Implement: Decentralized Opportunistic Coflow-Aware Scheduling for Data Center Networks

Agenda:

1. How to Build the new Linux kernel
2. How to generate the coflow traffic
3. Code in Linux kernel
4. Demo

Author: HaoJIN

2015-07-03

How to build new Linux kernel

STEPS:

1. Download kernel source code: https://www.kernel.org/
2. tar xvJf ***.tar.xz –C /usr/src/
3. sudo chown -R haojin linux-***
4. sudo chmod 775 -R linux-***
5. make -j5
6. make -j5 modules
7. sudo make -j5 modules_install
8. sudo make install
9. sudo shutdown –r now
10. Verify (uname -r)

```
haojin@sing064:/usr/src/linux-4.0$ sudo make install
sh ./arch/x86/boot/install.sh 4.0.0 arch/x86/boot/bzImage \
                System.map "/boot"
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.0.0 /boot/vmlinuz-4.0.0
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.0.0 /boot/vmlinuz-4.0.0
update-initramfs: Generating /boot/initrd.img-4.0.0
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.0.0 /boot/vmlinuz-4.0.0
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.0.0 /boot/vmlinuz-4.0.0
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.0.0
Found initrd image: /boot/initrd.img-4.0.0
Found linux image: /boot/vmlinuz-4.0.0.old
Found initrd image: /boot/initrd.img-4.0.0
Found memtest86+ image: /memtest86+.elf
Found memtest86+ image: /memtest86+.bin
done
```

Add the taskid in ip Options field

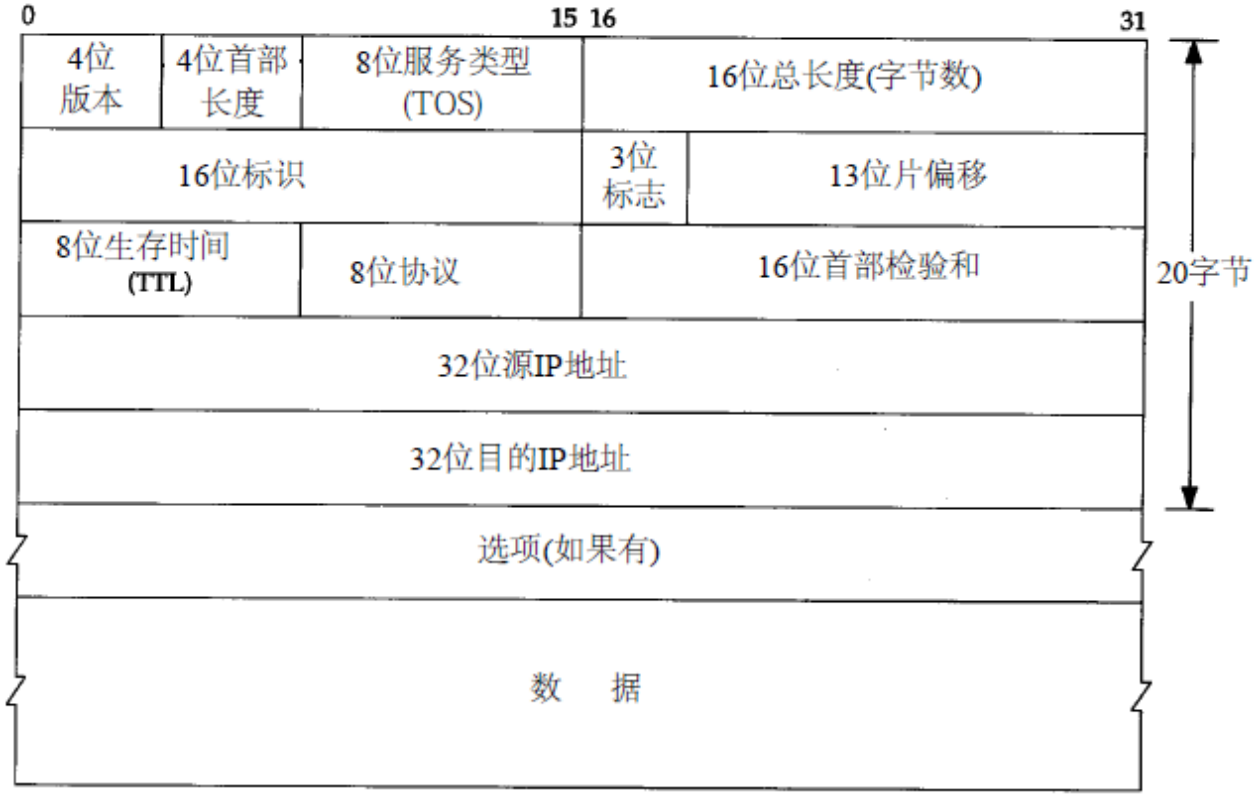
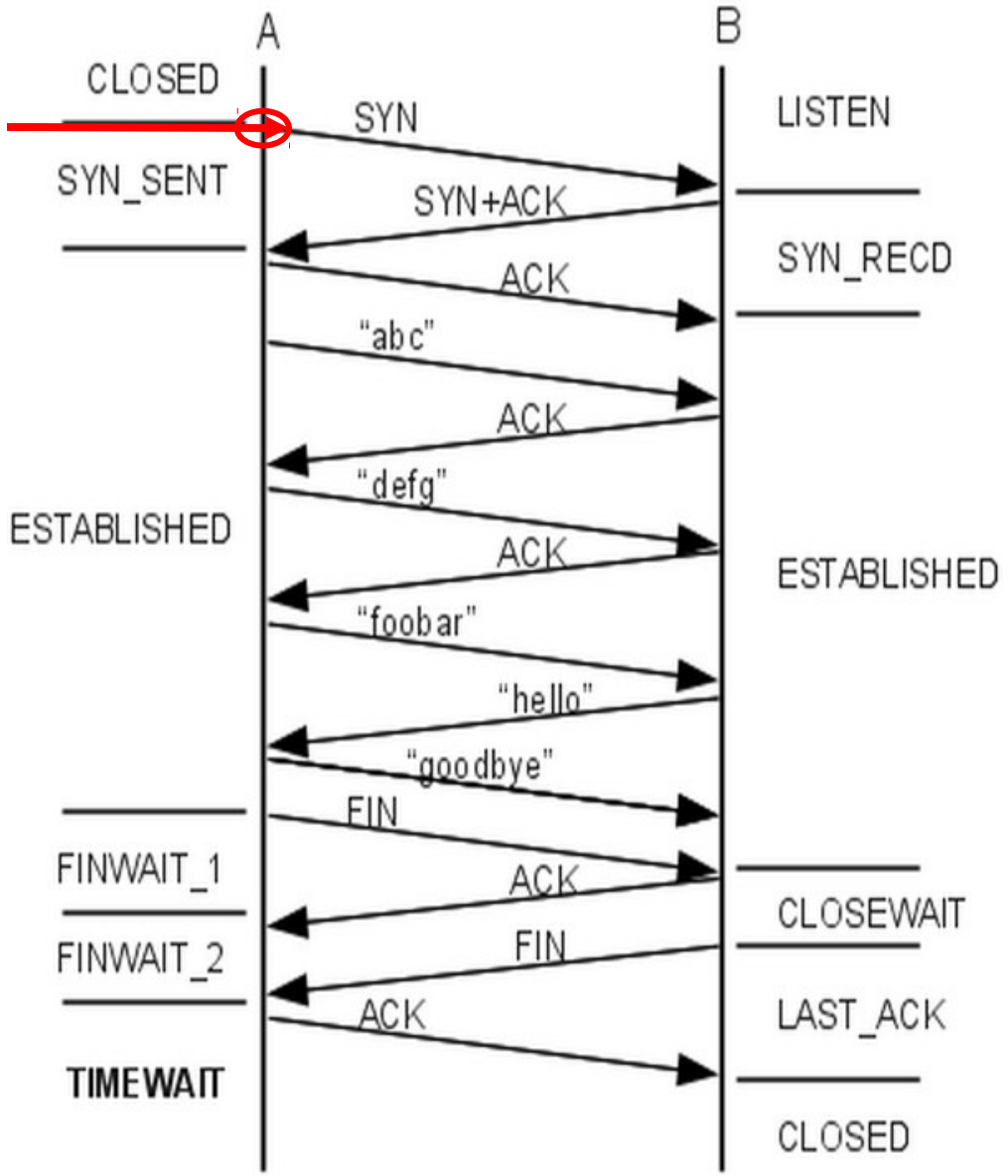


