

# Python发HTTPs请求踩坑记录

## 目的

在最近的项目中，需要发送大量HTTPs请求，要求可设置服务端IP、Client Hello中SNI值、HTTP头部字段。

## 现状分析

一个HTTPs请求主要由两部分组成

- SSL握手
- HTTP通信

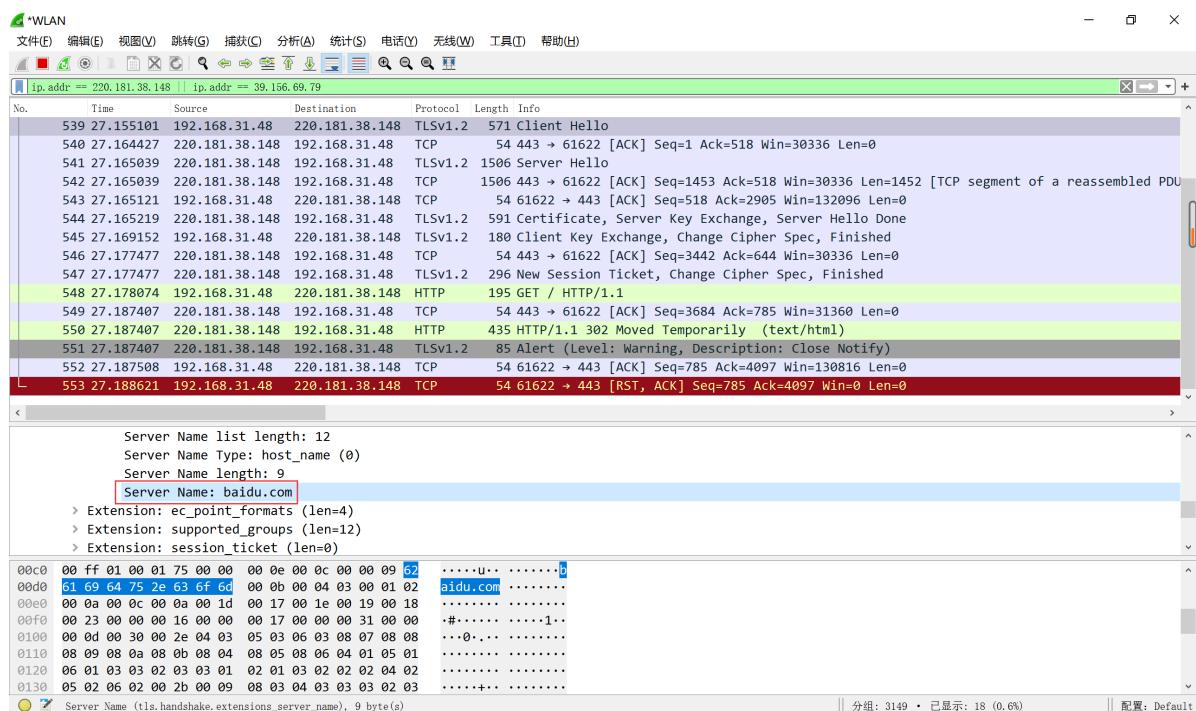
现有Python库，比如urllib，对HTTP协议有较好支持，可以很方便设置IPATH、HTTP请求内容、HTTP header等各种参数。但相反的是，urllib等库无法设置SSL握手时的参数，包括Client Hello中SNI值。

