Qun Lou contribution ---- Proxy side
Jinhuan Liu contribution ---- Server and Client side

Server Side:

Server Application mainly uses the Libtls library to wrap regular sockets. Initialize with tls_init in the library, then create a new TLS configuration, import the CA certificate, server certificate, and the server private key. Call the tls_server () function to initialize the context. Next, according to the socket process to establish a socket for monitoring (TCP connection). Use accept to block and wait until a connection is obtained. Then, it transfer the socket as an argument to the tls_accept_socket function, and then accept and send the message using the wrapped tls_read and tls_write functions.

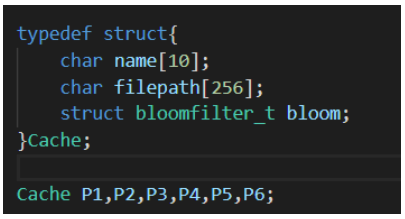
The received message is the file name, and we searched the file name in the local folder "./server ". Use the Access function under Linux C to determine whether the file exists, if the file exists, read the file and return. If the file does not exist, returns the constructed string "file@not~exit$". Then close the connection, go back to the beginning of the loop, and continue to accept listening.

Client Side:

The client mainly uses the libtls library to wrap the regular socket for normal communication. The TLS and Socket initialization process refers to the server side. Client connect to the proxy and send it own constructed packet. The contents of the packet consist of a char type of 10 bytes named cache and a char type of 1024 bytes named filename. Cache holds the cache name (such as P1,P2) and filename holds the filename you want to download. Before the package is sent, the package is constructed, mainly using consistent hash to select cache. It concatenates the string into cache·filename (e.g., P1testfile), then we calculate its hash value and sort it after we calculate all hash value. Pick the highest hash value and fill it I the send package. There are three types of return data: fil@not ~exit$ mean the file is not in the server, file@in ~blacklist$ indicates that the file name is on the blacklist by the proxy, and the file is written to the local clientfile folder except in the two cases which I mention before.

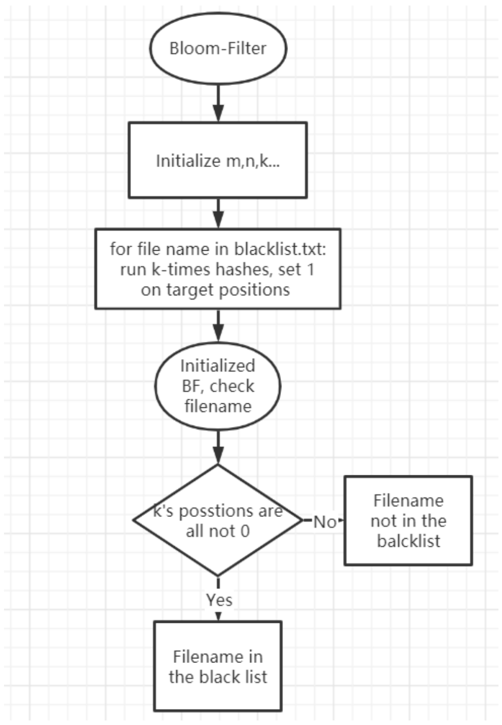
Proxy Side:

The TLS part of Proxy works just like its counterpart on Server. And we realized 3 functions: (1) listen and accept request from Server; (2) judge the filename of request whether is in blacklist or not; (3) Communicate with Server.
Start to listen locally, wait until receive request from Client, then split the package into 2 parts: cache_name and filename. Select cache according to cache_name. In fact, we use Struct to realize cache, with 3 elements: (1)name; (2) filepath; (3)bloomfilter_t



```c
typedef struct{
    char name[10];
    char filepath[256];
    struct bloomfilter_t bloom;
}Cache;

Cache P1,P2,P3,P4,P5,P6;
```