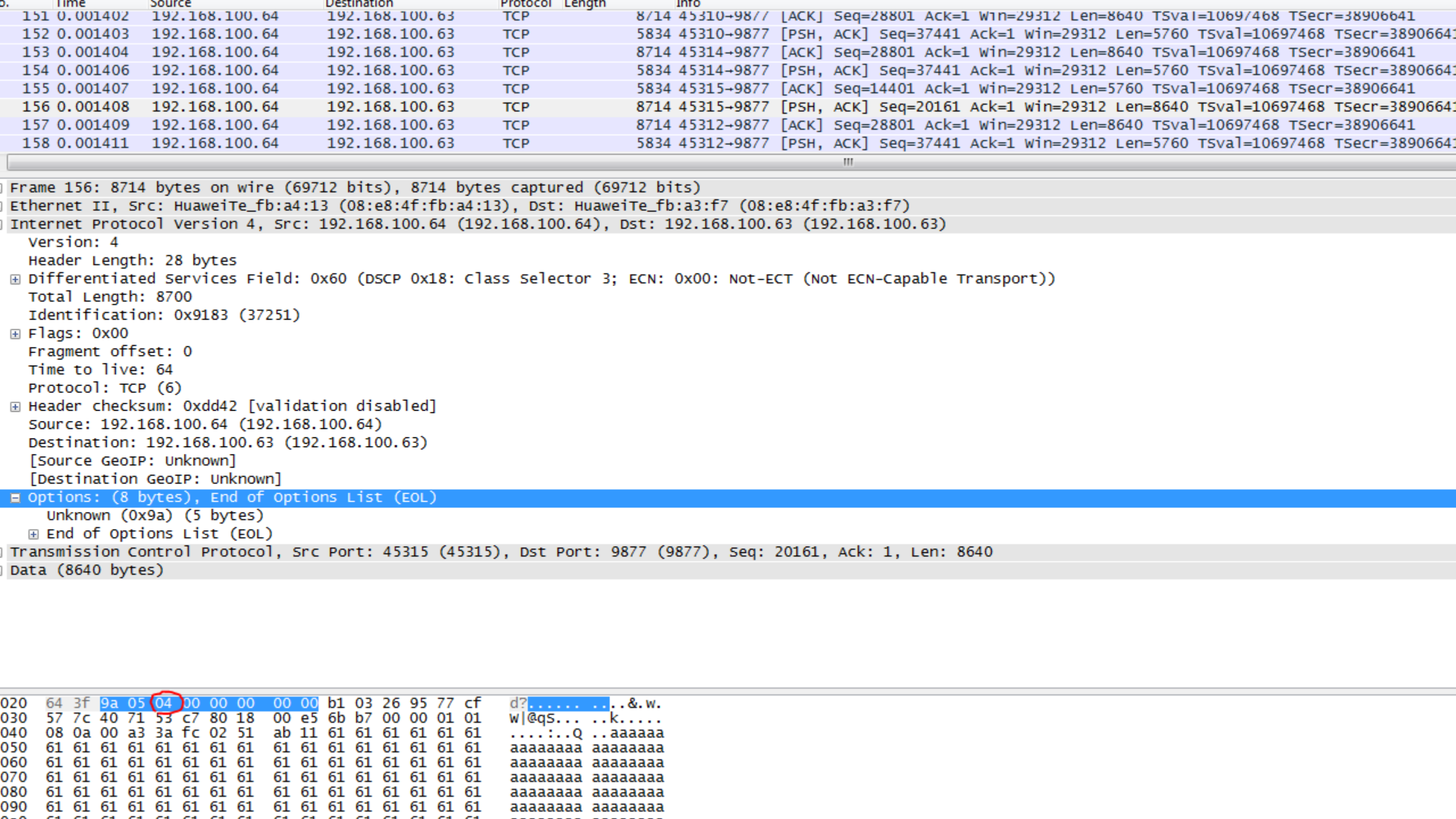
图3-1　IP数据报格式及首部中的各字段

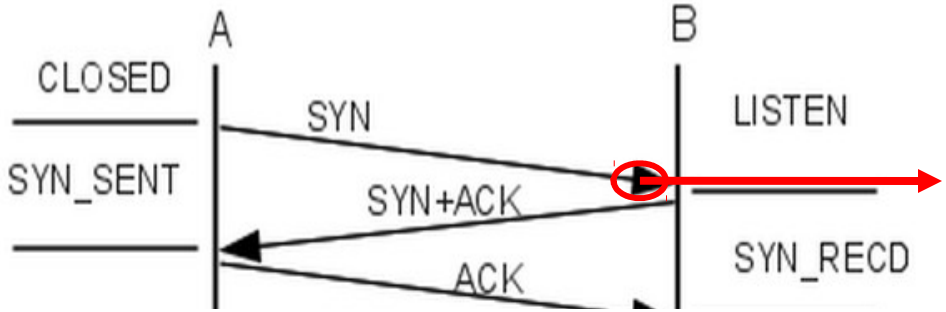
How to generate the coflow

```
00176: void handle(){
00177:
00178:     int sockfd;
00179:     struct sockaddr_in  servaddr;
00180:     TASKID option;
00181:     int n,m;
00182:     unsigned int coflowid;
00183:     int df_value = IP_PMTUDISC_DONT;
00184:
00185:     sockfd = Socket(AF_INET, SOCK_STREAM, 0);
00186:
00187:     bzero(&servaddr, sizeof(servaddr));
00188:     servaddr.sin_family = AF_INET;
00189:     servaddr.sin_port = htons(SERV_PORT);
00190:     inet_pton(AF_INET, DESTIP, &servaddr.sin_addr);
00191:
00192:     srand (time(NULL));
00193: //  coflowid = expon_random(1,coflowid_lamda)*coflowid_range + 1;
00194:     coflowid = 2;
00195:
00196: //  printf("coflowid: %d    ", coflowid);
00197:
00198:     bzero(&option, sizeof(TASKID));
00199:     option.Code = 154;
00200:     option.Len = 5;
00201:     option.Taskid = coflowid;
00202:
00203:     n = setsockopt(sockfd, IPPROTO_IP, IP_MTU_DISCOVER, &df_value, sizeof(df_value));
00204: //  printf("n:%d\n", n);
00205:
00206:     n = setsockopt(sockfd,IPPROTO_IP,IP_OPTIONS,(char *)&option, option.Len);
00207: //  printf("n:%d\n", n);
00208:
00209:
00210:     m = connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
00211: //  printf("connect() m:%d\n", m);
00212:
00213:     str_cli2(sockfd);
00214:
00215: } « end handle »
```