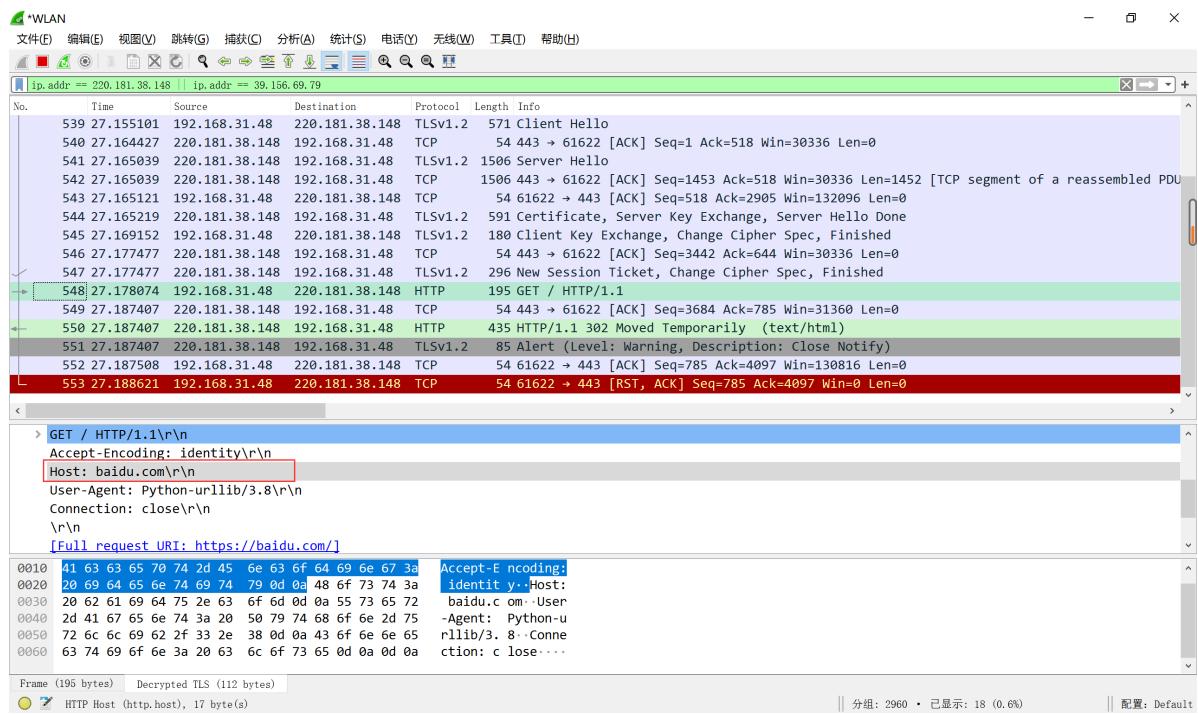
## urllib

urllib库对HTTPs请求有着较好支持。如下简单代码即可发送一个HTTPs请求

```
from urllib import request

if __name__ == "__main__":
    url = "https://baidu.com"
    req = request.Request(url=url, data=None)
    request.urlopen(req)
```





捕包发现：

- 此时能捕获到发往baidu服务端IP的包
- SNI为baidu.com
- HTTP头部Host字段为baidu.com

对于这样一个简单的HTTPS请求，可以看到一下几点：

- 对于https:{domain}/path类型的请求，SNI与domain保持一致
- 对于https:{domain}/path类型的请求，HOST与domain保持一致
- 对于https:{domain}/path类型的请求，服务端IP由domain解析得到

同时也有几个问题，也是我们的需求：

- 能否手动设置HTTP头部Host字段值？(问题1)
- 如何手动设置服务端IP？(问题2)
- 能否手动设置SNI值？(问题3)

