

网络传输机制实验

一、 实验内容

1.1 实现 TCP 协议栈，节点之间能够在 TCP 协议之上正确建立连接和传输数据。

1.2 完成应用程序的编写，完成节点之间传输短消息和文件的操作。

二、 实验实现

在网络传输机制实验中，我们需要实现最基本的 TCP 连接管理功能，使得节点之间能够在无丢包网络环境中建立和断开连接；实现最基本的 TCP 数据收发功能，使得节点之间能够在无丢包网络环境中传输数据。

2.1 TCP 协议分析

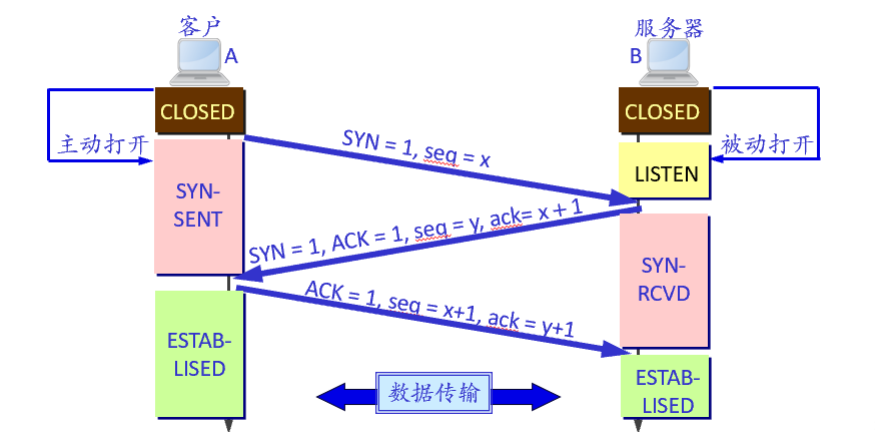
TCP 协议提供端到端的、面向连接的、可靠的、有序的字节流传输服务。TCP 协议实现点对点的双工通信，TCP 将应用程序交付下来的数据看作一连串无结构的有序的字节流，每条 TCP 连接是一对点对点字节流，每个方向一个字节流，TCP 两端设有发送缓存和接收缓存，存放双向通信的数据：发送时，应用进程把数据传入 TCP 的发送缓存，TCP 在合适时候将数据发出；接收时，TCP 把数据放入接收缓存，应用进程在合适时候读取数据。

2.1.1 TCP 建立连接

在 TCP 通信的过程中，通信双方先建立逻辑连接，再进行双向数据流传输，通信结束后撤销连接，同时利用滑动窗口机制实现可靠传输——对一个连接上传输的每个字节编号，通过接收确认和重传保证可靠传输；流量控制——防止发送方发送的数据超出接收方的接收能力；拥塞控制——防止过多数据注入网络造成网络结点或链路超载。

TCP 通过三次握手机制建立连接：

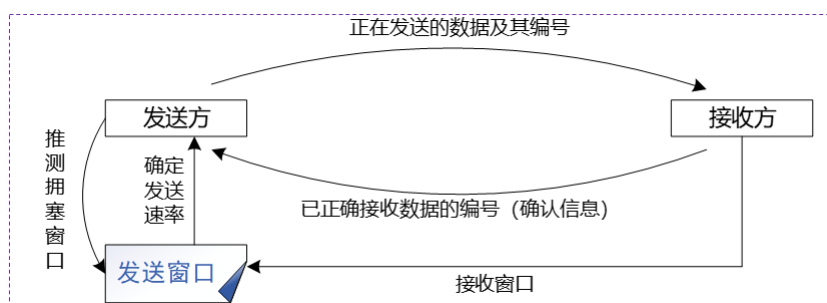
- (a) 最初，两端的 TCP 进程都处于 CLOSED 状态
- (b) 服务器 B 进程被动打开连接，进入 LISTEN 状态，等待客户进程的连接请求
- (c) TCP 客户进程 A 主动打开连接，创建传输控制块 TCB，向 B 发送连接请求报文段，进入 SYN-SENT 状态。
- (d) B 收到请求后，应答确认报文段，进入 SYN-RCVD 状态。
- (e) A 收到 B 的确认后，向 B 应答确认，进入 ESTABLISHED 状态。
- (f) B 收到 A 的确认后，也进入 ESTABLISHED 状态
- (g) 连接建立