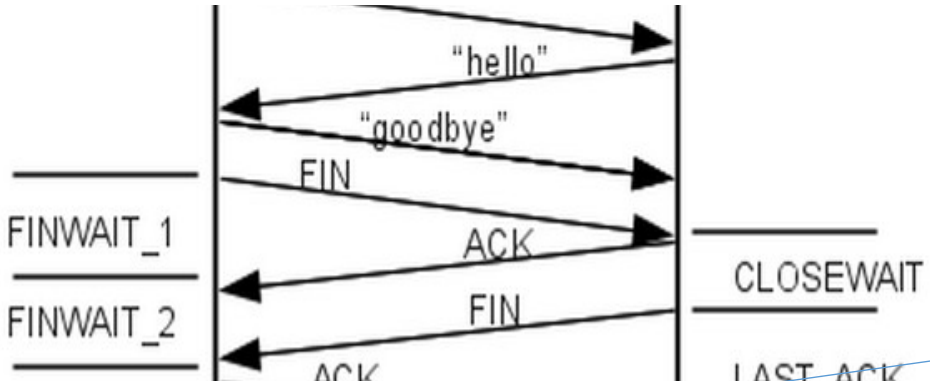
```
00020: typedef struct
00021: {
00022: unsigned char Code;
00023: unsigned char Len;
00024: unsigned char Taskid;
00025: unsigned long Reserved;
00026: }TASKID;
```




No. Time Source Destination Protocol Length Info
151 0.001402 192.168.100.64 192.168.100.63 TCP 8714 45310→9877 [ACK] Seq=28801 Ack=1 Win=29312 Len=8640 TSval=10697468 TSecr=38906641
152 0.001403 192.168.100.64 192.168.100.63 TCP 5834 45310→9877 [PSH, ACK] Seq=37441 Ack=1 Win=29312 Len=5760 TSval=10697468 TSecr=3890664
153 0.001404 192.168.100.64 192.168.100.63 TCP 8714 45314→9877 [ACK] Seq=28801 Ack=1 Win=29312 Len=8640 TSval=10697468 TSecr=38906641
154 0.001406 192.168.100.64 192.168.100.63 TCP 5834 45314→9877 [PSH, ACK] Seq=37441 Ack=1 Win=29312 Len=5760 TSval=10697468 TSecr=3890664
155 0.001407 192.168.100.64 192.168.100.63 TCP 5834 45315→9877 [ACK] Seq=14401 Ack=1 Win=29312 Len=5760 TSval=10697468 TSecr=38906641
156 0.001408 192.168.100.64 192.168.100.63 TCP 8714 45315→9877 [PSH, ACK] Seq=20161 Ack=1 Win=29312 Len=8640 TSval=10697468 TSecr=3890664
157 0.001409 192.168.100.64 192.168.100.63 TCP 8714 45312→9877 [ACK] Seq=28801 Ack=1 Win=29312 Len=8640 TSval=10697468 TSecr=38906641
158 0.001411 192.168.100.64 192.168.100.63 TCP 5834 45312→9877 [PSH, ACK] Seq=37441 Ack=1 Win=29312 Len=5760 TSval=10697468 TSecr=3890664
Frame 156: 8714 bytes on wire (69712 bits), 8714 bytes captured (69712 bits)
Ethernet II, Src: HuaweiTe_fb:a4:13 (08:e8:4f:fb:a4:13), Dst: HuaweiTe_fb:a3:f7 (08:e8:4f:fb:a3:f7)
Internet Protocol Version 4, Src: 192.168.100.64 (192.168.100.64), Dst: 192.168.100.63 (192.168.100.63)
Version: 4
Header Length: 28 bytes
Differentiated Services Field: 0x60 (DSCP 0x18: Class Selector 3; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
Total Length: 8700
Identification: 0x9183 (37251)
Flags: 0x00
Fragment offset: 0
Time to live: 64
Protocol: TCP (6)
Header checksum: 0xdd42 [validation disabled]
Source: 192.168.100.64 (192.168.100.64)
Destination: 192.168.100.63 (192.168.100.63)
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Options: (8 bytes), End of Options List (EOL)
Unknown (0x9a) (5 bytes)
End of Options List (EOL)
Transmission Control Protocol, Src Port: 45315 (45315), Dst Port: 9877 (9877), Seq: 20161, Ack: 1, Len: 8640
Data (8640 bytes)
0020 64 3f 9a 05 04 00 00 00 00 00 b1 03 26 95 77 cf d?..... ..&.w.
0030 57 7c 40 71 53 c7 80 18 00 e5 6b b7 00 00 01 01 w|@qS... ..k.....
0040 08 0a 00 a3 3a fc 02 51 ab 11 61 61 61 61 61 61 ....:..Q ..aaaaaa
0050 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aaaaaaaa aaaaaaaa
0060 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aaaaaaaa aaaaaaaa
0070 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aaaaaaaa aaaaaaaa
0080 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aaaaaaaa aaaaaaaa
0090 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 aaaaaaaa aaaaaaaa

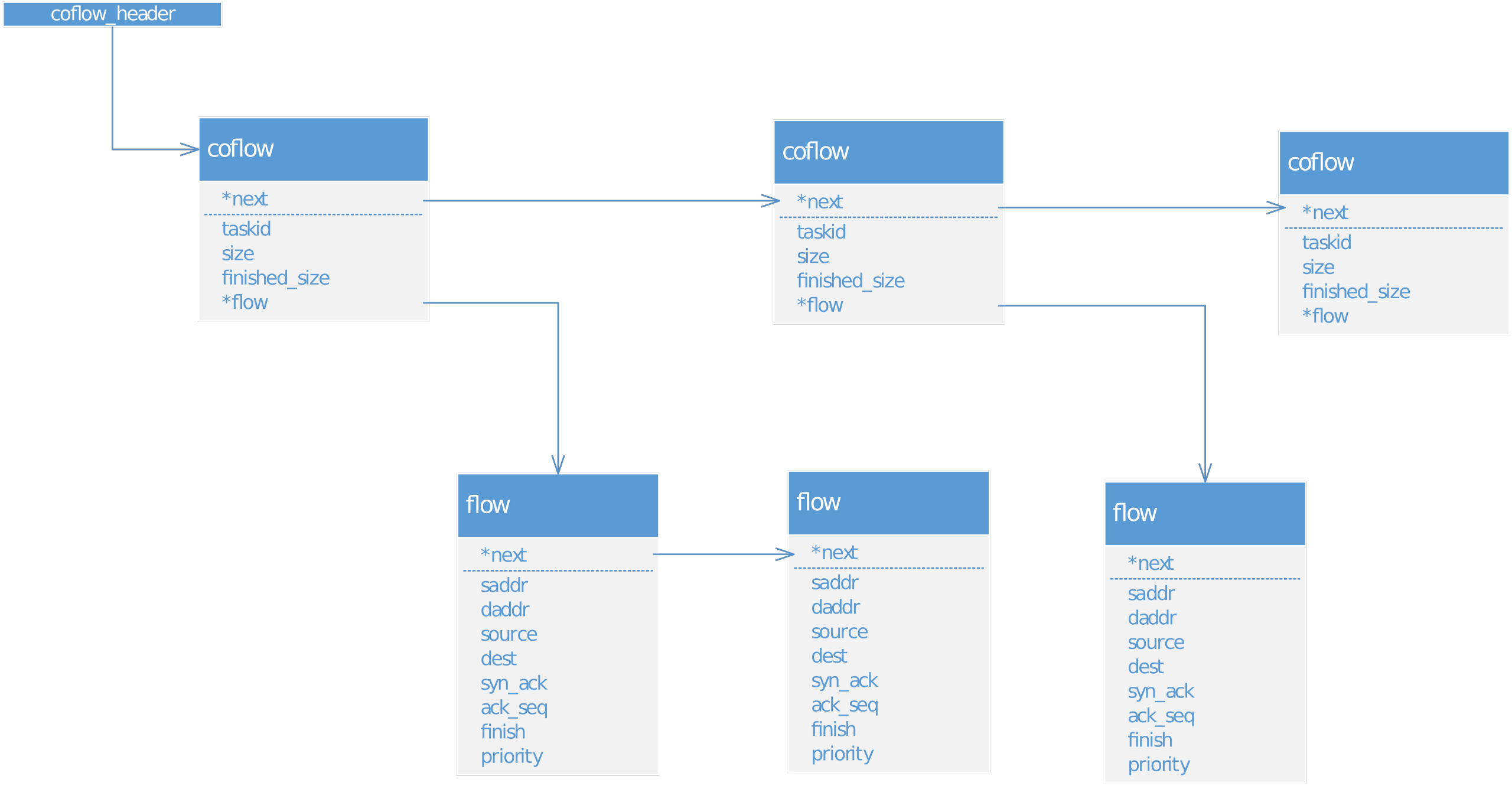



```c
/*haojin*/
struct flow{
    struct flow *next; /*flows constitute a list*/
    __be32  saddr;  /*source address of flow*/
    __be32  daddr;  /*destination address of flows*/
    __be16  source;  /*tcp source port*/
    __be16  dest;   /*tcp dest port*/
    __be32  syn_ack;    /*tcp three way handshake syn ack value*/
    __be64  ack_seq;    /*current ack sequence value*/
    __be32  finish;     /*this flow finished or not. if finisn, assign fin ack. Otherwise; unfinish assign 0*/
    __be32  priority;   /*flow priority that is change by flow size*/
};
```

