REPORT (RDT3.0 설계)

제 출 일 : 2013 년 11 월 13 일

과 목 명 : 컴퓨터네트워크 및 실습

담당 교수 : 홍 진 표 교수님

학 과 : 정보통신학과

학 번 : 200901435, 200902719

성 명 : 박 헌 일, 이 준 하

HANKUK UNIVERSITY OF FOREIGN STUDIES

1. 문제 정의

RDT 3.0 Protocol 을 아래의 다음에 따라 설계하고, 최종 구현 물을 작성하여, 두 개의 Ununtu VM 에서 구동되는 sender 와 receiver 가 잘 작동함을 검증하고(기능시험), timeout 기간을 달리 설정하면서 성능을 측정하라(성능시험).

제약조건

1. RDT 3.0 sender 는 application data 를 test file 에서 읽고, <LF>로 구분되는 한 줄(최대 1,400 bytes)을 읽고, packet 을 만들어 보낸다. RDT 3.0 receiver 는 받은 packet 을 file 로 출력한다.
2. Sender 는 EOF 을 만나면 receivers 에게 종료를 알려주기 위해 END 라는 control packet 을 전송하고, 이를 받으면 receiver 는 ACK 로 응답한다. 물론 END 와 ACK control packet 도 lost 또는 corrupt 될 수 있다.
3. Packet 은 하나도 빠짐없이, 순서에 맞게, 그리고 bit error 없이 정확하게 전달되어야 한다.
4. 요즘 네트워크에서는 packet loss 가 거의 발생하지 않기 때문에, udt_send()로 packet 을 전송할 때 1/10 확률로 random 하게 packet 을 lost 시켜야 한다. (다시 말해, 보내지 않으면 된다.)
5. 또한, Packet error rate (packet 중에 bit error 가 하나 있을 확률)은 1/10 로 보내기 전에 bit error 를 발생시켜야 한다. (참고: udt_send.c)

2. 설계

- 설계 명세를 포함해야 함
- State diagram, information flow(sequence diagram)은 반드시 포함되어야 함

- 교과서에는 sender 나 receiver 가 종료하는 event 가 없다. Sender 는 EOF 를 읽으면 END 라는 control packet 을 보내고, ACK 를 받아야 종료되게 하자. RDT receiver 는 END packet 을 받으면 ACK packet 을 연속하여 3 개를 보내고 종료한다. 3 개를 보내는 이유는 ACK 도 손실될 수 있기 때문에 3 개 중 하나라도 sender 에게 도착할 가능성을 높이려는 시도다. 물론, 종료하기 전에 receiver 는 성능 측정치들을 output 해야 할 것이다.
- sequence number 를 저장하는 variable 'seq'를 도입하여 state 개수를 교과서의 절반으로 줄여보자. 그리고, 교과서와 달리 ACK 번호를 잘 받은 packet 번호가 아니라, 다음에 받을 번호로 하면 구현하기 더 편리한지도 알아보자. (UML state diagram 표준을 따라 그리세요.)