2.1.2 TCP 数据传输

在 TCP 协议中，双方可以同时发送数据，有特定序列号值的数据，从发送方向接收方流动，发送数据的过程中每字节顺序编号，每个报文段中的序列号值指的是本报文段所发送的数据的第一个字节的序号。

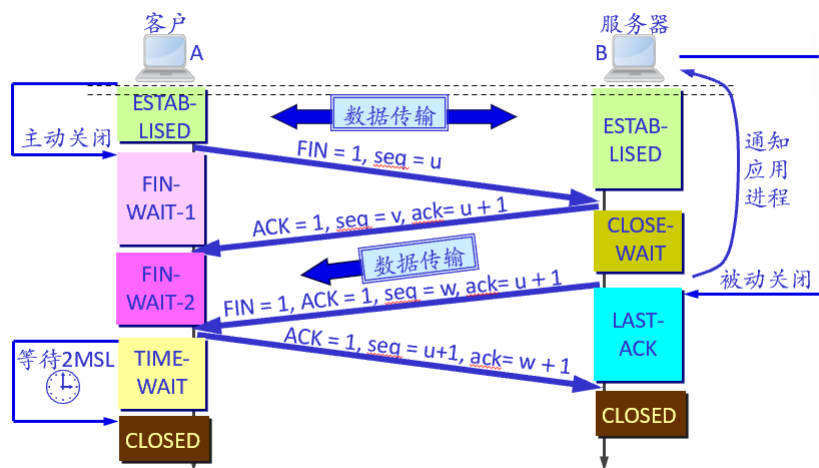
对数据的接收确认、接收窗口大小由接收方向发送方应答。



2.1.3 TCP 释放连接

TCP 连接的任何一方都可以通过发送 FIN 报文段主动关闭连接：

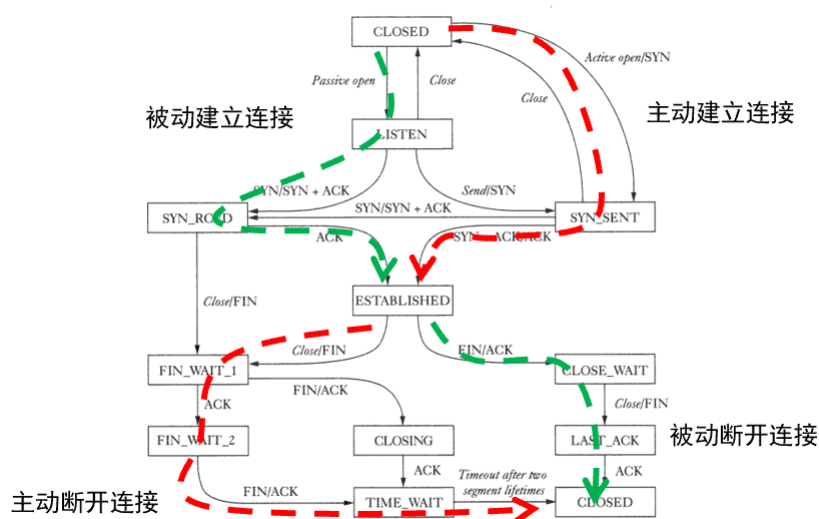
- A、B 都处于 ESTABLISHED 状态，A 向 B 发送释放报文段，进入 FIN-WAIT-1 状态
- B 收到 A 的连接释放报文段后，应答确认，进入 CLOSE-WAIT 状态
- A 收到 B 的确认后，进入 FIN-WAIT-2 状态，B 到 A 方向的连接未关闭，B 若发送数据，A 仍要接收
- 若 B 已没有向 A 发送的数据，其应用进程通知 TCP 释放连接
- B 向 A 发送连接释放报文段后，进入 LAST-ACK 状态
- A 收到 B 的连接释放报文段后，回复确认，进入 TIME-WAIT 状态
- B 收到 A 的确认后，撤销传输控制块 TCB，（等待超过 2MSL 后）进入 CLOSED 状态



