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

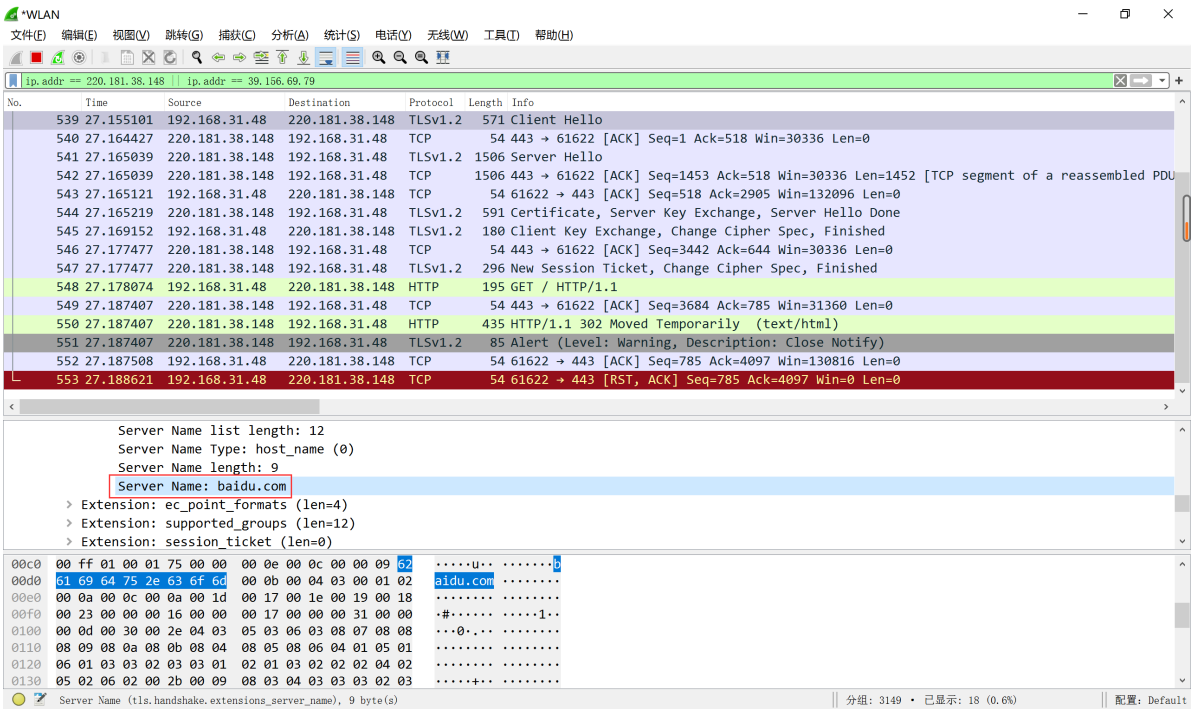
- SSL握手
- HTTP通信

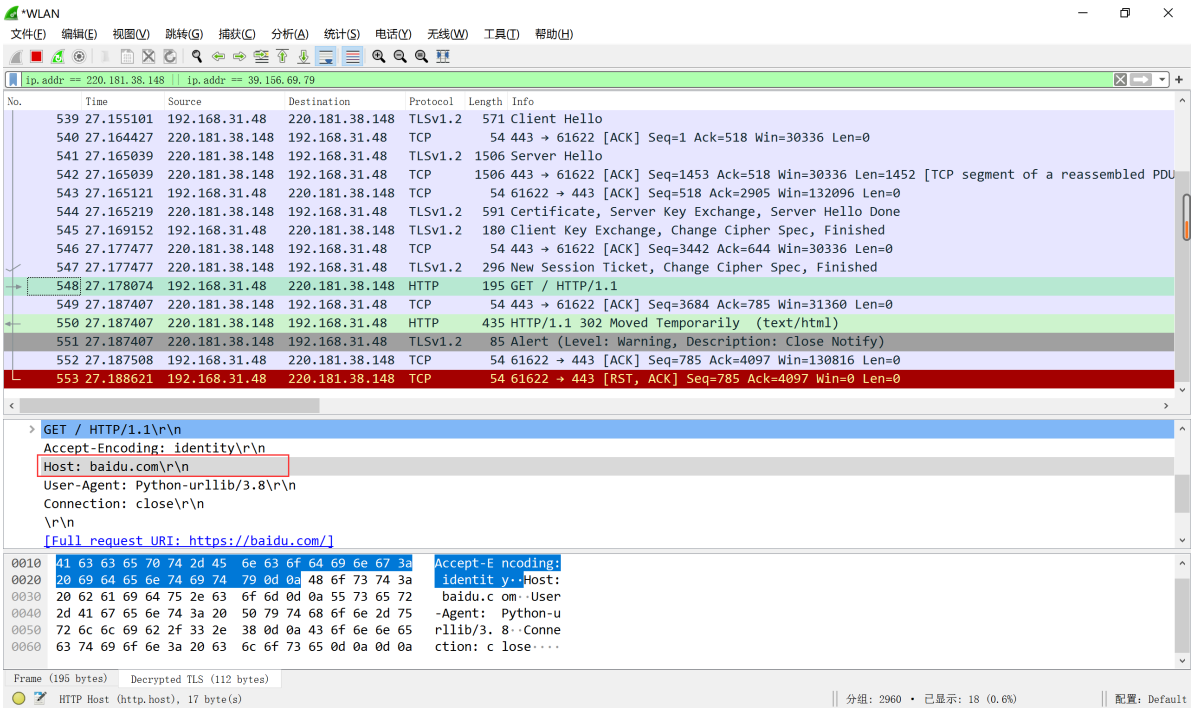
现有Python库，比如urllib，对HTTP协议有较好支持，可以很方便设置IPATH、HTTP请求内容、HTTP header等各种参数。但相反的是，urllib等库无法设置SSL握手时的参数，包括Client Hello中SNI值。

