**TCP内核代码**

**tcp内核结构体**

tcp控制块（每个tcp连接，64字节）

|  |
| --- |
| C struct tcpcb {  /\* Cache line 1 \*/  struct inpcb \*t\_inpcb; /\* back pointer to internet pcb \*/  struct tcp\_function\_block \*t\_fb;/\* TCP function call block \*/  void \*t\_fb\_ptr; /\* Pointer to t\_fb specific data \*/  uint32\_t t\_maxseg:24, /\* maximum segment size \*/  t\_logstate:8; /\* State of "black box" logging \*/  uint32\_t t\_port:16, /\* Tunneling (over udp) port \*/  t\_state:4, /\* state of this connection \*/  t\_idle\_reduce : 1,  t\_delayed\_ack: 7, /\* Delayed ack variable \*/  t\_fin\_is\_rst: 1, /\* Are fin's treated as resets \*/  t\_log\_state\_set: 1,  bits\_spare : 2;  u\_int t\_flags;  tcp\_seq snd\_una; /\* sent but unacknowledged \*/  tcp\_seq snd\_max; /\* highest sequence number sent;  \* used to recognize retransmits  \*/  tcp\_seq snd\_nxt; /\* send next \*/  tcp\_seq snd\_up; /\* send urgent pointer \*/  uint32\_t snd\_wnd; /\* send window \*/  uint32\_t snd\_cwnd; /\* congestion-controlled window \*/  uint32\_t t\_peakrate\_thr; /\* pre-calculated peak rate threshold \*/  /\* Cache line 2 \*/  u\_int32\_t ts\_offset; /\* our timestamp offset \*/  u\_int32\_t rfbuf\_ts; /\* recv buffer autoscaling timestamp \*/  int rcv\_numsacks; /\* # distinct sack blks present \*/  u\_int t\_tsomax; /\* TSO total burst length limit in bytes \*/  u\_int t\_tsomaxsegcount; /\* TSO maximum segment count \*/  u\_int t\_tsomaxsegsize; /\* TSO maximum segment size in bytes \*/  tcp\_seq rcv\_nxt; /\* receive next \*/  tcp\_seq rcv\_adv; /\* advertised window \*/  uint32\_t rcv\_wnd; /\* receive window \*/  u\_int t\_flags2; /\* More tcpcb flags storage \*/  int t\_srtt; /\* smoothed round-trip time \*/  int t\_rttvar; /\* variance in round-trip time \*/  u\_int32\_t ts\_recent; /\* timestamp echo data \*/  u\_char snd\_scale; /\* window scaling for send window \*/  u\_char rcv\_scale; /\* window scaling for recv window \*/  u\_char snd\_limited; /\* segments limited transmitted \*/  u\_char request\_r\_scale; /\* pending window scaling \*/  tcp\_seq last\_ack\_sent;  u\_int t\_rcvtime; /\* inactivity time \*/  /\* Cache line 3 \*/  tcp\_seq rcv\_up; /\* receive urgent pointer \*/  int t\_segqlen; /\* segment reassembly queue length \*/  uint32\_t t\_segqmbuflen; /\* Count of bytes mbufs on all entries \*/  struct tsegqe\_head t\_segq; /\* segment reassembly queue \*/  struct mbuf \*t\_in\_pkt;  struct mbuf \*t\_tail\_pkt;  struct tcp\_timer \*t\_timers; /\* All the TCP timers in one struct \*/  struct vnet \*t\_vnet; /\* back pointer to parent vnet \*/  uint32\_t snd\_ssthresh; /\* snd\_cwnd size threshold for  \* for slow start exponential to  \* linear switch  \*/  tcp\_seq snd\_wl1; /\* window update seg seq number \*/  /\* Cache line 4 \*/  tcp\_seq snd\_wl2; /\* window update seg ack number \*/   tcp\_seq irs; /\* initial receive sequence number \*/  tcp\_seq iss; /\* initial send sequence number \*/  u\_int t\_acktime; /\* RACK and BBR incoming new data was acked \*/  u\_int t\_sndtime; /\* time last data was sent \*/  u\_int ts\_recent\_age; /\* when last updated \*/  tcp\_seq snd\_recover; /\* for use in NewReno Fast Recovery \*/  uint16\_t cl4\_spare; /\* Spare to adjust CL 4 \*/  char t\_oobflags; /\* have some \*/  char t\_iobc; /\* input character \*/  int t\_rxtcur; /\* current retransmit value (ticks) \*/   int t\_rxtshift; /\* log(2) of rexmt exp. backoff \*/  u\_int t\_rtttime; /\* RTT measurement start time \*/   tcp\_seq t\_rtseq; /\* sequence number being timed \*/  u\_int