■ RDT 3.0 설명: 비트오류가 있는 손실 채널에서의 데이터 전송

RDT3.0의 특징

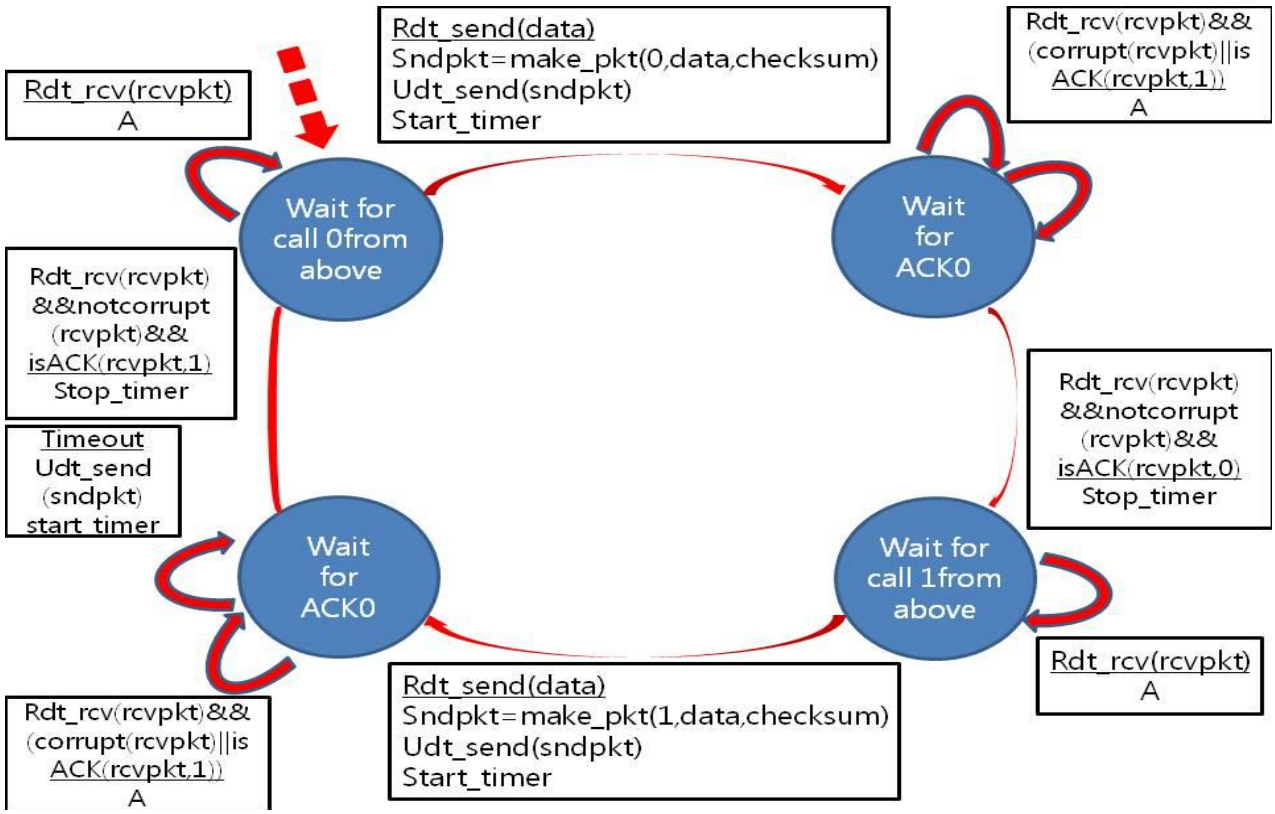
- 비트 오류가 있는 손실 채널상에서도 **신뢰적 데이터 전송**이 가능하다.
- **Stop-and-Wait** protocol 이다
- 송신자가 손실된 packet 의 검출과 회복 책임이 있다.
- Packet 의 **순서 번호**가 0 과 1 이 번갈아 일어나므로, alternating-bit protocol 라고도 한다
- 송신자가 합리적인 시간 동안 기다렸다가 **ACK 가 오지 않으면 재전송**함
- 중복 데이터 packet 이 재전송에 의해서 발생 할 수 있으나 순서번호로써 중복 packet 의 문제점을 해결할 수 있다. **수신자는 ACK 에 순서번호를 명시** 해야 한다.
- 주어진 시간 경과 후 기다리는 송신자를 인터럽트 할 수 있는 카운트 다운 **timer** 가 필요함.

