南京大学

LAB 7：Content Delivery Network

课程名称：计算机网络

姓名：孙文博

学号：201830210

学院：计算机科学与技术系

Email：201830210@smail.nju.edu.cn

任课教师：李文中

实验时间：2022.5.19 - 2022.6.2

一、 实验名称

Content Delivery Network

二、 实验目的

- ✧ 构建一个内容交付网络
- ✧ 巩固对应用层知识的理解
- ✧ 提高使用 pdb 等方法 debug 的能力

三、 实验内容

Task 1: Preparation

配置实验环境

Task 2: DNS Server

模拟 DNS 服务器，实现 Remote DNS Server 和 CDN DNS Server 的功能

Task 3: Cacheing Server

模拟 CDN 缓存服务器，实现处理 HTTP 请求和向主机发送请求等功能

Task 4: Deployment

测试 DNS Server 和 Caching Server 实际运行情况

四、 实验过程

Task 1: Preparation

搭建好实验环境。

Task 2: DNS Server

DNS 服务器是 CDN 的入口点，负责负载均衡和智能 DNS 解析。当用户

访问远程资源时，通常是通过域名而不是直接 IP 地址，用户在建立 TCP 连接之前访问 DNS 服务将域名转换为 IP 地址。为了提供更好的服务质量，DNS 服务器还需要根据客户端的相对位置回复最优目标服务器或 CDN Cache 节点的 IP。

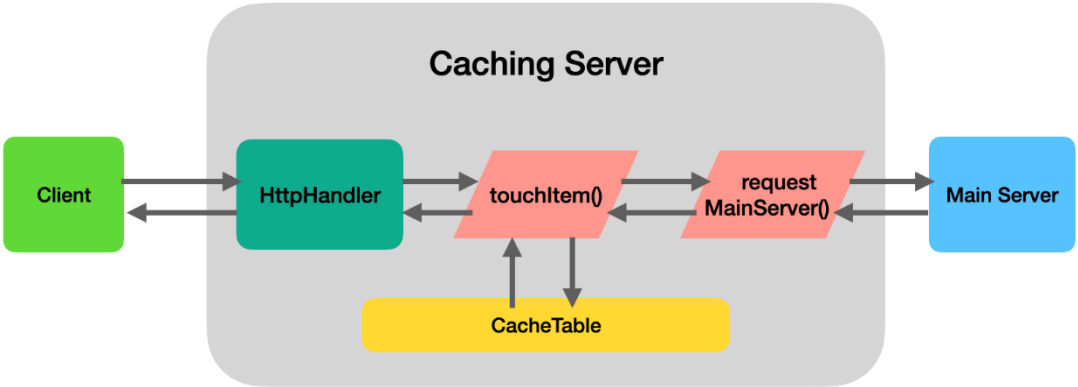
我们需要实现的主要功能有两个。第一个是从已有的 dns_table.txt 文件中读取现有 DNS 缓存表并将其转化为对应的数据结构；第二个功能则是根据要求的选择逻辑给用户返回请求的域名对应的 IP 地址/上层域名，注意当有多个可能的 IP 地址时，还需要根据用户与该 IP 的实际位置关系决定返回最近的那个。

实现方法见代码解析部分。

Task 3: Caching Server

缓存服务器是 CDN 的核心，缓存服务器的工作原理是有选择地将网站文件存储在 CDN 的缓存代理服务器上，这样从附近位置浏览的网站访问者可以快速访问这些文件。它维护一个本地缓存表（例如数据库）来存储所有缓存的内容。

在这个任务中，我们更重要的任务是读懂框架代码，这样才能更准确的实现未完成的功能。我们需要关注的文件是 CachingServer.py，有两个类：CachingServer 和 CachingServerHttpHandler，简单来说，前者包括了与客户端以及主服务器进行交互的功能，而后者则负责处理 HTTP 请求，可以用一张图说明他们的功能：



