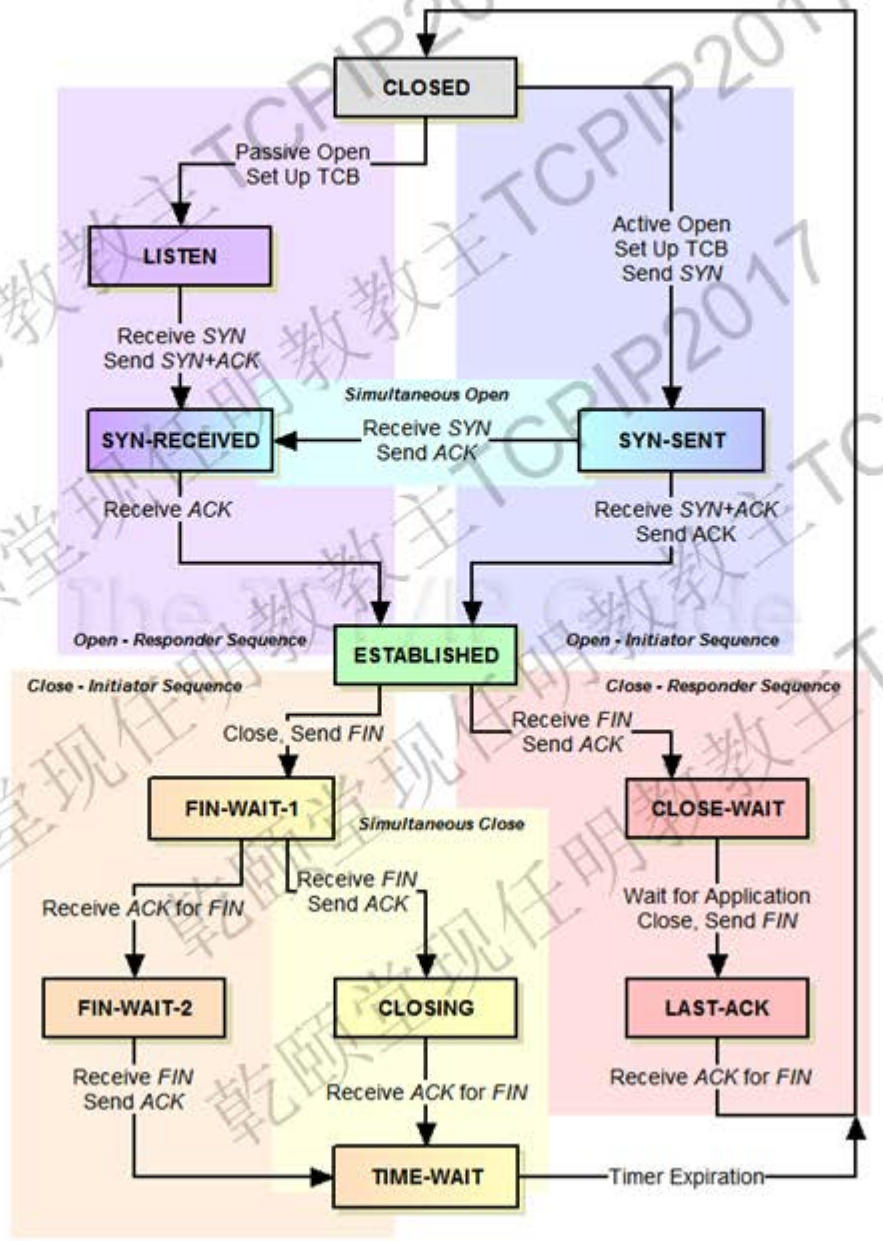
第五部分.6
TCP状态迁移

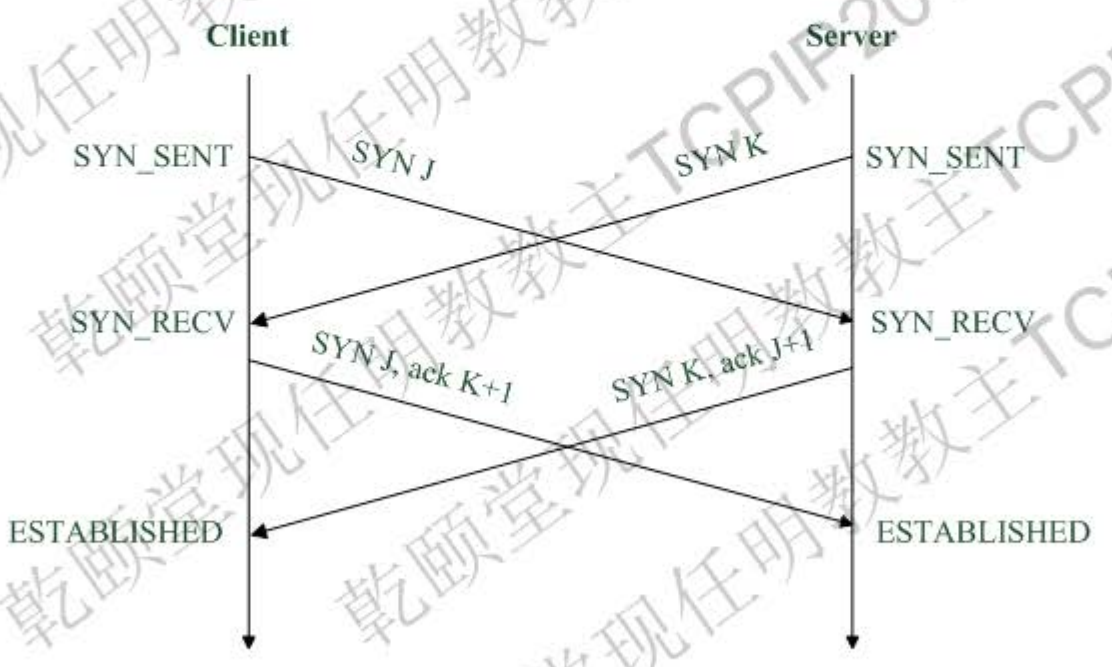
TCP状态迁移图



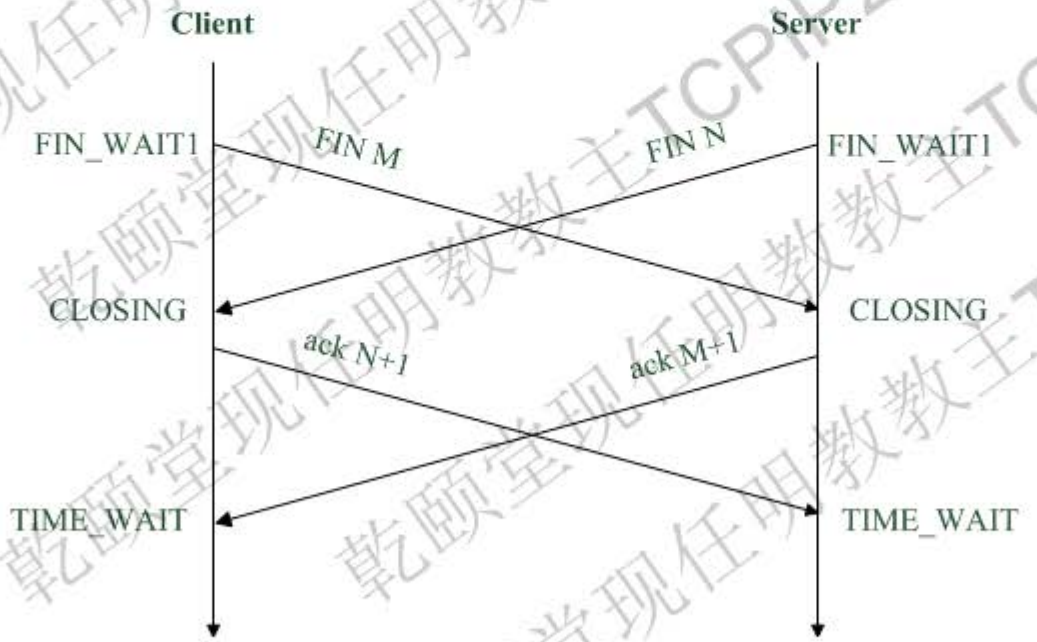
现任明教教主
乾颐堂TCP/IP 2017

正常序列

同时打开



同时关闭



Python模拟Client主动结束

Server

```
from socket import *
myHost = '172.16.12.101'
myPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.bind((myHost, myPort))
sockobj.listen(5)
connection, address = sockobj.accept()
```

Client

```
from socket import *
serverHost = '172.16.12.101'
serverPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.connect((serverHost, serverPort))
sockobj.close()
```

Server	Client
LISTEN	
ESTABLISHED	ESTABLISHED
CLOSE_WAIT	FIN_WAIT2
CLOSED	TIME_WAIT

connection.close()