■ 설계 명세

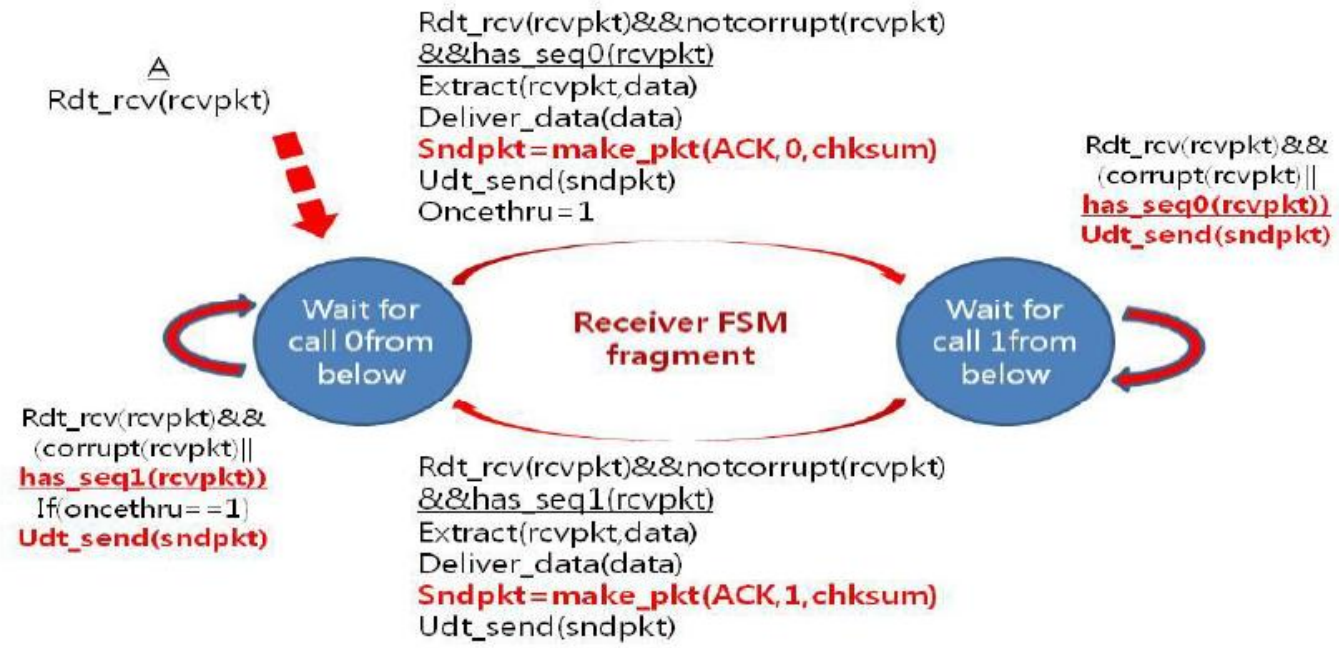
① State을 절반으로 줄여야 한다.
state을 절반으로 줄이기 위해 variable 'seq'을 도입해야 한다.
② sender 의 state closing 설계해야 한다.
③ Sender는 EOF를 읽으면 END 라는 control packet을 보내고, ACK를 받아야 종료되게 하자.
RDT receiver는 END packet을 받으면 ACK packet을 연속하여 3개를 보내고 종료한다.
④ 종료하기 전에 receiver는 성능 측정치들을 output해야 할 것이다.

다음 그림과 같이 컴퓨터 네트워킹 하향식접근 교재에서의 RDT 3.0 FSM 특징은 Sender의 State가 4개이며 Receiver의 FSM가 State가 2개입니다.

■ . 교재의 RDT 3.0 sender



■ **교재의 RDT 3.0 receiver**



상세 설계

① sequence variable 'seq' 도입, State개수 절반으로 줄이기

RDT3.0 의 특징 중 Packet 의 sequence number(순서 번호)가 0 과 1 이 번갈아 일어나므로, alternating-bit protocol 라고 하는데, 이 특징 때문에 교재의 RDT3.0 의 state 의 수가 sender 는 4 개가 되고 receiver 는 2 개가 되는 것 입니다.

따라서 저희 팀은 state 을 1/2 로 줄이기 위해서 sequence number(순서 번호)를 저장하는 variable 'seq'를 도입하여 state 개수를 Sender 의 state 를 2 개로 줄이고 Receiver 는 state 를 1 개로 줄였습니다.

저희 팀은 설계를 할 때 Sender 쪽은 variable 'seq'의 unsigned char sndSeq; 을 이용하고 Receiver 쪽은 variable 'seq'의 unsigned char rcvSeq; 을 이용하여 반으로 줄였습니다.

alternating-bit protocol 특징을 이용하기 위해서 0 과 1 이 번갈아 가게 하기 위해서 처음에 sndSeq 과 rcvSeq 을 0 으로 두었고 (sndSeq=0 과 rcvSeq=0) 1 로 바꾸기 위해서 Sender 쪽에서는 State 1: Wait_for_ACK 에서 sndSeq= (sndSeq+1)%2 에서 나머지가 0, 1 이 번갈아 가게 하였습니다. Receiver 쪽에서 if (isType(rcvpkt, DATA, rcvSeq))에서 rcvSeq= (rcvSeq+1)%2 나머지가 0, 1 이 번갈아 가게 하였습니다.

따라서, State 는 Sender 는 2State 로 / Receiver 는 1State 로 1 / 2 줄었습니다.

