Master Thesis: Software Engineering Study on Reliability of ARQ wireless protocol

Kevin Yao

Stevens Institute of Technology

Schedule: Jan 14 - May 14 (Tuesday), total 18 weeks

Table of Contents

1 Study Introduction

1.1 purpose

Develop a proposal for Software Engineering study on Reliability of ARQ wireless protocol, based on Sha's reliability model and Prof. Bernstein's Effectiveness Extension of Sha's model.

1.2 Scope

1. Software Reliability theories in Software Engineering
2. Software Development including Fault Tolerance and Error Correction libraries.
3. Computer Network transmission and error correction protocols and implementations
4. Wireless transmission and error correction protocols and implementations

1.3 Software Reliability Overview

Reference**:** Software fault tolerance forestalls crashes: to err is human; to forgive is fault tolerant by Larry Bernstein

The most common reliability model is:

$$R(t) = e^{-\lambda \cdot t},$$

Where λ is the failure rate. It is reasonable to assure that the failure rate is constant even though faults tend to be clustered in a few software components. The software execution is very sensitive to initial conditions and external data driving the software. What appear to be random failures are actually repeatable. The problem in finding and fixing these problems is the difficulty of doing the detective work needed to discover the particular initial conditions and data sequences can trigger the fault so that it becomes a failure. [Musa87]

In a two-state continuous-time Markov chain the parameters to be estimated are failure rate λ and repair rate υ.

The Mean Time Between Failures (MTTF) = $1/\lambda$

The Mean Time To Repair (MTTR) = $1/\upsilon$

The steady-state availability is:

Availability = MTTF/ (MTTF + MTTR) = 1/ (1+ λ / υ)

The goal of Software Fault Tolerance is to make Availability = 1

Prof. Lui Sha's model of reliability based on these postulates:

1. Complexity begets faults. For a given execution time software reliability decreases as complexity increases.
2. Faults are not equal, some are easy to find and fix and others are Hisenbugs. Faults are not random.
3. All budgets have limits so that there is not unlimited time or money to pay for exhaustive testing.

Sha chooses the MTTF = E/kC and the reliability of the system is $R(t) = e^{-kCt/E}$,where

k is a scaling constant,

C is the complexity, where Sha defines complexity as the effort needed to verify the reliability of a software system made up of both new and reused components,

t is the continuous execution time for the program,

E is the development effort that can be estimated by such tools as Checkpoint, COCOMO or Putnam's approach. Development effort is a function of the complexity of the software so the numerator and denominator of the exponent must be factored and R(0) = 1 because all the startup failures are assumed to be removed through classical unit, block and system testing.

Prof. Bernstein's Effectiveness Extension of Reliability Model adds an effectiveness factor to the denominator:

$$R(t) = e^{-kCt/E\varepsilon}$$

ε reflects the investment in software engineering tools, processes and code expansion that makes the work of one programmer more effective. Let ε be the expansion factor that expresses the ability to solve a program with fewer instructions with a new tool such as a complier.

The longer the software system runs the lower the reliability and more likely a fault will be executed to become a failure. Reliability can be improved by investing in tools(ε), simplifying the design (C), or increasing the effort (E) in development to do more inspections or testing than required.

SE Reliability Study on ARQ protocol is the implementation of this model, is the study on the factor C, ε and other factors affecting to a real software development.

2 General Description

2.1 Study Perspective

Quantify the factors of Reliability model and get real data of how these factors work in a real software project

2.2 Study Procedures

1. Study Requirements
2. Requirement Review
3. Software Architecture(4+1architecture)
4. Software Design
5. Software Architecture and Design Review
6. Software Implementation (development), realizing ARQ protocols in C++ with standard library. Prof. Yao will define the algorithm.
7. Code Inspection and software testing
8. Get reliability data from running system testing program in the lab (by running this code in the testing environment)
9. Implement in different C++ libraries, such as fault tolerance library, and Forward Error Correction library for an adoption approach, and simple garbage collection to avoid memory leaks.
10. Code Inspection and software testing
11. Get reliability data of each library-built-in implementation by running in the laboratory (need to integrate the code with the testing bed first)
12. Fix code errors.
13. Analyze the reliability data of various implementations to study how Reliability Model works in the project.

2.3 General Constrains

1. Only has one developer.

2. Selective Repeat ARQ protocol implementation in software only, no communication hardware, such as serial port, wire network equipments, wireless instruments, is involved.
3. The language using for software implementation is C/C++.
4. Development platform is Visual C++ 6.0 on Windows 2000.
5. Testing platform is based on software simulation environment.
6. Software engineering procedures are implemented in all project development phases.
7. The project should be finished by May 14.

2.4 Assumptions and Dependencies

1. We assert that the protocols are correct at the development phase.
2. We project that all the defects, either from protocols, coding or other sources, will be detected during the coming software engineering procedure.

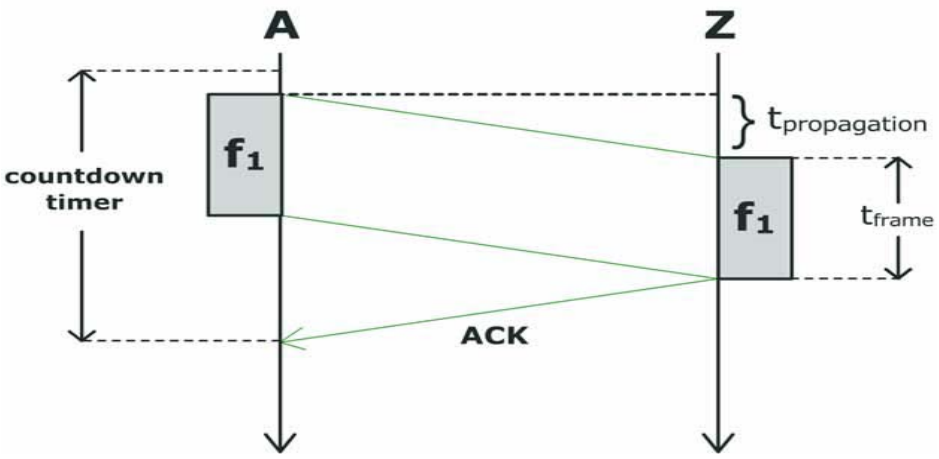
3 Software Requirements

The code will simulate standard Selective Repeat ARQ protocol. It will stop only after correctly sending the assigned number of frames. (Following description of Stop-N-Wait ARQ and Selective Repeat ARQ are refered to CS666/CpE678: Information Networks I, Lecture Notes 10)

3.1 background knowledge

Stop and Wait Protocol

Problem : Low link usage, Low throughput



Reference : http://attila.stevens-tech.edu/~lbernste/cs666lect9fall.html

In the stop and wait protocol A can only have one unacked frame the link to Z at any time. The bandwidth x delay product gives the number of bits (m) that can be in transit from A to Z. For peak link utilization, A needs to send m bits without waiting for the first ack. This is called pipelining or keeping the pipe full. The stop and wait protocol is not very good at this and provides poor link utilization for a >1 and for a < 1.

For error free stop and wait, let T be the total time needed to send one frame A to Z. Then,

T = Tproc + Tprop + TSN+ Tproc + Tprop + Tack ,

Where Tproc = processing time for header analysis, buffering and CRC computation and Tproc << Tprop,

Tprop = propagation time from A to Z,

TSN= time to transmit all the bits in the frame

= (l bits/frame) / b bits/sec and l = m + h

Tack = time to transmit the ack from Z to A and Tack << Tframe

So : T ~ TSN+ 2 Tprop and

Utilization (U) = TSN/ T

= TSN/ (TSN+ 2 Tprop) or letting a = Tprop / TSN

U = TSN/ (1 + 2a), error free stop and wait

Example : ATM Frame = 424 bits; TSN = 424 bits/cell / $1.552.10^9$ bits/sec (OC-3)
= 2.276 μsec/cell

For an error prone link use the results of problem 1-14 and assume that a time out equals twice the propagation delay (2 Tprop).

Then for one failure, the frame must be sent twice ,

T1 = TSN+ Time-out + TSN+ 2 Tprop

= (TSN+ Time-out + TSN+ 2 Tprop)

= (TSN+ 2 Tprop + TSN+ 2 Tprop)

= 2 (TSN+ 2 Tprop) and

for k-1 failures the SNmust be sent E(k) times

Tk = E(k) (TSN+ 2 Tprop)

= (1/ 1-p)(TSN+ 2 Tprop),

where p is the probability that a SNis damaged

then,

U = TSN/ Tk

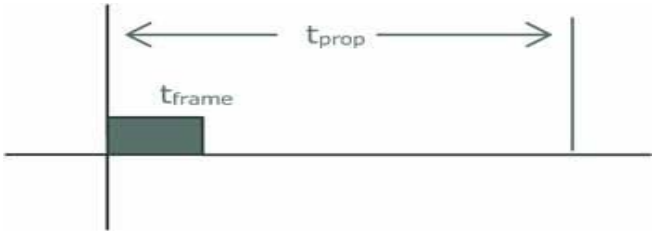
= TSN/ (1/ 1-p)(TSN+ 2 Tprop)

= (1- p) TSN/ (TSN+ 2 Tprop)

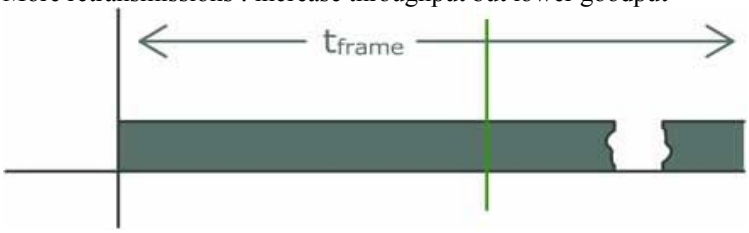
and letting a = Tprop / TSN

U = (1-p) / (1 + 2a), for stop and wait with errors

- **Case a>1** : the link is under utilized



- **Case a<1** : high probability of a SNerror
  More retransmissions : increase throughput but lower goodput



Problem : Low channel – Utilization with Stop and Wait Protocol
Solution : Sliding Window Protocol

Send a number of frames in a transmission window by pipelining.