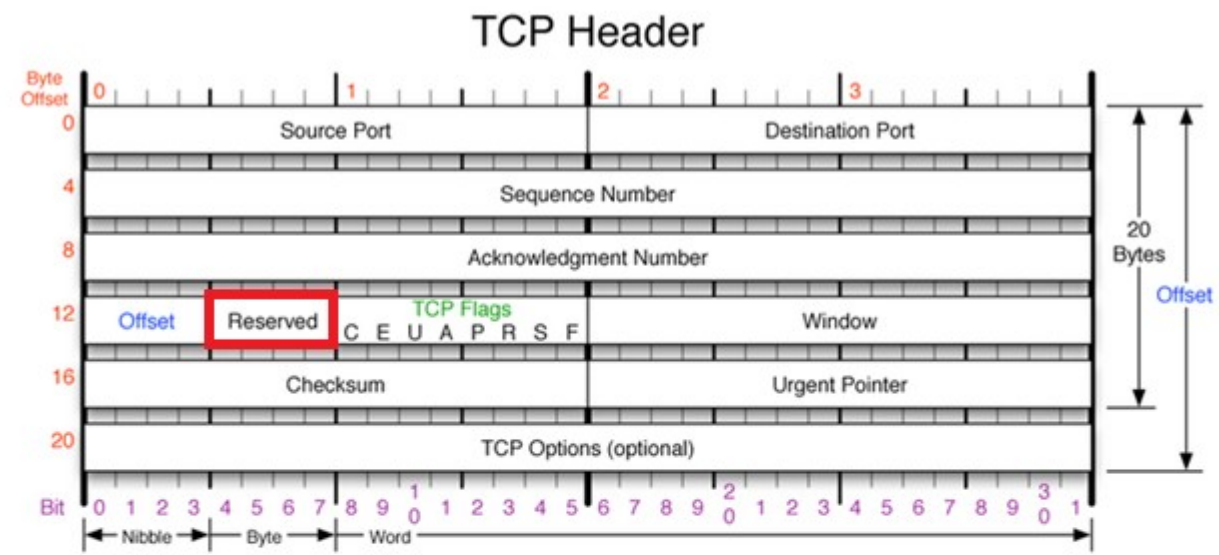



```c
struct coflow {
    struct coflow *next; /*coflow constitute a list*/
    __u8    taskid;     /*task id in ip header option*/
    __u32   size;       /*how many flows belong to one coflow*/
    __u32   finished_size;      /*how many flows have finished*/
    __u32   length;     /*how many coflow length, if two flows belong to one coflow, means add these two flows finished length*/
    __be64 start;       /*cofow start time*/
    __be64 stop;        /*coflow stop time*/
    struct flow *flow;  /*coflow contains a lot of flows. This is the entry of flow point*/
};
```

```c
int tcp_conn_request(struct request_sock_ops *rsk_ops,
            const struct tcp_request_sock_ops *af_ops,
            struct sock *sk, struct sk_buff *skb)
{
        /*haojin: create coflow structure*/
            optptr = (unsigned char *)&(ip_hdr(skb)[1]);
            if (*(optptr+1) == IPOPT_RA) {
                id = *(optptr + 3);
                printk(KERN_ALERT "The taskid: %x\n", id);
            }

            if (id >=1 && id <128){
                if (flow1 = kmalloc(sizeof(struct flow),  GFP_KERNEL)){
                    printk(KERN_ALERT "kmalloc struct flow\n");
                    flow1->next = NULL;
                    flow1->saddr = ip_hdr(skb)->saddr;
                    printk(KERN_ALERT "flow1->saddr: %x\n", flow1->saddr);
                    flow1->daddr = ip_hdr(skb)->daddr;
                    printk(KERN_ALERT "flow1->daddr: %x\n", flow1->daddr);
                    flow1->source = tcp_hdr(skb)->source;
                    flow1->dest = tcp_hdr(skb)->dest;
                    flow1->syn_ack = tcp_rsk(req)->rcv_isn + 1;
                    printk(KERN_ALERT "syn_ack:%x\n", tcp_rsk(req)->rcv_isn + 1);
                    flow1->ack_seq = tcp_rsk(req)->rcv_isn + 1;
                    flow1->finish = 0;
                    flow1->priority = 8;
                    printk(KERN_ALERT "flow build\n");
                    }

                p = coflow_header;
                prev = coflow_header;

                while (p != NULL && p->taskid != id){
                    prev = p;
                    p = p->next;
                };

              if(p == NULL){  /*it is a new coflow*/
                  q = kmalloc(sizeof(struct coflow),  GFP_KERNEL);
                  printk(KERN_ALERT "kmalloc struct coflow\n");
                  /*assign*/
                  if (coflow_header == NULL) {
                      coflow_header = q;

                                        q;



                                        = 0;

                                    T "coflow build\n");
                                    = id){     /*it is a old coflow*/
```


coflow_header
coflow
*next
taskid
size
finished_size
*flow
coflow
*next
taskid
size
finished_size
*flow
coflow
*next
taskid
size
finished_size
*flow
flow
*next
saddr
daddr
source
dest
syn_ack
ack_seq
finish
priority
flow
*next
saddr
daddr
source
dest
syn_ack
ack_seq
finish
priority
flow
*next
saddr
daddr
source
dest
syn_ack
ack_seq
finish
priority

Add priority in TCP Reserved field



TCP Flags

C E U A P R S F

Congestion Window
C 0x80 Reduced (CWR)
E 0x40 ECN Echo (ECE)
U 0x20 Urgent
A 0x10 Ack
P 0x08 Push
R 0x04 Reset
S 0x02 Syn
F 0x01 Fin

Congestion Notification

ECN (Explicit Congestion Notification). See RFC 3168 for full details, valid states below.