使用“netstat -an | grep 6666”查看TCP状态

2MSL等待时间

- TIME_WAIT状态也称为2MSL等待状态。每个具体TCP实现必须选择一个报文段最大生存时间**MSL（Maximum Segment Lifetime）。它是任何报文段被丢弃前在网络内的最长时间。**
- RFC 793 [Postel 1981c] 指出MSL为2分钟。然而，实现中的常用值是30秒，1分钟，或2分钟**（与操作系统有关）**。
- 对一个具体实现所给定的MSL值，处理的原则是：**当TCP执行一个主动关闭，并发回最后一个ACK，该连接必须在TIME_WAIT状态停留的时间为2倍的MSL。**这样可**让TCP再次发送最后的ACK以防这个ACK丢失（另一端超时并重发最后的FIN）。**
- 这种2MSL等待的另一个结果是这个TCP连接在2MSL等待期间，**定义这个连接的插口（客户的IP地址和端口号，服务器的IP地址和端口号）不能再被使用。**这个连接只能在2MSL结束后才能再被使用。
- 遗憾的是，大多数TCP实现（如伯克利版）强加了**更为严格的限制。在2MSL等待期间，插口中使用的本地端口在默认情况下不能再被使用。**
- 在连接处于2MSL等待时，任何迟到的报文段将被丢弃。因为处于2MSL等待的、由该插口对(socket pair)定义的连接在这段时间内不能被再用，因此当要建立一个有效的连接时，来自该连接的一个较早替身（incarnation）的迟到报文段作为新连接的一部分不可能不被曲解

2MSL等待时间（续）

- 客户执行主动关闭并进入TIME_WAIT是正常的。服务器通常执行被动关闭，不会进入TIME_WAIT状态。这暗示如果我们终止一个客户程序，并立即重新启动这个客户程序，则这个新客户程序将不能重用相同的本地端口。这不会带来什么问题，因为客户使用本地端口，而并不关心这个端口号是什么。
- 对于服务器，情况就有所不同，因为服务器使用熟知端口。如果我们终止一个已经建立连接的服务器程序，并试图立即重新启动这个服务器程序，服务器程序将不能把它的这个熟知端口赋值给它的端点，因为那个端口是处于2MSL连接的一部分。

Python模拟Server主动结束

Server

```
from socket import *
myHost = '172.16.12.101'
myHost = '172.16.12.101'
myPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.bind((myHost, myPort))
sockobj.listen(5)
connection, address = sockobj.accept()
```

LISTEN

Client

使用“netstat -an | grep 6666”查看TCP状态

```
from socket import *
serverHost = '172.16.12.101'
serverPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.connect((serverHost, serverPort))
```

ESTABLISHED — ESTABLISHED

```
connection.close()
```

FIN_WAIT2 — CLOSE_WAIT

```
sockobj.close()
```

TIME_WAIT — CLOSED

```
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.bind((myHost, myPort))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 98] Address already in use
```

平静时间的概念

- 对于来自某个连接的较早替身的迟到报文段，2MSL等待可防止将它解释成使用相同插口对的新连接的一部分。但这只有在处于2MSL等待连接中的主机处于正常工作状态时才有效。
- 如果使用处于**2MSL**等待端口的主机出现故障，它会在**MSL**秒内重新启动，并立即使用故障前仍处于2MSL的插口对来建立一个新的连接吗？如果是这样，在故障前从这个连接发出而迟到的报文段会被错误地当作属于重启后新连接的报文段。无论如何选择重启后新连接的初始序号，都会发生这种情况。
- 为了防止这种情况，**RFC 793**指出**TCP**在重启动后的**MSL**秒内不能建立任何连接。这就称为平静时间**(quiet time)**。
- 只有极少的实现版遵守这一原则，因为大多数主机重启动的时间都比MSL秒要长。

FIN_WAIT_2状态

- 在FIN WAIT_ 2状态我们已经发出了FIN，并且另一端也已对它进行确认。除非我们在实行半关闭，否则将等待另一端的应用层意识到它已收到一个文件结束符说明，并向我们发一个FIN来关闭另一方向的连接。只有当另一端的进程完成这个关闭，我们这端才会从FIN_WAIT_2状态进入TIME_WAIT状态。
- 这意味着我们这端可能永远保持这个状态。另一端也将处于CLOSE_WAIT状态，并一直保持这个状态直到应用层决定进行关闭。

一般防火墙都有解决半闭连接（**FIN_WAIT_2**状态）的超时时间设置，如果超时，防火墙会向双方发送**RSET**（伪装源）来踢掉连接。

Python模拟FIN_WAIT_2

Server

```
from socket import *
myHost = '172.16.12.101'
myHost = '172.16.12.101'
myPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.bind((myHost, myPort))
sockobj.listen(5)
connection, address = sockobj.accept()
```

LISTEN

Client

```
from socket import *
serverHost = '172.16.12.101'
serverPort = 6666
sockobj = socket(AF_INET, SOCK_STREAM)
sockobj.connect((serverHost, serverPort))

sockobj.close()
```

ESTABLISHED — ESTABLISHED

CLOSE_WAIT — FIN_WAIT2

服务器不发送FIN，只能说
服务器代码有问题！