

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	202220013	姓名	徐简
Email	161200063@smail.nju.edu.cn	开始/完成日期	2021.5.19-2021.6.2

1. 实验名称

Lab 7: Content Delivery Network

2. 实验目的

1. build a demo of Content Delivery Network (CDN)
2. build a dns server and a caching server
3. deploy servers in OpenNetLab

3. 实验内容、代码与结果

Step 1: DNS Server

- Load DNS Records Table

Read the txt file in and parse it into a dns records table, which is a python list. Each line of the txt file has three components, domain name, record type and record values. There are two cases we should consider. First, if the type is CNAME, append a list consists of name, type and value to the dns table. Second, if the type is A, which means multiple values, append a list consists of name type and a list of values to the dns table.

```
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.
    # -----
    with open("./dnsServer/dns_table.txt") as file:
        while 1:
            line=file.readline()
            if not line:
                break
            else:
                line=line.strip('\n')
                d=line.split(" ")
```

```

if d[1]=="CNAME":
    #dns_table[d[0]]=d[1],d[2]]
    self._dns_table.append([d[0],d[1],d[2]])
else:
    l=len(d)
    temp=[]
    for i in range(2,l):
        temp.append(d[i])
    #dns_table[d[0]]=d[1],temp]
    self._dns_table.append([d[0],d[1],temp])

```

- Reply Clients' DNS Request

If the dns server receives a request, it should search the table and send an appropriate reply back.

Iterate through the dns table to find a match, using the request_domain_name. There are two match cases, with or without a '*'. In both situation, check the type, if it is CNAME, just return the value. If it is A, choose the nearest server in the value list. PS, if the request_domain_name cannot be found in the distance database, pick a random one from the value list.

```

def get_response(self, request_domain_name):
    response_type, response_val = (None, None)
    client_ip, _ = self.client_address
    for i in self.table:
        a=i[0]
        b=i[1]
        if a[0]=='*':
            aa=a[1:]
            if aa in request_domain_name:
                flag=True
                if b=="CNAME":
                    response_type="CNAME"
                    response_val=i[2]
                else:
                    response_type="A"
                    res_candidate=i[2]
                    try:
                        loc=IP_Uutils.getIpLocation(client_ip)
                        response_val=res_candidate[0]
                        for i in res_candidate:
                            iloc=IP_Uutils.getIpLocation(i)
                            rloc=IP_Uutils.getIpLocation(response_val)
                            idis=abs(iloc[0]-loc[0])*abs(iloc[0]-
loc[0])+abs(iloc[1]-loc[1])*abs(iloc[1]-loc[1])
                            rdis=abs(rloc[0]-loc[0])*abs(rloc[0]-
loc[0])+abs(rloc[1]-loc[1])*abs(rloc[1]-loc[1])
                            if(idis<rdis):
                                response_val=i
                    except:

```

```

        rd=random.randint(0,len(res_candidate)-1)
        response_val=res_candidate[rd]

    break
else:
    if a==request_domain_name:
        flag=True
        if b=='CNAME':
            response_type="CNAME"
            response_val=i[2]
        else:
            response_type="A"
            res_candidate=i[2]
        try:
            loc=IP_Utils.getIpLocation(client_ip)
            response_val=res_candidate[0]
            for i in res_candidate:
                iloc=IP_Utils.getIpLocation(i)
                rloc=IP_Utils.getIpLocation(response_val)
                idis=abs(iloc[0]-loc[0])*abs(iloc[0]-
loc[0])+abs(iloc[1]-loc[1])*abs(iloc[1]-loc[1])
                rdis=abs(rloc[0]-loc[0])*abs(rloc[0]-
loc[0])+abs(rloc[1]-loc[1])*abs(rloc[1]-loc[1])
                if(idis<rdis):
                    response_val=i
                    log_info(response_val)
        except:
            rd=random.randint(0,len(res_candidate)-1)
            response_val=res_candidate[rd]

    break
return (response_type, response_val)

```

Step 2: Caching server

- Caching Server

`touchItem()` ,called by `HttpHandler`, serves as a bridge of server and handler.

Search the path in the `cacheTable`, if fails to match, then request from the main server. If the path is already in the table and it is not expired yet, return its corresponding headers and body. Else, request from the main server and update the cache table.

```

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 exist or expires, fetch from main server.
    Write the headers to local cache and return the body.