Packet State	DSB	ECN bits
Syn	0 0	1 1
Syn-Ack	0 0	0 1
Ack	0 1	0 0
No Congestion	0 1	0 0
No Congestion	1 0	0 0
Congestion	1 1	0 0
Reciever Response	1 1	0 1
Sender Response	1 1	1 1

TCP Options

0 End of Options List
1 No Operation (NOP, Pad)
2 Maximum segment size
3 Window Scale
4 Selective ACK ok
8 Timestamp

Checksum

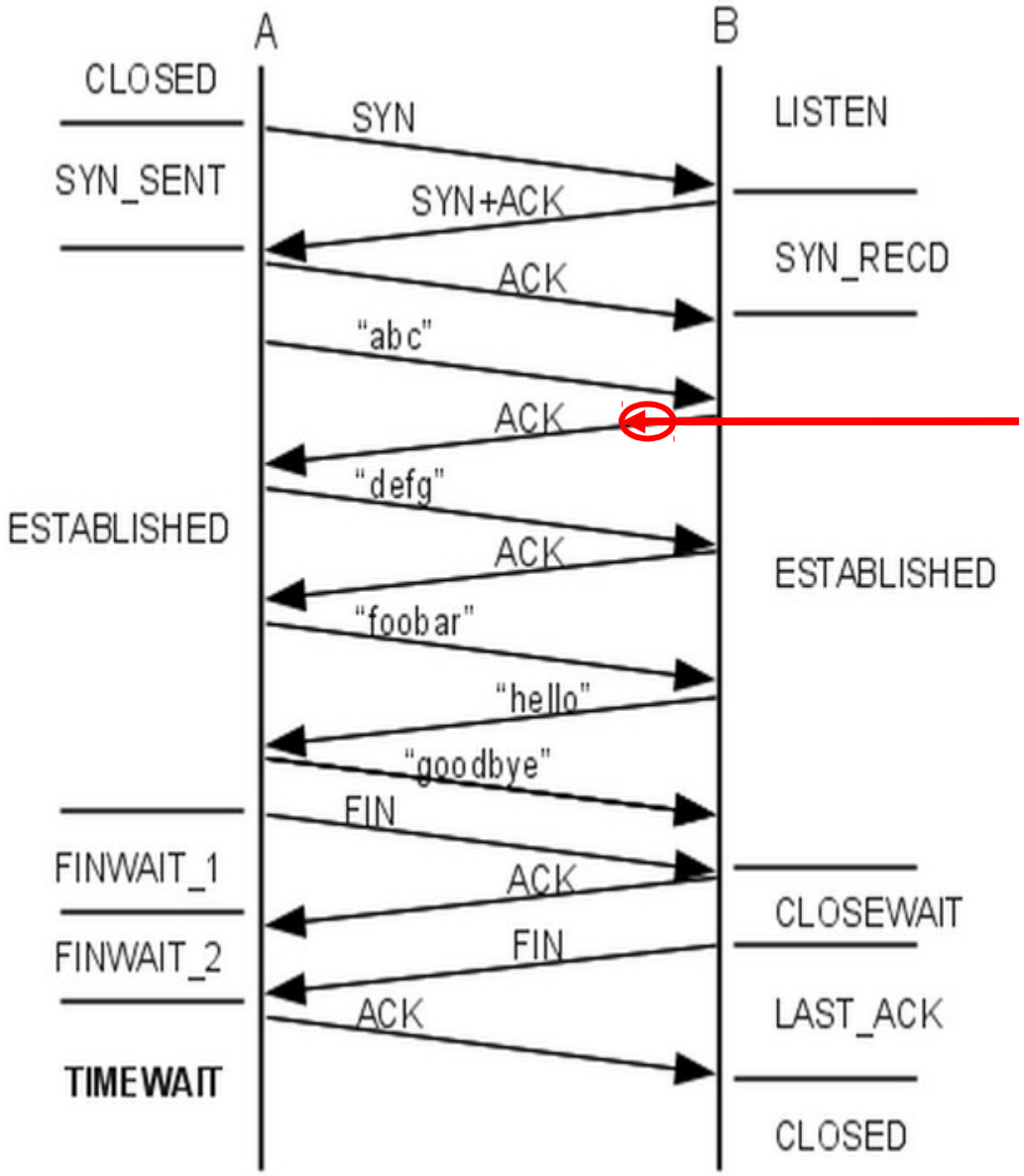
Checksum of entire TCP segment and pseudo header (parts of IP header)

Offset

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

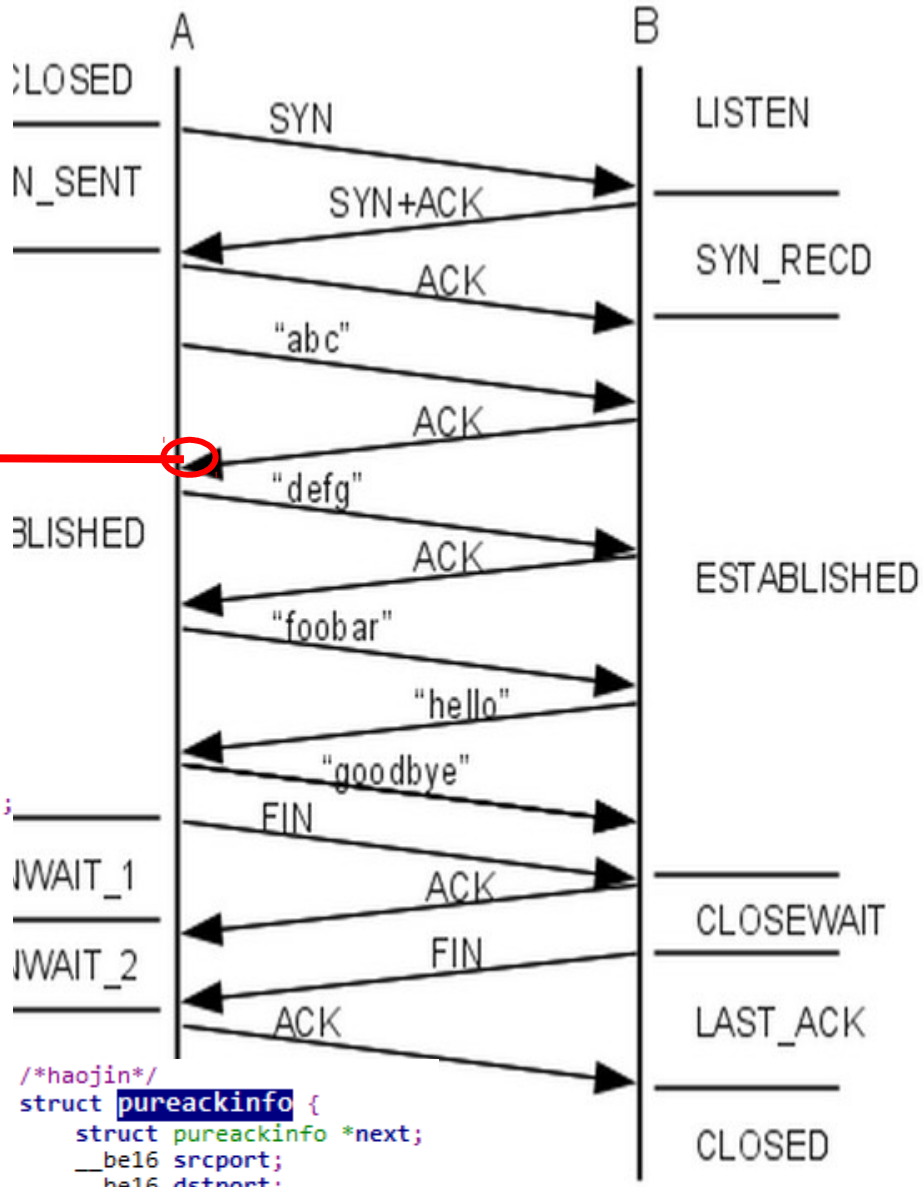
RFC 793

Please refer to RFC 793 for the complete Transmission Control Protocol (TCP) Specification.