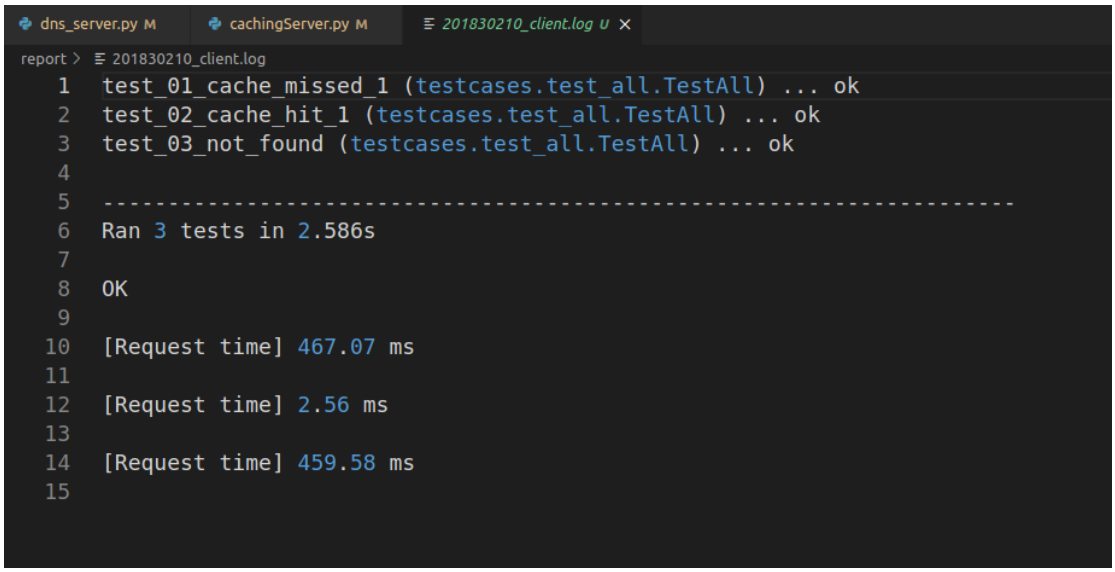
我们主要实现 touchItem 和 CachingServerHttpHandler 中的 do_get 等方法，具体实现见代码解析部分。

Task 4: Deployment

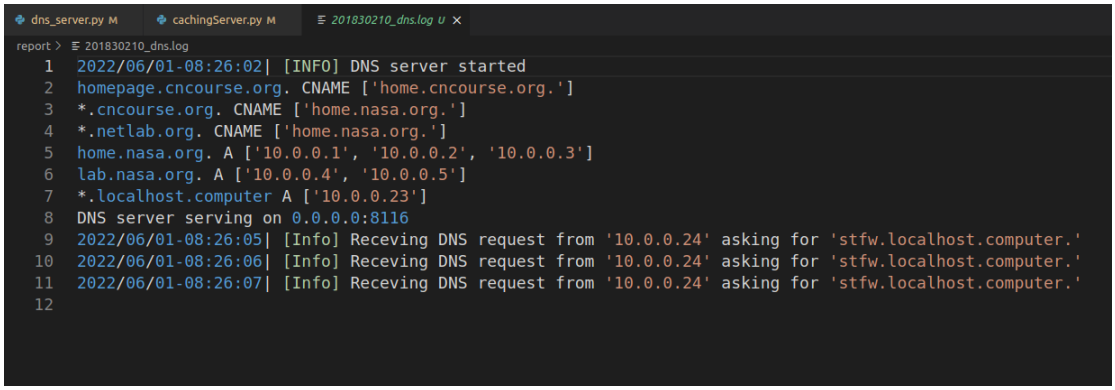
通过 testcase 测试 DNS Server 和 Caching Server 实际运行情况如下：

```
njucs@njucs-VirtualBox:~/switchyard/lab-07-Florentino-73$ python3 test_entry.py dns
2022/06/02-12:01:00| [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 0.016s

OK
2022/06/02-12:01:01| [INFO] DNS server terminated
njucs@njucs-VirtualBox:~/switchyard/lab-07-Florentino-73$ python3 test_entry.py cache
2022/06/02-12:11:47| [INFO] Main server started
2022/06/02-12:11:47| [INFO] RPC server started
2022/06/02-12:11:47| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_cache.TestCache) ...
[Request time] 20.25 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...
[Request time] 59.87 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ...
[Request time] 57.46 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 35.54 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 20.43 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 25.08 ms
ok

----------------------------------------------------------------------
Ran 6 tests in 4.448s

OK
2022/06/02-12:11:52| [INFO] Caching server terminated
2022/06/02-12:11:52| [INFO] PRC server terminated
2022/06/02-12:11:52| [INFO] Main server terminated
```