```

```

...
# TODO: implement the logic described in doc-string
ct=self.cacheTable
if path in ct.keys():
    #h=ct.getHeaders(path)
    if not ct.expired(path):
        print("cache")
        return [ct.getHeaders(path),ct.getBody(path)]
    else:
        rp=self.requestMainServer(path)
        if rp is not None:
            print("main")
            #print(rp.getheaders())
            h=rp.getheaders()
            b=rp.read()
            ct.setHeaders(path,h)
            ct.appendBody(path,b)
            #return [ct.getHeaders(path),ct.getBody(path)]
            return [h,b]
        else:
            return None;
else:
    rp=self.requestMainServer(path)
    if rp is not None:
        print("main")
        h=rp.getheaders()
        b=rp.read()
        ct.setHeaders(path,h)
        ct.appendBody(path,b)
        return [h,b]
        #print(rp.getheaders())
        #ct.setHeaders(path,rp.getheaders())
        #ct.appendBody(path,rp.read())
        #return [ct.getHeaders(path),ct.getBody(path)]
    else:
        return None;

```

- HTTPRequestHandler

Send headers, stored in self.headers and body, passed in as a parameter.

```

def sendHeaders(self):
    ''' Send HTTP headers to client'''
    # TODO: implement the logic of sending headers
    #res=self.server.touchItem(self.path)
    #h=res[0]
    for i in self.headers:
        self.send_header(i[0],i[1])

```

```

        self.end_headers()

    def sendBody(self, body):
        ''' Send HTTP body to client.
        Should be called after calling self.sendHeaders(). Else you may get
        broken pipe error.
        '''
        self.wfile.write(body)

```

When receive a http get request, call touchItem. It return value of touchItem is none, then send error messages. Otherwise, store headers in self.header and call sendHeaders and sendBody.

```

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.

    p=self.path
    body=self.server.touchItem(p)

    if body is not None:
        self.headers=body[0]
        self.send_response(200)
        self.sendHeaders()
        self.sendBody(body[1])
    else:
        self.send_error(HTTPStatus.NOT_FOUND, "'File not found'")

```

```

class Item():
    def __init__(self, pkt, seq):
        self.pkt=pkt
        self.seq=seq
        self.ackflag=0
    #store Items in the queue
    q=[]

```

When the `Blaster` receives a packet, i.e an ACK. If the sequence number of the ACK equals a packet's sequence number in the queue, then mark it.

If the head of the queue is ACKed, then delete it and slide the window ahead. Also, the new time should be recorded to monitor timeouts.

```

def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    _, fromIface, packet = recv
    log_debug("I got a packet")
    seq=packet[3].to_bytes()[0:4]
    seq0=int.from_bytes(seq,byteorder='big',signed=False)
    #extract the sequence num of the ACK received
    for i in self.q:
        if i.seq==seq0:
            print(seq0,"got ack")
            i.ackflag=1
    while len(self.q)>0:
        # move the window forward
        if self.q[0].ackflag==1:
            self.LHS=self.q[0].seq+1
            del(self.q[0])
            self.curTime=time.time()
        else:
            break

```

Step 4: Deploy and Test

```
python3 test_entry.py dns
```

```

kirakiraakira@KiraKiraAkiraMacBook-Pro lab-7-KiraKiraAkira % python3 test_entr
y.py dns
2021/06/16-01:43:47| [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 0.019s

OK
2021/06/16-01:43:49| [INFO] DNS server terminated

```

```
python3 test_entry.py cache
```

```

2021/06/16-01:44:44| [INFO] Main server started
2021/06/16-01:44:44| [INFO] RPC server started
2021/06/16-01:44:44| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_cache.TestCache) ...
[Request time] 15.76 ms
ok
test_02_cache_hit_1 (testcases.test_cache.TestCache) ...
[Request time] 6.11 ms
ok
test_03_cache_missed_2 (testcases.test_cache.TestCache) ...
[Request time] 10.45 ms
ok
test_04_cache_hit_2 (testcases.test_cache.TestCache) ...
[Request time] 6.65 ms
ok
test_05_HEAD (testcases.test_cache.TestCache) ...
[Request time] 6.88 ms
ok
test_06_not_found (testcases.test_cache.TestCache) ...
[Request time] 8.30 ms
ok