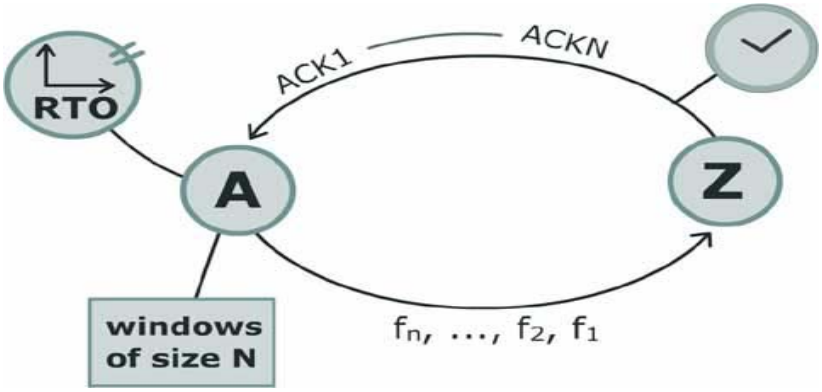
TSN= C_{actual} / 8 x SNsize

Let f be the number of frames
Let M be the number of bits in the header for the sequence number (SN)

Buffer at Z is equal to : f (frames) – RTT / Tframe

$f \leq 2^m - 1$

e.g. f ≤ Z + (RTT / Tframe) mod (SNsize)



⇒ Piggybacking the SN in header from Z to A increases utilization but also increases latency x retransmission for RTO=0

Problem : With piggybacking, Z may not have a message to send to A so Z does not ACK + A resends
Solution :

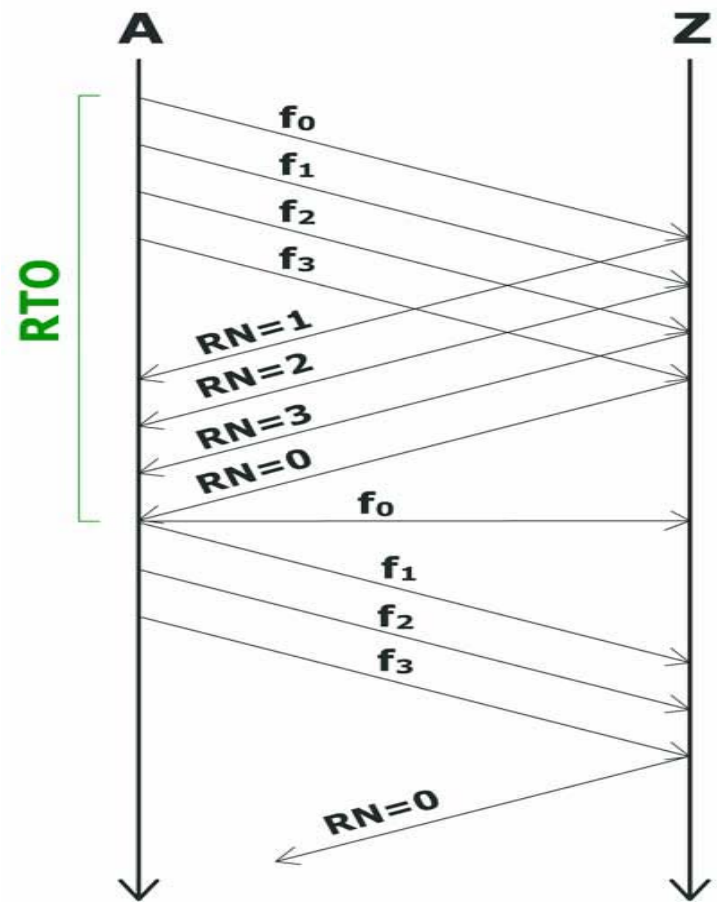
1- Add a RTO to Z that launches an independent ACK frame
2- Add windows and counters to A and Z for coordinating and receiving data frames

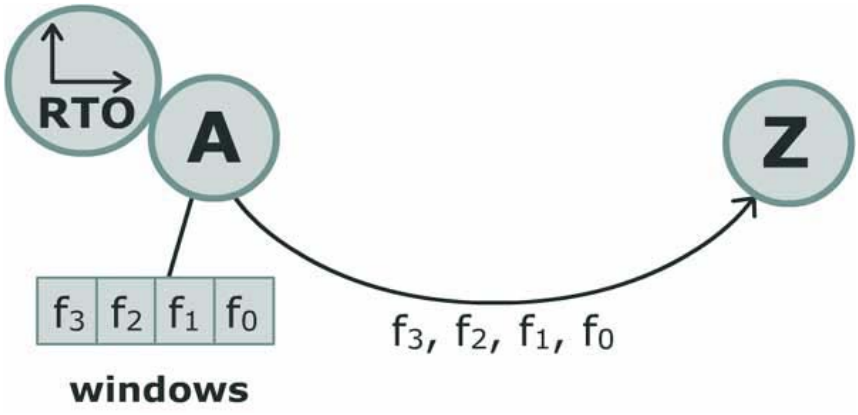
Problem : T_{ACK} must be faster than the retransmit RTO by at least $T_{prop} + T_{frame}$.
Solution :

1- Set T_{ACK} = RTO / 2 at the cost of extra ACKS.
2- Establish a sending window
SWS = Sending Window Size
RWS = Receiving Window Size

Let Z acks with RN (Request Number) = SN+1

RN = SN+1 means "explicit acknowledgment of message SN and implicit ACK of message ≤ SN-1"



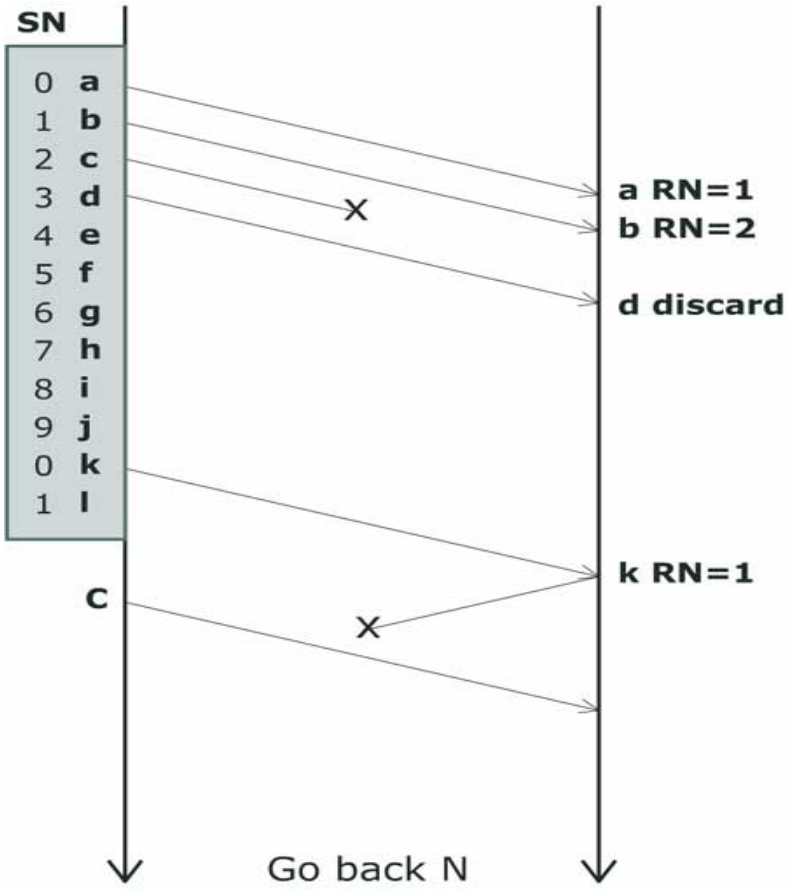


⇒ What happens if a SNis lost ?

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
SN	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

These frames are waiting to be acked
These frames are waiting to be sent

S_{LAST} (last transmitted frame) = 6
S_{RECENT} (most recently transmitted frame) = 7
SWS = 7 (avoid ambiguity)



SN = [0,7]

m = 3
SWS = $2^m - 1 = 7$

⇒ In general **SWS = 2^m – 1** (RWS=1) to prevent Sequence Number Rollover

Selective Repeat ARQ :

Selective Repeat error recovery is a procedure which is implemented in some communications protocols to provide reliability. It is the most complex of a set of procedures which may provide error recovery, it is however the most efficient scheme. Selective repeat is employed by the TCP transport protocol.

Features required for Selective Repeat ARQ

- To support Go-Back-N ARQ, a protocol must number each PDU which is sent. (PDUs are normally numbered using modulo arithmetic, which allows the same number to be re-used after a suitably long period of time. The time period is selected to ensure the same PDU number is never used again for a different PDU, until the first PDU has "left the network" (e.g. it may have been acknowledged)).
- The local node must also keep a buffer of all PDUs which have been sent, but have not yet been acknowledged.
- The receiver at the remote node keeps a record of the highest numbered PDU which has been correctly received. This number corresponds to the last acknowledgement PDU which it may have sent.

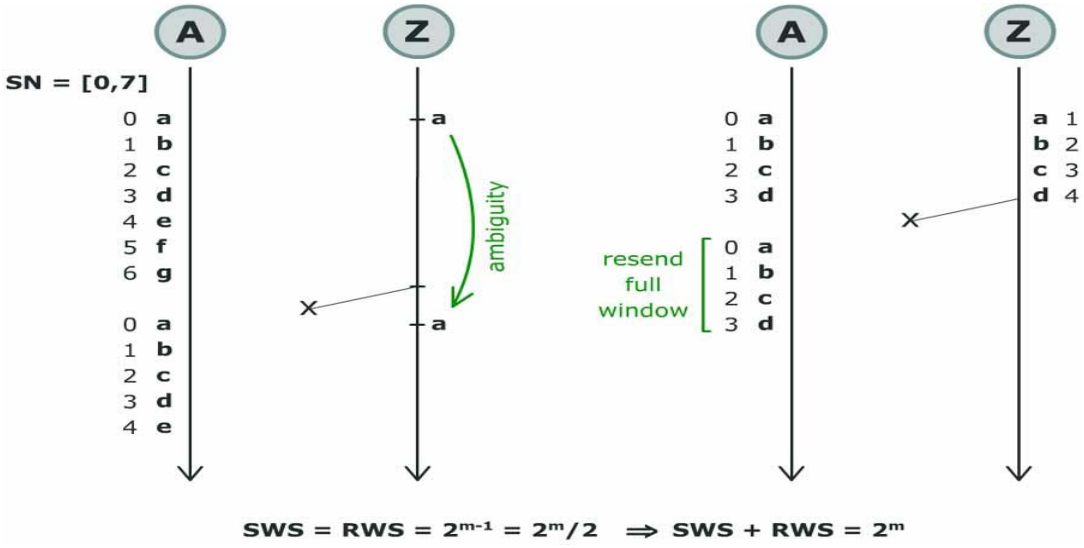
The above features are also required for Go-Back-N, however for selective repeat, the receiver must also maintain a buffer of frames which have been received, but not acknowledged.

