```
  for (;;) {
    len = sizeof(fromAddr);
    if ( (n= ?? (rawSocket, rcvbuffer, BUFSIZE,0,
                  (struct sockaddr *)&fromAddr, &len)) < 0 )  {
      printf ("erreur recvfrom");
      exit (1);}
    /*Commentaire 2*/
    if ( inet_ntop (AF_INET, (const void *)&fromAddr.sin_addr,source ,sizeof(source)) <0)
{
      printf ("erreur inet_ntop");
      exit (1);
    }
    printf( "%d octets ICMP de %s: \n", n, source);
    ip = (struct ip *) rcvbuffer ;
    lenIPHeader = ip->ip_hl * 4;
    /*Commentaire 3*/
    icmp = (struct icmp *) (rcvbuffer + lenIPHeader);
    /*Commentaire 4*/
    ip2 = (??) (rcvbuffer + lenIPHeader + 8);
    lenIPHeader2 = ip2->ip_hl * 4;
    if (ip2->ip_p == IPPROTO_UDP) {
      udp = (struct udphdr *) (?? + lenIPHeader2);
      sport = ntohs(udp->uh_sport);
      dport = ntohs(udp->uh_dport) ;
       printf (" Reponse ICMP a un paquet UDP avec port source = %d et port destination =
%d \n",
      sport, dport);
    }
    switch (??->icmp_type) {
    case ICMP_UNREACH: {
      printf ("destination unreachable \n");

      switch (icmp->icmp_code){
      case ICMP_UNREACH_PORT:
             printf (" bad port \n");
             break;
      default:
       printf ("type %d, code = %d\n", icmp->icmp_type,
             icmp->icmp_code);
             break;
      }
   }
      break;
   case ICMP_ECHO:
      printf (" echo service  \n");
      break;
   case ICMP_ECHOREPLY :
      printf (" echo reply  \n");
      break;
  case ICMP_TIMXCEED :
      printf (" Time Exceed \n");
      break;
   default:
      printf ("type %d, code = %d\n", icmp->icmp_type,
          icmp->icmp_code);
      }
    }

    close(rawSocket);
}
```