t\_starttime; /\* time connection was established \*/  u\_int t\_fbyte\_in; /\* ticks time when first byte queued in \*/  u\_int t\_fbyte\_out; /\* ticks time when first byte queued out \*/   u\_int t\_pmtud\_saved\_maxseg; /\* pre-blackhole MSS \*/  int t\_blackhole\_enter; /\* when to enter blackhole detection \*/  int t\_blackhole\_exit; /\* when to exit blackhole detection \*/  u\_int t\_rttmin; /\* minimum rtt allowed \*/   u\_int t\_rttbest; /\* best rtt we've seen \*/   int t\_softerror; /\* possible error not yet reported \*/  uint32\_t max\_sndwnd; /\* largest window peer has offered \*/  /\* Cache line 5 \*/  uint32\_t snd\_cwnd\_prev; /\* cwnd prior to retransmit \*/  uint32\_t snd\_ssthresh\_prev; /\* ssthresh prior to retransmit \*/  tcp\_seq snd\_recover\_prev; /\* snd\_recover prior to retransmit \*/  int t\_sndzerowin; /\* zero-window updates sent \*/  u\_long t\_rttupdated; /\* number of times rtt sampled \*/  int snd\_numholes; /\* number of holes seen by sender \*/  u\_int t\_badrxtwin; /\* window for retransmit recovery \*/  TAILQ\_HEAD(sackhole\_head, sackhole) snd\_holes;  /\* SACK scoreboard (sorted) \*/  tcp\_seq snd\_fack; /\* last seq number(+1) sack'd by rcv'r\*/  struct sackblk sackblks[MAX\_SACK\_BLKS]; /\* seq nos. of sack blocks \*/  struct sackhint sackhint; /\* SACK scoreboard hint \*/  int t\_rttlow; /\* smallest observerved RTT \*/  int rfbuf\_cnt; /\* recv buffer autoscaling byte count \*/  struct toedev \*tod; /\* toedev handling this connection \*/  int t\_sndrexmitpack; /\* retransmit packets sent \*/  int t\_rcvoopack; /\* out-of-order packets received \*/  void \*t\_toe; /\* TOE pcb pointer \*/  struct cc\_algo \*cc\_algo; /\* congestion control algorithm \*/  struct cc\_var \*ccv; /\* congestion control specific vars \*/  struct osd \*osd; /\* storage for Khelp module data \*/  int t\_bytes\_acked; /\* # bytes acked during current RTT \*/  u\_int t\_maxunacktime;  u\_int t\_keepinit; /\* time to establish connection \*/  u\_int t\_keepidle; /\* time before keepalive probes begin \*/  u\_int t\_keepintvl; /\* interval between keepalives \*/  u\_int t\_keepcnt; /\* number of keepalives before close \*/  int t\_dupacks; /\* consecutive dup acks recd \*/  int t\_lognum; /\* Number of log entries \*/  int t\_loglimit; /\* Maximum number of log entries \*/  int64\_t t\_pacing\_rate; /\* bytes / sec, -1 => unlimited \*/  struct tcp\_log\_stailq t\_logs; /\* Log buffer \*/  struct tcp\_log\_id\_node \*t\_lin;  struct tcp\_log\_id\_bucket \*t\_lib;  const char \*t\_output\_caller; /\* Function that called tcp\_output \*/  struct statsblob \*t\_stats; /\* Per-connection stats \*/  uint32\_t t\_logsn; /\* Log "serial number" \*/  uint32\_t gput\_ts; /\* Time goodput measurement started \*/  tcp\_seq gput\_seq; /\* Outbound measurement seq \*/  tcp\_seq gput\_ack; /\* Inbound measurement ack \*/  int32\_t t\_stats\_gput\_prev; /\* XXXLAS: Prev gput measurement \*/  uint8\_t t\_tfo\_client\_cookie\_len; /\* TCP Fast Open client cookie length \*/  uint32\_t t\_end\_info\_status; /\* Status flag of end info \*/  unsigned int \*t\_tfo\_pending; /\* TCP Fast Open server pending counter \*/  union {  uint8\_t client[TCP\_FASTOPEN\_MAX\_COOKIE\_LEN];  uint64\_t server;  } t\_tfo\_cookie; /\* TCP Fast Open cookie to send \*/  union {  uint8\_t t\_end\_info\_bytes[TCP\_END\_BYTE\_INFO];  uint64\_t t\_end\_info;  }; #ifdef TCPPCAP  struct mbufq t\_inpkts; /\* List of saved input packets. \*/  struct mbufq t\_outpkts; /\* List of saved output packets. \*/ #endif }; |

