实验报告

计算机网络-Lab7

专业：计算机科学与技术

任课教师：田臣

周迅 201220037

目录

一、 实验目的 2

二、 实验结果 2

三、 实验内容 2

1. Preparation 2

2. DNS server 2

3. Caching server 4

4. Testing 6

5. (Optional) Stream forwarding 7

6. Deployment 11

四、 核心代码 12

五、 总结与感想 12

一、 实验目的

- 建立一个 CDN 的简化演示模型；
- 具体地，完成 DNS 和 Caching 的简化基础功能；
- 使用 OpenNetLab 部署我们的设计，并验证其功能。

二、 实验结果

- 完成了 DNS server 和 Caching server 的基础功能；
- 通过了实验提供的几个测试样例；
- **完成了可选任务，实现了流转发(Stream forwarding)**；
- 使用 OpenNetLab 部署设计，通过分析 Log file 验证其功能。

三、 实验内容

1. Preparation

```
git clone!
python3 -m pip install -r requirements.txt!
```

2. DNS server

- *Load DNS Records Table*

  对于 DNS 记录和匹配，我们使用类来封装。表项信息的获取从文件获得。

```python
class DNSRecord:
    def __init__(self, **kwargs):
        self.domain_name: str = None
        self.type: str = None
        self.values: list[str] = None
        if 'line' in kwargs:
            line = kwargs['line'].split()
            self.domain_name = line[0]
            self.type = line[1]
            self.values = line[2:]
        elif 'domain_name' in kwargs and 'type' in kwargs :
            self.domain_name = kwargs['domain_name']
            self.type = kwargs['type']
```

```python
def parse_dns_file(self, dns_file) -> None:
    # ------------------------------------------------
    # TODO: your codes here. Parse the dns_table.txt file
    # and load the data into self._dns_table.
    # ------------------------------------------------
    with open(dns_file, 'r') as f:
        for line in f:
            line = line.strip()
            if line:
                self._dns_table.append(DNSRecord(line=line))
```

域名匹配这件事，我的实现非常粗糙，只有一个简陋的字符串查找（所以对于 jsb&*cjh.baidu.com.iyf#/nb 这种东西还真可以回复），我暂时也想不到什么高级而简洁的算法了……为了，字符串匹配，表项中的’ *’ 和末尾的’ .’ 都要去掉。

```python
def match(self, request_domain_name: str) -> Optional[List[str or List[str]]]:
    match_str = self.domain_name[:]
    if match_str.endswith('.'):
        match_str = match_str[:-1]
    if match_str.startswith('*.'):
        match_str = match_str[2:]
    if match_str in request_domain_name:
        return [self.type, self.values]
    else:
        return None
```

- *Reply Clients' DNS Request*

  手册上已经说得非常清楚了，我就当一下翻译员吧！

  没有匹配时，返回`(None, None)`；

```python
for record in self.table:
    if record.match(request_domain_name):
        response_type, optional_response_val = record.match(request_domain_name)
        break
else:
    return None, None # no match
```

  匹配项 `type` 为 `CNAME` 时，直接返回该项；

```
if response_type == 'CNAME':
    response_val = optional_response_val[0]
    assert(response_val is not None)
```

匹配项 `type` 为 `A` 且客户地理位置不知道，随便从备选的 `values` 选一个返回；

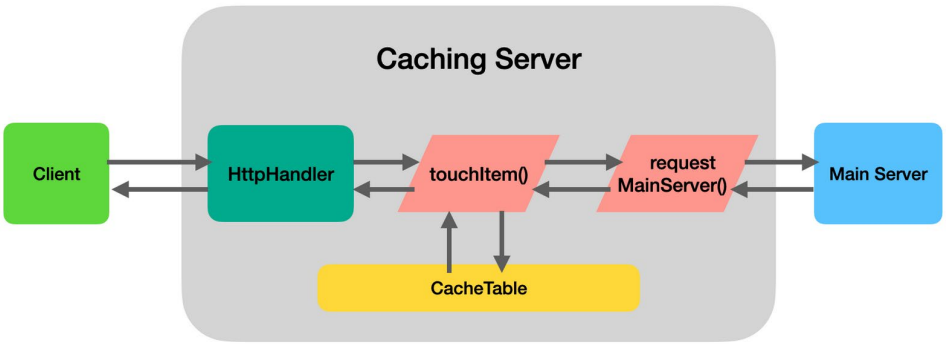
```
elif response_type == 'A':
    client_location = IP_Utils.getIpLocation(client_ip)
    # we don't know the location of the client, so we just select a random ip
    if client_location is None:
        response_val = random.choice(optional_response_val)
```