对于问题1，urllib可设置host值，代码如下。但设置host值后，事情变得有趣起来

```
from urllib import request

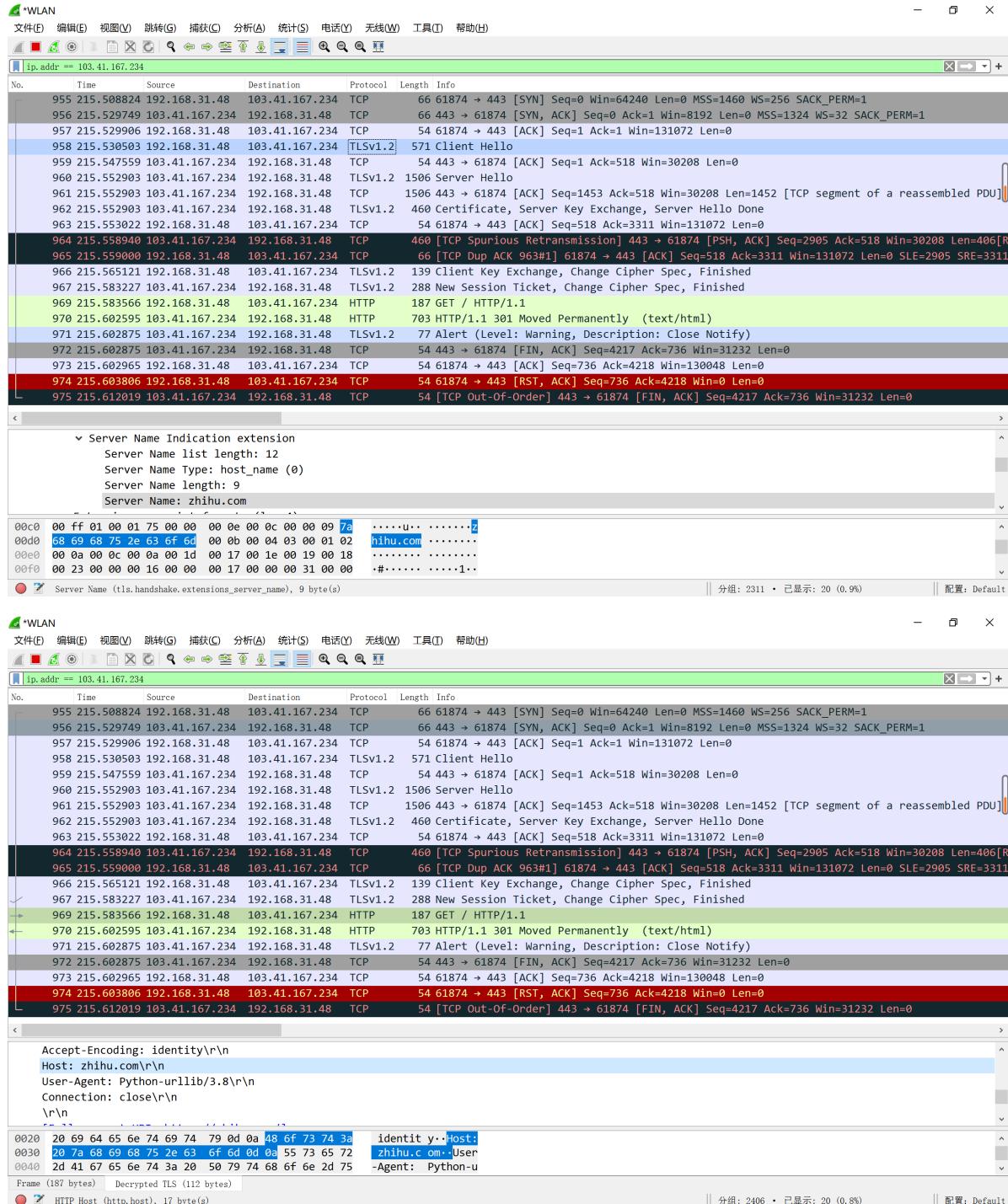
if __name__ == "__main__":
    url = "https://baidu.com"
    req = request.Request(url=url, data=None)
    req.host = "zhihu.com"
    request.urlopen(req)
```

将host值设置为zhihu.com后，无法捕获到发往baidu的包(ip.addr == 220.181.38.148 || ip.addr == 39.156.69.79)，但能捕获到发往zhihu的包(ip.addr == 103.41.167.234)，并且

- SNI与host值保持一致
- HTTP header中host字段与host值保持一致

也就是说，url中的domain参数完全失效，host完全取代了domain的作用，用作：

- 得到服务端IP
- 充当SNI
- 充当HTTP头部HOST字段值



urllib还支持设置header，因此可以在header中设置HOST字段，代码如下：

```
from urllib import request

if __name__ == "__main__":
    url = "https://baidu.com"
    req = request.Request(url=url, data=None)
    # req.host = "zhihu.com"
    req.add_header("Host", "zhihu.com")
    request.urlopen(req)
```

这时情况与前述又不相同

- 此时能捕获到发往baidu服务端的数据包，说明domain用作获取服务端IP
- SNI为baidu.com，说明domain充当了SNI值
- HTTP头部Host字段为zhihu.com，与手动设置的header保持一致，说明可以手动设置host。

一个基本结论是直接设置HTTP header不会影响到domain的作用，因此可以手动设置HTTP头部Host字段值。

## 至此，我们解决了问题1，手动设置HTTP头部Host字段值。

The figure consists of two screenshots of the NetworkMiner tool interface. Both screenshots show a list of network frames and their details.