tcp头部结构体

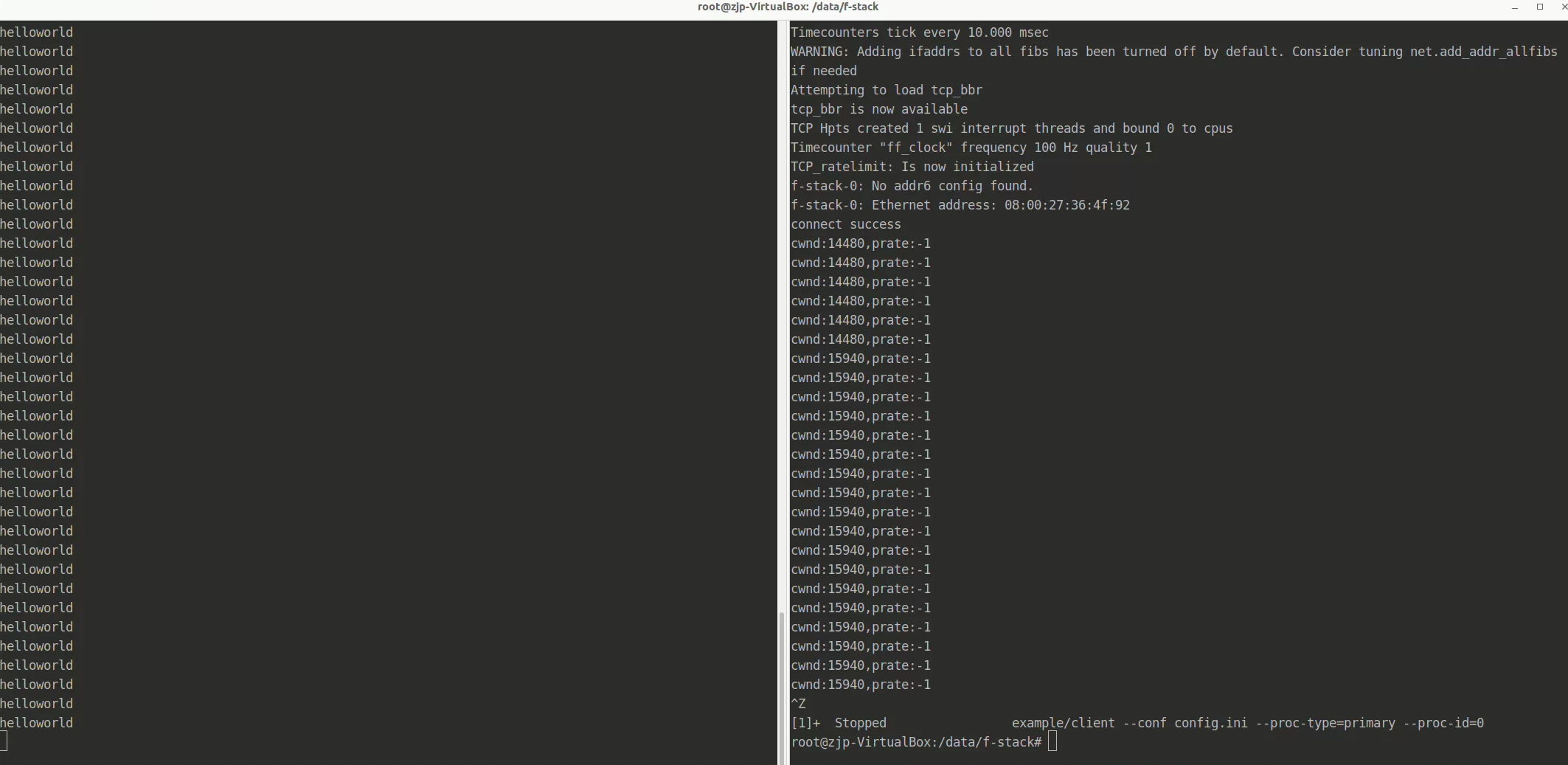
|  |
| --- |
| C struct tcphdr {  u\_short th\_sport; /\* source port \*/  u\_short th\_dport; /\* destination port \*/  tcp\_seq th\_seq; /\* sequence number \*/  tcp\_seq th\_ack; /\* acknowledgement number \*/ #if BYTE\_ORDER == LITTLE\_ENDIAN  u\_char th\_x2:4, /\* (unused) \*/  th\_off:4; /\* data offset \*/ #endif #if BYTE\_ORDER == BIG\_ENDIAN  u\_char th\_off:4, /\* data offset \*/  th\_x2:4; /\* (unused) \*/ #endif  u\_char th\_flags; #define TH\_FIN 0x01 #define TH\_SYN 0x02 #define TH\_RST 0x04 #define TH\_PUSH 0x08 #define TH\_ACK 0x10 #define TH\_URG 0x20 #define TH\_ECE 0x40 #define TH\_CWR 0x80 #define TH\_AE 0x100 /\* maps into th\_x2 \*/ #define TH\_FLAGS (TH\_FIN|TH\_SYN|TH\_RST|TH\_PUSH|TH\_ACK|TH\_URG|TH\_ECE|TH\_CWR) #define PRINT\_TH\_FLAGS "\20\1FIN\2SYN\3RST\4PUSH\5ACK\6URG\7ECE\10CWR\11AE"   u\_short th\_win; /\* window \*/  u\_short th\_sum; /\* checksum \*/  u\_short th\_urp; /\* urgent pointer \*/ }; |