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

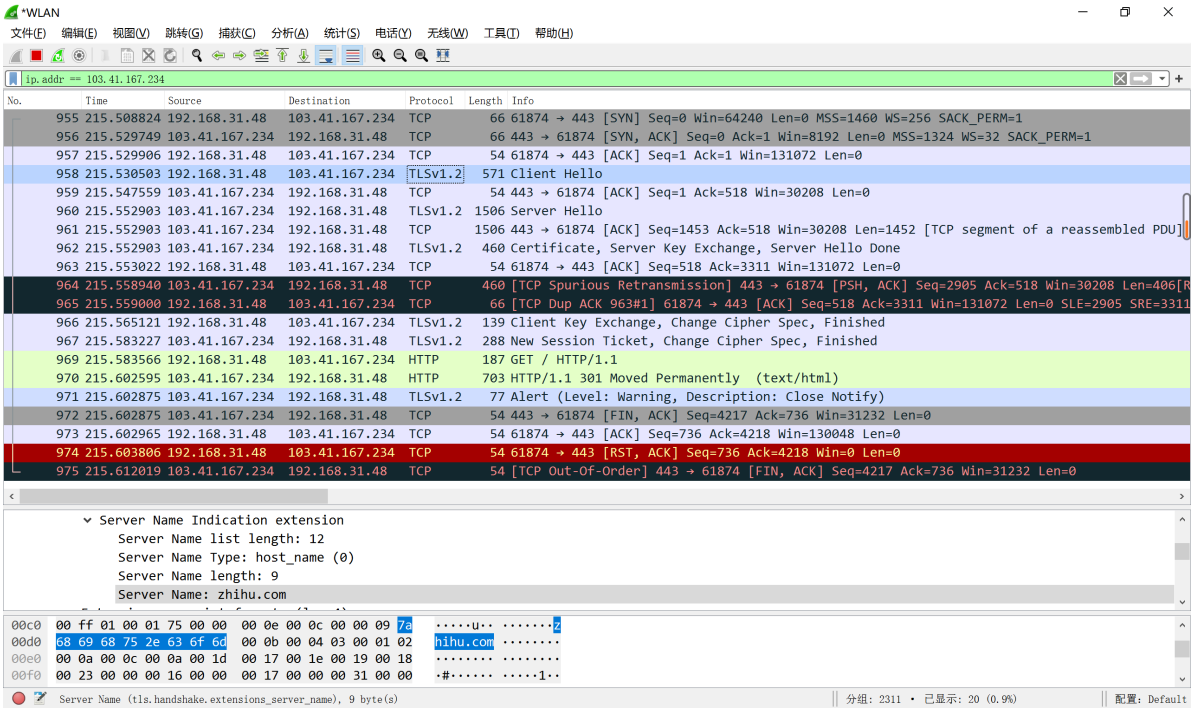
```
from urllib import request


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

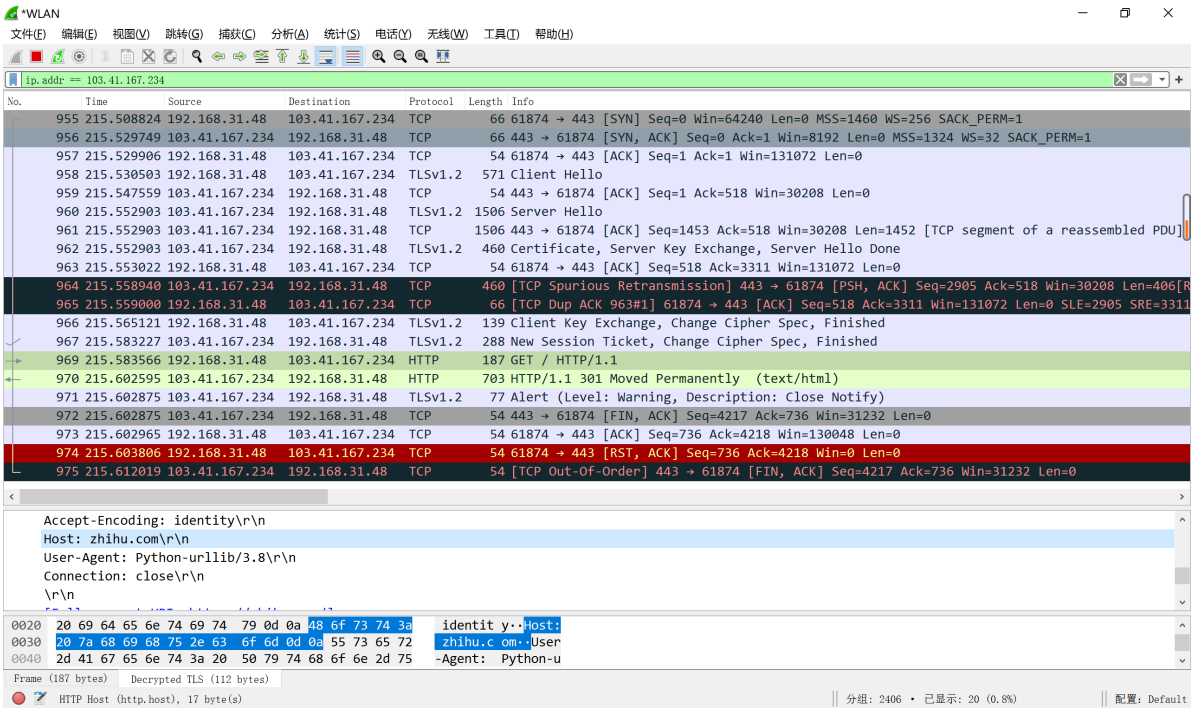
将host值设置为`zhihu.com`后，无法捕获到发往baidu的包(`ip.addr == 220.181.38.148 || ip.addr == 