南京大学本科生实验报告

课程名称：**计算机网络**　　任课教师：田臣/李文中　　助教：

学院	**计算机科学与技术系**	专业（方向）	**计算机科学与技术**
学号	**191220154**	姓名	**张涵之**
Email	**1683762615@qq.com**	开始/完成日期	**2021/6/5-2021/6/5**

1. 实验名称：Lab 7: Content Delivery Network

2. 实验目的：

Build a demo of Content Delivery Network (CDN).
Provide high availability and performance by distributing service spatially relative.

3. 实验内容

a) Task 2: DNS server
Implement the functions of Remote DNS Server and CDN DNS Server.
b) Task 3: Caching server
Fill in the blanks to complete a functional caching server.

4. 实验结果

a) Task 2: DNS server

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ python3 test_entry.py dns
2021/06/05-20:39:02| [INFO] DNS server started
test_cname1 (testcases.test_dns.TestDNS) ... ok
test_cname2 (testcases.test_dns.TestDNS) ... ok
test_location1 (testcases.test_dns.TestDNS) ... ok
test_location2 (testcases.test_dns.TestDNS) ... ok
test_non_exist (testcases.test_dns.TestDNS) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.036s

OK
2021/06/05-20:39:03| [INFO] DNS server terminated
```

b) Task 3: Caching server
Testing the code manually:

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ python3 mainServer/mainServer.py -d mainServer/
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
127.0.0.1 - - [05/Jun/2021 20:41:29] "GET /doc/success.jpg HTTP/1.1" 200 -
127.0.0.1 - - [05/Jun/2021 20:41:45] code 404, message File not found
127.0.0.1 - - [05/Jun/2021 20:41:45] "GET /nonexist HTTP/1.1" 404 -
127.0.0.1 - - [05/Jun/2021 20:41:57] code 404, message File not found
127.0.0.1 - - [05/Jun/2021 20:41:57] "GET /doc/test.pdf HTTP/1.1" 404 -
```

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ python3 runCachingServer.py localhost:8000
Caching server serving on http://0.0.0.0:1222
2021/06/05-20:41:29| [Info] Fetched '/doc/success.jpg' from main server 'localhost:8000'
2021/06/05-20:41:29| [From 127.0.0.1:51514] "GET /doc/success.jpg HTTP/1.1" 200 -
2021/06/05-20:41:45| [Error] File not found on main server 'localhost:8000'
2021/06/05-20:41:45| [From 127.0.0.1:51518] code 404, message File not found
2021/06/05-20:41:45| [From 127.0.0.1:51518] "GET /nonexist HTTP/1.1" 404 -
2021/06/05-20:41:57| [Error] File not found on main server 'localhost:8000'
2021/06/05-20:41:57| [From 127.0.0.1:51522] code 404, message File not found
2021/06/05-20:41:57| [From 127.0.0.1:51522] "HEAD /doc/test.pdf HTTP/1.1" 404 -
```

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ curl -O http://localhost:1222/doc/success.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2125  100  2125    0     0   296k      0 --:--:-- --:--:-- --:--:--  296k
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ curl http://localhost:1222/nonexist
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
        <title>Error response</title>
    </head>
    <body>
        <h1>Error response</h1>
        <p>Error code: 404</p>
        <p>Message: File not found.</p>
        <p>Error code explanation: HTTPStatus.NOT_FOUND - Nothing matches the given URI.</p>
    </body>
</html>
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ curl -I http://localhost:1222/doc/test.pdf
HTTP/1.0 404 File not found
Server: CachingServerHTTP/0.1
Date: Sat, 05 Jun 2021 12:41:57 GMT
Connection: close
Content-Type: text/html;charset=utf-8
Content-Length: 469
```

The picture has been downloaded successfully.
See folder mainServer/doc, the file test.pdf does not exist.

Running the provided testcase:

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ python3 test_entry.py cache
2021/06/05-20:50:41| [INFO] Main server started
2021/06/05-20:50:41| [INFO] RPC server started
2021/06/05-20:50:41| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_cache.TestCache) ...
[Request time] 66.58 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...
[Request time] 61.49 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ...
[Request time] 57.14 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 9.83 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 12.15 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 56.70 ms
ok

----------------------------------------------------------------------
Ran 6 tests in 4.862s

OK
2021/06/05-20:50:47| [INFO] Caching server terminated
2021/06/05-20:50:47| [INFO] PRC server terminated
2021/06/05-20:50:47| [INFO] Main server terminated
```

c) Running test all to test DNS server and caching server at the same time:

```
njucs@njucs-VirtualBox:~/switchyard/lab-7-RainTreeCrow$ python3 test_entry.py all
2021/06/05-20:54:00| [INFO] DNS server started
2021/06/05-20:54:00| [INFO] Main server started
2021/06/05-20:54:00| [INFO] RPC server started
2021/06/05-20:54:00| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 58.41 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 57.19 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 48.73 ms
ok

----------------------------------------------------------------------
Ran 3 tests in 2.286s

OK
2021/06/05-20:54:04| [INFO] DNS server terminated
2021/06/05-20:54:04| [INFO] Caching server terminated
2021/06/05-20:54:04| [INFO] PRC server terminated
2021/06/05-20:54:04| [INFO] Main server terminated
```

Submitting the code on OpenNetLab:
DNS log file: received three DNS requests

```
2021/06/05-09:31:29| [INFO] DNS server started
DNS server serving on 0.0.0.0:8175
2021/06/05-09:31:32| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
2021/06/05-09:31:33| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
2021/06/05-09:31:34| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
```

Cache server log file: When first request to get success.jpg arrives, it is not yet found in the cache, so the sever fetches it from main server. When it arrives at a second time, however, it is directly accessed from the cache. A request for a nonexistent domain arrives, and the server replies with a 404 not found error.

```
2021/06/05-09:31:30| [INFO] Caching server started
Caching server serving on http://0.0.0.0:8175
2021/06/05-09:31:32| [Info] Fetched '/doc/success.jpg' from main server '20.188.122.123:8888'
2021/06/05-09:31:32| [From 10.0.0.24:49036] "GET /doc/success.jpg HTTP/1.1" 200 -
2021/06/05-09:31:33| [From 10.0.0.24:49038] "GET /doc/success.jpg HTTP/1.1" 200 -
2021/06/05-09:31:34| [Error] File not found on main server '20.188.122.123:8888'
2021/06/05-09:31:34| [From 10.0.0.24:49040] code 404, message File not found
2021/06/05-09:31:34| [From 10.0.0.24:49040] "GET /noneexist HTTP/1.1" 404 -
```

Client log file: it can be seen that the request time for missed cache (where the server fetches file from main server) is about the same as the request time for file not found error (as both of them goes through the entire cache and tries to fetch from main server, only one succeeded and the other failed). The request time for cache hit is however much shorter, for it directly gets the file from the cache and does not need to interact with the main server.

```
test_01_cache_missed_1 (testcases.test_all.TestAll) ... ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ... ok
test_03_not_found (testcases.test_all.TestAll) ... ok

----------------------------------------------------------------------
Ran 3 tests in 3.148s

OK

[Request time] 707.19 ms

[Request time] 2.65 ms

[Request time] 703.68 ms
```

Through the test implemented in the real network, we feel certain to arrive at the conclusion that the goal to provide high availability and performance can be achieved by distributing the service spatially relative to end users.

5. 核心代码

a) Task 2: DNS server

Step 1 : Load DNS Records Table

```
class DnsEntry():
    def __init__(self, name, _type, values):
        self.domainName = name
        self.recordType = _type
        self.recordValues = values
```

```
    with open(dns_file) as DnsFile:
        for line in DnsFile:
            entry = line.split()
            if entry:
                self._dns_table.append(
                    DnsEntry(entry[0], entry[1], entry[2:])
                )
```

The DNS entries are read from the DNS text file. Each line is split with default argument (blank space), the first two are the domain name and record type, the rest are treated as a list containing record values.

Step 2 : Reply Clients' DNS Request

```
client_ip, _ = self.client_address
request_split = request_domain_name.split('.')
if request_split[-1] == '':
    request_split.pop()
for entry in self.table:
    entry_split = entry.domainName.split('.')
    if entry_split[-1] == '':
        entry_split.pop()
```