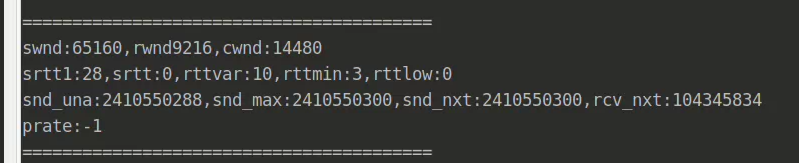
struct inpcb是一种用于表示网络连接的结构体。它包含了与该连接相关的所有信息，如本地和远程IP地址、本地和远程端口号、状态等。该结构体通常用于创建、维护和管理TCP套接字。

|  |
| --- |
| C struct inpcb {  /\* Cache line #1 (amd64) \*/  CK\_LIST\_ENTRY(inpcb) inp\_hash; /\* [w](h/i) [r](e/i) hash list \*/  CK\_LIST\_ENTRY(inpcb) inp\_pcbgrouphash; /\* (g/i) hash list \*/  struct rwlock inp\_lock;  /\* Cache line #2 (amd64) \*/ #define inp\_start\_zero inp\_hpts #define inp\_zero\_size (sizeof(struct inpcb) - \  offsetof(struct inpcb, inp\_start\_zero))  TAILQ\_ENTRY(inpcb) inp\_hpts; /\* pacing out queue next lock(b) \*/   uint32\_t inp\_hpts\_request; /\* Current hpts request, zero if  \* fits in the pacing window (i&b). \*/  /\*  \* Note the next fields are protected by a  \* different lock (hpts-lock). This means that  \* they must correspond in size to the smallest  \* protectable bit field (uint8\_t on x86, and  \* other platfomrs potentially uint32\_t?). Also  \* since CPU switches can occur at different times the two  \* fields can \*not\* be collapsed into a signal bit field.  \*/ #if defined(\_\_amd64\_\_) || defined(\_\_i386\_\_)  volatile uint8\_t inp\_in\_hpts; /\* on output hpts (lock b) \*/  volatile uint8\_t inp\_in\_input; /\* on input hpts (lock b) \*/ #else  volatile uint32\_t inp\_in\_hpts; /\* on output hpts (lock b) \*/  volatile uint32\_t inp\_in\_input; /\* on input hpts (lock b) \*/ #endif  volatile uint16\_t inp\_hpts\_cpu; /\* Lock (i) \*/  u\_int inp\_refcount; /\* (i) refcount \*/  int inp\_flags; /\* (i) generic IP/datagram flags \*/  int inp\_flags2; /\* (i) generic IP/datagram flags #2\*/  volatile uint16\_t inp\_input\_cpu; /\* Lock (i) \*/  volatile uint8\_t inp\_hpts\_cpu\_set :1, /\* on output hpts (i) \*/  inp\_input\_cpu\_set : 1, /\* on input hpts (i) \*/  inp\_hpts\_calls :1, /\* (i) from output hpts \*/  inp\_input\_calls :1, /\* (i) from input hpts \*/  inp\_spare\_bits2 : 4;  uint8\_t inp\_numa\_domain; /\* numa domain \*/  void \*inp\_ppcb; /\* (i) pointer to per-protocol pcb \*/  struct socket \*inp\_socket; /\* (i) back pointer to socket \*/  uint32\_t inp\_hptsslot; /\* Hpts wheel slot this tcb is Lock(i&b) \*/  uint32\_t inp\_hpts\_drop\_reas; /\* reason we are dropping the PCB (lock i&b) \*/  TAILQ\_ENTRY(inpcb) inp\_input; /\* pacing in queue