2.2 tcp 协议栈实现

2.2.1 状态机实现

根据之前对 tcp 协议建立连接以及释放连接过程的分析，我们得到 TCP 状态机的状态转换图：



在本实验中我们实现虚线标注的部分，具体代码实现如下所示：

```
void tcp_process(struct tcp_sock *tsk, struct tcp_cb *cb, char *packet)
{
    if (cb->flags & TCP_RST) {
        tcp_sock_close(tsk);
        return;
    }
    if (cb->flags & TCP_ACK) {
        tsk->snd_una = cb->ack;
    }
    tsk->rcv_nxt = cb->seq_end;

    switch (tsk->state) {
```

```

case TCP_LISTEN:
    if (cb->flags == TCP_SYN) {
        struct tcp_sock *csk = alloc_child_tcp_sock(tsk, cb);
        tcp_send_control_packet(csk, TCP_SYN | TCP_ACK);
        tcp_set_state(csk, TCP_SYN_RECV);
    }
    break;
case TCP_SYN_SENT:
    if (cb->flags == (TCP_SYN | TCP_ACK)) {
        tcp_update_window_safe(tsk, cb);
        tcp_send_control_packet(tsk, TCP_ACK);
        tcp_set_state(tsk, TCP_ESTABLISHED);
        wake_up(tsk->wait_connect);
    }
    break;
case TCP_SYN_RECV:
    if (cb->flags == TCP_ACK) {
        tcp_sock_accept_enqueue(tsk);
        tcp_set_state(tsk, TCP_ESTABLISHED);
        wake_up(tsk->wait_accept);
    }
    break;
case TCP_ESTABLISHED:
    if (cb->flags & TCP_FIN) {
        tcp_send_control_packet(tsk, TCP_ACK);
        tcp_set_state(tsk, TCP_CLOSE_WAIT);
        if (tsk->wait_empty->sleep) {
            wake_up(tsk->wait_empty);
        }
    }
    if (cb->flags & TCP_ACK) {
        tcp_update_window_safe(tsk, cb);
        if (cb->pl_len != 0) {
            pthread_mutex_lock(&tsk->rcv_buf_lock);
            while (ring_buffer_free(tsk->rcv_buf) < cb->pl_len) {
                pthread_mutex_unlock(&tsk->rcv_buf_lock);
                sleep_on(tsk->wait_full);
                pthread_mutex_lock(&tsk->rcv_buf_lock);
            }

            write_ring_buffer(tsk->rcv_buf, cb->payload,
cb->pl_len);

            tsk->rcv_wnd = ring_buffer_free(tsk->rcv_buf);

```

```

        pthread_mutex_unlock(&tsk->rcv_buf_lock);

        if (tsk->wait_empty->sleep) {
            wake_up(tsk->wait_empty);
        }
        tcp_send_control_packet(tsk, TCP_ACK);
    }
}
break;
case TCP_FIN_WAIT_1:
    if (cb->flags == TCP_ACK) {
        tcp_set_state(tsk, TCP_FIN_WAIT_2);
    } else if (cb->flags == (TCP_ACK | TCP_FIN)) {
        tcp_send_control_packet(tsk, TCP_ACK);
        tcp_set_state(tsk, TCP_TIME_WAIT);
        tcp_set_timewait_timer(tsk);
    }
    break;
case TCP_FIN_WAIT_2:
    if (cb->flags & TCP_FIN) {
        tcp_send_control_packet(tsk, TCP_ACK);
        tcp_set_state(tsk, TCP_TIME_WAIT);
        tcp_set_timewait_timer(tsk);
    }
    break;
case TCP_LAST_ACK:
    if (cb->flags & TCP_ACK) {
        tcp_unhash(tsk);
        tcp_set_state(tsk, TCP_CLOSED);
    }
    break;
default:
    break;
}
}
}