```
    if entry.domainName[0] == '*' and entry_split[1:] == request_split[1:]\
        or entry_split[0:] == request_split[0:]:
        response_type = str(entry.recordType)
        if entry.recordType == "CNAME":
            response_val = str(entry.recordValues[0])
        elif entry.recordType == "A":
            if (len(entry.recordValues) == 1):
                response_val = str(entry.recordValues[0])
            else:
                client_la, client_long = IP_Utils.getIpLocation(client_ip)
                if (client_la, client_long) == (None, None):
                    index = random.randint(0, len(entry.recordValues) - 1)
                    response_val = str(entry.recordValues[index])
                else:
                    selected_ip = None
                    min_dist = float('inf')
                    for ip in entry.recordValues:
                        temp_la, temp_long = IP_Utils.getIpLocation(ip)
                        temp_dist = math.sqrt((temp_la - client_la) ** 2 +\
                            (temp_long - client_long) ** 2)
                        if (temp_dist < min_dist):
                            min_dist = temp_dist
                            selected_ip = ip
                    response_val = str(selected_ip)
# ------------------------------------------------
return (response_type, response_val)
```

The sever goes through the cache table looking for a match. for every entry in the table, the name is split with '.', with the empty item '' deleted from the list if the original name ends with '.'. Then if the entry domain name begins with

a '*', that is to say, it is a wildcard mask, and any sub-domain under the name should be matched to it. Thus, only those items with index above 1 in the lists are compared. Otherwise, it is a common name, and all items in the lists should be compared. Once the match is found, the response type is set accordingly. If it is a "CNAME" type, there must be only one item in the record values list, so we just take it out and pass it as the response value. If it is a "A" type, however, we must check the length of the values list first. If the length is 1, pass the one and only value; otherwise, we get the IP location of the client. If the location is unknown, we have no choice but to simply select a random value from the list. With a large enough quantity of entries and requests, the load balance for multiple servers should be relatively equal. If the location is sure, we calculate the client's distance to each cache node, choosing the one closest to it.

b) Task 3: Caching server

Step1: HTTPRequestHandler

Complete CachingServerHttpHandler.sendHeaders():

Simply calling the send_header and ender_headers functions

```
for header in headers:
    self.send_header(header[0], header[1])
self.end_headers()
```

Complete CachingServerHttpHandler.do_GET():

If the header is None (the touchItem function returns none, that is to say, the path is not found in the cache or fetched from the main server), the server shall raise a 404 not found error. Otherwise, it sends response with the code 200 to indicate that everything is ok, then it sends the headers and body in order.

```
headers, body = self.server.touchItem(self.path)
if headers:
    self.send_response(200)
    self.sendHeaders(headers)
    self.sendBody(body)
else:
    self.send_error(HTTPStatus.NOT_FOUND, "File not found")
```

Complete CachingServerHttpHandler.do_HEAD():

Almost the same as do_HEAD, just don't send the body in this case.

Step2: Caching Server

Complete touchItem(): If path not in the cache table or expired (timeout), fetch it from the main server, if no response from main server, no worries, just return None. Otherwise, get headers and body from the cache table.

```
headers, body = None, None
if path not in self.cacheTable or self.cacheTable.expired(path):
    response = self.requestMainServer(path)
    if response:
        headers = response.getheaders()
        body = response.read()
        self.cacheTable.setHeaders(path, headers)
        self.cacheTable.appendBody(path, body)
else:
    headers = self.cacheTable.getHeaders(path)
    body = self.cacheTable.getBody(path)
return headers, body
```

6. 总结与感想

a) This lab is surprisingly (unexpectedly?) NOT as difficult as I thought it would be (after all, it is completely different and separate from the first six, and it is tested and implemented in the "real" network world, where more unexpected things might occur, but thanks to the detailed and precise lab manual, the links to python methods and fields, I did not come across a lot of confusion.

b) One point I did got stuck is to handle wildcard masts that begins with '*', at first, I thought I might just split the names with '.', check if the first item in the list is '*', and compare the other parts in order. This idea can pass the separate testcases for DNS and cache, but cannot pass "all" where two combined.

I tired printing the lists out:

```
testFile = open("test.txt", "w")
for item in entry_split:
    testFile.write(item + " ")
testFile.write("\n")
for item in request_split:
    testFile.write(item + " ")
testFile.write("\n")
testFile.close()
```

And found a redundant blank space at the end of the line that only appears with several certain lines, then it occurred to me that I might had misunderstood the functions of the method split, for example, the manual suggests that the . at the end of "*.netlab.nju.edu.cn." represents the root domain name, and we usually omit it. For this name, if I split it with '.', the list will contain an item '' at the end of it, which cannot be seen by eyes in the printed test txt file. However, as I compare entry_split[1:] with request_split[1:], the redundant '' will cause the failure to match correctly. Thus, I checked the last item of the list after splitting and pop it if it is '', then I can pass all the testcases successfully.

Another funny mistake occurred in this stage, where I submitted the code that passed all testcases locally. I forgot to delete the part where I write log info to a test file as shown above. When my local program finds the "test.txt" file does not exist, it simply creates a new file with the name. I suppose the real network does not support this, so the first submission failed. I apologize for submitting this kind of wrong code to OpenNetLab, I did not think about it before.

c) Generally speaking, this lab is very fun and educational, with the real network I feel I understood the logic and performance of the program better than simply imitating a network in switchyard. As this is the last lab of the semester, I want to thank you for all the hard work and wish you a happy summer vacation. 😉