39.156.69.79`)，但能捕获到发往zhihu的包(`ip.addr == 103.41.167.234`)，并且

- 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：

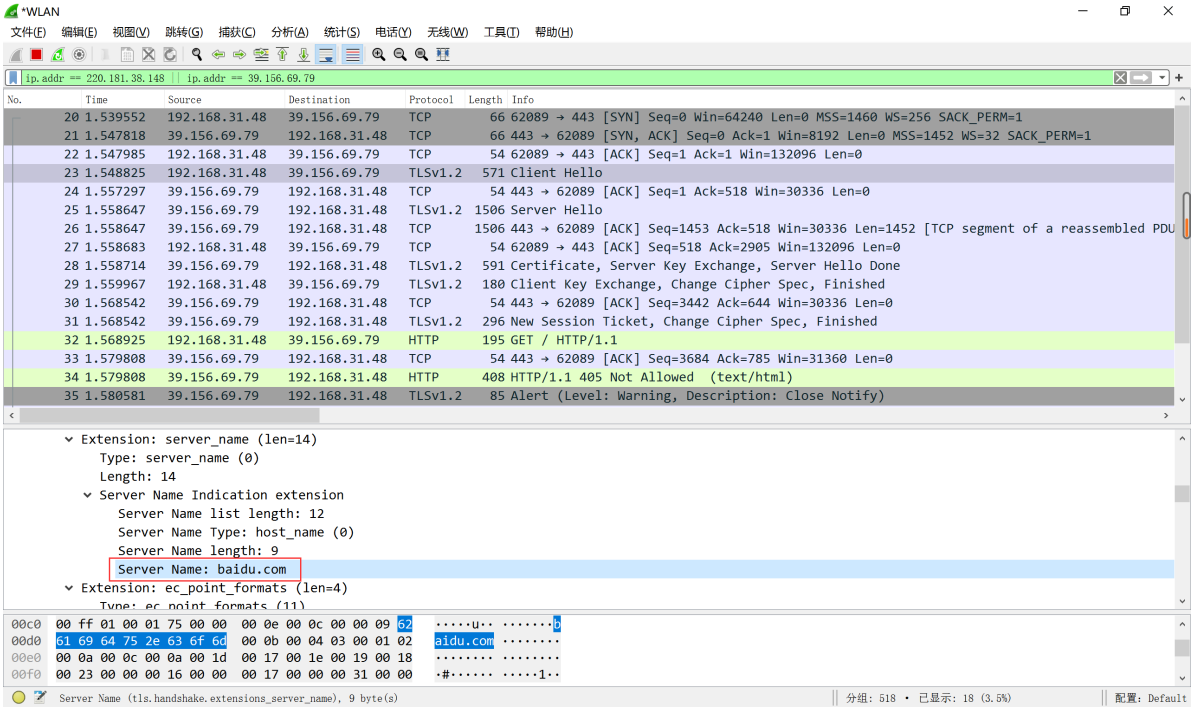
```
from urllib import request


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)
```

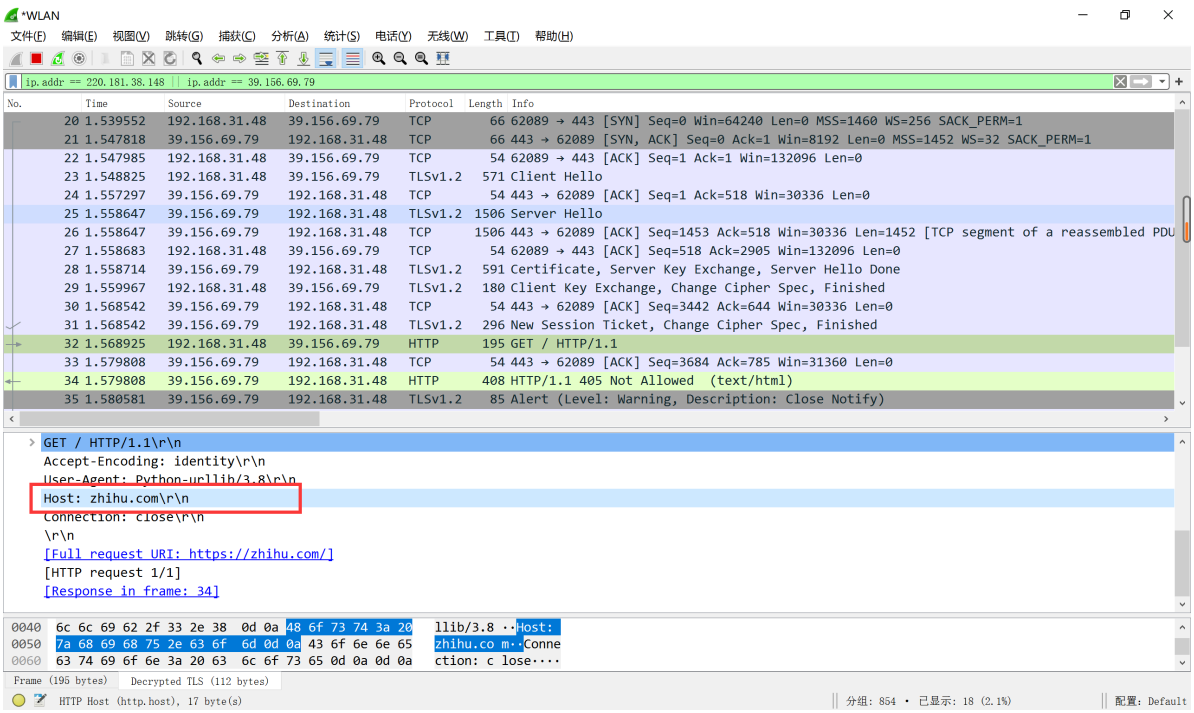