Recovery of lost PDUs using Selective Repeat ARQ

The recovery of a corrupted PDU proceeds in four stages:

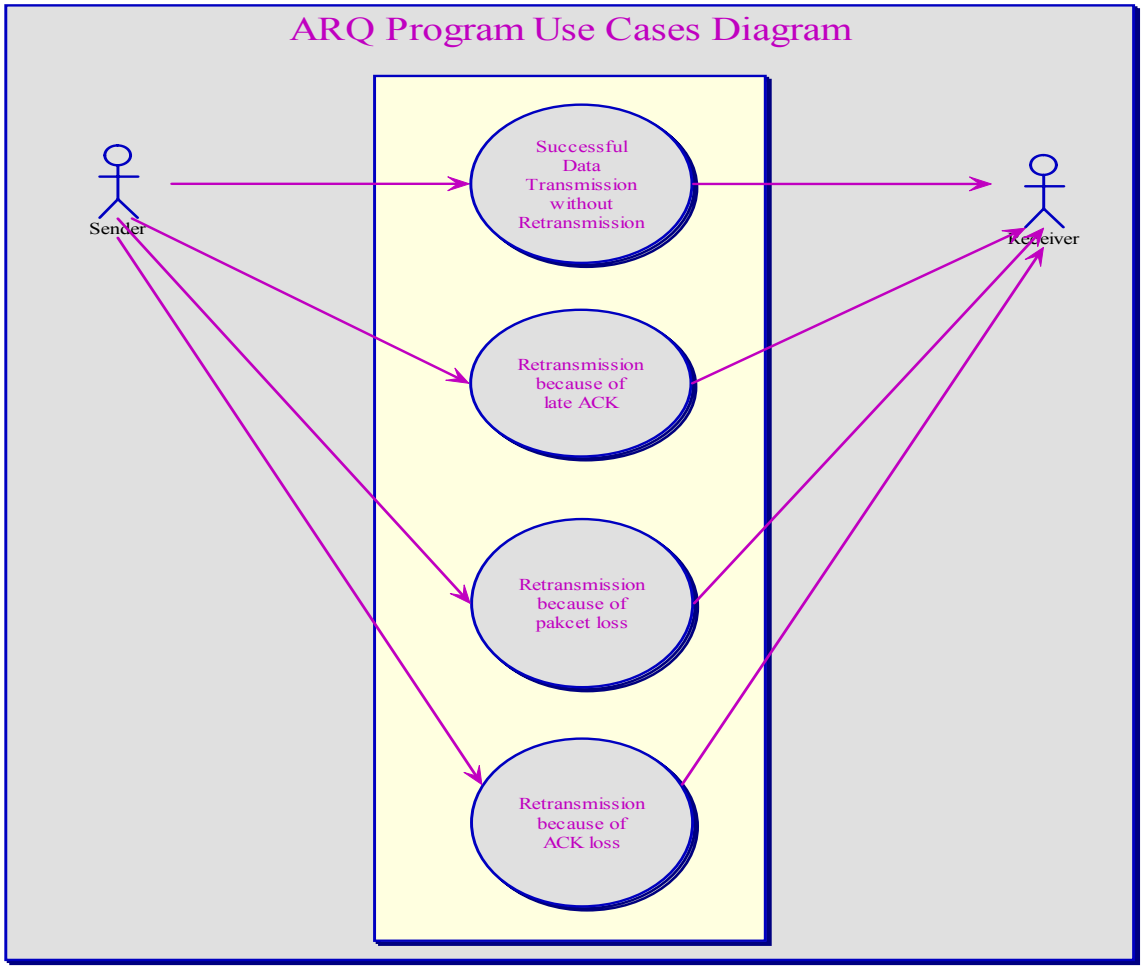
- First, the corrupted PDU is discarded at the remote node's receiver.
- Second, the remote node requests retransmission of the missing PDU using a control PDU (sometimes called a Selective Reject). The receiver then stores all out-of-sequence PDUs in the receive buffer until the requested PDU has been retransmitted.
- The sender receives the retransmission request and then transmits the lost PDU(s).
- The receiver forwards the retransmitted PDU, and all subsequent in-sequence PDUs which are held in the receive buffer. ”

PDU = Protocol Data Unit – Usually we talk about PDU (send by A) and ACK PDU (send by Z)

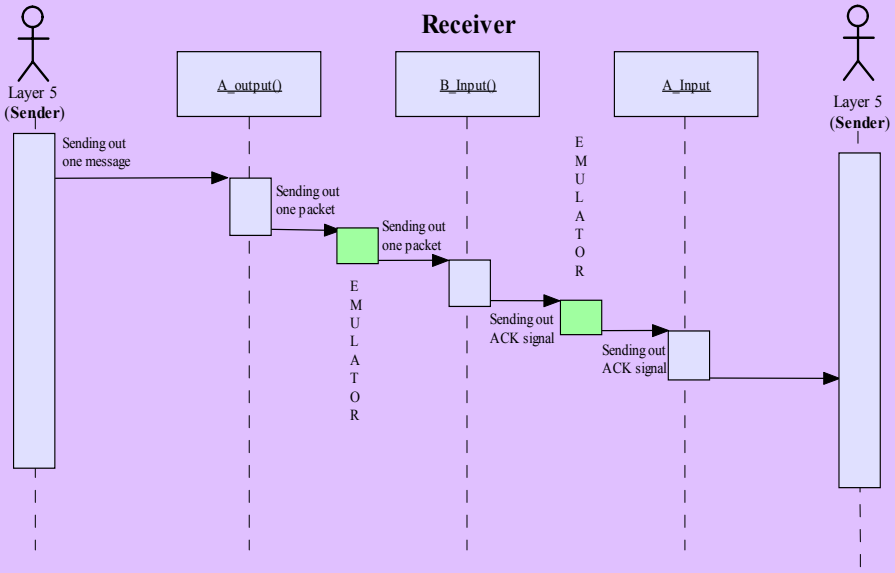


3.2 Functional Requirements

Use Cases Diagram:

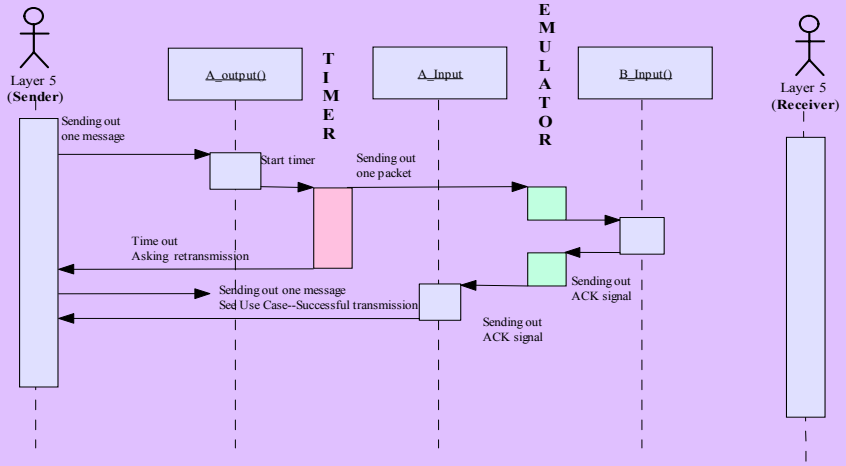


Use Case -- Successful Transmission

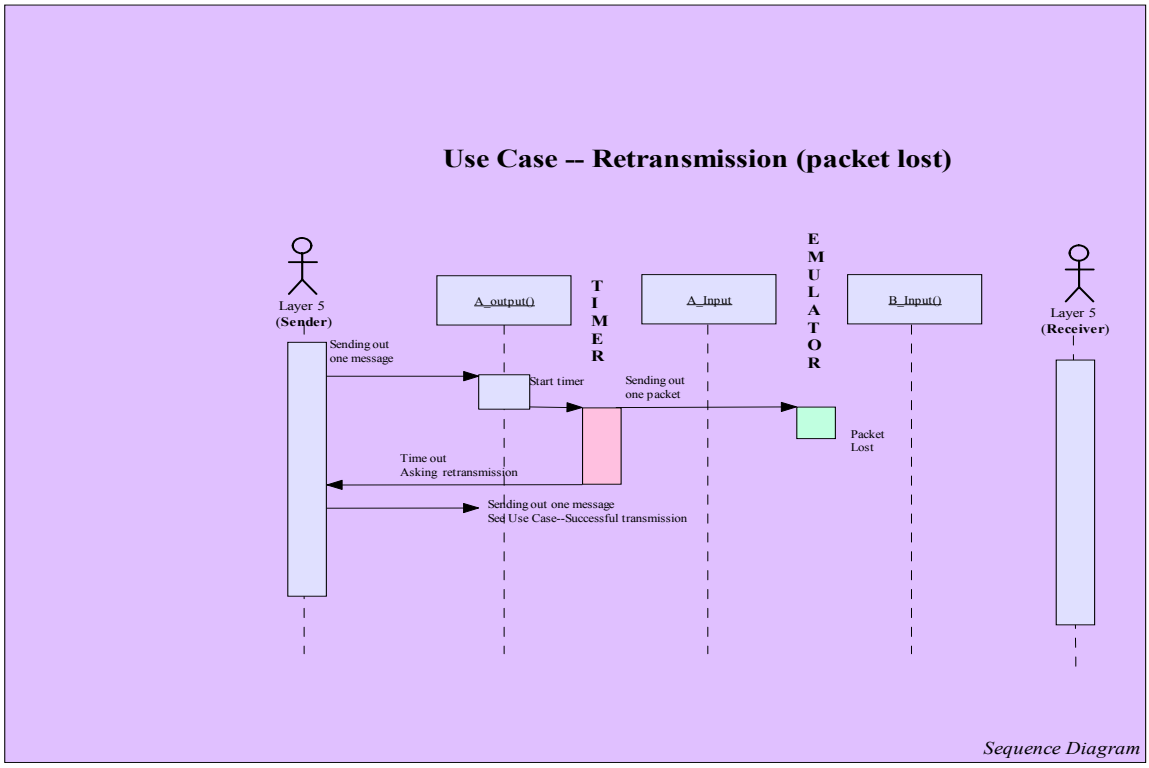


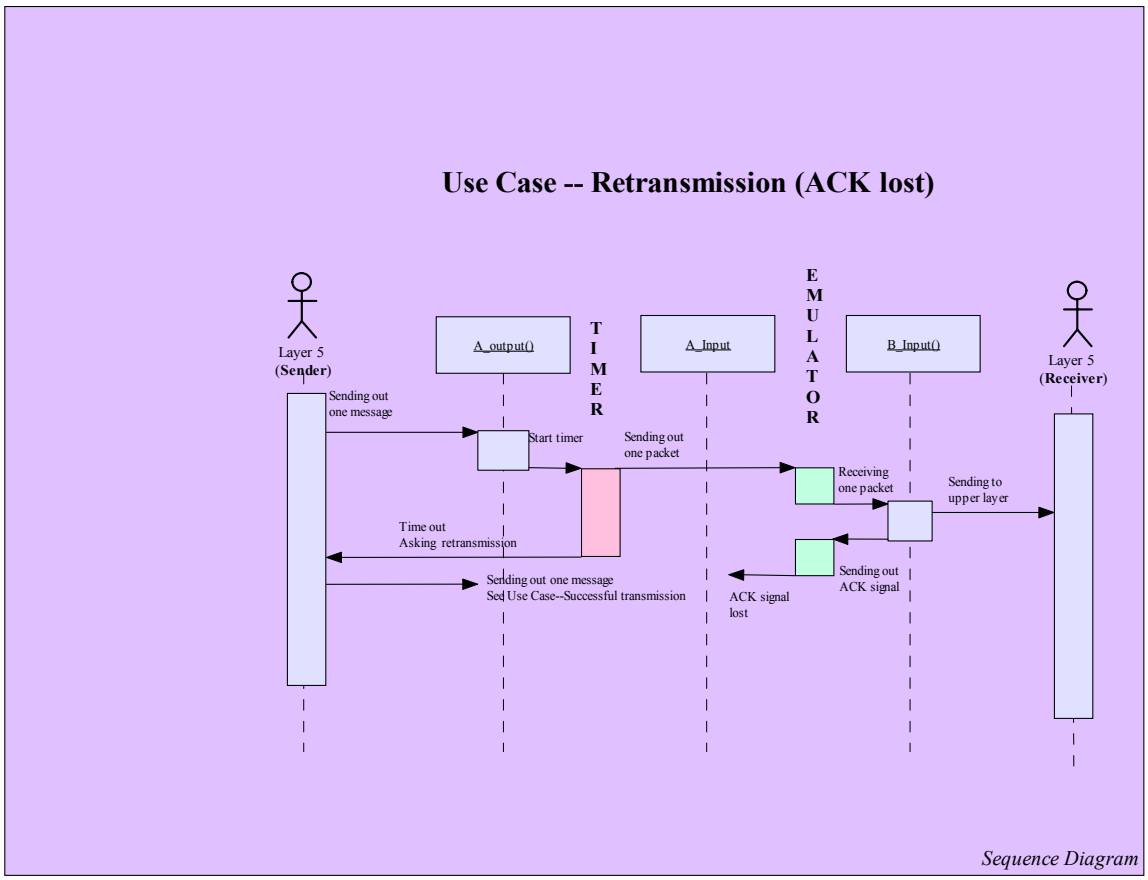
Sequence Diagram

Use Case -- Retransmission (late ACK)



Sequence Diagram





3.2 Implementation requirements:

Transmitter side:

1. Generate data messages
   i. create a 20 bytes long string as a message
   ii. generate assigned number of messages
2. Encapsulate message into frames (under a certain format)
   i. Encapsulate a message into a frame which has sequence number, ACK number, checksum and a message string.
3. Window control
   i. Control a window with a fixed size.
   ii. Under following routines:
      a. Keep sending frames in the window until there is no frame not yet been sent within a window.
      b. Only as soon as the first N (1<=N<=WindowSize) number of frames are acknowledged, the window start point will be moving to the position of StartPoint + N +1 and end point will be StartPoint + WindowSize. No other condition can a window change its position.
      c. There is only one window on the transmitter side.

4. Timer control
   i. Only a single timer is at the transmitter side.
   ii. Timer is set only for the oldest frame that has been transmitted but not yet acknowledged.
   iii. The timer will count down a user assigned time.
   iv. When timer = 0, the sender will resend the frame on which the timer is set.
   v. When an ACK is received for the frame on which the timer is set, the timer will be stopped and restarted for the next unacknowledged frame if existed.
   vi. No timer limit is set to quit the program when the same frame has been sent out at timer limit times without ACK received yet.

Transmission simulation:

1. Simulate the lowest OSI/ISO three layers data transmission from sender to receiver.

2. Simulate delayed (more time than average transmission time) frame transmission.
3. Simulate frame loss transmission.
4. Simulate ACK transmission from receiver to sender.

Receiver side:

1. receiver side window control
    i. Only when the receiver get windowSize number of correct frames, can it transfer them to the upper layer. Otherwise, the receiver should do nothing but wait.
    ii. Send an ACK to the sender when a frame is successfully received with no error.
    iii. Discard the duplicated frame that already correctly received.
    iv. Discard the frame that is received corrupt.
2. Validate checksums to decide if frame is correctly transmitted.
3. Reorder frames received in the window.
4. Draw messages from frames and send them to the network layer
5. Connect messages with original order.

3.4 Configuration control requirements

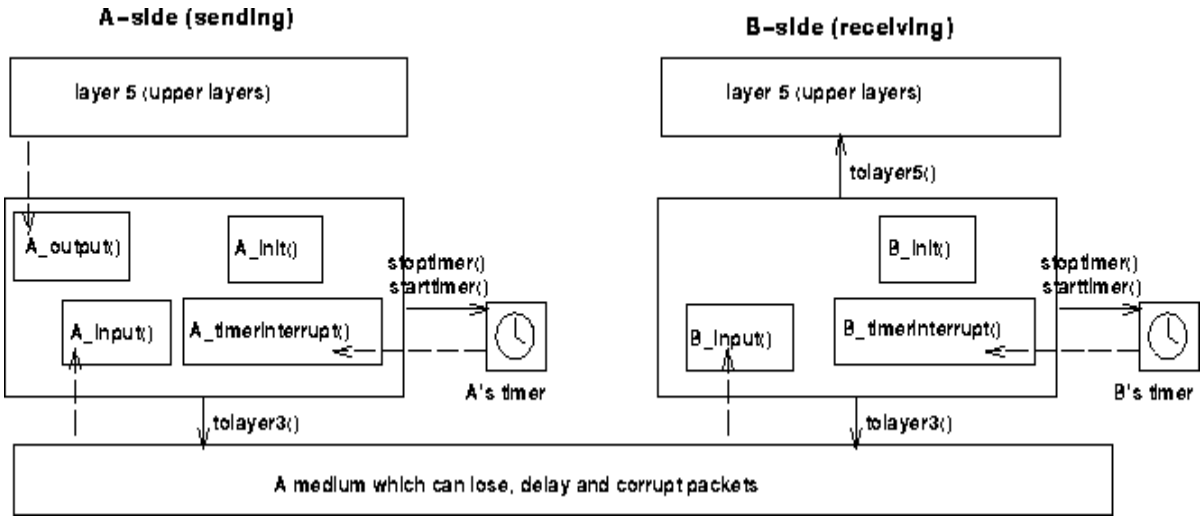
1. Development Environment: MS Visual C++ 6.0, Windows 2000 Prof., PC PIII 866, 512MB RAM.
2. Testing Environment: Windows NT 4.0 with SP6a, PC PIII 866, 128MB RAM.

3.5 External Interface requirements

1. The Interface between the ARQ code and the testing system platform --- Libft fault tolerance library that would be built into the ARQ code.
2. Interface between the people and ARQ code during the development phase.
    i. DOS-look interface prompting user to input frames number that need to transmit; frame loss rate; frame corruption rate; average transmission time from sender to receiver; detail level of the log that record the process during the transmission.
    ii. Log file for user to review the transmission.

4 Architecture and Design

Reference: Northeastern University COM 3515 Programming Assignment 2



Layered structure and design diagram

Based on the requirement, the program will run in a simulated hardware/software environment. It will have three parts: A-side(sender), B-side(receiver) and a network emulator to simulate the network environment. The overall structure of the environment is showing above.

The program only implements unidirectional transfer of data (from A to B). Of course, the B side will have to send frames to A to acknowledge receipt of data. Following procedures are to be implemented.

The unit of data passed between the upper layers and the transmission layer is a *message,* which is declared as:

```
struct msg {
  char data[20];
  };
```

The unit of data passed between transmission layer and the lower layer is the *frame,* which is declared as:

```
struct pkt {
   int seqnum;
   int acknum;
```

```
  int checksum;
  char payload[20];
  };
```

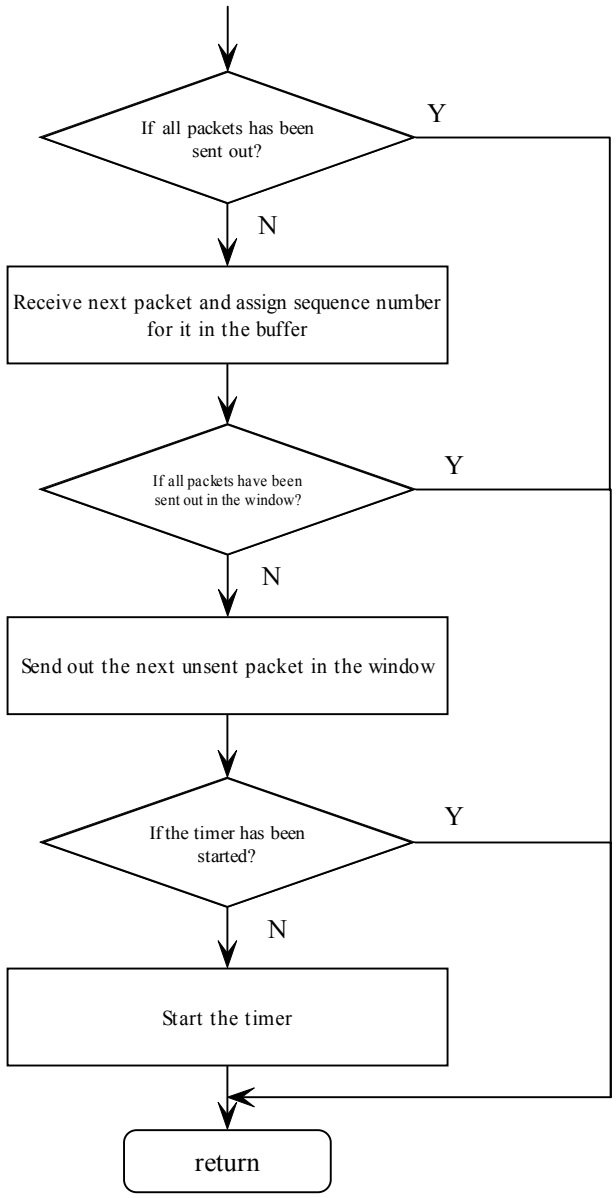
A_output(message),

where message is a structure of type msg, containing data to be sent to the B-side.

This routine will be called whenever the upper layer at the sending side (A) has a message to send. It is the job of Selective ARQ protocol to ensure that the data in such a message is delivered in-order, and correctly, to the receiving side upper layer.

A_output() routine will be called when there are outstanding, unacknowledged messages in the medium - implying that there needs to buffer multiple messages in the sender. Also, window control is needed because of the nature of Selective-Repeat: sometimes the sender will be called but it won't be able to send the new message because the new message falls outside of the window.