next lock(b) \*/  struct inpcbinfo \*inp\_pcbinfo; /\* (c) PCB list info \*/  struct inpcbgroup \*inp\_pcbgroup; /\* (g/i) PCB group list \*/  CK\_LIST\_ENTRY(inpcb) inp\_pcbgroup\_wild; /\* (g/i/h) group wildcard entry \*/  struct ucred \*inp\_cred; /\* (c) cache of socket cred \*/  u\_int32\_t inp\_flow; /\* (i) IPv6 flow information \*/  u\_char inp\_vflag; /\* (i) IP version flag (v4/v6) \*/  u\_char inp\_ip\_ttl; /\* (i) time to live proto \*/  u\_char inp\_ip\_p; /\* (c) protocol proto \*/  u\_char inp\_ip\_minttl; /\* (i) minimum TTL or drop \*/  uint32\_t inp\_flowid; /\* (x) flow id / queue id \*/  struct m\_snd\_tag \*inp\_snd\_tag; /\* (i) send tag for outgoing mbufs \*/  uint32\_t inp\_flowtype; /\* (x) M\_HASHTYPE value \*/  uint32\_t inp\_rss\_listen\_bucket; /\* (x) overridden RSS listen bucket \*/   /\* Local and foreign ports, local and foreign addr. \*/  struct in\_conninfo inp\_inc; /\* (i) list for PCB's local port \*/   /\* MAC and IPSEC policy information. \*/  struct label \*inp\_label; /\* (i) MAC label \*/  struct inpcbpolicy \*inp\_sp; /\* (s) for IPSEC \*/   /\* Protocol-dependent part; options. \*/  struct {  u\_char inp\_ip\_tos; /\* (i) type of service proto \*/  struct mbuf \*inp\_options; /\* (i) IP options \*/  struct ip\_moptions \*inp\_moptions; /\* (i) mcast options \*/  };  struct {  /\* (i) IP options \*/  struct mbuf \*in6p\_options;  /\* (i) IP6 options for outgoing packets \*/  struct ip6\_pktopts \*in6p\_outputopts;  /\* (i) IP multicast options \*/  struct ip6\_moptions \*in6p\_moptions;  /\* (i) ICMPv6 code type filter \*/  struct icmp6\_filter \*in6p\_icmp6filt;  /\* (i) IPV6\_CHECKSUM setsockopt \*/  int in6p\_cksum;  short in6p\_hops;  };  CK\_LIST\_ENTRY(inpcb) inp\_portlist; /\* (i/h) \*/  struct inpcbport \*inp\_phd; /\* (i/h) head of this list \*/  inp\_gen\_t inp\_gencnt; /\* (c) generation count \*/  void \*spare\_ptr; /\* Spare pointer. \*/  rt\_gen\_t inp\_rt\_cookie; /\* generation for route entry \*/  union { /\* cached L3 information \*/  struct route inp\_route;  struct route\_in6 inp\_route6;  };  CK\_LIST\_ENTRY(inpcb) inp\_list; /\* (p/l) list for all PCBs for proto \*/  /\* (e[r]) for list iteration \*/  /\* (p[w]/l) for addition/removal \*/  struct epoch\_context inp\_epoch\_ctx; }; |