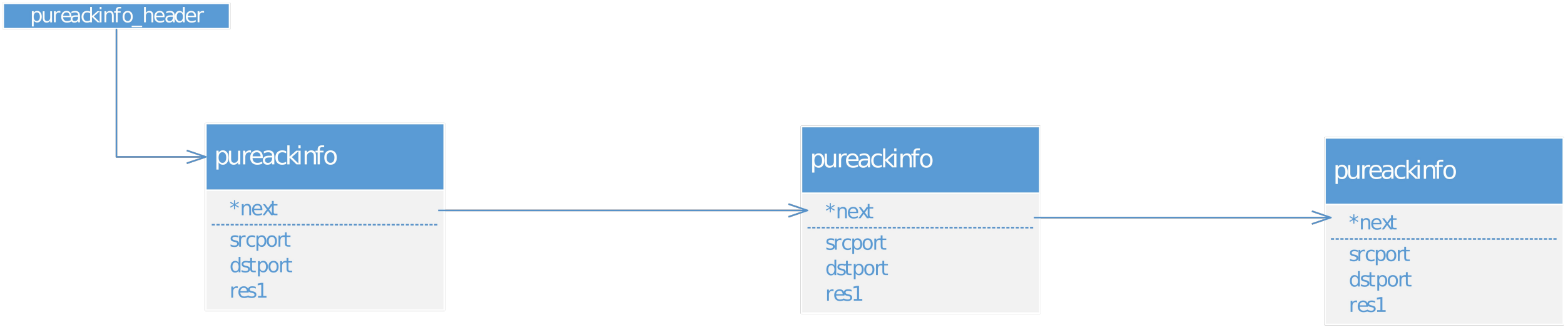

```
static int tcp_transmit_skb(struct sock *sk, struct sk_buff *skb, int clone_it,
                gfp_t gfp_mask)
{

 /*haojin*/
     if (tcb->tcp_flags & TCPHDR_ACK){

          //iph = ip_hdr(skb);
          fl4 = &inet->cork.fl.u.ip4;

          p = coflow_header;

          while(p != NULL){

              f = p->flow;

              while (f != NULL){


                  if((f->saddr == fl4->daddr) && (f->daddr == fl4->saddr) && (f->source == inet->inet_dport)
                  {
                      break;
                  }
                  f = f->next;
              };

              if(f == NULL) {
                  p = p->next;
              }else if((f->saddr == fl4->daddr) && (f->daddr == fl4->saddr) && (f->source == inet->inet_dpor
//                    printk(KERN_ALERT "Found this flow\n");
                    break;
              }
          } « end while p!=NULL »

          if(f != NULL){

              /*haojin: deal with the sequence rollback*/
              ack_sequence = htonl(th->ack_seq);
              while(ack_sequence - f->ack_seq < 0 ){
                  ack_sequence += MAXLEN32;
              }

              f->ack_seq = ack_sequence;

              len = f->ack_seq - f->syn_ack;

                  if ( len >= 70){
                      th->res1 = 1;
                  }else if(len >= 60){
                      th->res1 = 2;
                  }else if (len >= 50){
                      th->res1 = 3;
                  }else if(len >= 40){
                      th->res1 = 4;
```

```
void tcp_rcv_established(struct sock *sk, struct sk_buff *skb,
            const struct tcphdr *th, unsigned int len)
{


if((th->res1 >= 1) && (th->res1 <=8))
{
    printk(KERN_ALERT "ACK res1: %x  source port: %x  dest port: %x\n", th->res1, th->source, th->dest);
    p = pureackinfo_header;
    prev = p;

    while((p != NULL) && !(p->srcport == th->source && p->dstport == th->dest)){

        prev = p;
        p = p->next;
        printk(KERN_ALERT "pureack p: %x  pureack prev: %x\n", p, prev);
    }


    if(p != NULL && p->srcport == th->source && p->dstport == th->dest){

        if (p->res1 != th->res1){
            p->res1 = th->res1;
            printk(KERN_ALERT "update res1: %x\n", th->res1);
        }

    }else if (p == NULL){

        if (pureack = kmalloc(sizeof(struct pureackinfo), GFP_KERNEL)){

            pureack->dstport = th->dest;
            pureack->srcport = th->source;
            pureack->res1 = th->res1;
            pureack->next = NULL;
            printk(KERN_ALERT "pureackinfo create: dstport: %x  srcport: %x\n", pureack->dstport, pureack->srcport);
        }

        if (pureackinfo_header == NULL) {

            pureackinfo_header = pureack;
            printk(KERN_ALERT "add in the header\n");

        } else {

            prev->next = pureack;
            printk(KERN_ALERT "add in the list\n");
        }


    } « end if p==NULL »
```



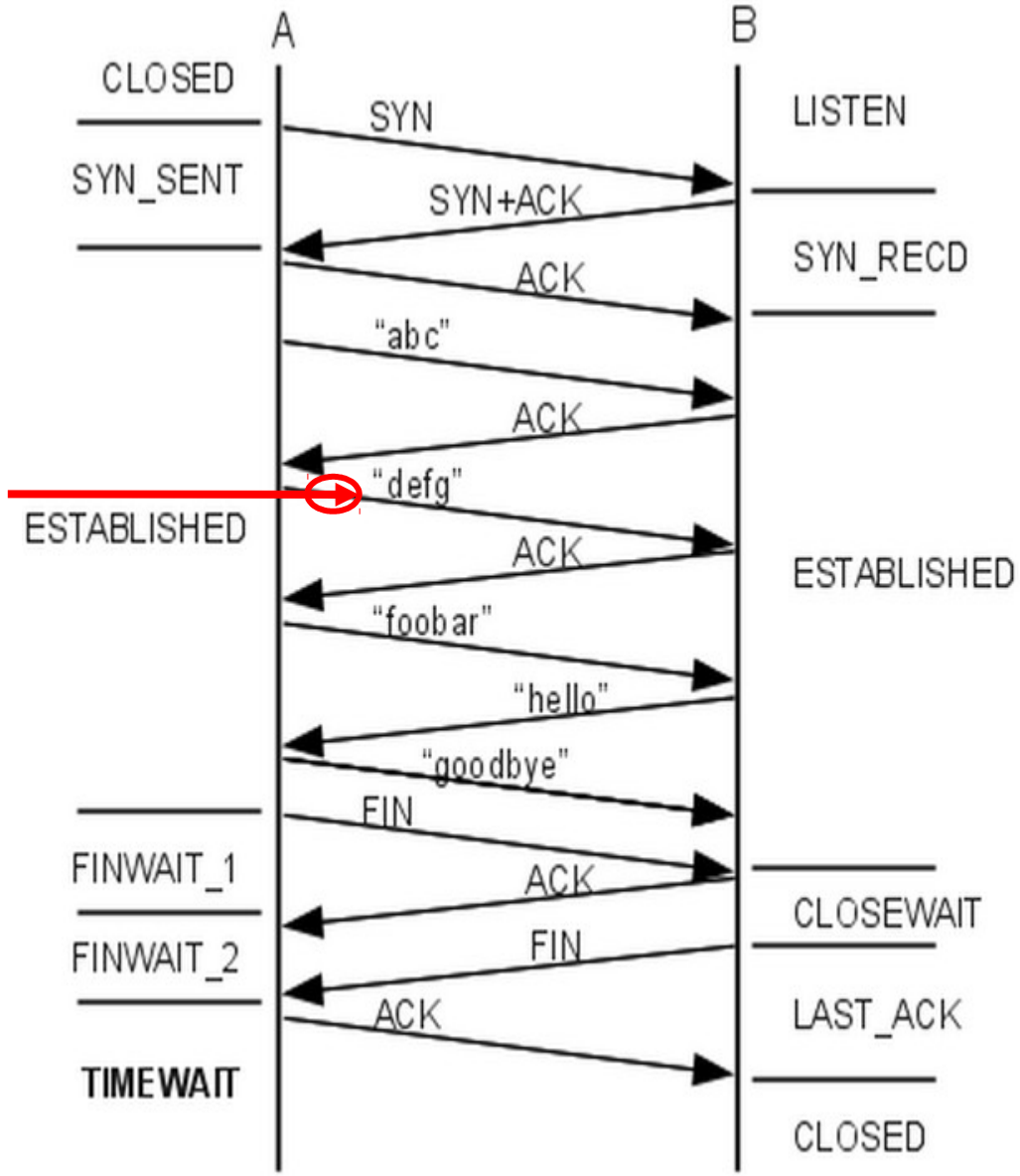

```
/*haojin*/
struct pureackinfo {
    struct pureackinfo *next;
    __be16 srcport;
    __be16 dstport;
    __u16 res1;
};
```