**Screenshot 1 (Top):** This screenshot shows the initial TLS handshake. Frame 32 (highlighted in green) is an HTTP GET request to '192.168.31.48' (port 39.156.69.79). The details pane shows the request headers, including 'Host: baidu.com'. The expanded 'server\_name' extension in the details pane also lists 'baidu.com'.

**Screenshot 2 (Bottom):** This screenshot shows the continuation of the handshake. Frame 32 (highlighted in green) is the same GET request. The details pane shows the full request URI: 'https://zhihu.com/'. The expanded 'Host' header in the details pane also lists 'zhihu.com'.

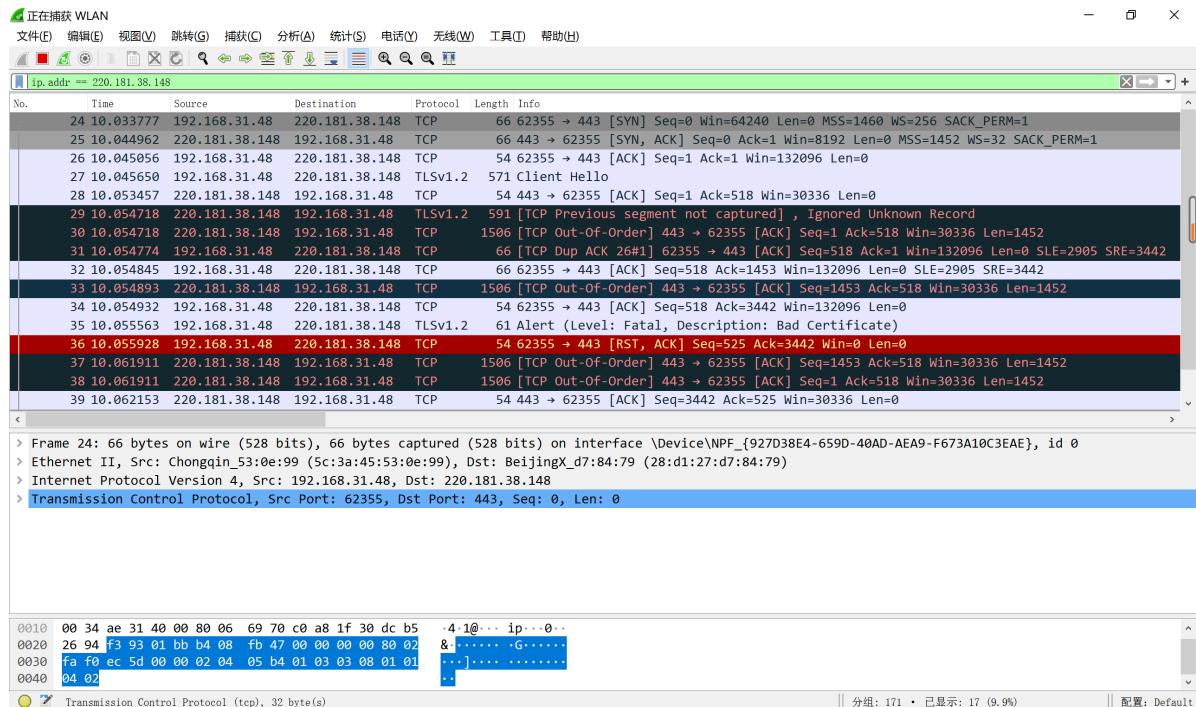
但如何手动设置服务端IP呢？换言之，如何分开设置IP和SNI，不要求SNI是IP的域名。对于问题2，我们尝试将url中domain修改为IP，事情再次有趣了起来。

```

from urllib import request

if __name__ == "__main__":
    url = "https://220.181.38.148"
    req = request.Request(url=url, data=None)
    # req.host = "zhihu.com"
    req.add_header("HOST", "zhihu.com")
    request.urlopen(req)