```

2.2.2 tcp_sock 管理函数实现

free_tcp_sock 函数释放 tcp_sock 所占用的所有资源，为了确保协议栈的安全运行，每次 socket 被使用时，ref_cnt+1，每次调用 free_tcp_sock 函数时，ref_cnt-1，当 ref_cnt 减至 0 时，释放所有的资源。

```

void free_tcp_sock(struct tcp_sock *tsk)
{
    tsk->ref_cnt -= 1;
    if (tsk->ref_cnt <= 0) {

```

```

        free_wait_struct(tsk->wait_accept);
        free_wait_struct(tsk->wait_connect);
        free_wait_struct(tsk->wait_empty);
        free_wait_struct(tsk->wait_full);
        free_wait_struct(tsk->wait_send);
        free_ring_buffer(tsk->rcv_buf);
        free(tsk);
    }
}

```

tcp_sock_lookup_established 函数利用四元组信息 (saddr, daddr, sport, dport) 在 established_table 中查找 tcp_sock, 如果找到了, 则返回相应的 socket, 如果没有, 则返回 NULL。

具体代码实现如下所示:

```

struct tcp_sock *tcp_sock_lookup_established(u32 saddr, u32 daddr, u16
sport, u16 dport)
{
    int idx = tcp_hash_function(saddr, daddr, sport, dport);
    struct tcp_sock *ptr;
    list_for_each_entry (ptr, &tcp_established_sock_table[idx],
hash_list) {
        if (saddr == ptr->sk_sip && sport == ptr->sk_sport &&
            daddr == ptr->sk_dip && dport == ptr->sk_dport) {
            return ptr;
        }
    }
    return NULL;
}

```

tcp_sock_lookup_listen 函数的实现与 tcp_sock_lookup_established 函数类似, 同样利用四元组信息在 listen_table 中查找 socket, 具体实现代码如下所示:

```

struct tcp_sock *tcp_sock_lookup_listen(u32 saddr, u16 sport)
{
    int idx = tcp_hash_function(0, 0, sport, 0);
    struct tcp_sock *ptr;
    list_for_each_entry (ptr, &tcp_listen_sock_table[idx], hash_list) {
        if (sport == ptr->sk_sport) {
            return ptr;
        }
    }
    return NULL;
}

```

tcp_sock_connect 函数主动发起连接，其首先确定四元组信息，分配源端口，然后发送 SYN 数据包，转到 TCP_SYN_SENT 状态，将 socket 加入到哈希表中，唤醒 wait_connect 进程，等待客户端响应。

具体代码实现如下所示：

```
int tcp_sock_connect(struct tcp_sock *tsk, struct sock_addr *skaddr)
{
    tsk->sk_dip = ntohl(skaddr->ip);
    tsk->sk_dport = ntohs(skaddr->port);
    tsk->sk_sip = longest_prefix_match(ntohl(skaddr->ip))->iface->ip;
    if (tcp_sock_set_sport(tsk, 0) < 0) {
        log(ERROR, "No available port");
        return -1;
    }
    tcp_bind_hash(tsk);
    tcp_send_control_packet(tsk, TCP_SYN);
    tcp_set_state(tsk, TCP_SYN_SENT);
    tcp_hash(tsk);
    sleep_on(tsk->wait_connect);
    return tsk->sk_sport;
}
```

tcp_sock_accept 函数首先判断接收队列是否为空，如果为空，唤醒 wait_connect 进程，等待接收接下来的连接请求，否则弹出第一个 tcp_sock 并且接收它，这里调用 tcp_sock_accept_dequeue 函数完成该操作，具体实现代码如下所示：

```
struct tcp_sock *tcp_sock_accept(struct tcp_sock *tsk)
{
    while (list_empty(&tsk->accept_queue)) {
        sleep_on(tsk->wait_accept);
    }

    return tcp_sock_accept_dequeue(tsk);
}
```