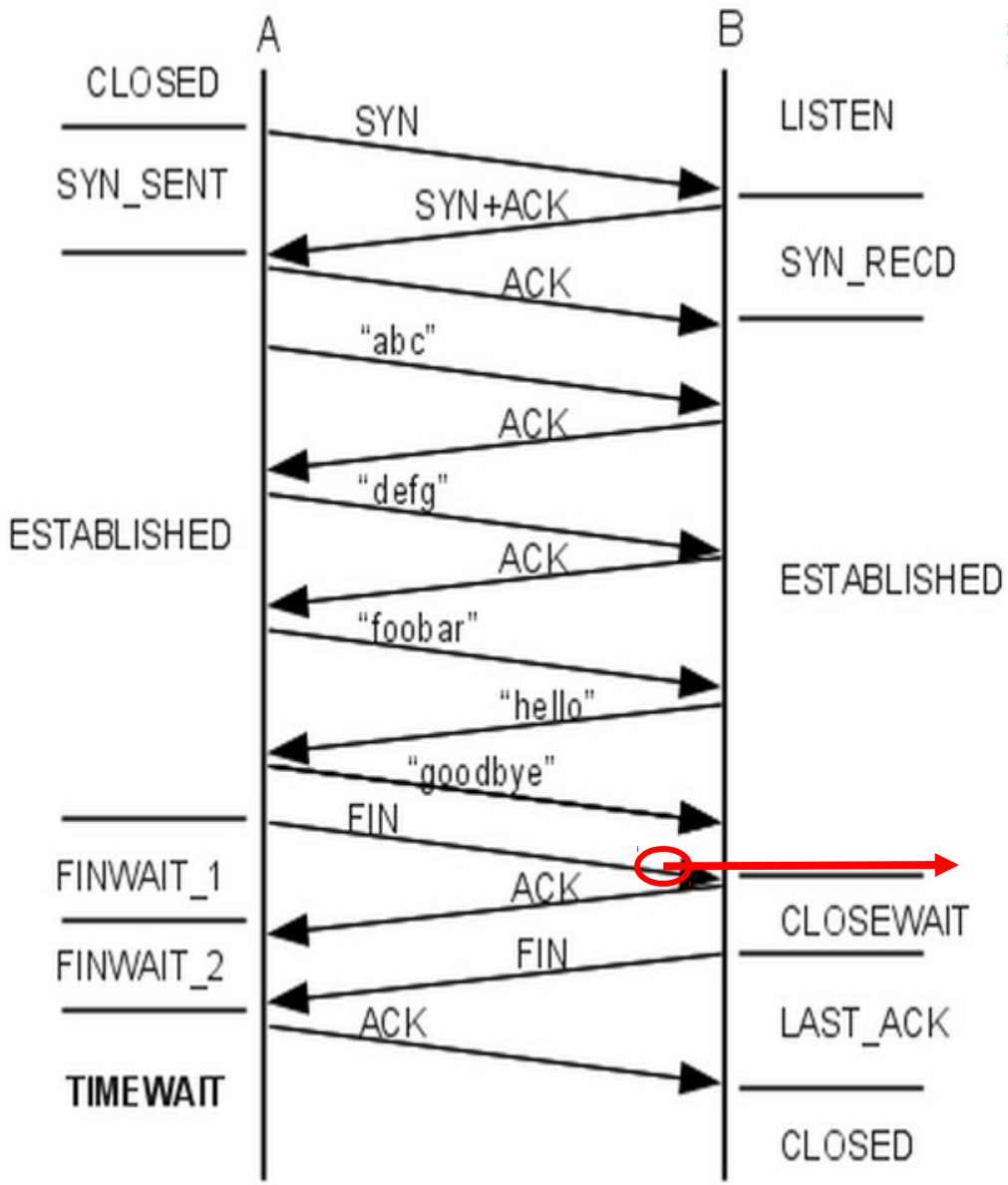
Record tcp ACK res1 != 0.
It is used to modify ip data dscp value when send the next packet.



```
int ip_queue_xmit(struct sock *sk, struct sk_buff *skb, struct flowi *fl)
{
```

```
    /*haojin*/
        th = tcp_hdr(skb);
    //  printk(KERN_ALERT "tcp header source port: %x\n", th->source);
    //  printk(KERN_ALERT "tcp header dest port: %x\n", th->dest);


        info = pureackinfo_header;

        while ((info != NULL) && !(info->dstport == th->source && info->srcport == th->dest)){

            info = info->next;


        }

        if(info != NULL && info->dstport == th->source && info->srcport == th->dest){

            iph->tos = dscp[info->res1];
    //      printk(KERN_ALERT "add tos: %x\n", dscp[info->res1]);
        }

    /*end*/
```