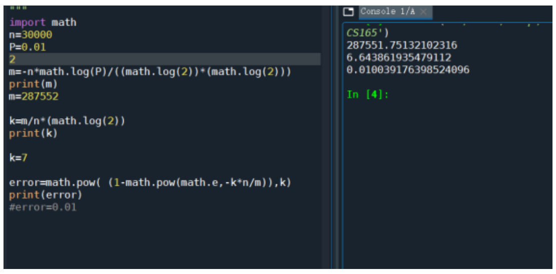
About Bloom-Filter:
The main process run as the following diagram:


Bloom-Filter
Initialize m,n,k...
for file name in blacklist.txt: run k-times hashes, set 1 on target positions
Initialized BF, check filename
k's posstions are all not 0
No
Filename not in the balcklist
Yes
Filename in the black list

To satisfy the need for 0.01 error-rate (Pfp) and 30000-item (n) in blacklist, we design as follows.

According to data, we can determine the size of Bloom-Filter's array, m, as $m = -\frac{n \ln P_{fp}}{(\ln 2)^2}$

Also, we don't hard-code k=5 here, instead we calculate it as $k = \frac{m}{n} ln2$, and the theoretical error rate is 0.01 as the following calculation.

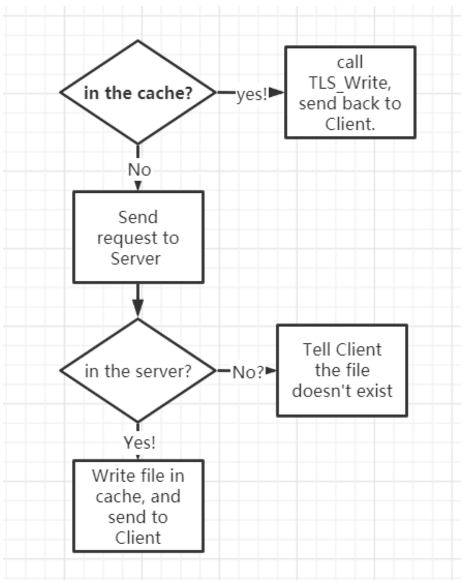


```python
"""
import math
n=30000
P=0.01
2
m=-n*math.log(P)/((math.log(2))*(math.log(2)))
print(m)
m=287552

k=m/n*(math.log(2))
print(k)

k=7

error=math.pow( (1-math.pow(math.e,-k*n/m)),k)
print(error)
#error=0.01
```

```
CS165')
287551.75132102316
6.643861935479112
0.010039176398524096

In [4]:
```

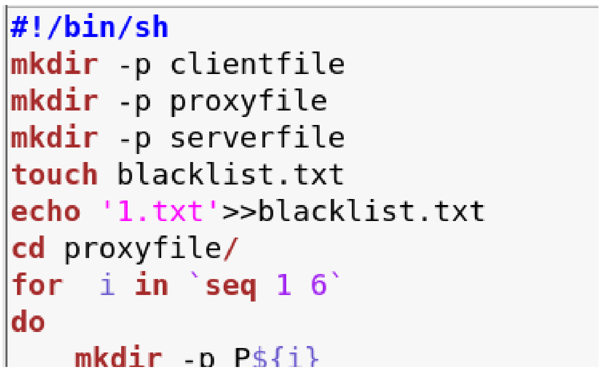
Proxy receives requests and separate the filename. Using Bloom Filter, proxy can check whether it's in blacklist or not. If the file is away from blacklist, Proxy will first find whether it's in cache:

Then do as the follows:


in the cache?
yes!
call TLS_Write, send back to Client.
No
Send request to Server
in the server?
No?
Tell Client the file doesn't exist
Yes!
Write file in cache, and send to Client

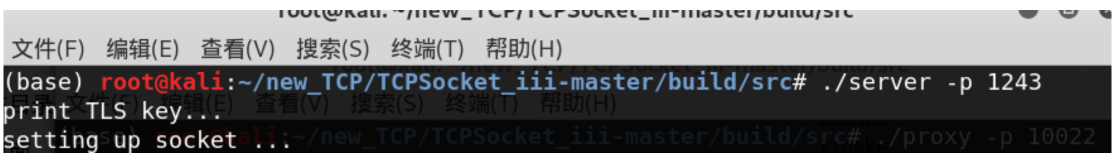
Result report:

Initialze: run 1.sh

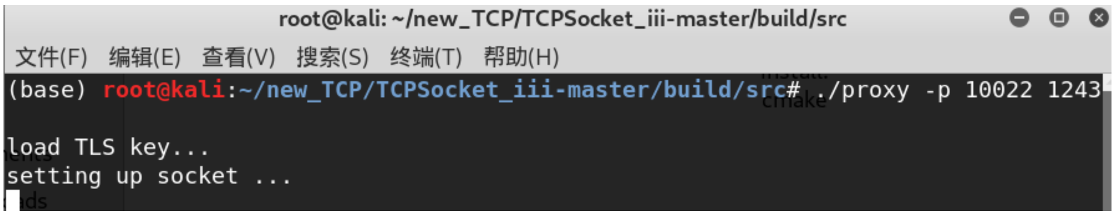


```
#!/bin/sh
mkdir -p clientfile
mkdir -p proxyfile
mkdir -p serverfile
touch blacklist.txt
echo '1.txt'>>blacklist.txt
cd proxyfile/
for  i in `seq 1 6`
do
    mkdir -p P${i}
```

Run Server and Proxy:


root@kali: ~/new_TCP/TCPSocket_iii-master/build/src
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

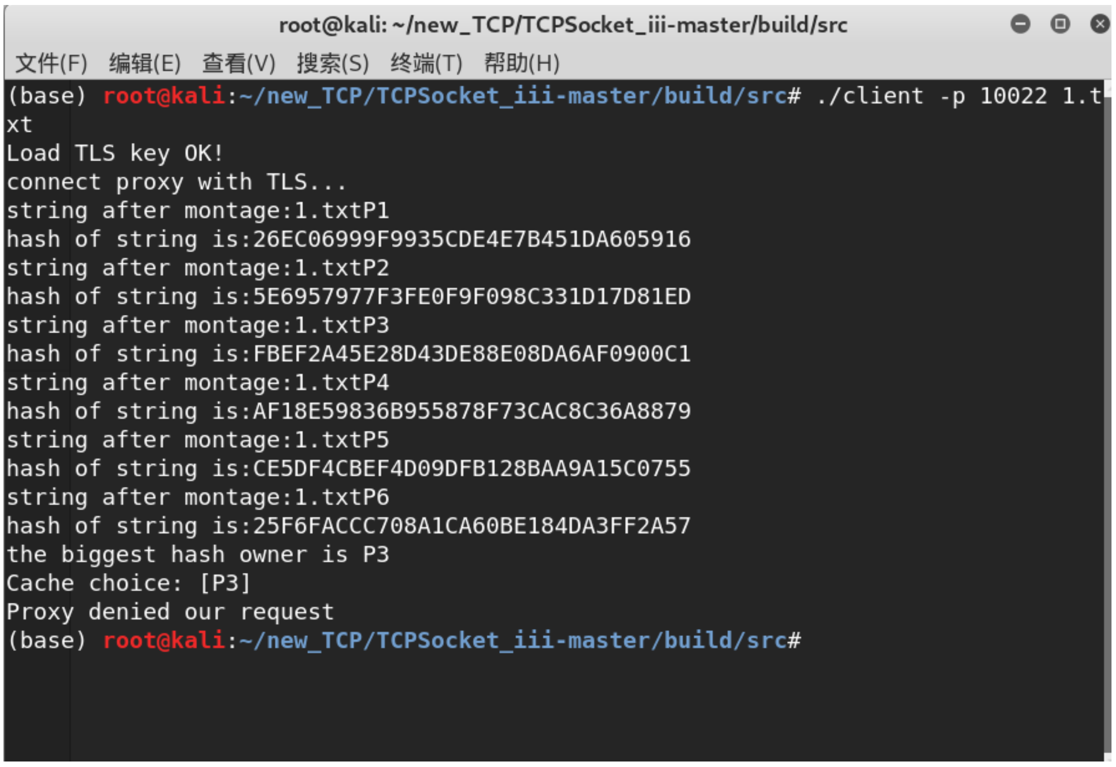
```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./server -p 1243
print TLS key...
setting up socket ...
```