tcp_sock_close 函数完成主动断开连接处理和被动断开连接处理，当此时为 TCP_CLOSE_WAIT 状态时，为被动断开连接，发送 FIN&ACK 数据包，将状态设置为 TCP_LAST_ACK，等待连接关闭；当此时是 TCP_CLOSE_WAIT 状态时，主动断开连接，发送 FIN&ACK 数据包，将状态设置为 TCP_FIN_WAIT_1，等待对方的确认信号，具体代码实现如下所示：

```
void tcp_sock_close(struct tcp_sock *tsk)
{
    if (tsk->state == TCP_CLOSE_WAIT) {
```

```

        tcp_send_control_packet(tsk, TCP_FIN | TCP_ACK);
        tcp_set_state(tsk, TCP_LAST_ACK);
    } else if (tsk->state == TCP_ESTABLISHED) {
        tcp_send_control_packet(tsk, TCP_FIN | TCP_ACK);
        tcp_set_state(tsk, TCP_FIN_WAIT_1);
    } else {
        tcp_unhash(tsk);
        tcp_set_state(tsk, TCP_CLOSED);
    }
}

```

2.2.3 TCP 协议栈数据收发函数实现

tcp_sock_read 函数将环形缓存中的数据读取到应用程序中，返回正在读取的数据长度，具体实现代码如下所示：

```

int tcp_sock_read(struct tcp_sock *tsk, char *buf, int len) {
    pthread_mutex_lock(&tsk->rcv_buf_lock);
    while (ring_buffer_empty(tsk->rcv_buf)) {
        pthread_mutex_unlock(&tsk->rcv_buf_lock);
        if (tsk->state == TCP_CLOSE_WAIT) {
            return 0;
        }
        sleep_on(tsk->wait_empty);
        if (tsk->state == TCP_CLOSE_WAIT) {
            if (!ring_buffer_empty(tsk->rcv_buf)) {
                pthread_mutex_lock(&tsk->rcv_buf_lock);
            } else {
                return 0;
            }
        } else {
            pthread_mutex_lock(&tsk->rcv_buf_lock);
        }
    }
    int rcv_len = read_ring_buffer(tsk->rcv_buf, buf, len);
    tsk->rcv_wnd = ring_buffer_free(tsk->rcv_buf);
    if (ring_buffer_free(tsk->rcv_buf) > 0 && tsk->wait_full->sleep) {
        wake_up(tsk->wait_full);
    }
    pthread_mutex_unlock(&tsk->rcv_buf_lock);
    return rcv_len;
}

```

tcp_sock_write 函数将应用程序中的数据组成数据包发送出去，返回发送数据的长度，具体实现代码如下所示：


```

int tcp_sock_write(struct tcp_sock *tsk, char *buf, int len) {
    int len_write = len;
    while (len > 0) {
        int len_sent = tcp_send_data(tsk, buf, len);
        buf += len_sent;
        len -= len_sent;
    }
    return len_write;
}

int tcp_send_data(struct tcp_sock *tsk, char *buf, int len) {
    len = min(len, ETH_FRAME_LEN - ETHER_HDR_SIZE - IP_BASE_HDR_SIZE -
TCP_BASE_HDR_SIZE);
    int pkt_len = ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE
+ len;
    char *pkt = malloc(pkt_len);
    memcpy(pkt + ETHER_HDR_SIZE + IP_BASE_HDR_SIZE + TCP_BASE_HDR_SIZE,
buf, len);
    while (tsk->snd_wnd < len) {
        tsk->snd_wnd = 0;
        sleep_on(tsk->wait_send);
    }
    tcp_send_packet(tsk, pkt, pkt_len);
    return len;
}