```



可以看到SSL握手无法完成。原因也比较明显，此时host和domain同时不存在，而IP又无法充当SNI，故SNI值不存在，握手被服务端拒绝。

so，如何同时解决问题2和问题3？

## socket

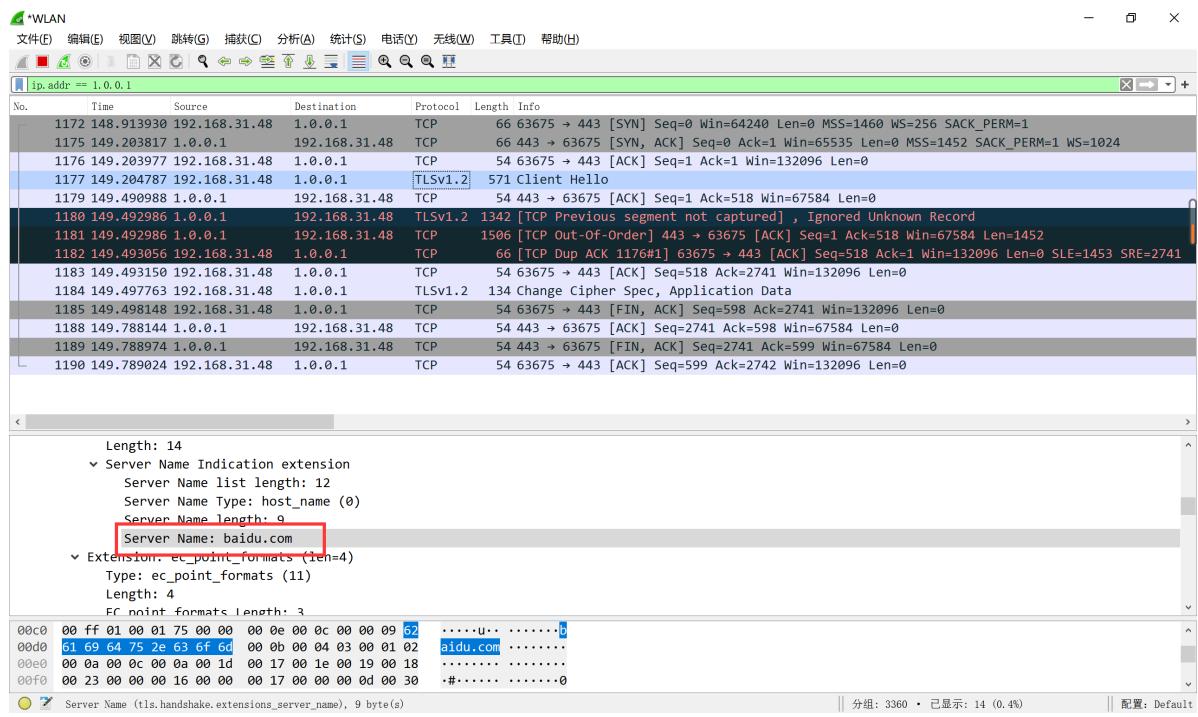
首先验证一点，Python可以同时手动设置服务端IP和SNI。

```

import socket
import ssl

if __name__ == "__main__":
    ssl_ctx = ssl.create_default_context()
    ssl_ctx.check_hostname = False
    tcp_sockt = socket.create_connection(("1.0.0.1", 443))
    ssl_socket = ssl_ctx.wrap_socket(tcp_sockt, server_hostname="baidu.com")
    ssl_socket.close()