root@kali: ~/new_TCP/TCPSocket_iii-master/build/src
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./proxy -p 10022 1243

load TLS key...
setting up socket ...
```

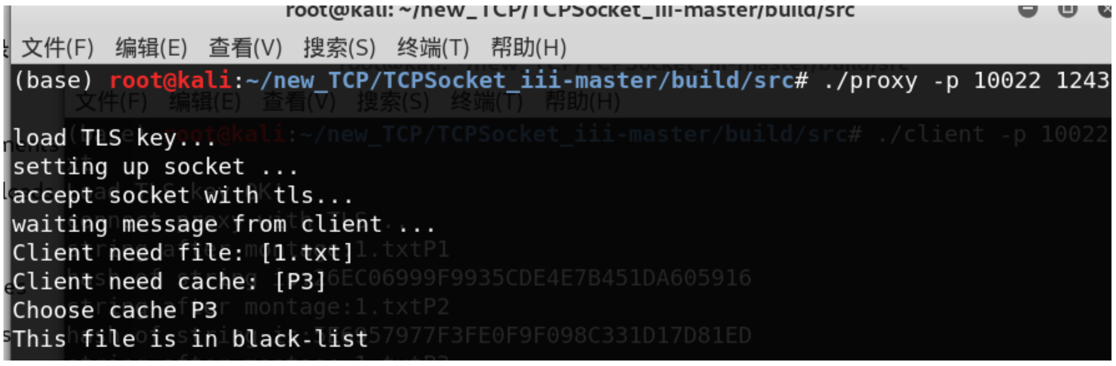
（1） A fail request for 1.txt is in blacklist.txt

Client:


root@kali: ~/new_TCP/TCPSocket_iii-master/build/src
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./client -p 10022 1.t
xt
Load TLS key OK!
connect proxy with TLS...
string after montage:1.txtP1
hash of string is:26EC06999F9935CDE4E7B451DA605916
string after montage:1.txtP2
hash of string is:5E6957977F3FE0F9F098C331D17D81ED
string after montage:1.txtP3
hash of string is:FBEF2A45E28D43DE88E08DA6AF0900C1
string after montage:1.txtP4
hash of string is:AF18E59836B955878F73CAC8C36A8879
string after montage:1.txtP5
hash of string is:CE5DF4CBEF4D09DFB128BAA9A15C0755
string after montage:1.txtP6
hash of string is:25F6FACCC708A1CA60BE184DA3FF2A57
the biggest hash owner is P3
Cache choice: [P3]
Proxy denied our request
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src#
```

Proxy:

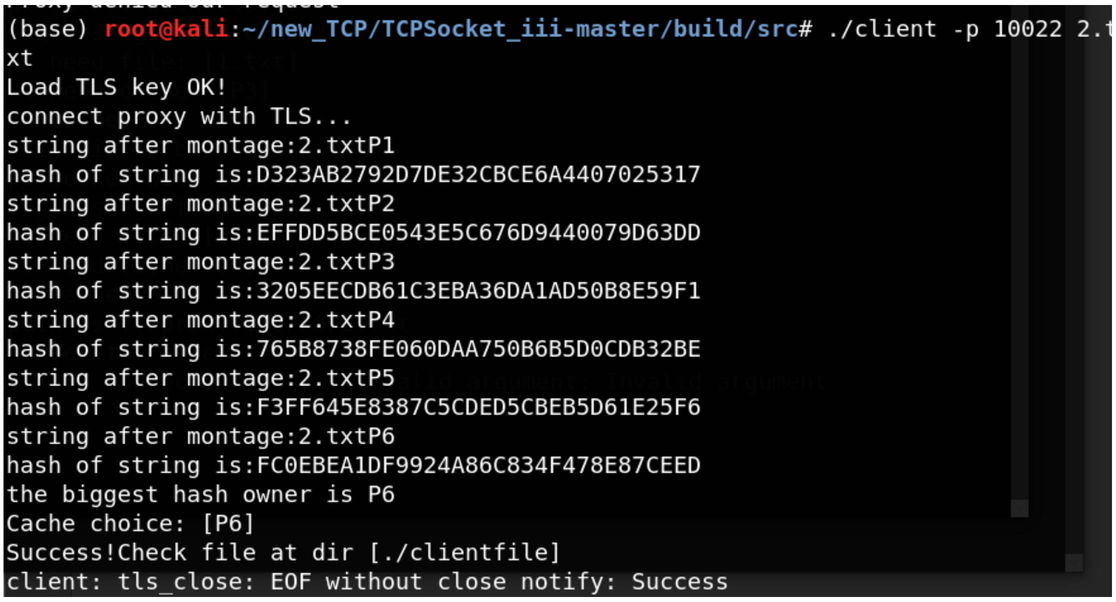

root@kali: ~/new_TCP/TCPSocket_iii-master/build/src
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./proxy -p 10022 1243

load TLS key...
setting up socket ...
accept socket with tls...
waiting message from client ...
Client need file: [1.txt]
Client need cache: [P3]
Choose cache P3
This file is in black-list
```

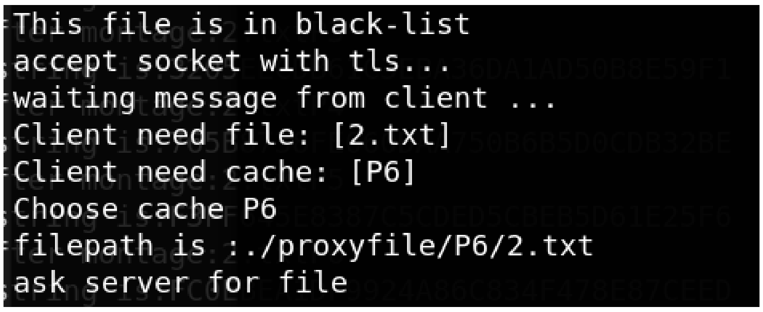
Server

(2) A successful request for 2.txt

Client:

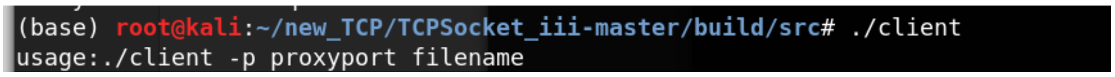

```
Proxy denied our request
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./client -p 10022 2.t
xt
Load TLS key OK!
connect proxy with TLS...
string after montage:2.txtP1
hash of string is:D323AB2792D7DE32CBCE6A4407025317
string after montage:2.txtP2
hash of string is:EFFDD5BCE0543E5C676D9440079D63DD
string after montage:2.txtP3
hash of string is:3205EECDB61C3EBA36DA1AD50B8E59F1
string after montage:2.txtP4
hash of string is:765B8738FE060DAA750B6B5D0CDB32BE
string after montage:2.txtP5
hash of string is:F3FF645E8387C5CDED5CBEB5D61E25F6
string after montage:2.txtP6
hash of string is:FC0EBEA1DF9924A86C834F478E87CEED
the biggest hash owner is P6
Cache choice: [P6]
Success!Check file at dir [./clientfile]
client: tls_close: EOF without close notify: Success
```

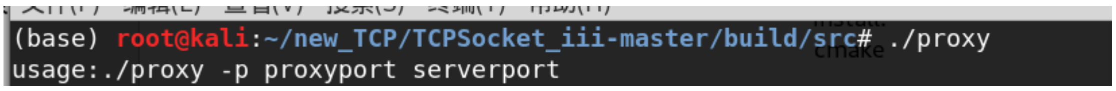
Proxy:

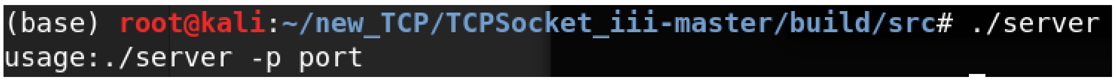