```

2.2.4 TCP 定时器实现

tcp_scan_timer_list 函数扫描定时器列表，查找每个存在时间超过 2MSL 的 tcp_sock 并且释放它，具体实现代码如下所示：

```

void tcp_scan_timer_list()
{
    struct tcp_timer *p, *q;
    list_for_each_entry_safe (p, q, &timer_list, list) {
        p->timeout += TCP_TIMER_SCAN_INTERVAL;
        if (p->enable == 1 && p->timeout >= TCP_TIMEWAIT_TIMEOUT) {
            list_delete_entry(&p->list);
            struct tcp_sock *tsk = timewait_to_tcp_sock(p);
            tcp_set_state(tsk, TCP_CLOSED);
            tcp_bind_unhash(tsk);
            tcp_unhash(tsk);
        }
    }
}

```

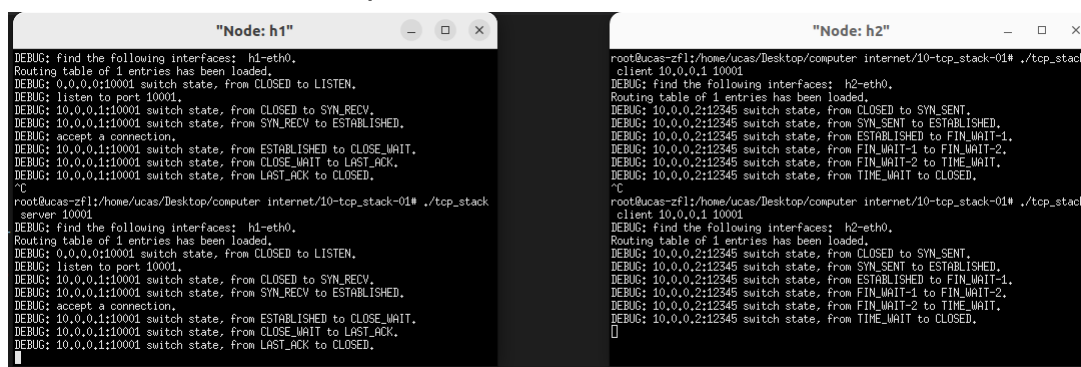
tcp_set_timewait_timer 函数设置 socket 定时器，并将其添加到定时器列表中，具体代码实现如下所示：

```
void tcp_set_timewait_timer(struct tcp_sock *tsk)
{
    tsk->timewait.enable = 1;
    tsk->timewait.type = 0;
    tsk->timewait.timeout = 0;
    list_add_tail(&tsk->timewait.list, &timer_list);
}
```

三、实验结果

3.1 连接管理

3.1.1 在节点 h1 h2 上执行 tcp_stack



```
"Node: h1"
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
^C
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.

"Node: h2"
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
^C
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
[]
```