A_output Diagram



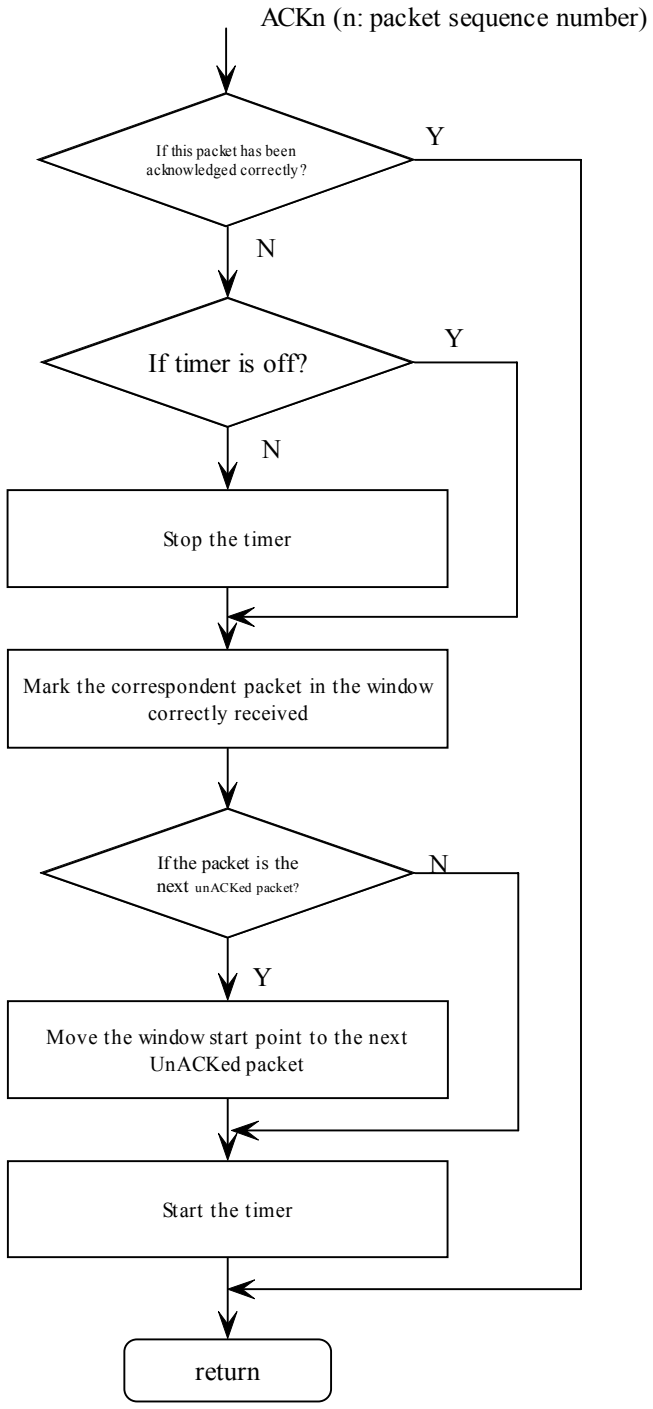
A_input(frame),

where frame is a structure of type pkt;

ACK is a structure of type pkt with acknum = 1.

This routine will be called whenever a frame sent from the B-side (i.e., as a result of a tolayer3() being done by a B-side procedure) arrives at the A-side. frame is the (possibly corrupted) frame sent from the B-side.

A_input Diagram

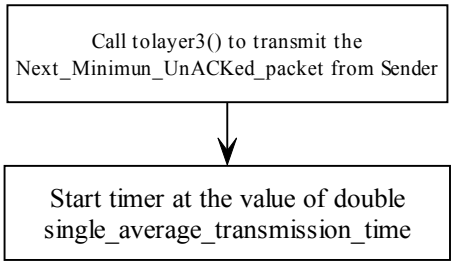


A_timerinterrupt()

This routine will be called when A's timer expires (thus generating a timer interrupt). Control the retransmission of frames is needed here. See starttimer() and stoptimer() below for how the timer is started and stopped.

Note that there is only one timer.

A_timerinterrupt Diagram



A_init()

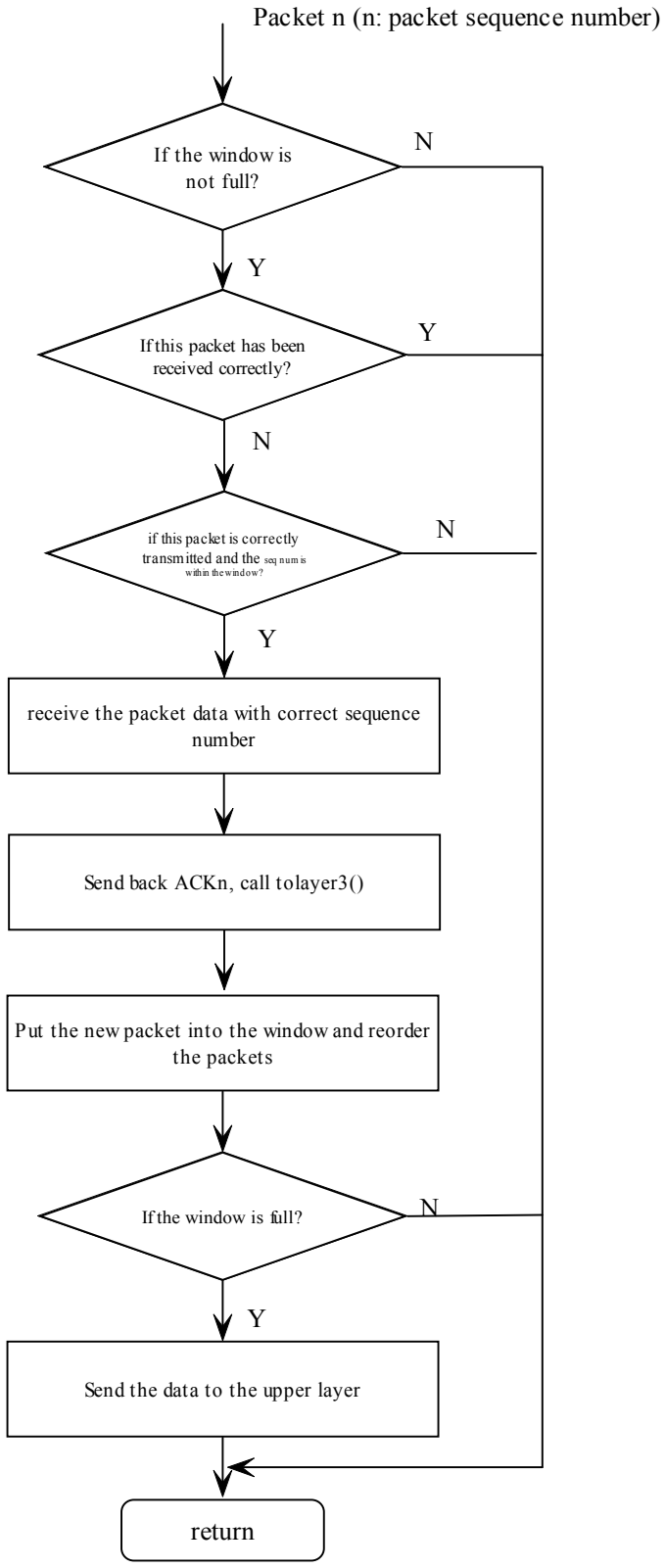
This routine will be called once, before any of other A-side routines are called. It can be used to do any required initialization.

B_input(frame),

where frame is a structure of type pkt.

This routine will be called whenever a frame sent from the A-side (i.e., as a result of a tolayer3() being done by a A-side procedure) arrives at the B-side. frame is the (possibly corrupted) frame sent from the A-side.

B_input Diagram



B_init()

This routine will be called once, before any of other B-side routines are called. It can be used to do any required initialization.

Software Internal Interfaces

The procedures described above are the ones which are used to control the transmission and implement the Selective Repeat ARQ. Following routines are part of the network emulator:

starttimer(calling_entity, increment),

where calling_entity is either 0 (for starting the A-side timer) or 1 (for starting the B-side timer), and increment is a *float* value indicating the amount of time that will pass before the timer interrupts. A's timer should only be started (or stopped) by A-side routines, and similarly for the B-side timer.

Note that when the timer expires, this automatically stops the timer. it should be restarted again if another timer is needed in the future.

stoptimer(calling_entity),

where calling_entity is either 0 (for stopping the A-side timer) or 1 (for stopping the B-side timer).

tolayer3(calling_entity, frame),

where calling_entity is either 0 (for the A-side send) or 1 (for the B-side send), and frame is a structure of type pkt.

This routine will have the frame sent into the network, destined for the other entity.

tolayer5(calling_entity, message),

where calling_entity is either 0 (for A-side delivery to layer 5) or 1 (for B-side delivery to layer 5), and message is a structure of type msg.

This routine will cause data to be passed up to layer 5.

The simulated network environment

A call to procedure tolayer3() sends frames into the medium (i.e., into the network layer). Your procedures A_input() and B_input() are called when a frame is to be delivered from the medium to your protocol layer.

The medium is capable of corrupting and losing frames. It will not reorder frames.

Following values are related to the simulated network environment:

- **Number of messages to simulate.** The emulator will stop as soon as this number of messages have been passed down from layer 5, regardless of whether or not all of the messages have been correctly delivered.

  Note that if this value is set to 1, the program will terminate immediately, before the message is delivered to the other side. Thus, this value should always be greater than 1.

- **Loss.** Users are asked to specify a frame loss probability. A value of 0.1 would mean that one in ten frames (on average) will be lost. Frame loss is meaning that the frame is lost, failing to reach the destination because of a noisy channel, external interference, etc.

- **Corruption.** Users are asked to specify a frame corruption probability. A value of 0.2 would mean that one in five frames (on average) will be corrupted. Frame corruption is the frame that reaches the destination with wrong data, or wrong sequence number, and so on due to reasons such as a noisy channel, external interference.

  Note that the contents of payload, sequence, ack, or checksum fields can be corrupted.

- **Tracing.** Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to frames and timers). A tracing value of 0 will turn this off.

- **Average time between messages from sender's layer5.** Users can set this value to any non-zero, positive value. Note that the smaller the value be chosen, the faster frames will be arriving to the sender.

5 Software Development and process

1. Write documentation, trying to finish requirement document, architecture document and even test plan first before coding. But under time pressure I started coding before all documents are ready.
2. Rush into programming without well understanding.
3. Stuck after finishing Stop-N-Wait ARQ, because old simulation design cannot meet the requirement of Selective Repeat ARQ
4. Have a design change --- Event driven solution for network simulation
5. New implementation of the Stop-N-Wait ARQ
6. Selective Repeat ARQ with flow control
7. Unit Test, bug fixing
8. A bug showed when sending 1000 frames with 50% lost rate, see code in Appendix 4, Bug1 Fix.
9. By Apr.1st, the program can run smoothly transmitting 1000 frames with 50% frame loss rate.
10. April.17th, I finished making up documentation of the process, and architecture and design document as well.
11. Test configuration set up.
12. Building fault tolerance program.

See Appendix 1 for detailed experiences.

6 Software Test

6.1 Test Configuration

Libft (Swift) Fault Tolerance software library runs on MS Windows NT 4.0 with service pack SP6a.

6.2 Feature test

Test cases were executed under the fixed window size of 8.