否则，选择距离客户最近的 CDN 服务器的 IP。

```
else:
    min_distance = math.inf
    for ip_str in optional_response_val:
        dns_location = IP_Utils.getIpLocation(ip_str)
        distance = self.calc_distance(client_location, dns_location)
        if distance < min_distance:
            min_distance = distance
            response_val = ip_str
assert(response_val is not None)
```

3. Caching server

这一阶段的布置如下图所示：



Caching server structure

- *HTTPRequestHandler*

  主要任务就是把通过 `touchItem()`获取的内容传给 `client`，下面讲一下具体实现。

  首先，尝试调用 `touchItem()`获取 `path` 下的对象，如果没拿到，`404` 送上。

```
headers_vals, body = self.server.touchItem(self.path)
if headers_vals is None:
    self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")
    return
```

否则，先后发送 headers 和 body（如果是 `do_HEAD()`，body 也省了）。

```
self.sendHeaders(headers_vals)
self.sendBody(body)
```

对于 headers，我们要考虑三部分：

- `Status code`;
- `touchItem()`传过来的首部;
- 其他一些首部。

`code` 直接用 `HTTPStatus.OK`；`touchItem()`给的依次 send 即可（每个 header 是个 header-value 元组）；其他的嘛……好像随便填并不会影响测试，所以我们参考 `MainServer` 的字段（下图 1）做了点补充（下图 2）。

```
Server: SimpleHTTP/0.6 Python/3.6.9
Date: Thu, 14 Apr 2022 00:35:38 GMT
Content-type: image/jpeg
Content-Length: 2125
Last-Modified: Mon, 21 Jun 2021 11:59:36 GMT
```

```
self.send_response(HTTPStatus.OK)
self.send_header('Server', self.server_version + self.sys_version)
self.send_header('Date', datetime.now().strftime("%a, %d %b %Y %H:%M:%S GMT+8"))
self.send_header('Connection', 'close')
```

- *Caching Server*

总的来说，只要分成两大类：在缓存表里的和不在表里的。

在表里的，就是查得到的并且没有过时，那么我们直接发送出去。

```
if path in self.cacheTable and not self.cacheTable.expired(path):
    return (self.cacheTable.getHeaders(path), self.cacheTable.getBody(path))
```

否则，我们向 `MainServer` 发出 request，得到相应内容，转发并存入缓存。

```
reponse = self.requestMainServer(path)
if reponse is None:
    return (None, None)
headers = self._filterHeaders(reponse.getheaders())
body = reponse.read()
self.cacheTable.setHeaders(path, headers)
self.cacheTable.appendBody(path, body)
return (headers, body)
```

4. Testing

- *DNS server*

  按照手册指导，部署 DNS server 后，使用 Python 检验。

```
12
13  result = resolveDomain("home.nasa.org")
14  print(result)
15
16  result = resolveDomain("test.nasa.org")
17  print(result)
```

```
问题  输出  调试控制台  终端  端口

njucs@njucs-VirtualBox:~/NetLabWork/lab-07-SkyerWalkery$ /usr/bin/pytho
ery/testDNS.py
10.0.0.1
None
home.nasa.org.
```

  官方测试通过。

```
----------------------------------------------------------------------
Ran 5 tests in 0.007s

OK
2022/04/14-09:06:02| [INFO] DNS server terminated
```