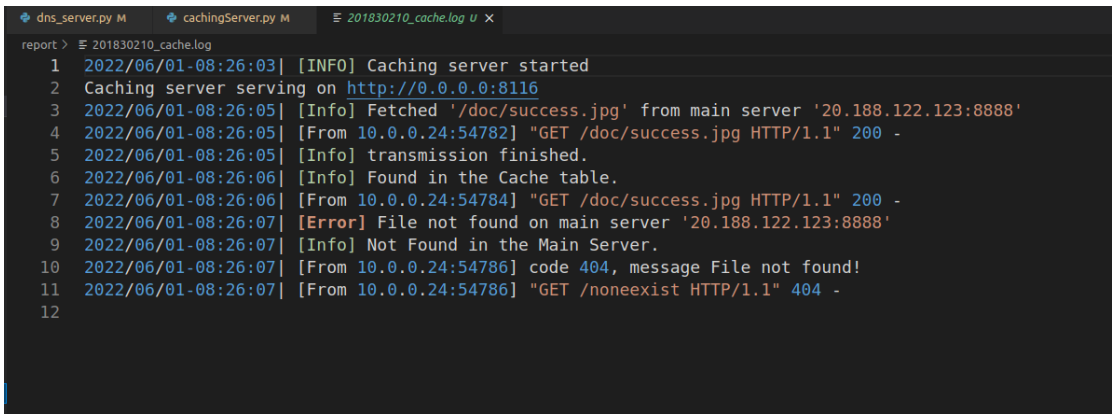
通过 test all 并上传到 OpenNetLab 上进行检测得到日志：



```
dns_server.py M   cachingServer.py M   201830210_client.log U
report > 201830210_client.log
1  test_01_cache_missed_1 (testcases.test_all.TestAll) ... ok
2  test_02_cache_hit_1 (testcases.test_all.TestAll) ... ok
3  test_03_not_found (testcases.test_all.TestAll) ... ok
4
5  ----------------------------------------------------------------------
6  Ran 3 tests in 2.586s
7
8  OK
9
10 [Request time] 467.07 ms
11
12 [Request time] 2.56 ms
13
14 [Request time] 459.58 ms
15
```



```
dns_server.py M   cachingServer.py M   201830210_dns.log U
report > 201830210_dns.log
1  2022/06/01-08:26:02| [INFO] DNS server started
2  homepage.cncourse.org. CNAME ['home.cncourse.org.']
3  *.cncourse.org. CNAME ['home.nasa.org.']
4  *.netlab.org. CNAME ['home.nasa.org.']
5  home.nasa.org. A ['10.0.0.1', '10.0.0.2', '10.0.0.3']
6  lab.nasa.org. A ['10.0.0.4', '10.0.0.5']
7  *.localhost.computer A ['10.0.0.23']
8  DNS server serving on 0.0.0.0:8116
9  2022/06/01-08:26:05| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
10 2022/06/01-08:26:06| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
11 2022/06/01-08:26:07| [Info] Receving DNS request from '10.0.0.24' asking for 'stfw.localhost.computer.'
12
```



```
dns_server.py M   cachingServer.py M   201830210_cache.log U
report > 201830210_cache.log
1  2022/06/01-08:26:03| [INFO] Caching server started
2  Caching server serving on http://0.0.0.0:8116
3  2022/06/01-08:26:05| [Info] Fetched '/doc/success.jpg' from main server '20.188.122.123:8888'
4  2022/06/01-08:26:05| [From 10.0.0.24:54782] "GET /doc/success.jpg HTTP/1.1" 200 -
5  2022/06/01-08:26:05| [Info] transmission finished.
6  2022/06/01-08:26:06| [Info] Found in the Cache table.
7  2022/06/01-08:26:06| [From 10.0.0.24:54784] "GET /doc/success.jpg HTTP/1.1" 200 -
8  2022/06/01-08:26:07| [Error] File not found on main server '20.188.122.123:8888'
9  2022/06/01-08:26:07| [Info] Not Found in the Main Server.
10 2022/06/01-08:26:07| [From 10.0.0.24:54786] code 404, message File not found!
11 2022/06/01-08:26:07| [From 10.0.0.24:54786] "GET /noneexist HTTP/1.1" 404 -
12
```

可以看到缓存服务器先从主服务器中下载文件，之后客户从缓存服务器中顺利的获得了一张已缓存的照片 success.jpg，此时 Caching Server 返回命中 Cache 等信息。而请求的文件不在缓存服务器和主服务器中时，系统报 404 错误信息。