-Sender 설계-

Sender 에서 typedef enum {

1. Wait_for_call,
2. Wait_for_ACK,
3. Closing (②번에서 설명하겠습니다.)

} State; 로 나타내었는데 Sender 는 세가지 State 가 존재합니다.

Sender State 첫 번째 : **Wait_for_Call**

Event 1 : RDT_SEND(DATA)

1. **파일에 NULL 이 아닐 때 DATA 패킷을 보내고 Wait_for_ACK 로 이동**

   sndpkt = make_pkt(DATA, sndSeq, &sndPacket, datalen);

   packet.c파일을 보면 static char*typeName [ ] =
   {"NOOP", "DATA", "ACK", "END" }; 에서 힌트를 제공받았고
   설계하는 것에 "DATA", "ACK", "END"을 이용하였습니다.

   udt_send(s, sndpkt, (struct sockaddr *)&peer, sizeof(peer));

   udt_send.c 파일을 보면 Pcket loss, Packet Corurpted,
   Packet Not Corurpted 인지 자세하게 나타나있습니다.
   그리고 pktLossRate ,pktLossRate 등이 나타나 있습니다.

   start_timer(ReTxTimeout);

   print_pkt(">>",sndpkt); 패킷을 보내는 것을 프린트합니다.

   state = Wait_for_ACK; (Wait for ACK 으로 이동합니다.)

2. **파일에 EOF 이 일 때 END 패킷을 보내고 Closing 으로 이동**

   sndpkt = make_pkt(END, sndSeq, &sndPacket, 0);

   udt_send(s, sndpkt, (struct sockaddr *)&peer, sizeof(peer));

   start_timer(ReTxTimeout);

   print_pkt(">>",sndpkt); 패킷을 보내는 것을 프린트합니다.

   state = Closing; (Closing 으로 이동합니다.)

Sender State 두 번째 : **Wait_for_Call**

Event 2 : RDT_RCV(rcvpkt)

rcvlen = recvfrom(s, rcvpkt, sizeof(Packet), 0, NULL, NULL);

recvfrom 함수로 rcvpkt 와 rcvlen 을 받아옵니다.

Corrupt(rcvpkt))

오류가 있을 경우 입니다

notCorrupt(rcvpkt)&&isType(rcvpkt, ACK, sndSeq)

stop_timer( );

sndSeq = (sndSeq+1)%2; 2 로 나눈 나머지 처음에 sndSeq=0 이므로
0 을 넣으면 나머지가 1 이 되서 Wait for call 으로 이동
그 다음에 sndSeq=1 이므로 1 을 넣으면 2 로 나눈 2 의 나머지는
0 이므로 다시 sndSeq=0 이 됨

state = Wait_for_call;

Event 3: Wait_for_Call 에서 TIME OUT

udt_send(s, sndpkt, (struct sockaddr *)&peer, sizeof(peer));

start_timer(ReTxTimeout); (기존 RDT3.0 과 똑같음)

②Sender 에서 closing state 설계

Sender State 세 번째

Event 4 : RDT_RCV(rcvpkt)

notCorrupt(rcvpkt)&&isType(rcvpkt, END, sndSeq)

exit();