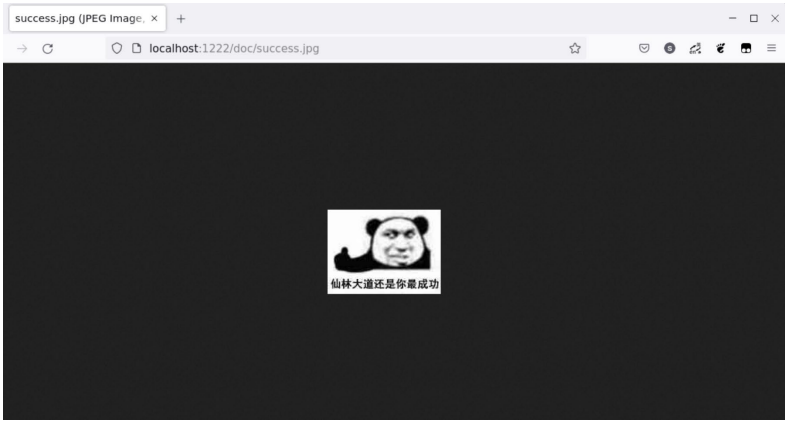
- *Caching server*

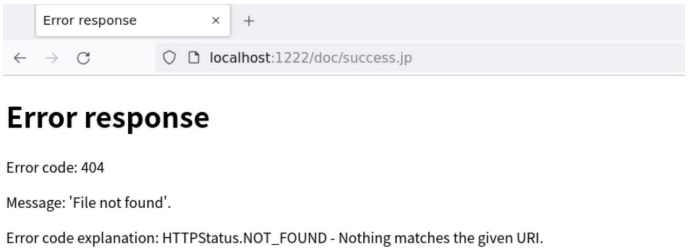
  按照手册指导，部署 main server、caching server，然后自己发出命令。正常获取，404 均表现正常。

```
njucs@njucs-VirtualBox:~/NetLabWork/lab-07-SkyerWalkery$ 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   259k      0 --:--:-- --:--:-- --:--:--  259k
```

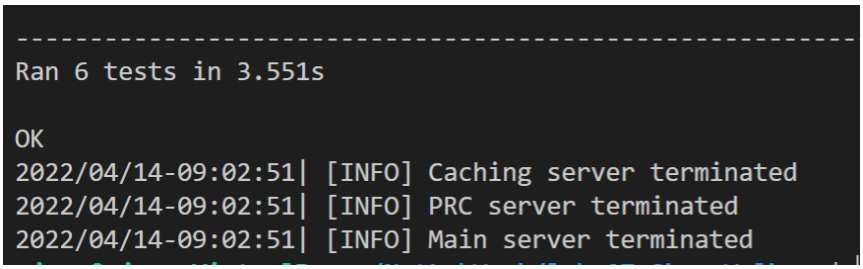
```
127.0.0.1 - - [14/Apr/2022 08:59:56] code 404, message File not found
127.0.0.1 - - [14/Apr/2022 08:59:56] "GET /do/ds HTTP/1.1" 404 -
```

在浏览器中，键入对应 URL 可以直接获取图片/404。





官方测试通过。



5. (Optional) Stream forwarding

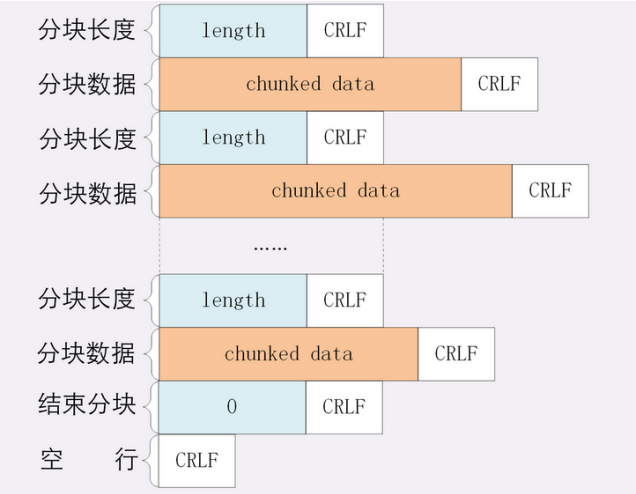
经过手册提示，我们决定使用 HTTP 的分块传输来实现流转发。首先，我们了解一

下分块传输。

分块传输的格式和一般传输大致相同，主要差异有如下几点：

- 含有首部行”`Tranfer-Encoding: chunked`”，用于指明为分块传输；
- 没有首部”`Content-Length`”（都分块了，哪来的实际长度？）；
- 传输的编码分为两部分：本块的数据长度（16 进制）和数据。两者均以”`\r\n`”结尾；
- 最后一个分块为”`0\r\n\r\n`”，用于指示传输结束。

分块可用下图示意：



在 Python 中，使用生成器可以非常直观地实现分块传输，我们的实现细节如下。

在`touchItem()`中，对于已经在缓存中的数据，无需分块传输；不在缓存中的数据，需要逐个块向主机 Pull，然后传给客户端，循环往复。所以，我们需要一个信号来告诉`do_GET()`是否分块。例如，在一开始就 yield 字符串”`whole`”，表明不分块，后面再依次传输首部和数据。

```
if path in self.cacheTable and not self.cacheTable.expired(path):
    yield 'whole'
    yield self.cacheTable.getHeaders(path)
    yield self.cacheTable.getBody(path)
    raise StopIteration
```