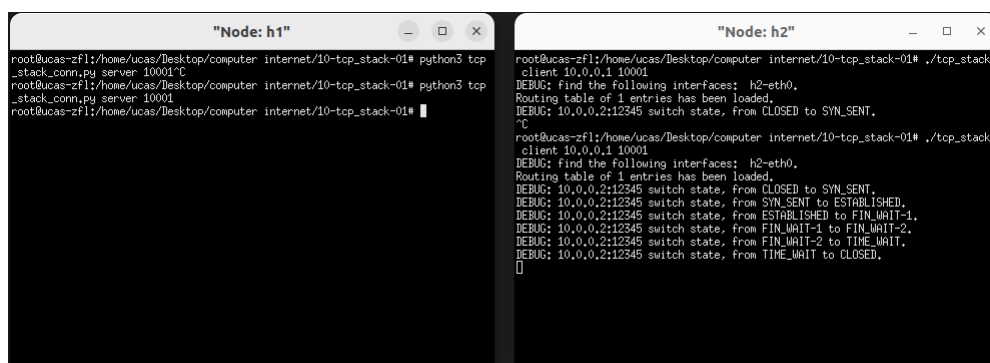
如上图所示，连接成功建立之后成功断开。

wireshark 抓包结果：

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	9e:ab:5f:14:0e:0e	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
2	0.010129396	f2:04:39:8b:bc:96	9e:ab:5f:14:0e:0e	ARP	42	10.0.0.1 is at f2:04:39:8b:bc:96
3	0.010149282	f2:04:39:8b:bc:96	9e:ab:5f:14:0e:0e	ARP	42	10.0.0.1 is at f2:04:39:8b:bc:96
4	0.020377608	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [SYN] Seq=0 Win=65535 Len=0
5	0.030593069	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
6	0.040919112	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=1 Ack=1 Win=65535 Len=0
7	1.040978361	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [FIN, ACK] Seq=1 Ack=1 Win=65535 Len=0
8	1.051078663	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [ACK] Seq=1 Ack=2 Win=65535 Len=0
9	5.051440642	10.0.0.1	10.0.0.2	TCP	54	10001 → 12345 [FIN, ACK] Seq=1 Ack=2 Win=65535 Len=0
10	5.061823810	10.0.0.2	10.0.0.1	TCP	54	12345 → 10001 [ACK] Seq=2 Ack=2 Win=65535 Len=0

通过 wireshark 我们看到发送数据包的过程符合我们之前分析的连接建立与释放的过程。

3.1.2 利用 tcp_stack_conn.py 替换服务端



```
"Node: h1"
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# python3 tcp_stack_conn.py server 10001^C
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# python3 tcp_stack_conn.py server 10001
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01#

"Node: h2"
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
^C
root@ucas-zf1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
[]
```

客户端工作正常。

3.1.3 利用 tcp_stack_conn.py 替换客户端

"Node: h1"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# ./tcp_stack
server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

"Node: h2"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# python3 tcp_stack_conn.py client 10.0.0.1 10001
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01#
```

服务端工作正常。

3.2 短消息收发

3.2.1 在 h1 h2 节点上执行 tcp_stack

"Node: h1"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# ./tcp_stack
server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

"Node: h2"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# ./client 10.0.0.1 10001
bash: ./client: No such file or directory
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyz
server echoes: 123456789abcdefghijklmnopqrstuvwxyz
server echoes: 23456789abcdefghijklmnopqrstuvwxyz
server echoes: 3456789abcdefghijklmnopqrstuvwxyz
server echoes: 456789abcdefghijklmnopqrstuvwxyz
server echoes: 56789abcdefghijklmnopqrstuvwxyz
server echoes: 6789abcdefghijklmnopqrstuvwxyz
server echoes: 789abcdefghijklmnopqrstuvwxyz
server echoes: 89abcdefghijklmnopqrstuvwxyz
server echoes: 9abcdefghijklmnopqrstuvwxyz
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

从图中我们看到，客户端和服务端之前成功完成了消息的收发。

3.2.2 利用 tcp_stack_trans.py 替换客户端

"Node: h2"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# python3 tcp_stack_trans.py client 10.0.0.1 10001
server echoes: 0123456789abcdefghijklmnopqrstuvwxyz
server echoes: 123456789abcdefghijklmnopqrstuvwxyz
server echoes: 23456789abcdefghijklmnopqrstuvwxyz
server echoes: 3456789abcdefghijklmnopqrstuvwxyz
server echoes: 456789abcdefghijklmnopqrstuvwxyz
server echoes: 56789abcdefghijklmnopqrstuvwxyz
server echoes: 6789abcdefghijklmnopqrstuvwxyz
server echoes: 789abcdefghijklmnopqrstuvwxyz
server echoes: 89abcdefghijklmnopqrstuvwxyz
server echoes: 9abcdefghijklmnopqrstuvwxyz
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01#
```

"Node: h1"

```
root@ucas-zf1:/home/ucas/Desktop/computer_internet/10-tcp_stack-01# ./tcp_stack
server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 0.0.0.0:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
```