Passed: in black Failed: in pink

1. Sending out 0 frames with 0% loss rate, 0% corruption rate.
2. Sending out 0 frames with 20% loss rate, 0% corruption rate.
3. Sending out 0 frames with 50% loss rate, 0% corruption rate.
4. Sending out 0 frames with 100% loss rate, 0% corruption rate.
5. Sending out 0 frames with 0% loss rate, 20% corruption rate.
6. Sending out 0 frames with 0% loss rate, 50% corruption rate.
7. Sending out 0 frames with 0% loss rate, 100% corruption rate.
8. Sending out 0 frames with 20% loss rate, 20% corruption rate.
9. Sending out 0 frames with 50% loss rate, 50% corruption rate.
10. Sending out 0 frames with 100% loss rate, 100% corruption rate.

11. Sending out 1 frame with 0% loss rate, 0% corruption rate.
12. Sending out 1 frame with 20% loss rate, 0% corruption rate.
13. Sending out 1 frame with 50% loss rate, 0% corruption rate.
14. Sending out 1 frame with 100% loss rate, 0% corruption rate.
15. Sending out 1 frame with 0% loss rate, 20% corruption rate.
16. Sending out 1 frame with 0% loss rate, 50% corruption rate.
17. Sending out 1 frame with 0% loss rate, 100% corruption rate.
18. Sending out 1 frame with 20% loss rate, 20% corruption rate.
19. Sending out 1 frame with 50% loss rate, 50% corruption rate.
20. Sending out 1 frame with 100% loss rate, 100% corruption rate.

21. Sending out 7 frames with 0% loss rate, 0% corruption rate.
22. Sending out 7 frames with 20% loss rate, 0% corruption rate.
23. Sending out 7 frames with 50% loss rate, 0% corruption rate.
24. Sending out 7 frames with 100% loss rate, 0% corruption rate.
25. Sending out 7 frames with 0% loss rate, 20% corruption rate.
26. Sending out 7 frames with 0% loss rate, 50% corruption rate.
27. Sending out 7 frames with 0% loss rate, 100% corruption rate.

28. Sending out 7 frames with 20% loss rate, 20% corruption rate.
29. Sending out 7 frames with 50% loss rate, 50% corruption rate.
30. Sending out 7 frames with 100% loss rate, 100% corruption rate.

31. Sending out 8 frames with 0% loss rate, 0% corruption rate.
32. Sending out 8 frames with 20% loss rate, 0% corruption rate.
33. Sending out 8 frames with 50% loss rate, 0% corruption rate.
34. Sending out 8 frames with 100% loss rate, 0% corruption rate.
35. Sending out 8 frames with 0% loss rate, 20% corruption rate.
36. Sending out 8 frames with 0% loss rate, 50% corruption rate.
37. Sending out 8 frames with 0% loss rate, 100% corruption rate.
38. Sending out 8 frames with 20% loss rate, 20% corruption rate.
39. Sending out 8 frames with 50% loss rate, 50% corruption rate.
40. Sending out 8 frames with 100% loss rate, 100% corruption rate.

41. Sending out 9 frames with 0% loss rate, 0% corruption rate.
42. Sending out 9 frames with 20% loss rate, 0% corruption rate.
43. Sending out 9 frames with 50% loss rate, 0% corruption rate.
44. Sending out 9 frames with 100% loss rate, 0% corruption rate.
45. Sending out 9 frames with 0% loss rate, 20% corruption rate.
46. Sending out 9 frames with 0% loss rate, 50% corruption rate.
47. Sending out 9 frames with 0% loss rate, 100% corruption rate.
48. Sending out 9 frames with 20% loss rate, 20% corruption rate.
49. Sending out 9 frames with 50% loss rate, 50% corruption rate.
50. Sending out 9 frames with 100% loss rate, 100% corruption rate.

51. Sending out 100 frames with 0% loss rate, 0% corruption rate.
52. Sending out 100 frames with 20% loss rate, 0% corruption rate.
53. Sending out 100 frames with 50% loss rate, 0% corruption rate.

54. Sending out 100 frames with 100% loss rate, 0% corruption rate.
55. Sending out 100 frames with 0% loss rate, 20% corruption rate.
56. Sending out 100 frames with 0% loss rate, 50% corruption rate.
57. Sending out 100 frames with 0% loss rate, 100% corruption rate.
58. Sending out 100 frames with 20% loss rate, 20% corruption rate.
59. Sending out 100 frames with 50% loss rate, 50% corruption rate.
60. Sending out 100 frames with 100% loss rate, 100% corruption rate.

---

61. Leave the program there for 4 hours, doing nothing.

---

62. Sending out 100 frames with 200% loss rate, 0% corruption rate
63. Sending out 100 frames with 0% loss rate, 200% corruption rate

Result validation checked out one more bug: one of the parameters used in the window control for pointing out the next being sent frame exceeded the window size. It led to sending the wrong frames. During the test, this bug never broke the program. See code in Appendix 5, Bug2 Fix.

6.3 Stress test with/without Rejuvenation

1. Sending out 1000 frames 1000 times, with 0% loss rate and 0% corruption rate.
2. Sending out 1000 frames 1000 times, with 10% loss rate and 10% corruption rate.
3. Sending out 1000 frames 1000 times, with 50% loss rate and 50% corruption rate.
4. Sending out 1000 frames 1000 times, with 100% loss rate and 100% corruption rate.

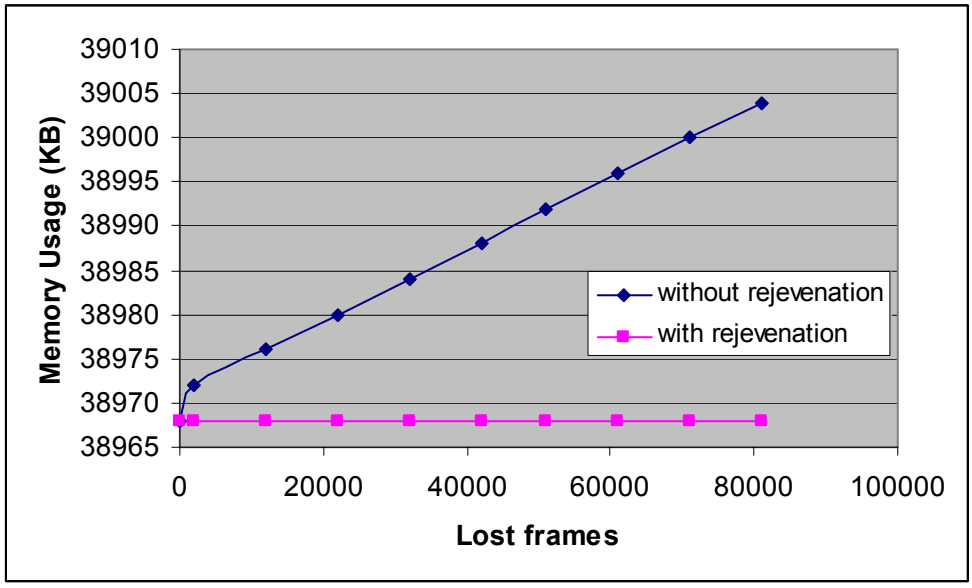
There is a memory leak in the code (See attachment A). The memory leak operates under all conditions but does not cause the software to fail except under the most stressful conditions. The stress test uncovered the fault.

If the system was used in production, a random amount of memory would have been consumed .Other programs running in the same memory space might have randomly failed for leak of memory. The fault becomes coupled from one program to another.

The stress test was effective in detecting the fault so that a potential system failure could be avoided.

After fault tolerance library (libft) was built in (See code in Appendix 3), above stress test cases were repeated. At the situation with 100% loss rate, the program was survived, although it never stopped. The memory usage was never going up and kept steady. The memory leak failure was avoided.

The chart following shows the result of the testing:



6.4 Reliability Test:

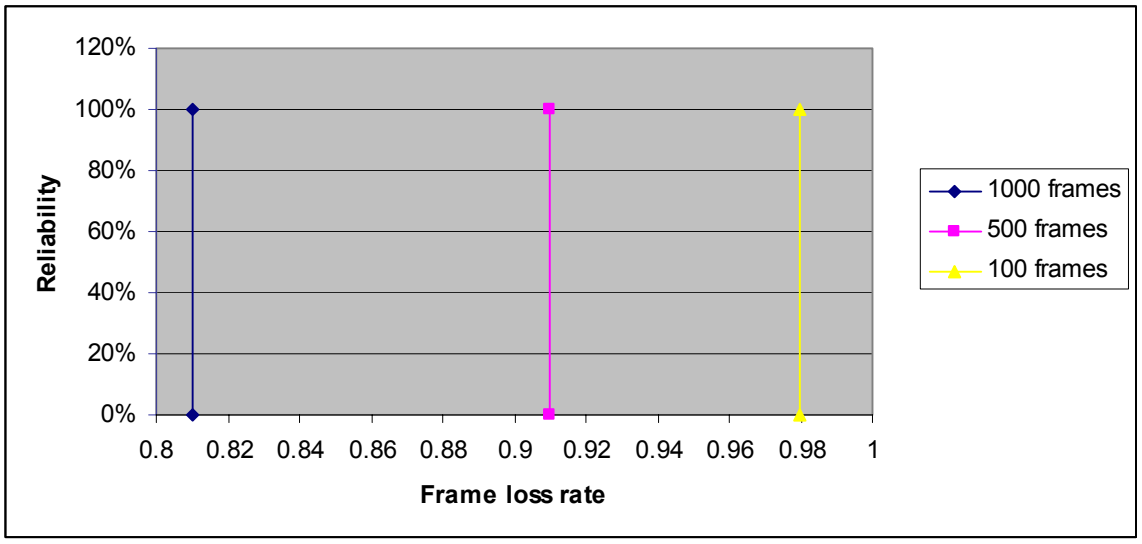
1. Sending out 1000 frames, with 90% loss rate.
2. Sending out 1000 frames, with 80% loss rate.
3. Sending out 1000 frames, with 81% loss rate.
4. Sending out 1000 frames, with 82% loss rate.
5. Sending out 1000 frames, with 81.5% loss rate.

6. Sending out 500 frames, with 90% loss rate.
7. Sending out 500 frames, with 91% loss rate.
8. Sending out 500 frames, with 92% loss rate.

9. Sending out 100 frames, with 90% loss rate.
10. Sending out 100 frames, with 95% loss rate.
11. Sending out 100 frames, with 97% loss rate.
12. Sending out 100 frames, with 98% loss rate.
13. Sending out 100 frames, with 99% loss rate.

After fault tolerance library (libft) was built in (See code in Attachment2) to reset the program every 100 frames were sent, above 1000 frames test cases were repeated. At the situation with 82% loss rate, the program successfully finished. The program can finish under 98% loss rate, the same result as 100 frames.

The following chart shows the result (assuming all test cases with loss rate between 81.5% and 82% when 1000 frames are sent, loss rate between 91% and 92% when 500 frames are sent and loss rate between 98% and 99% when 100 frames are sent, are all failed):



The tests above are for selective repeat program. Same test cases were executed for the Stop-and-Wait program. It failed also at the 100% frame loss rate, with fluctuant memory usage (200KB difference).