```
This file is in black-list
accept socket with tls...
waiting message from client ...
Client need file: [2.txt]
Client need cache: [P6]
Choose cache P6
filepath is :./proxyfile/P6/2.txt
ask server for file
```

And the 2.txt has been in the cache for Proxy as well as in Clientfile folder.

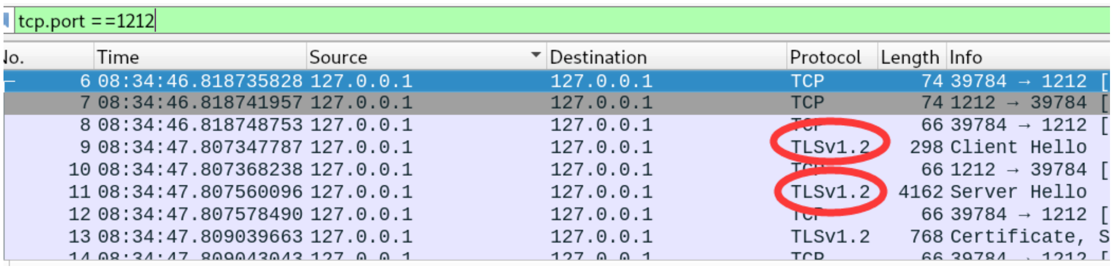
We also have a error check when you run wrong commands.

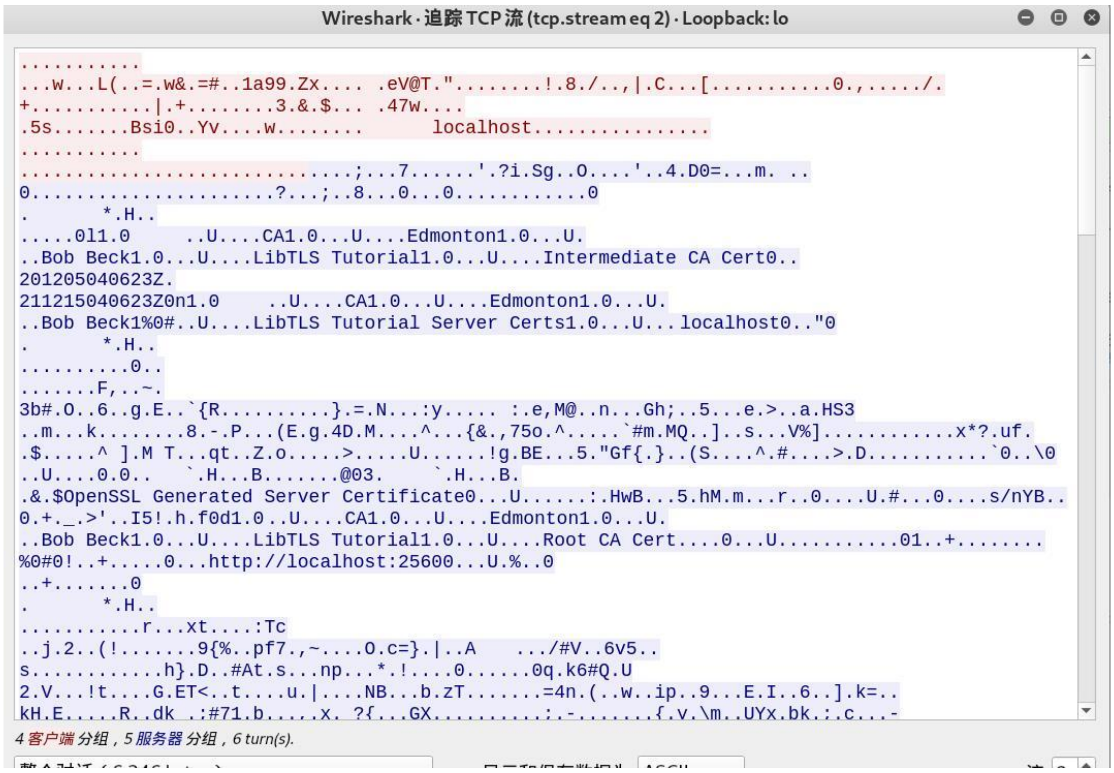