如果分块，则每次都要从主机申请`BUFFER_SIZE`大小的数据再发出去。

```
yield 'chunk'
headers = self._filterHeaders(reponse.getheaders())
self.cacheTable.setHeaders(path, headers)
yield headers
# send BUFFER_SIZE bytes at a time
body_parts = bytearray(b'\x00' * BUFFER_SIZE)
read_len = reponse.readinto(body_parts)
while read_len:
    self.cacheTable.appendBody(path, body_parts[:read_len])
    yield (read_len, body_parts[:read_len])
    read_len = reponse.readinto(body_parts)
```

do_GET()和 do_HEAD()行为类似，我们将其代码合并后分析。

调用 touchItem()后，首先使用 next()，判断发过来的是什么类型（分块、不分块、404)。

```
        ret_gen: Union[str, None] = self.server.touchItem(self.path)
        ret_type = next(ret_gen) # generator gives return type first
        assert ret_type is None or isinstance(ret_type, str)
        if ret_type is None:…
        elif ret_type == 'chunk':…
        elif ret_type == 'whole':…
```

若分块，则对于首部字段，删去”content-length”，添加”Transfer-Encoding”，然后发出。

```
# send headers
headers_vals = next(ret_gen)
headers_vals = [header for header in headers_vals
    if header[0].lower() != 'content-length']
# must add 'Transfer-Encoding: chunked'
headers_vals.append(('Transfer-Encoding', 'chunked'))
self.sendHeaders(headers_vals)
```

对于数据字段，则利用生成器，逐块传输，最后发送结束分块的标志。

```
for content_len, body in ret_gen:
    # remove '0x' from hexadecimal, add '\r\n'
    body = (hex(content_len)[2:] + '\r\n').encode() + body + ('\r\n').encode()
    self.sendBody(body)
# indicate the end of the body
body = (hex(0)[2:] + '\r\n').encode() + ('\r\n').encode()
self.sendBody(body)
```

- ***Testing***

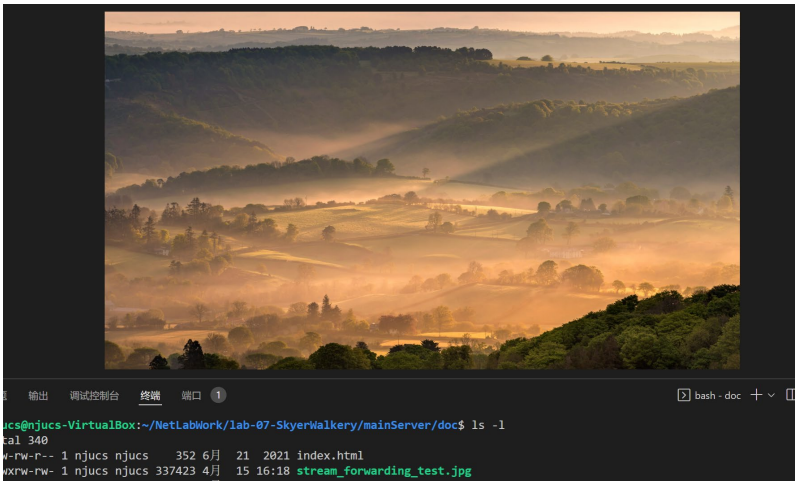
在分块模式下，官方测试集可以正常通过（怎么速度还变慢了？）。

```
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 2.15 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 2.55 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 3.58 ms
ok

----------------------------------------------------------------------
Ran 6 tests in 3.705s

OK
2022/04/16-08:44:17| [INFO] Caching server terminated
2022/04/16-08:44:17| [INFO] PRC server terminated
2022/04/16-08:44:17| [INFO] Main server terminated
```

部署测试中，由于测试图片大小，还填不满一个 BUFFER……为此，我们找来一张三百多 KB 的“巨型”文件。

```
njucs@njucs-VirtualBox:~/NetLabWork/lab-07-SkyerWalkery/mainServer/doc$ ls -l
total 340
-rw-rw-r-- 1 njucs njucs    352 6月  21  2021 index.html
-rwxrw-rw- 1 njucs njucs 337423 4月  15 16:18 stream_forwarding_test.jpg
-rw-rw-r-- 1 njucs njucs   2125 6月  21  2021 success.jpg
```