7 Other tests and analysis

Measurements

1. Source Lines of Code.
2. Repeat times: for Stop-n-Wait = 2, Selective Repeat = 1
3. Reliability $R(t) = e^{-kCt/E\varepsilon}$

Window size -- Efficiency analysis

During testing, I found window size can affect much of the transmission efficiency. When I tested transferring 1000 frames with window-size 8 and data loss rate 0%, it finished at about 106 time units. If window size is changed to 20, it finished at 99 time units; with 40 window, 97 time units; 80 window size, 96 time units; 500 window size, 96 time units. It needs more testing data to find the regular relationship between window size and transmission number of frames.

With the current selective repeat algorithm, by the time passing, the program behavior is more and more like stop-and-wait ARQ in the small window size, such as 8 in sending 1000 frames totally. Proper bigger window size will help the program improve its efficiency.

With 10% loss rate, 1000 frames were sent and received in nearly 232 time units. With 50% loss rate, about 913 time units are needed. The transmission with 80% loss rate needs 6012 time units to finish. Loss rate is definitely affect the time significantly. (All time units data are the average of 10 test results.)

Software reliability analysis

Until now, after I started testing, I found three errors for selective repeat. It didn't pass 17 test cases out of 63. It is related to my effort and time. It shows that in software reliability theory, increasing the effort (E) to do more inspections and testing will add more reliability.

Reliability model for Selective Repeat ARQ(no fault tolerance):

$$R(t) = e^{-kCt/E\varepsilon}$$

C: complexity---represented by LOC = 745

t: time for running

E: development effort, use 1 to represent first time development.

Finally the reliability is

$$R(t) = e^{-745kt/\varepsilon}$$

After Fault Tolerance ability was built in, the LOC is 820

$$R(t) = e^{-820kt/\varepsilon}$$

I found only one error in Stop-n-Wait ARQ program. I implemented it two times and it has much simpler algorithm than selective repeat ARQ.

Reliability for stop and wait ARQ is:

E: use 2 to represent that the effort has been put 2 times

$$R(t) = e^{-620\,kt/2\varepsilon} = e^{-310\,kt/\varepsilon}$$

Fault tolerance capability

Fault tolerance library libft helps to prevent program crush from memory exhaustion. But it does not help memory insensitive bugs, such as the Bug2 (Appendix 5). However, it does help to handle known memory bugs and gives rejuvenation ability.

It is very useful on stress test.

Software development effort analysis

I found Software testing time spent was the same as for coding.

Reference:

1. Software fault tolerance forestalls crashes: to err is human; to forgive is fault tolerant, Larry Bernstein

2. Stevens Institute of Technology CS666/CpE678: Information Networks I, Lecture Notes 10

3. http://attila.stevens-tech.edu/~lbernste/cs666lect9fall.html

4. Northeastern University COM 3515 Programming Assignment 2

APPENDIX 1

Detailed Experience Description

- Requirement elicitation---Selective Repeat ARQ, Sliding Window control, time out control.
- Architecture design---Layered architecture
- Implementation:

This description is based on memory and logbook I made during implementation, written after I finished coding. Here is my record of activities:

1. Write documentation, trying to finish requirement document, architecture document and even test plan first before coding. But under time pressure I started coding before all documents are ready.
2. Rush into programming without well understanding.

I talked with Prof. Yao. He described how ARQ works and gave me good reference document about ARQ. I browse some data communication books, didn't ask Prof. Yao many questions. After I thought I understand what ARQ is, I wrote some requirements roughly and started to code. It is about Feb.20, 2003. (At the time, I didn't understand this project in whole. I knew I must implement the ARQ, then do testing, and collect reliability data. But how, where, or when to test, what to get, and so on were not clear enough for writing. Even after Prof. Bernstein gave me some of the answers, because I understood only a little about data communication and had no software engineering experience, I did not fully understand. I thought I'd better get the code done ASAP, and then I would know what's next.

I knew, as a software engineering major student, I should have finished requirement, architecture and design document first, and all should be reviewed and passed by professors or some other experts before I code. I didn't because I thought documentation is only for helping me understand the problem. Since I understand the problem, I am going to start coding, leaving documentation to later. (Actually I say I understand it, it's just the behaviors of ARQs, but I thought it is good enough to start coding. But now I know I should have gotten the detail level as what is going to handle in coding, so that I would not have to go back to do it again.) (Commented by prof.: both building a simple prototype and doing the design are important.)

I believe by coding, I would learn more and get deeper understanding. At the time I didn't quite sure what the relationships are among buffer control, flow control, sliding windows, and timer control and how they

interact. I understood the easy Stop-N-Wait and I can finish it soon. I believed by doing SW would lead me to SR easily.

There was a conflict, I say I understand ARQ but later I say I could learn more by coding. I dig hard in my mind trying to connect those two things and find another element was working there: code's value to a programmer.

I am a programmer. Only code represents value to a programmer, not documentation. Having code done quickly will show a good programming skill. Doing documentation will help me get to the understanding level I need for programming. Coding will also get me there, but plus, code will be ready at the same time. Considering code's value for a programmer, start coding earlier rather than wait all documents ready would save time and show my value.)

3. Stuck after finishing Stop-N-Wait ARQ, because old simulation design cannot meet the requirement of Selective Repeat ARQ

During the implementation, I started thinking if my design is right: copying struct array from one to another. Then I am at a dead-end. It seemed that I cannot use the design for SW to implement SR ARQ. SR is much more complicated. I rushed into coding in SW. I thought it would lead me to SR, but I was wrong. I had to reconsider the whole project and redesign again. That happened at about Mar. 4th.

I consider this was because I didn't completely understand all those ARQs. I wanted to start simple and didn't prepare that much of complication in my design: layered architecture with no solution for simulation of the sender sending data while waiting for acknowledgement from the receiver at the same time, which is requirement in SR but not in SW. That gives a big difference for design. However, I didn't think of that before.

Lesson: know as much as possible before coding. But do some coding to help understanding by building a throw-away prototype, I was able to formu the areas of my lack of understanding. But Without complete understanding about what is going to do, you cannot start coding. See, although I learned software engineering procedures, I know what I need to do; I just thought I have understood it and wanted to have the code done ASAP. Actually rush into coding may delay the project rather than saving time. For me it's an important experience, the first real software engineering development.

4. Have a design change --- Event driven solution for network simulation

I started to study ARQ word by word, knowing how and why ARQ evolving from SW, NB, to SR, knowing sliding window, timer control, ACK, NAK are all part of flow control. From the last mistake, I have to learn in detail how ARQ works first.

Then I had to decide how I am going to implement, multithread programming or real socket programming. I asked some friends for advice and searched the Internet, comparing solutions coming from different sources. At about Mar.11th, I found one that I should use: layered architecture with the event driven design using event controller based on random timing to simulate sending out frame while waiting for acknowledgement (ACK). It is a simulation model, no socket programming, no socket library involved, no need for more than one computer to test, can simulate frame loss or corruption, perfect for my program.

5. New implementation of the Stop-N-Wait ARQ

I started from simple: SW first to try the new design if it's working for SW. It was done soon and convinced me to continue to use this design.

6. Selective Repeat ARQ with flow control

After I passed SW, I add window control first, then timer control, and so on.

I have to say I didn't finish documentation either this time. I just wanted to save some time from making good diagrams, but now I think that's procedures to make my mind clear and solid. Because during coding, I had to spend time more or less to keep my understanding right every now and then. Maybe that is how people learn, wherever you spend time, in making documentation or coding. That amount of time is there. But the earlier you have a clear mind, the easier way you will get things done, and the less trouble you will go through. It was done at about Mar. 18th.

7. Unit Test, bug fixing

I spent a lot of time on unit testing and modifying code. Further more, because of the random timing, it gives frames-through patterns great uncertainty, plus the layered architecture and design that I have to fine-tune proper pointer operations within different sub routines, testing challenged me so much. I started to use some documentation to keep my mind clear.

Because I have taken testing courses, I used some skills I learned in class, setting boundary conditions. That helps some, predicting some errors. But the thing is the frame-through patterns are varied, because of random timing, I might get a new pattern (such as frame gets acknowledged after time out, where I expect

to get ACK in time) which is not what I want to test at the time. I have to run the program again until I get what I want. Meanwhile, recorded the new problem and started to consider the solution. This is so time-consuming that it frustrated me. Finally it's kind of settled that I can move on to the next stage. It finished on Mar.25th.

8. A bug showed when sending 1000 frames with 50% lost rate

I found I failed to consider about one pointer operation of one boundary condition:

After the first three frames are sent out, sender doesn't get the ACK1 in time, so sender resends frame1, which causes sender getting ACK2 and ACK3 first rather than getting ACK1 first. In my window controller, I set pointer, after sender gets ACKn, move to the next (n+1) frame instead of moving to the next UNACKNOWLEDGED frame. After I change it moving to the next unacknowledged frame, the problem solved. (See Attachment3, including code)

After I showed my program to professors, I found another bug showing above. Professor Bernstein asked me: don't touch it anymore. He wanted to use some tools such as rejuvenation scripts to prevent it from happening, and some other tools to find the problem automatically.

But I felt embarrassing and could not help to troubleshooting it and fix it. Because he reiterated me not touching the code, I thought I would not overwrite the original code with error and fix the error in another copy of the code. He wanted to use other techniques to find it and prevent it from happening at the condition that I don't know the cause of the error. I didn't realize that it's so important to this project, the experiment that expects errors, expects real information that cannot be collected in the real software production house or anywhere else where errors are not welcomed and expected to be removed and fixed as soon as possible.

I noticed that I have been doing this project for almost two months, still have not understood the purpose of this project. I have to consider if it's the language itself or the procedures causing such an understanding gap. Now, I know why I didn't get many instructions such as what to do next or when to do what. My professor wants real information, wants the detailed procedures that I have been through, all primitive and organic from a developer's nature.

There must be errors, my professor said. I have to record what I have done, especially the mistakes.

Hopefully the gap is smaller enough.

Above four paragraphs are from my logbook, it shows part of my frustration for the problem caused by the communication between me and professors.

About communication:

I consider one of the reasons I made a lot of mistakes is communication problem. I didn't keep tight communication with professors. I didn't ask many questions, didn't request feedback much about what I have done. Professors didn't know how I did this and when I was going to finish until it's been behind the schedule a lot, I didn't quite understand the purpose of the whole project and how it is going to run until recently.

I think the reasons are following:

Objectively my language skills are part of the problems. When I learn in class, I may have something I didn't hear clearly or understand immediately, I can make up after class and I have the textbook, notes slides and classmates helping me. Request and feedback between me and the professor are not necessarily happening very often. But in the project, things are totally different, it needs immediate discussion and decision making during the face to face communication. It challenged my English skills and really pushed me to improve a lot. I can express my ideas better and faster than before, but still, something is there not so smooth and straight.