```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./client
usage:./client -p proxyport filename
```


```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./proxy
usage:./proxy -p proxyport serverport
```


```
(base) root@kali:~/new_TCP/TCPSocket_iii-master/build/src# ./server
usage:./server -p port
```

Use Wireshark to check TCP connect


```
tcp.port ==1212
No.  Time               Source     Destination  Protocol  Length Info
   6 08:34:46.818735828 127.0.0.1  127.0.0.1    TCP           74 39784 → 1212 [
   7 08:34:46.818741957 127.0.0.1  127.0.0.1    TCP           74 1212 → 39784 [
   8 08:34:46.818748753 127.0.0.1  127.0.0.1    TCP           66 39784 → 1212 [
   9 08:34:47.807347787 127.0.0.1  127.0.0.1    TLSv1.2      298 Client Hello
  10 08:34:47.807368238 127.0.0.1  127.0.0.1    TCP           66 1212 → 39784 [
  11 08:34:47.807560096 127.0.0.1  127.0.0.1    TLSv1.2     4162 Server Hello
  12 08:34:47.807578490 127.0.0.1  127.0.0.1    TCP           66 39784 → 1212 [
  13 08:34:47.809039663 127.0.0.1  127.0.0.1    TLSv1.2      768 Certificate, S
  14 08:34:47.809043043 127.0.0.1  127.0.0.1    TCP           66 39784 → 1212 [
```



Wireshark · 追踪 TCP 流 (tcp.stream eq 2) · Loopback: lo

```
...........
...w...L(..=.w&.=#..1a99.Zx.... .eV@T."........!.8./..,|.C...[...........0.,...../.
+...........|.+........3.&.$... .47w....
.5s.......Bsi0..Yv....w........      localhost...............
...........
...........................;...7......'.?i.Sg..O....'..4.D0=...m. ..
0.......................?...;..8...0...0...........0
.       *.H..
.....0l1.0     ..U....CA1.0...U....Edmonton1.0...U.
..Bob Beck1.0...U....LibTLS Tutorial1.0...U....Intermediate CA Cert0..
201205040623Z.
211215040623Z0n1.0     ..U....CA1.0...U....Edmonton1.0...U.
..Bob Beck1%0#..U....LibTLS Tutorial Server Certs1.0...U...localhost0.."0
.       *.H..
..........0..
.......F,..~.
3b#.O..6..g.E..`{R..........}.=.N...:y..... :.e,M@..n...Gh;..5...e.>..a.HS3
..m...k........8.-.P...(E.g.4D.M....^...{&.,75o.^.....`#m.MQ..]..s...V%]............x*?.uf.
.$.....^ ].M T...qt..Z.o.....>.....U......!g.BE...5."Gf{.}..(S....^.#....>.D...........`0..\0
..U....0.0..   `.H...B.......@03.     `.H...B.
.&.$OpenSSL Generated Server Certificate0...U......:.HwB...5.hM.m...r..0....U.#...0....s/nYB..
0.+._.>'..I5!.h.f0d1.0..U....CA1.0...U....Edmonton1.0...U.
..Bob Beck1.0...U....LibTLS Tutorial1.0...U....Root CA Cert....0...U...........01..+........
%0#0!..+.....0...http://localhost:25600...U.%..0
..+.......0
.       *.H..
...........r...xt....:Tc
..j.2..(!.......9{%..pf7.,~....O.c=}.|..A    .../#V..6v5..
s............h}.D..#At.s...np...*.!....0......0q.k6#Q.U
2.V...!t....G.ET<..t....u.|....NB...b.zT.......=4n.(..w..ip..9...E.I..6..].k=..
kH.E.....R..dk .:#71.b.....x. ?{...GX..........:.-.......{.v.\m..UYx.bk.;.c...-
```

4 客户端 分组，5 服务器 分组，6 turn(s).