（图片来自 bing.com 的 2022/4/15 的每日一图，如有侵犯您的权益，请告知作者以删除。）

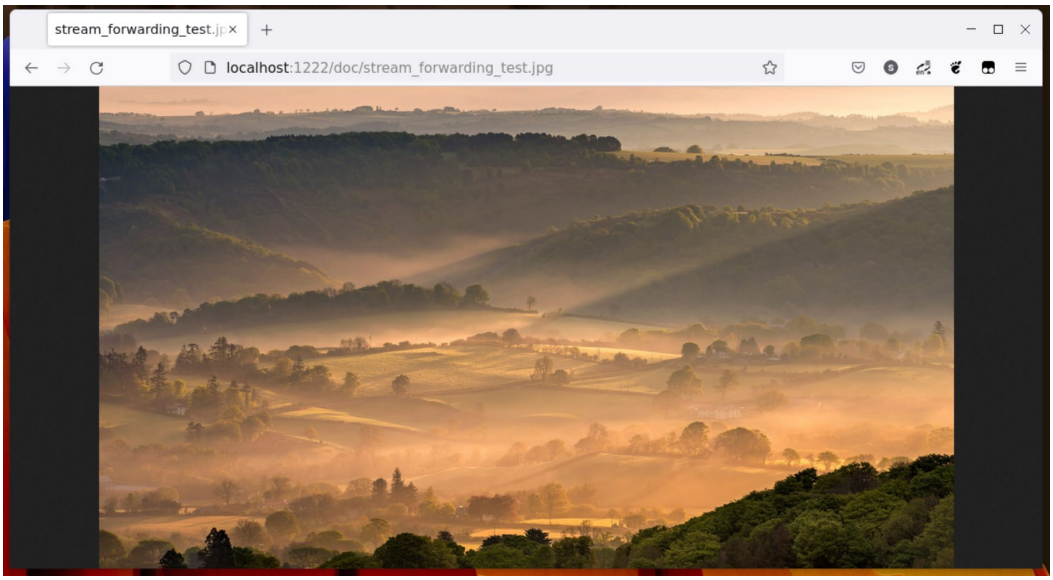


测试成功。先后两次分别为分块（从主机）和为分块（从缓存）。

```
njucs@njucs-VirtualBox:~/NetLabWork/lab-07-SkyerWalkery$ curl -O http://localhost:1222/doc/stream_forwardi
.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  329k    0  329k    0     0  24.7M      0 --:--:-- --:--:-- --:--:-- 26.8M
njucs@njucs-VirtualBox:~/NetLabWork/lab-07-SkyerWalkery$ curl -O http://localhost:1222/doc/stream_forwardi
.jpg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  329k  100  329k    0     0  40.2M      0 --:--:-- --:--:-- --:--:-- 40.2M
```

```
2022/04/16-08:51:04| [Info] Fetched '/doc/stream_forwarding_test.jpg' from main server 'localhost:8000'
2022/04/16-08:51:04| [From 127.0.0.1:37266] "GET /doc/stream_forwarding_test.jpg HTTP/1.1" 200 -
2022/04/16-08:51:10| [From 127.0.0.1:37272] "GET /doc/stream_forwarding_test.jpg HTTP/1.1" 200 -
```

使用浏览器测试，可以看到预期的图片。



6. Deployment

首先，同时测试 DNS server 和 Caching server（3 个样例，我严重怀疑测试的强度……）。

```
----------------------------------------------------------------------
Ran 3 tests in 1.656s

OK
2022/04/14-09:07:41| [INFO] DNS server terminated
2022/04/14-09:07:41| [INFO] Caching server terminated
2022/04/14-09:07:41| [INFO] PRC server terminated
2022/04/14-09:07:41| [INFO] Main server terminated
```

在 OpenNetLab 上运行，从 Client 的 log，可以看出，Caching Server 对于请求时间的缩短有着巨大帮助。

```
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 2.620s

OK

[Request time] 460.66 ms

[Request time] 2.66 ms

[Request time] 460.42 ms
```

四、 核心代码

要不您往上边翻一翻，都有截图；或者，上 Github 看看代码？

五、 总结与感想

- 缓存的设计无处不在，大大降低了各种任务的时间、经济成本；
- CDN 通过 DNS，实现了其功能的同时，对用户还具有高度的封装（你并能非常容易地知道你看的 B 站视频是从哪个 CDN 节点送过来的）；
- URL 的存在，隐藏了底层的技术复杂性，有利于互联网的普及；
- 撒花完结！🎉