tcp\_input.c

修改内核cc函数，打印struct tcpcb控制块相应参数





**tcp时间辍与报文长度**

*MSS就是TCP数据包每次能够传输的最大量。为了达到最佳的传输效能， TCP协议在建立连接的时候通常 要协商双方的MSS值，这个值TCP协议在实现的*

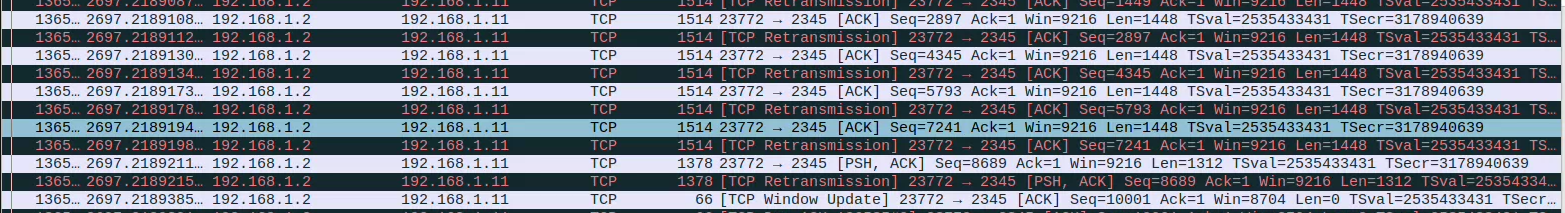
*时候往往用MTU值代替（需要减去IP数据包包头的大小20Bytes和TCP数据段的 包头 20Bytes ）所以往往 MSS 为 1460（ 如图1中红色方框所示的SYN包中的MSS值）。通讯双方会根据双方提供的 MSS 值得最小 值确定为这次连接的最大 MSS 值。*