```



上述程序完成了一次SSL握手，SNI和服务端IP都可手动设置。因此考虑调试并修改urllib库实现该功能

## urllib调试分析

```
from urllib import request
import ssl

def make_ssl_context():
    ssl_ctx = ssl.create_default_context()
    ssl_ctx.check_hostname = False
    ssl_ctx.verify_mode = ssl.CERT_NONE

if __name__ == "__main__":
    url = "https://220.181.38.148"
    req = request.Request(url=url, data=None)
    req.host = "zhihu.com"
    req.add_header("HOST", "zhihu.com")
    request.urlopen(req, context=make_ssl_context())
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help feature\_creator - D:\Anaconda\lib\urllib\request.py

test4.py test request.py

server\_hostname

```

503         if result is not None:
504             return result
505
506     def open(self, fullurl, data=None, timeout=socket._GLOBAL_DEFAULT_TIMEOUT): self: <urllib.request.OpenerDirector object at 0x00000253F3A3FF78>
507         # accept a URL or a Request object
508         if isinstance(fullurl, str):
509             req = Request(fullurl, data) req: <urllib.request.Request object at 0x00000253F3A18250>
510         else:
511             req = fullurl
512             if data is not None:
513                 req.data = data
514
515             req.timeout = timeout
516             protocol = req.type
517

```

OpenerDirector > open()

Debug: test

Frames Variables

- MainThread
- open, request.py:515
- urlopen, request.py:222
- <module>, test.py:16

host = [str] 'zhihu.com'

origin\_req\_host = [str] '220.181.38.148'

Event Log

调试上述代码，发现在Request类中有两个host：

- host: 与req.host值一致
- origin\_req\_host: 与IP\domain值一致

如果能将host用作设置SNI、origin\_req\_host用作设置服务端IP则问题解决。

继续调试，发现Request类型传递到http\_class类型后，origin\_req\_host丢失

Debug: test

Frames Variables

- MainThread
- connect, client.py:917
- connect, client.py:1402
- send, client.py:946
- send\_output, client.py:1006
- endheaders, client.py:1235
- send\_request, client.py:1286
- request, client.py:1240
- do\_open, request.py:1351
- https\_open, request.py:1394
- call\_chain, request.py:502
- open, request.py:525
- urlopen, request.py:222
- <module>, test.py:16

host = [str] 'baidu.com'

origin\_req\_host = [str] '220.181.38.148'

method = (str) 'GET'

auto\_open = [int] 1

blocksize = [int] 8192

cert\_file = [NoneType] None

debuglevel = [int] 0

default\_port = [int] 443

host = [str] 'baidu.com'

key\_file = [NoneType] None

port = [int] 443

response\_class = [ABCMeta] <class 'http.client.HTTPResponse'>

sock = [NoneType] None

source\_address = [NoneType] None

timeout = [object] <object object at 0x000001D1E1B64160>

url = [str] "

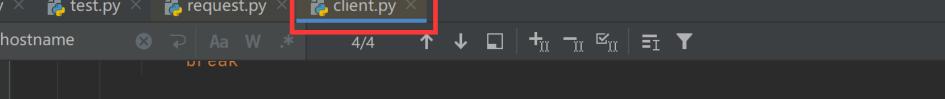
Event Log

因此在http\_class类中添加origin\_req\_host元素。然后使用origin\_req\_host创建SSL连接。问题解决

D: > Anaconda > Lib > urllib > request.py

Project test4.py × test.py × request.py × client.py ×

```
1343 # PROXY AUTHENTICATION SHOULD NOT BE SENT TO ORIGIN
1344 # server.
1345 del headers[proxy_auth_hdr]
1346 h.set_tunnel(req._tunnel_host, headers=tunnel_headers)
1347
1348 h.origin_req_host = req.origin_req_host
1349 try:
1350     try:
```



```
D: > Anaconda > Lib > http > client.py
D: > Anaconda > Lib > http > client.py
server_hostname
server_hostname
if self.debuglevel > 0:
    print('header:', line.decode())
def connect(self):
    """Connect to the host and port specified in __init__"""
    self.sock = self._create_connection(
        # (self.host,self.port), self.timeout, self.source_address
        (self.origin_req_host, self.port), self.timeout, self.source_address)
    self.sock.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
```