服务端工作正常。

3.2.3 利用 tcp_stack_trans.py 替换服务端

```
"Node: h1"
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# python3 tcp_stack_trans.py server 10001
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# []

"Node: h2"
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
server echoes: 0123456789abcdefghijklmnopqrstuvwxyz
server echoes: 123456789abcdefghijklmnopqrstuvwxyz
server echoes: 23456789abcdefghijklmnopqrstuvwxyz
server echoes: 3456789abcdefghijklmnopqrstuvwxyz
server echoes: 456789abcdefghijklmnopqrstuvwxyz
server echoes: 56789abcdefghijklmnopqrstuvwxyz
server echoes: 6789abcdefghijklmnopqrstuvwxyz
server echoes: 789abcdefghijklmnopqrstuvwxyz
server echoes: 89abcdefghijklmnopqrstuvwxyz
server echoes: 9abcdefghijklmnopqrstuvwxyz
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

服务端工作正常。

3.3 大文件传送

3.3.1 h1 h2 节点上均执行 tcp_stack

```
"Node: h1"
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
server 10001
DEBUG: find the following interfaces: h1-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to LISTEN.
DEBUG: listen to port 10001.
DEBUG: going to accept a connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSED to SYN_RECV.
DEBUG: 10.0.0.1:10001 switch state, from SYN_RECV to ESTABLISHED.
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.
[]

"Node: h2"
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-01# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

节点之间完成了大文件的传送。

```
ucas@ucas-zfl1:~/Desktop/computer internet/10-tcp_stack-01$ md5sum client-input.dat server-output.dat
fd4c2a9395472c4dada2f8466f6242ef client-input.dat
fd4c2a9395472c4dada2f8466f6242ef server-output.dat
```

利用 md5sum 比较两个文件，两个文件完全相同。

3.3.2 利用 tcp_stack_trans.py 替换服务端

```
"Node: h1"
DEBUG: read data from socket, length is 460
DEBUG: read data from socket, length is 1000
DEBUG: read data from socket, length is 460
DEBUG: read data from socket, length is 1000
DEBUG: read data from socket, length is 620
DEBUG: read data from socket, length is 1000
DEBUG: read data from socket, length is 460
DEBUG: read data from socket, length is 1000
DEBUG: read data from socket, length is 460
DEBUG: read data from socket, length is 1000
DEBUG: read data from socket, length is 172
DEBUG: 10.0.0.1:10001 switch state, from ESTABLISHED to CLOSE_WAIT.
DEBUG: read data from socket, length is 0
DEBUG: tcp_sock_read return 0, finish transmission.
DEBUG: close this connection.
DEBUG: 10.0.0.1:10001 switch state, from CLOSE_WAIT to LAST_ACK.
DEBUG: 10.0.0.1:10001 switch state, from LAST_ACK to CLOSED.

"Node: h2"
root@ucas-zfl1:/home/ucas/Desktop/computer internet/10-tcp_stack-02# ./tcp_stack
client 10.0.0.1 10001
DEBUG: find the following interfaces: h2-eth0.
Routing table of 1 entries has been loaded.
DEBUG: 10.0.0.2:12345 switch state, from CLOSED to SYN_SENT.
DEBUG: 10.0.0.2:12345 switch state, from SYN_SENT to ESTABLISHED.
DEBUG: 10.0.0.2:12345 switch state, from ESTABLISHED to FIN_WAIT-1.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-1 to FIN_WAIT-2.
DEBUG: 10.0.0.2:12345 switch state, from FIN_WAIT-2 to TIME_WAIT.
DEBUG: 10.0.0.2:12345 switch state, from TIME_WAIT to CLOSED.
```

客户端工作正常。

```
ucas@ucas-zfl1:~/Desktop/computer internet/10-tcp_stack-02$ md5sum client-input.dat server-output.dat
fd4c2a9395472c4dada2f8466f6242ef client-input.dat
fd4c2a9395472c4dada2f8466f6242ef server-output.dat
```

两个文件完全相同。