*MSS为1460是由1500-20（IP头）-20（TCP头）计算出的。*

*实际场景下，TCP包头中会带有12字节的选项----时间戳。*

*这样，单个TCP包实际传输的最大量就缩减为1448字节。1448=1500-20（IP头）-32（20字节TCP头和12字节TCP选项时间戳）*

*原文链接：https://blog.csdn.net/hfhhgfv/article/details/86511432*



**tcp拥塞控制算法搭建**

流程参考：

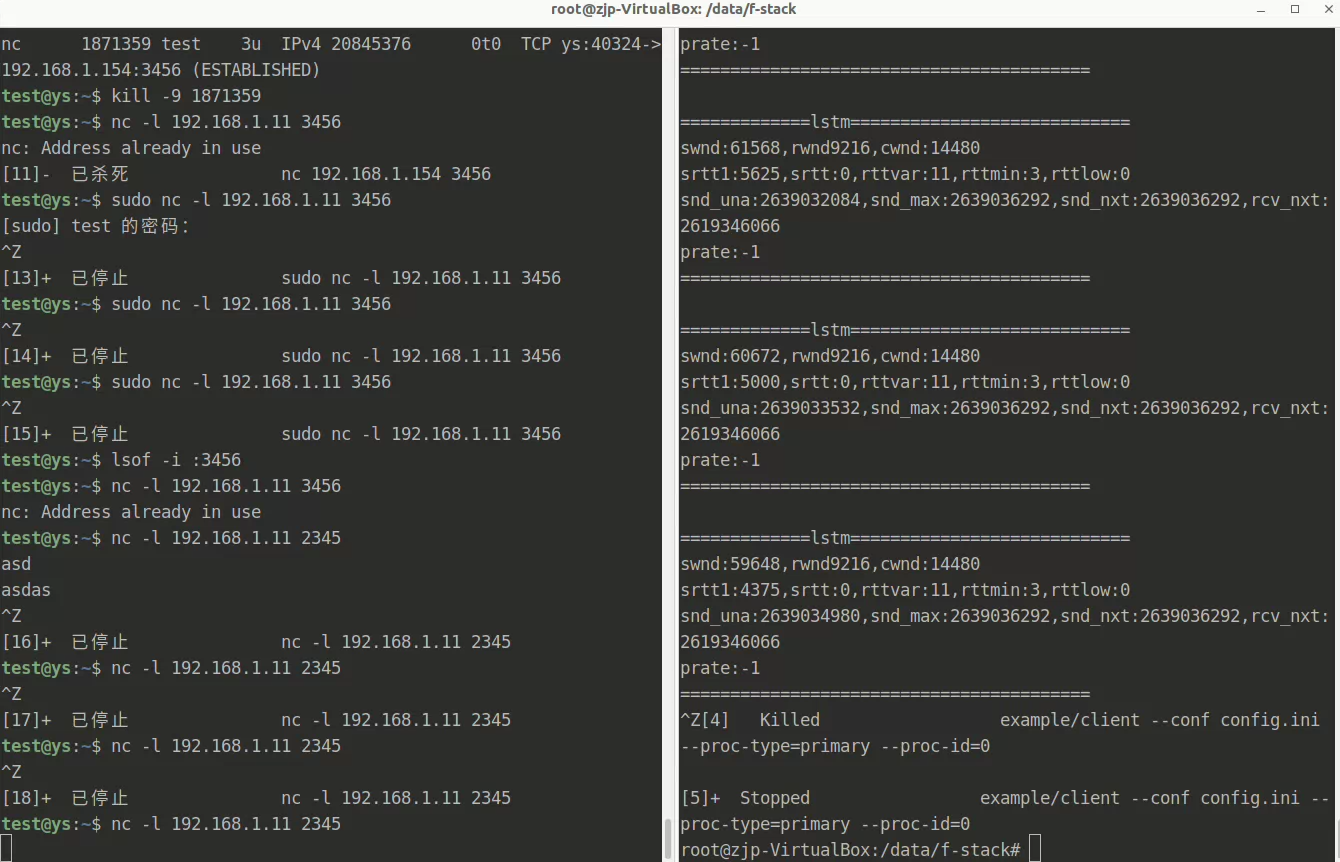
[cc](https://uestc.feishu.cn/docx/CNefdcLGcovK5TxTsxUcG25Wnve?from=from_copylink) “算法部署”

pidcc.c

|  |
| --- |
| C 按照其流程完成算法部署后 1.修改f-stack/lib文件中的Makefile,加入你的cc\_xxx.c文件，然后编译 2.重新编译APP，即example，不然直接运行会显示你新增的总赛控制算法报错，如下面小图所示 |



完成上述流程，拥塞控制算法成功部署，显示结果如下：



主机直连PING