这时情况与前述又不相同

- 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。

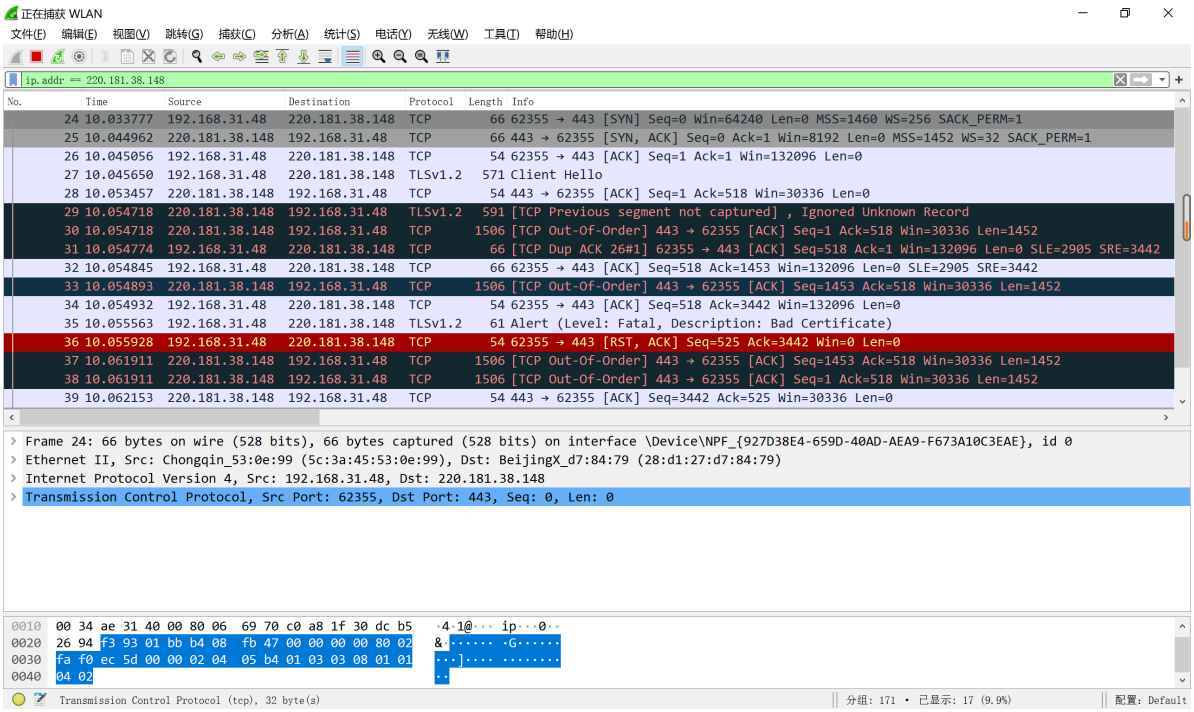
到此，我们解决了问题1，手动设置HTTP头部Host字段值。





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。

```python
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)
```