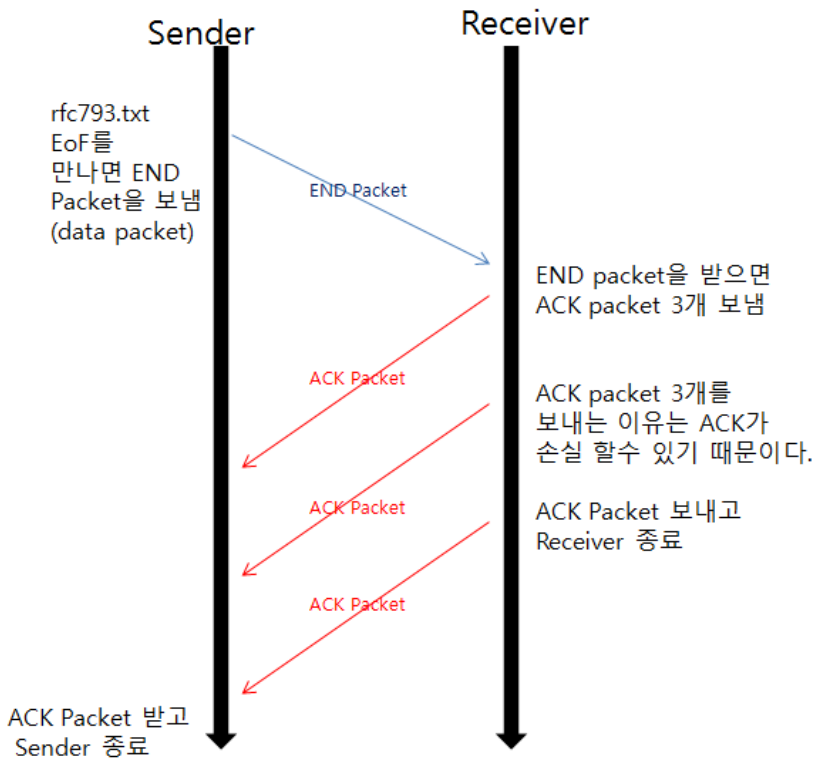
Event 5: closing 에서 TIME OUT

exit();

TIMEOUT 이 되면 종료합니다.

③ Sender는 EOF를 읽으면 END 라는 control packet을 보내고, ACK를 받아야 종료되게 하자. RDT receiver는 END packet을 받으면 ACK packet을 연속하여 3개를 보내고 종료한다.

-Sender 설계에서 다시 말하자면 파일에 EOF 이 일 때 END 패킷을 보내고 Closing 로 이동합니다.



Sender State 첫 번째 : **Wait_for_Call**

Event 1 : RDT_SEND(DATA)

sndpkt = make_pkt(END, sndSeq, &sndPacket, 0);

이부분은 Sender 는 EOF 를 읽으면 END 라는 control packet 을 receiver 에 보내는 것입니다.

udt_send(s, sndpkt, (struct sockaddr *)&peer, sizeof(peer));

start_timer(ReTxTimeout);

print_pkt(">>",sndpkt); 패킷을 보내는 것을 프린트합니다.

state = Closing; (Closing 으로 이동합니다.)

Receiver에서 END packet을 받으면 ACK packet을 연속하여 3개를 보내고 종료하는 구조입니다.

RDT_RCV(rcvpkt)&&isType(rcvpkt, END, rcvSeq)

sndpkt = make_pkt(ACK, rcvSeq, &sndPacket, 0);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

exit( ); ACK 을 3 번 보내고 종료를 합니다.

④ 종료하기 전에 receiver는 성능 측정치들을 output해야 할 것이다.

printStatistics 함수에서는 duration, throughput, nPackets, nDupPackets, nCorruptPackets, nCorrectPackets 를 구하는 식이 나와있고 timer.c 파일에서 보면 interval.tv_sec, interval.tv_usec 등을 구하는 식이 나와 있는 것을 알 수 있었습니다.

Udt_send.c 파일에서는 seed 를 구하는 공식이 나와있어 공부할 수 있었습니다.

```
void printStatistics(){
        struct timeval interval;
      float duration;              // in sec
      float throughput;            // in bytes per sec
      fprintf(stderr, "Total packets received: %d₩n", nPackets); // nPackets 출력합니다.
      fprintf(stderr, "₩tdulplicate packets: %d₩n", nDupPackets); // nDupPackets 출력합니다.
      fprintf(stderr, "₩tcorrupted packets: %d₩n", nCorruptPackets); // nCorruptPackets 출력합니다.
      fprintf(stderr, "₩tcorrect packets:%d₩n", nCorrectPackets); // nCorrectPackets 출력합니다.
      timersub(&endTime, &startTime, &interval);
      duration = ((float)interval.tv_sec) + interval.tv_usec/1000000. // duration 을 구하는 식입니다.;
      throughput = nCorrectBytes / duration                  // throughput 구하는 식입니다.;
      fprintf(stderr, "Total bytes correctly transfered: %d₩n",nCorrectBytes); // nCorrectBytes 출력합니다.
      fprintf(stderr, "Total time elapsed: %f sec₩n", duration); // duration 을 출력합니다.
      fprintf(stderr, "Throughput: %f Bytes/sec₩n", throughput);// throughput 출력합니다.
      return; } // 리턴
if ( isType(rcvpkt, END, rcvSeq) ) {
   ...   ~ printStatistics( ) ; 종료하기 전에 receiver 는 성능 측정치들을 output 합니다.
   exit(0); }
```