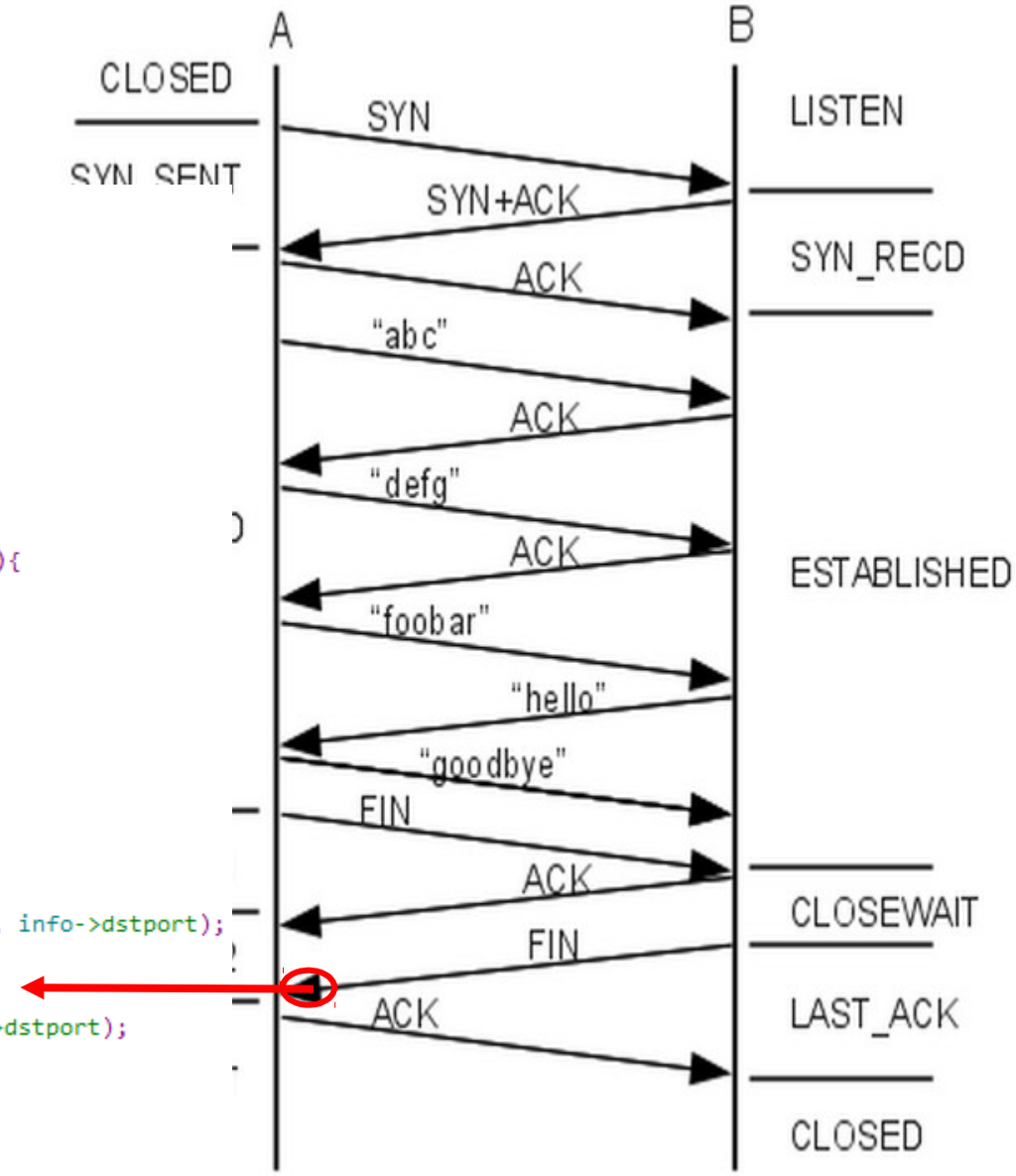




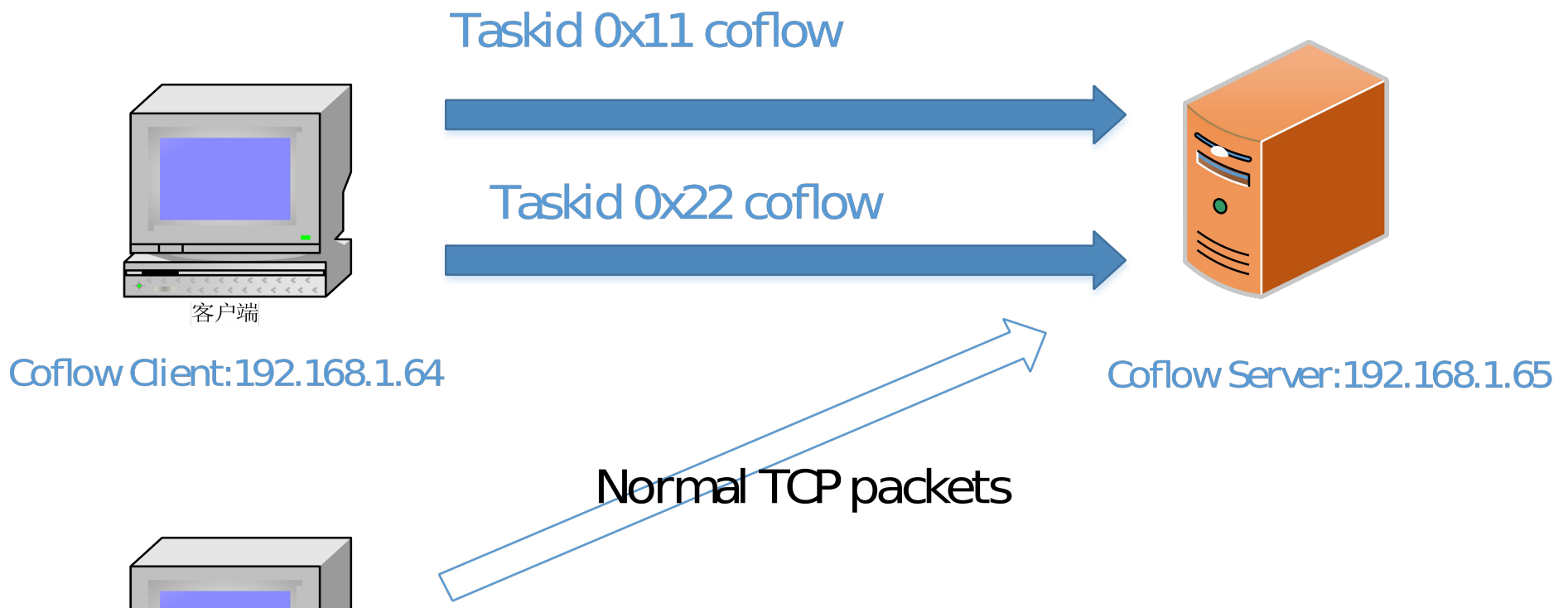

```
static void tcp_fin(struct sock *sk)
{

    case TCP_ESTABLISHED:

        /*haojin: remove flow or some coflow*/
        inet = inet_sk(sk);
        fl4 = &inet->cork.fl.u.ip4;
        p = coflow_header;
        prep = coflow_header;

        while(p != NULL){
            f = p->flow;
            pref = p->flow;
            activeflows = p->size - p->finished_size;

            while (f != NULL){

                if((f->saddr == fl4->daddr) && (f->daddr == fl4->saddr) && (f->source == inet->inet_dport) &
                {
                    break;
                }
                pref = f;
                f = f->next;
            };

            if(f == NULL) {
                prep = p;
                p = p->next;
            }else if((f->saddr == fl4->daddr) && (f->daddr == fl4->saddr) && (f->source == inet->inet_dport)
                if(p->flow == f){
                    p->flow = f->next;
                    kfree(f);
                }else {
                    pref->next = f->next;
                    kfree(f);
                }
                p->finished_size +=1;

                if(activeflows == 1) {
                    if(p == coflow_header){
                        coflow_header = p->next;
                    }else{
                        prep->next = p->next;
                    }
                    kfree(p);
                    printk(KERN_ALERT "Free this flow\n");
                }
                break;
            } « end if (f->saddr==fl4->daddr... »
        } « end while p!=NULL »
```

```
static void tcp_fin(struct sock *sk)
{
```

```
    case TCP_FIN_WAIT2:

    /*haojin*/
    //  printk(KERN_ALERT "TCP_FIN_WAIT2\n");
        inet = inet_sk(sk);
        info = pureackinfo_header;
        previous = pureackinfo_header;



    //  printk(KERN_ALERT "inet:source port: %x dest port: %x\n", inet->inet_sport, inet->inet_dport);
    //  printk(KERN_ALERT "source port: %x dest port: %x\n", info->srcport, info->dstport);


        while ((info != NULL) && !(info->dstport == inet->inet_sport && info->srcport == inet->inet_dport)){

            previous = info;
            info = info->next;

        }

        if(info != NULL && info->dstport == inet->inet_sport && info->srcport == inet->inet_dport){

    //      printk(KERN_ALERT "inet:source port: %x dest port: %x\n", inet->inet_sport, inet->inet_dport);
    //      printk(KERN_ALERT "source port: %x dest port: %x\n", info->srcport, info->dstport);
            if (info == pureackinfo_header) {
                pureackinfo_header = info->next;
                kfree(info);
                printk(KERN_ALERT "Free header pureackinfo:source port: %x dest port: %x\n", info->srcport, info->dstport);
            }else {
                previous->next = info->next;
                kfree(info);
                printk(KERN_ALERT "Free pureackinfo:source port: %x dest port: %x\n", info->srcport, info->dstport);
            }
        }
/*end*/
```



Demo & QA


Taskid 0x11 coflow
Taskid 0x22 coflow
客户端
Coflow Client:192.168.1.64
Coflow Server:192.168.1.65
Normal TCP packets
客户端
Normal Client:192.168.1.63