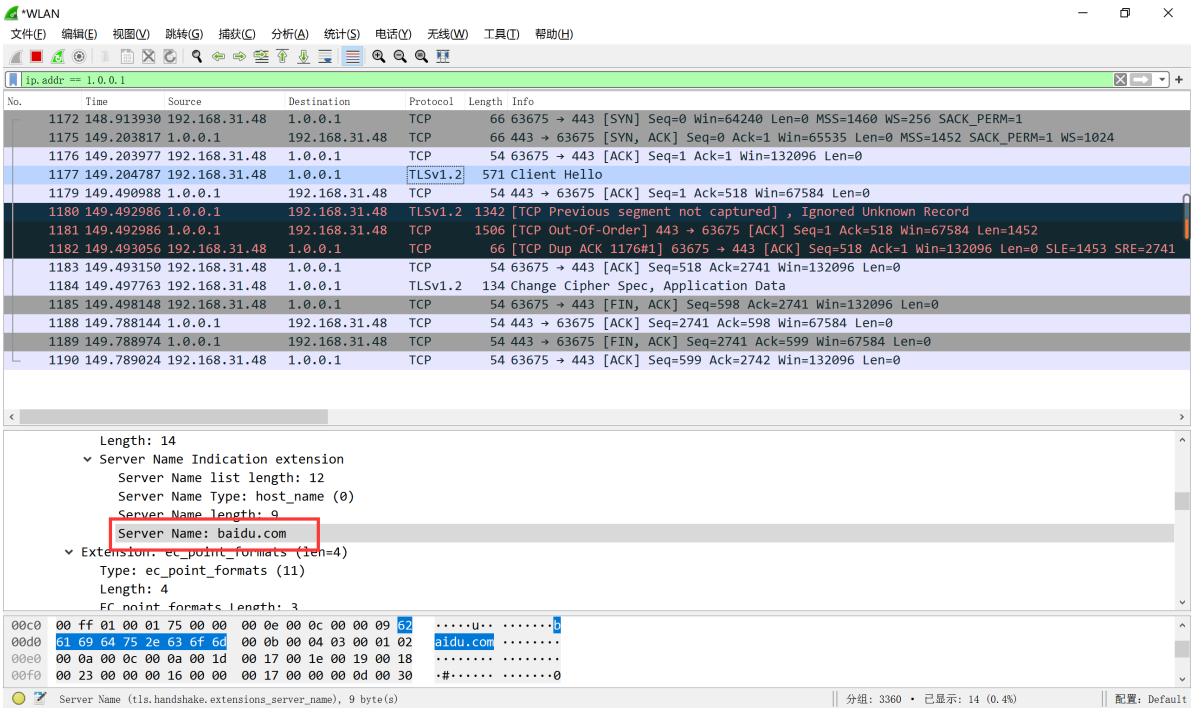
可以看到SSL握手无法完成。原因也比较明显，此时host和domain同时不存在，而IP又无法充当SNI，故SNI值不存在，握手被服务端拒绝。

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。

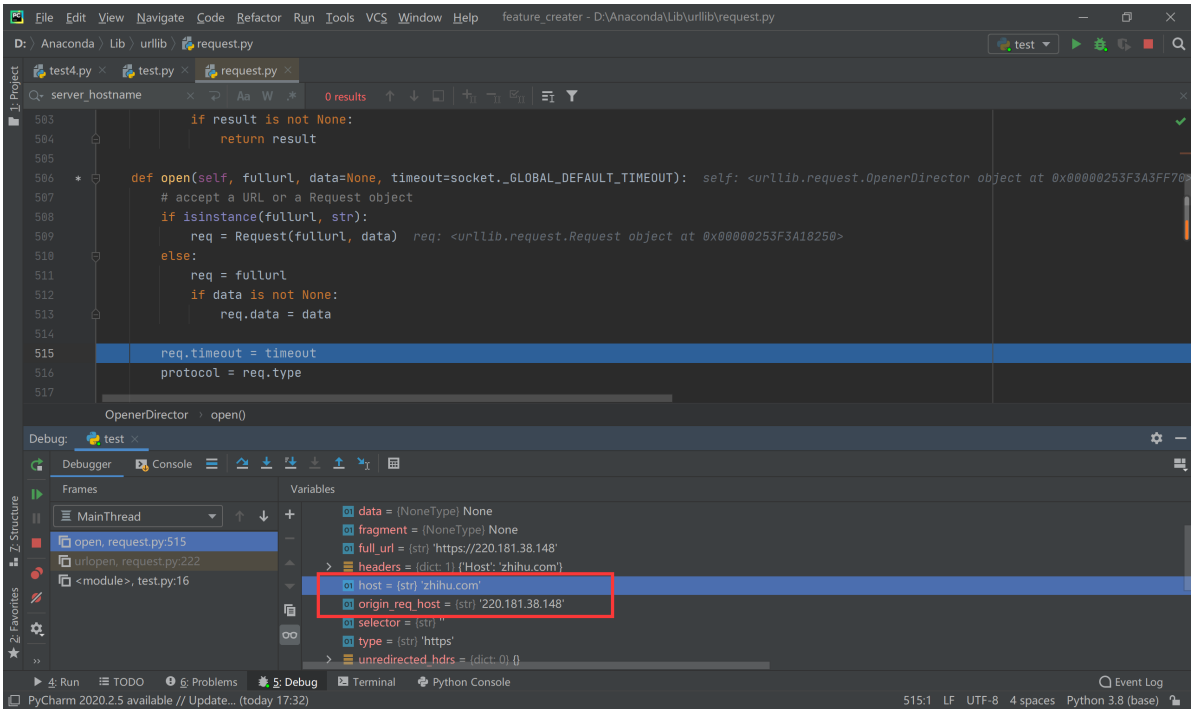
```python
import socket
import ssl

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())
```