-Receiver 설계-

Receiver State : **Wait for below state**

Event 1 : RDT_RCV(rcvpkt)&&Corrupt(rcvpkt)

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

Event 2 : RDT_RCV(rcvpkt)&&isType(rcvpkt, DATA, rcvSeq)&& notCorrupt(rcvpkt)

Extract(rcvpkt, data); //datalen = rcvlen - sizeof(Header);

데이터를 추출합니다.

deliver_data(data, datalen);

상위 계층에 데이터를 전달하려고 할 때 호출합니다.

ndpkt = make_pkt(ACK, rcvSeq, &sndPacket, 0);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

rcvSeq = (rcvSeq + 1) % 2;

2 로 나눈 나머지 처음에 rcvSeq =0 이므로 0 을 넣으면 나머지가 1 이됨
그 다음에 rcvSeq =1 이므로 1 을 넣으면 2, 2 로 나눈 나머지는 0 이므로
다시 rcvSeq =0 이 됨

Event 3 : RDT_RCV(rcvpkt)&& && notCorrupt(rcvpkt)

1. **Sender로부터 End packet을 받을 때**

if (isType(rcvpkt, END, rcvSeq))

sndpkt = make_pkt(ACK, rcvSeq, &sndPacket, 0);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen);

exit( ); 수신자는 ACK 을 3 번 보내고 종료를 합니다.

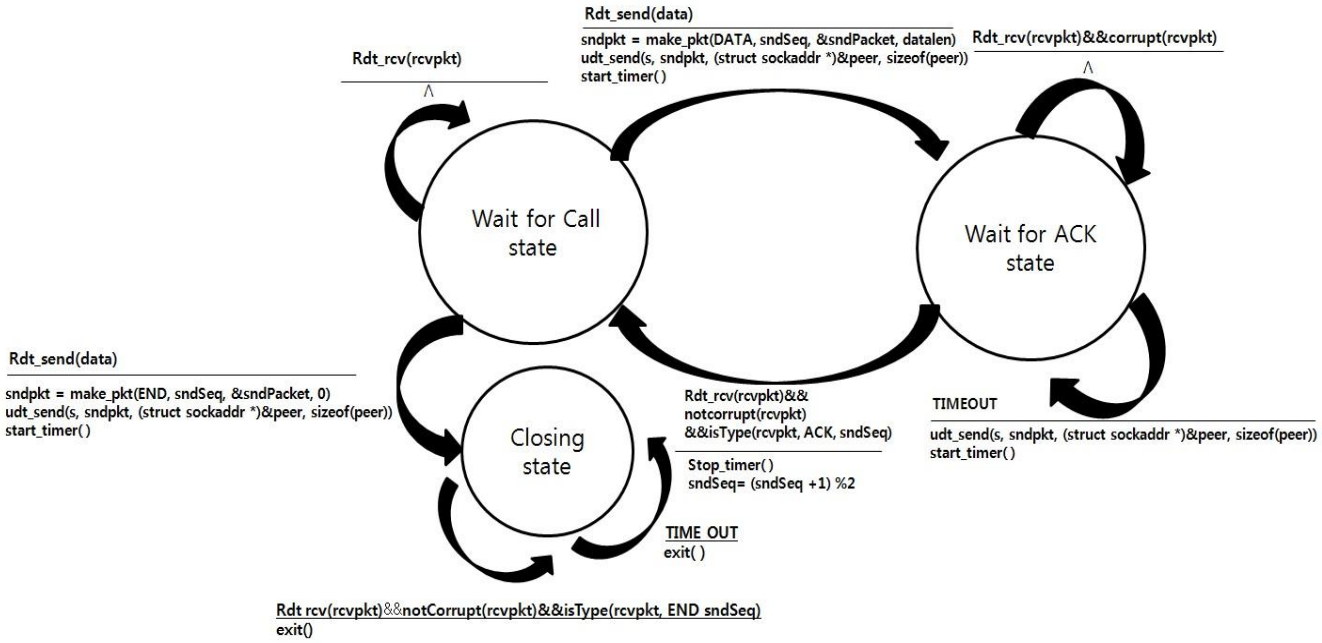
2. **Duplicated packet 일 때**

else

nDupPackets++;

udt_send(s, sndpkt, (struct sockaddr *)&sender, senderlen)

설계한 RDT3.0 Sender FSM



설계한 RDT3.0 Receiver FSM