五、 核心代码

DNS Server :

```python
# TODO:record the DNS
# each entry has three parts: domain_name, record_type, record_value
class DNS_record:
    def __init__(self, domain_name, record_type, record_value):
        self.name = domain_name
        self.type = record_type
        self.value = record_value

class DNSServer(UDPServer):
    def __init__(self, server_address, dns_file, RequestHandlerClass, bind_and_activate=True):
        super().__init__(server_address, RequestHandlerClass, bind_and_activate=True)
        self._dns_table = []
        self.parse_dns_file(dns_file)

    def parse_dns_file(self, dns_file):
        # -------------------------------------------------
        # TODO: your codes here. Parse the dns_table.txt file
        # and load the data into self._dns_table.
        # -------------------------------------------------
        file = open(dns_file)
        line = file.readline()
        while line:
            line = line.strip('\n') # delete '\n'
            entry = line.split(" ")
            self._dns_table.append(DNS_record(entry[0], entry[1], entry[2:]))
            line = file.readline()

        # print dns table
        # i.e. homepage.cncourse.org. CNAME [home.cncourse.org.]
        for item in self._dns_table:
            print(item.name, item.type, item.value)

```

第一步是加载现有的 DNS 记录表，并将其转换为对应的数据结构，方便之后使用，表项的结构为域名，类型（A 或者 CNAME），返回值（可能不止一个）。

```python
    def get_response(self, request_domain_name):
        response_type, response_val = (None, None)
        # -------------------------------------------------
        # TODO: your codes here.
        # Determine an IP to response according to the client's IP address.
        # Set "response_ip" to "the best IP address".
        client_ip, _ = self.client_address
        is_found = False
        for entry in self.table:
            # handle the root domain name '.'
            temp = ''
            if entry.name[len(entry.name)-1] == '.':
                temp = entry.name[:len(entry.name)-2]
            else:
                temp = entry.name + '.'

            if fnmatch(request_domain_name, entry.name) or fnmatch(request_domain_name, temp):
                is_found = True
                # print(request_domain_name, entry.name)
                if entry.type == "CNAME":
                    response_type = "CNAME"
                    response_val = entry.value[0]
                elif entry.type == "A":
                    response_type = "A"
                    response_val = self.find_best_ip_address(client_ip, entry)
                break

        if is_found == False:
            return(None, None)

        # -------------------------------------------------
        return (response_type, response_val)

```

```python
    def calc_distance(self, pointA, pointB):
        ''' TODO: calculate distance between two points '''
        return math.sqrt(pow(pointA[0] - pointB[0], 2) + pow(pointA[1] - pointB[1], 2))

    # TODO: find the nearest IP address
    def find_best_ip_address(self, client_ip, entry):
        best_ip_address = ''
        min_dist = float('inf') # initial dist as inifity
        client_location = IP_Utils.getIpLocation(client_ip)
        # print(client_location)
        if client_location == None:
            random_IP_index = randint(0,len(entry.value)-1)
            return entry.value[random_IP_index]
        else:
            for item in entry.value:
                server_location = IP_Utils.getIpLocation(item)
                if server_location is not None:
                    distance = self.calc_distance(client_location, server_location)
                    if(distance < min_dist):
                        # choose the nearest Cache Node (CDN Node)
                        min_dist = distance
                        best_ip_address = item

        return best_ip_address
```

第二步是对用户发送的域名请求提供解析服务，具体来说分 CNAME 和 A 两种类型进行讨论，对于 A 型记录，我们封装一个 find_best_ip_address 方法计算距离用户最近的那个 IP 地址并返回。值得注意的是，第 122 行使用 python 自带的 fnmatch 方法进行通配符处理，以应对表中的*.xxx.xxx.xxx 类型的域名。