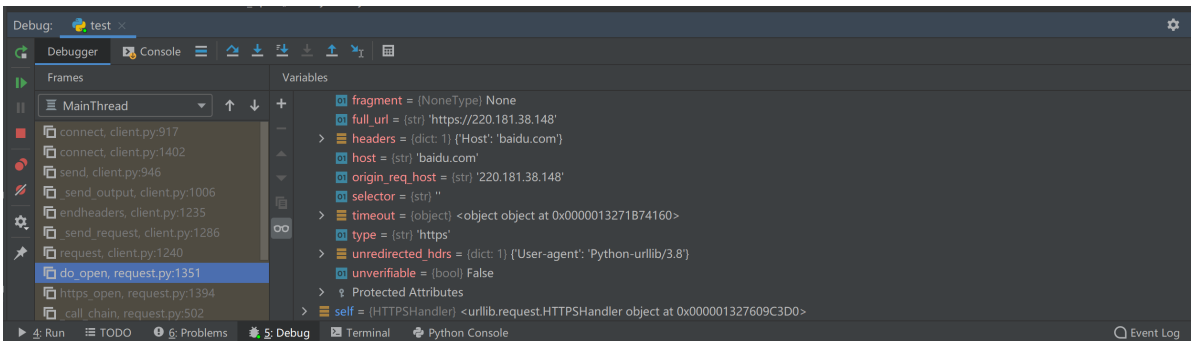


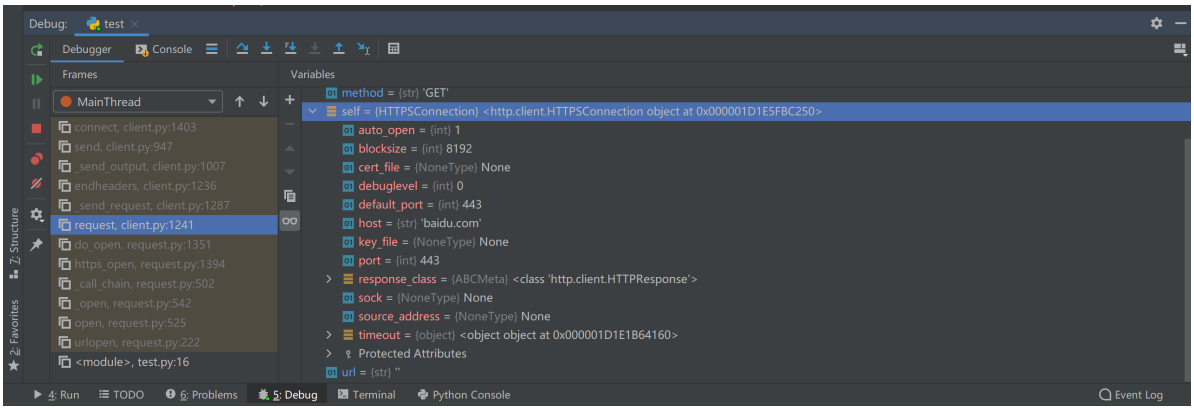
调试上述代码，发现在Request类中有两个host：

- host：与req.host值一致
- origin_req_host：与IP\domain值一致

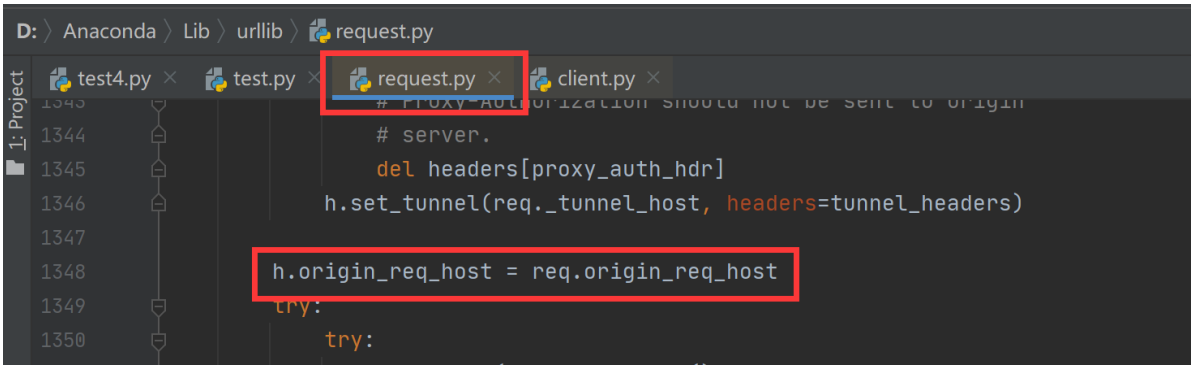
如果能将host用作设置SNI、origin_req_host用作设置服务端IP则问题解决。

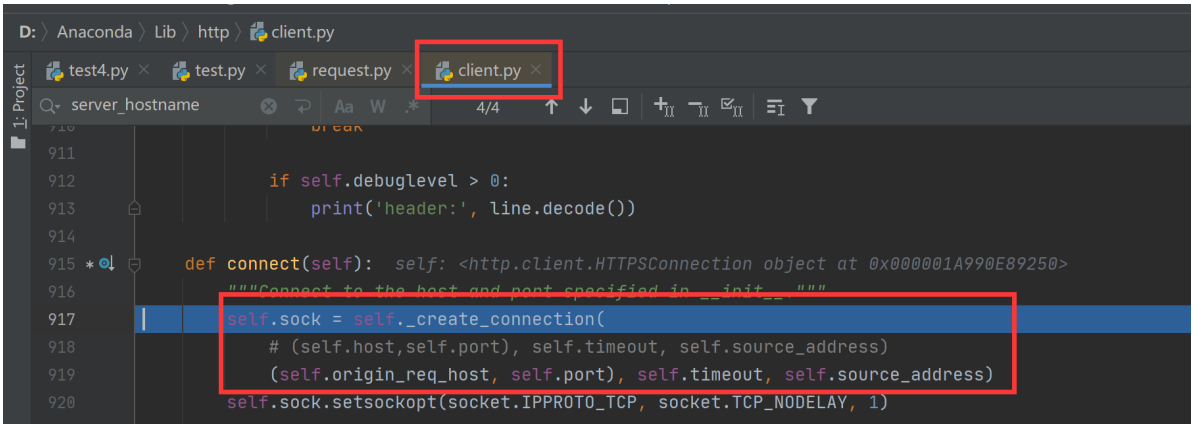
继续调试，发现Request类型传递到http_class类型后，origin_req_host丢失





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
test4.py
test.py
request.py
client.py
server.
del headers[proxy_auth_hdr]
h.set_tunnel(req._tunnel_host, headers=tunnel_headers)
h.origin_req_host = req.origin_req_host
try:
try:


D: Anaconda Lib http client.py
test4.py
test.py
request.py
client.py
server_hostname
4/4
if self.debuglevel > 0:
print('header:', line.decode())
def connect(self): self: <http.client.HTTPSConnection object at 0x000001A990E89250>
"""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)