```

Ran 6 tests in 2.549s

```
python3 test_entry.py all
```

```

2021/06/16-01:45:28| [INFO] DNS server started
2021/06/16-01:45:28| [INFO] Main server started
2021/06/16-01:45:28| [INFO] RPC server started
2021/06/16-01:45:28| [INFO] Caching server started
test_01_cache_missed_1 (testcases.test_all.TestAll) ...
[Request time] 15.56 ms
ok
test_02_cache_hit_1 (testcases.test_all.TestAll) ...
[Request time] 5.47 ms
ok
test_03_not_found (testcases.test_all.TestAll) ...
[Request time] 8.59 ms
ok

```

Ran 3 tests in 1.629s

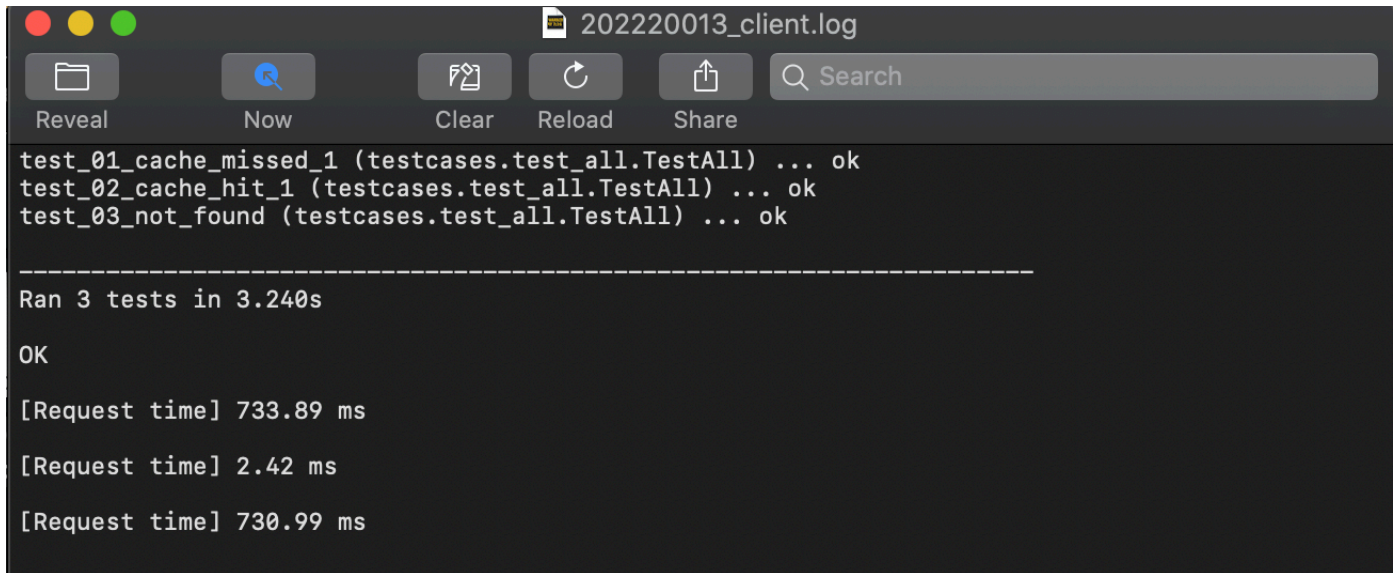
```

OK
2021/06/16-01:45:31| [INFO] DNS server terminated
2021/06/16-01:45:31| [INFO] Caching server terminated
2021/06/16-01:45:31| [INFO] PRC server terminated
2021/06/16-01:45:31| [INFO] Main server terminated

```

- deploy in OpenNetLab

As the log shows, cache shortened the request time from about 700ms to 2.42ms. A great improvement !



```
202220013_client.log
Reveal Now Clear Reload Share Search
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 3.240s
OK
[Request time] 733.89 ms
[Request time] 2.42 ms
[Request time] 730.99 ms
```

4. 总结与感想

The key of lab7 is to understand the skeleton and figure out when to use what API. Once you have an overall understanding of the skeleton, it is not hard to code.

To sum up, lab7 is really interesting and it helps to understand the CDN mechanism in the real network.