Caching Server:

```python
    @trace
    def do_GET(self):
        ''' Logic when receive a HTTP GET.
        Notice that the URL is automatically parsed and the path is stored in
        self.path.
        '''
        # TODO: implement the logic to response a GET.
        # Remember to leverage the methods in CachingServer.
        # update: use the Python Gernerator
        multi_res = self.server.touchItem(self.path)
        while True:
            try:
                response = next(multi_res)
                if response[0] is None: # header is not exist
                    self.send_error(HTTPStatus.NOT_FOUND, 'File not found!')
                    self.end_headers()
                    break
                else:
                    self.sendHeaders(response[0]) # header is exist
                self.sendBody(response[1])
            except StopIteration:
                break
```

```python
    @trace
    def do_HEAD(self):
        ''' Logic when receive a HTTP HEAD.
        The difference from self.do_GET() is that do_HEAD() only send HTTP
        headers.
        '''
        # TODO: implement the logic to response a HEAD.
        # Similar to do_GET()
        multi_res = self.server.touchItem(self.path)
        response = next(multi_res)
        if response[0] is None: # header is not exist
            self.send_error(HTTPStatus.NOT_FOUND, 'File not found!')
            self.end_headers()
        else:
            self.sendHeaders(response[0]) # header is exist


```

第一步我们要实现 CachingServerHttpHandler 中的 do_get(), do_head()等方法，这里需要根据手册内容去阅读相关 CPI，调用一些已有方法，如 send_header(), end_headers()等。

值得一提的是，这里我们要为缓存服务器设计一个 Stream Forwarding 的策略，即从主服务器中获取文件时，用户无需等待整个文件全部到达缓存服务器才能下载文件，而是缓存服务器可以一边下载一边传输给用户，通过一个 Buffer 实现（在 touchItem 方法中）。在这一部分，我们参考 Python 中的生成器 Gernerator，将发送方式写为 while 循环以便 touchItem 方法使用，参考 https://realpython.com/introduction-to-python-generators/。

```python
    def touchItem(self, path: str):
        ''' Touch the item of path.
        This method, called by HttpHandler, serves as a bridge of server and
        handler.
        If the target doesn't exsit or expires, fetch from main server.
        Write the headers to local cache and return the body.
        '''
        # TODO: implement the logic described in doc-string
        headers = None
        body = None
        buffer = bytearray(BUFFER_SIZE)
        # search the cache table
        if path in self.cacheTable: # find the file
            headers = self.cacheTable.getHeaders(path)
            self.log_info(f"Found in the Cache table.")
            body = self.cacheTable.getBody(path)
            yield (headers, body)
            raise StopIteration

```

```
        # expired or not found in cache table
        if (headers != None and self.cacheTable.expired(path)) or headers is None:
            response = self.requestMainServer(path)
            if response is None:
                # server is down or file not found
                self.log_info(f"Not Found in the Main Server.")
                yield (None, None)
                raise StopIteration
            else:
                # get the file from server and add it into the cache table
                # getheaders() and getheader() !!!!!!!! debug for an hour
                headers = self._filterHeaders(response.getheaders())
                self.cacheTable.setHeaders(path, headers)
                # print(headers)
                while True:
                    length = response.readinto(buffer)
                    self.cacheTable.appendBody(path, buffer)
                    if length > 0:
                        yield (headers, buffer[:length])
                    else:
                        self.log_info(f"Transmission finished.")
                        raise StopIteration
        else:
            self.log_error("something is wrong here in touchItem.")
            yield (None, None)
            raise StopIteration

```

第二步是实现 CacheServer 类中的 touchItem() 方法，其逻辑是首先判断用户请求文件是否在 Cache 中，若 HIT Cache 则直接返回文件；否则判断是否是因为主服务器不存在该文件或存在但是文件超时，之后从主服务器中下载文件到 buffer 数组中，对这部分进行不断的转发，其中 yield 函数和对应前面提到的流转发的策略，实现一个 Gernerator，yield 可以理解为 return 后继续进行，而 raise StopIteration 则用于终止。至此，一个完整的 Caching Server 成功实现！

六、 实验总结

随着屏幕上最后一个“ok”的出现，实验七走向了终结，也意味着一个学期的 Computer Network Lab 落下帷幕。回顾整个学期的七次实验，有配置环境时遇到各种各样的报错时的崩溃，有实现交换机或者路由器时无从下手的茫然，也有每次实验成功通过 testcase 时的欣喜，或是通过了助教验收时的快乐……

无论如何，这都是一段美妙而充满挑战的过程，也是（也许是）未来我们在计算机这个领域做各种工作，抑或是做各种科研的写照。那就带着这份期待去迎

接下一座高山吧，加油！💪

最后再次感谢一路以来相伴的老师和助教们，感谢参与编写实验手册的每一个人，你们都是这段旅途的引路人，respect！🫰

2022.6.2 日夜

于宿舍