem loss 1%`

And we fix the worker number to 4 and crack the same passcode. Meanwhile test loss ratio of 0, 1%, 2%, 5% and 10%. Then calculate the RTT and throughput of the communication between the management server and cracking workers.



Analyze:

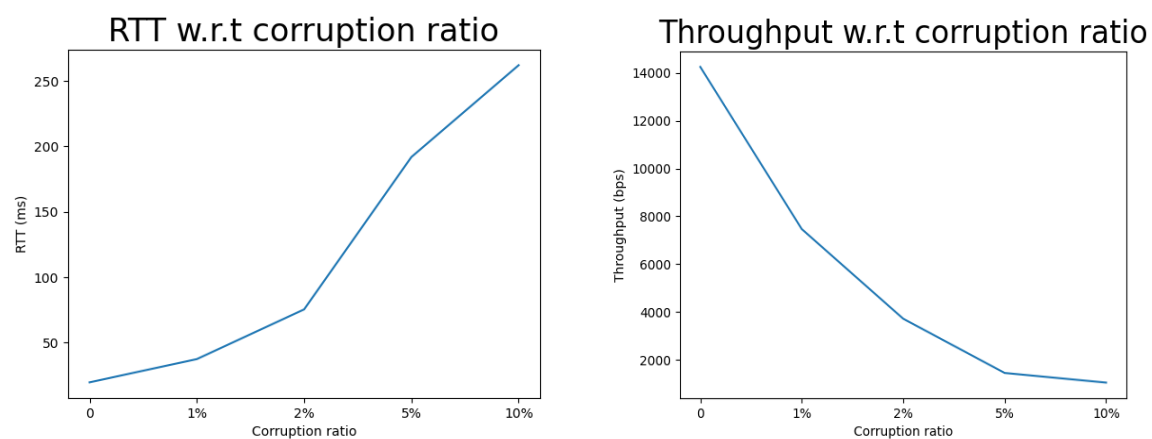
Socket uses TCP for transmitting packets. If loss is detected, TCP will take responsibility to retransmit the packet, therefore increasing RTT. As shown in our result, RTT increases when loss ratio increases, meanwhile throughput decreases.

### 3.2.4 RTT w.r.t corruption ratio

In this experiment, we simulate network delay on each worker node with tc command:

`tc qdisc add dev eth0 root netem corruption 1%`

And we fix the worker number to 4 and crack the same passcode. Meanwhile test loss ratio of 0, 1%, 2%, 5% and 10%. Then calculate the RTT and throughput of the communication between the management server and cracking workers.

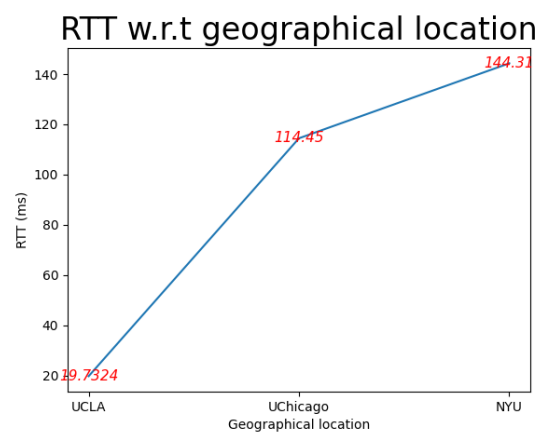


Analyze:

Same as above, TCP will retransmit corrupted packets. Therefore RTT increases and throughput decreases.

### 3.2.5 RTT w.r.t geographical location

In this experiment, we created 3 controller nodes at UCLA, UChicago and NYU respectively. Then we started the cracking tasks using worker nodes located at Stanford University and tested the average RTT for each location. The result is as below:



Analyze: Our worker nodes are reserved at Stanford stack, which is located in California on the West Coast. Geographically, it's closer to UCLA which is also located in California. As an opposite, Chicago University is in the Middle and NYU is on the East Coast, which is much farther than UCLA. So the RTT increases because of longer propagation delay.

## 4. Conclusion

Our expectation is to verify the effect of corruption ratio and loss ratio to TCP, also to prove that geographical distance will affect RTT markedly. As a test of our product, we want to show that increasing the number of workers does enhance cracking efficiency. The result shown above meets our expectations.

There are some possible extensions of our project if given some extra time. One extension we can think of is adding fault tolerance to our system, for example, by implementing map-reduce protocol. In this way, the manager system can detect dead nodes and reassign the task to other alive nodes.

## 5. Division of Labor

Hanyu Chen: Finish the frontend part.

Yichen Mu: Finish the worker part.

Zheng Jiang: Finish the management service.

Zixiong Cheng: Reserve and manage GENI nodes; create automated scripts; design and execute experiments