Writing is another problem. I could not write fast and use the right words to precisely express what I meant.

Culture difference is part of the problem, too. I am not getting used to ask all questions whenever I have them. Although Professor Bernstein knows part of the culture difference, if I don't say it, he also doesn't know whether or not I understand. Sometimes I say I understand maybe meaning I believe I will understand after I do some readings.

My attitude that I didn't response to these communication barriers actively was not quite right. What I did was doing what I think I should do, whether it's right or wrong, and waiting until something happened. I should have done more communication in order to improve work efficiency.

9. So far, it can run smoothly transmitting 1000 frames with 50% frame loss rate. It's about Apr.1st.

10. April.17, I finished making up documentation of the process, and architecture and design document as well.

11. Test configuration set up.

12. I was told I would be using Libft library to build fault tolerance into my program. When I started to learn libft, I found it's a UNIX-based library written in Perl. There is also a Windows NT version of libft---Swift, which I can download its evaluation version from Internet.

First, I thought it was going to be easier if it can work on Windows 2000. Memory conflict happened at the start up of Windows 2000. The system administrator spent some time trying to help me out, but he failed, either.

I started to try my second choice---installing Windows NT at the lab. Brian told me the school never had Windows NT license. That means no Windows NT can be run in the lab's computer.

After that, I have two choices: use BSD Unix on the school's server or Linux on the lab's computer; borrow a Windows NT machine to use by my own.

I intended to use the latter one because I know Windows NT more. While having some difficulties to find a Windows NT computer, I tried to compile my program on UNIX. I tried the BSD UNIX server in the department, local Linux, Cygwin UNIX emulation software in Windows 2000, other people's Linux computer. The results were so discouraging that I gave up the UNIX programming idea. At April 26th, I finally had a Windows NT 4.0 running in one computer with latest service pack 6a and Swift. There are many problems happening during preparing the Windows NT computer.

If I have known this earlier, I would have prepared for it from the beginning of my development. Maybe I would probably change to UNIX platform. Or at least, at this rush final period, I didn't need to spend much time to make the test computer ready. (Comment by professor: this is a classic resource dependency problem and configuration problem facing many software engineers.)

13. Building fault tolerance program.

Libft library provides software library using checkpoint approach to store critical memory statement at the given time during program running to give the fault tolerance ability.

My program didn't expect the situation with 100% frame loss rate. It broke when it exhausted the system memory. I built in libft library at the beginning of my program to record the statement. I asked the program to send 1000 frames with 100% frame loss rate. I reset the memory statement once every 100 frames are sent. This time, memory usage kept steady and never went up, the program were running all the time for at least half an hour, nothing broken.

At this point, the program has, say, 100% availability. It keeps running and is steady. This is the expected result because with 100% frame loss rate, the sender is supposed to resend frames until it gets the acknowledgement from the receiver, who will no longer get the frame and send the ACK back.

Libft gives the program the reliability that it never had before.

14. Memory Leak fix: it's because the fprintf() code all over the program. It is for debugging and statement recording to a text file. I will remove it to avoid memory usage exhaustion.

Appendix 2

Source Code without Rejuvenation

```
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <malloc.h>
#include <cstdlib>
#include <cmath>
/* ****************************************************************
  ARQ NETWORK EMULATOR:

  This code should be used for unidirectional or bidirectional
  data transfer protocols (from A to B. BIDIRECTIONAL TRANSFER OF DATA
  IS NOT REQUIRED).  Network properties:
  - one way network delay averages five time units (longer if there
    are other messages in the channel for Pipelined ARQ), but can be larger
  - packets can be corrupted (either the header or the data portion)
```

```
    or lost, according to user-defined probabilities
   - packets will be delivered in the order in which they were sent
    (although some can be lost).
**********************************************************************/

#define BIDIRECTIONAL 0    /* change to 1 for bidirectional */
                       /* and write a routine called B_output */

/* a "msg" is the data unit passed from layer 5 to layer  */
/* 4.  It contains the data (characters) to be delivered */
/* to layer 5 via transport level protocol entities.         */
struct msg {
  char data[20];
  }DataMsg[100];

/* a packet is the data unit passed from layer 4 to layer 3.*/

struct pkt {
   int seqnum;
   int acknum;
   int checksum;
   char payload[20];
    };

/**********************************************************
***************** NETWORK EMULATION CODE STARTS BELOW ***********
The code below emulates the layer 3 and below network environment:
  - emulates the tranmission and delivery (possibly with bit-level corruption
    and packet loss) of packets across the layer 3/4 interface
```

```
  - handles the starting/stopping of a timer, and generates timer
    interrupts
  - generates message to be sent (passed from later 5 to 4)

****************************************************************/

struct event {
   float evtime;           /* event time */
   int evtype;             /* event type code */
   int eventity;           /* entity where event occurs */
   struct pkt *pktptr;     /* ptr to packet (if any) assoc w/ this event */
   struct event *prev;
   struct event *next;
 };
struct event *evlist = NULL;   /* the event list */

/* possible events: */
#define  TIMER_INTERRUPT 0
#define  FROM_LAYER5     1
#define  FROM_LAYER3     2

#define  OFF             0
#define  ON              1
#define   A    0
#define   B    1



int TRACE = 1;              /* for debugging */
```

```
int nsim = 0;             /* number of messages from 5 to 4 so far */
int nsimmax = 0;          /* number of msgs to generate, then stop */
float time = 0.000;
float lossprob;           /* probability that a packet is dropped  */
float corruptprob;        /* probability that one bit is packet is flipped */
float lambda;             /* arrival rate of messages from layer 5 */
int   ntolayer3;          /* number sent into layer 3 */
int   nlost;              /* number lost in media */
int ncorrupt;             /* number corrupted by media*/



////////////////////////////////////////////////////////////////////////
//////////////Global Variables////////////////////////////////////////////
const int winSize_A = 8;
const int winSize_B = 8;

//const int FileSize = 50;

FILE *fp;
int runingTimer;

int sendbase;
int nextseqnum;
int numACKed;
int nMinUnACKed;
int winpktnum;
int pktnum, evnum;

int nBrecvd, nBwinpkt;
```

```
int bufend;

//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////

int ntolayer4;
struct pkt buffer_A[1000];
struct pkt buffer_B[5000];

generate_next_arrival();
init();
printevlist();
insertevent(struct event *p);

A_init();
B_init();
/* called from layer 5, passed the data to be sent to other side */
A_output(struct msg message);

B_output(struct msg message);  /* need be completed only for bidirectional transfer */

/* called from layer 3, when a packet arrives for layer 4 */
A_input(struct pkt packet);
B_input(struct pkt packet);
/* called when A's timer goes off */
A_timerinterrupt();
B_timerinterrupt();
```

```
main()
{
  struct event *eventptr;
  struct msg  msg2give;
  struct pkt  pkt2give;

  int i,j,loopCounter;
  //char c;

  init();
  A_init();
  B_init();

  loopCounter = 0;

  while (1)
  {
      eventptr = evlist;           /* get next event to simulate */
      if (eventptr==NULL)
                                goto terminate;
      evlist = evlist->next;       /* remove this event from event list */
      if (evlist!=NULL)
        evlist->prev=NULL;
                        if (TRACE>=2) {
        fprintf(fp, "\nEVENT time: %f,",eventptr->evtime);
        fprintf(fp, "  type: %d",eventptr->evtype);
        if (eventptr->evtype==0)
                                  fprintf(fp, ", timerinterrupt  ");
        else if (eventptr->evtype==1)
```

```
                        fprintf(fp, ", fromlayer5 ");
                                else
                                        fprintf(fp, ", fromlayer3 ");
                fprintf(fp, " entity: %d\n",eventptr->eventity);
}
time = eventptr->evtime;        /* update time to next event time */
if (nsim==nsimmax)
                //if( numACKed == nsimmax)
                break;                  /* all done with simulation */
if (eventptr->evtype == FROM_LAYER5 )
                {
   generate_next_arrival();   /* set up future arrival */
   /* fill in msg to give with string of same letter */
   //j = nsim % 26;
                        j = loopCounter % 26;
   for (i=0; i<20; i++)
     msg2give.data[i] = 97 + j;
   if (TRACE>2)
                        {
     fprintf(fp, "          MAINLOOP: data to be transmitting: ");
     for (i=0; i<20; i++)
       fprintf(fp, "%c", msg2give.data[i]);
     fprintf(fp, "\n");
                        }
   nsim++;
   if (eventptr->eventity == A)
                        {
                                //++ntolayer4;
                                A_output(msg2give);
```

```
                printevlist();
                //fprintf(fp, "ntolayer4 = %d", ntolayer4);
            }
  else
            {
                B_output(msg2give);
              printevlist();
            }
}
else if (eventptr->evtype ==  FROM_LAYER3)
        {
  pkt2give.seqnum = eventptr->pktptr->seqnum;
  pkt2give.acknum = eventptr->pktptr->acknum;
  pkt2give.checksum = eventptr->pktptr->checksum;
  for (i=0; i<20; i++)
     pkt2give.payload[i] = eventptr->pktptr->payload[i];
            if (eventptr->eventity ==A)      /* deliver packet by calling */
            {
                A_input(pkt2give);           /* appropriate entity */
                printevlist();
            }
            else
            {
                B_input(pkt2give);
                printevlist();
            }
            free(eventptr->pktptr);          /* free the memory for packet */
            --pktnum;
}
```

```
                else if (eventptr->evtype ==  TIMER_INTERRUPT)
                {
                        if (eventptr->eventity == A)
                                A_timerinterrupt();
                        else
                                B_timerinterrupt();
                }
                        else
                        {
                                fprintf(fp, "INTERNAL PANIC: unknown event type \n");
                        }
        free(eventptr);
                        --evnum;
                        ++loopCounter;
                        //printf("There are %d Lost frames\n", nlost);
        }

    terminate:
    fprintf(fp, " Simulator terminated at time %f\n after sending %d msgs from layer5\n",time,nsim);
    printevlist();
    for ( i=0;i<nsimmax;i++)
            fprintf(fp, "the buffer_A is %d, %c. \n", buffer_A[i].seqnum, buffer_A[i].payload[0]);
    for ( i=0;i<nBrecvd;i++)
            fprintf(fp, "the buffer_B is %d, %c. \n", buffer_B[i].seqnum, buffer_B[i].payload[0]);

    printf("event number left is %d\n", evnum);
    printf("packet number left is %d\n", pktnum);
    fclose(fp);
}
```

```
init()                  /* initialize the simulator */
{
 int i;
 float sum, avg;
 float jimsrand();


 if ((fp = fopen("testresult.txt", "w")) == NULL)       {
            fprintf(fp, "Cannot open file.\n");
            exit(1);
 }
  fprintf(fp, "**************************************************");
  fprintf(fp, "**********************************************************");
  fprintf(fp, "********************New Test Result************************");

  printf("Enter the number of messages to simulate: ");
  scanf("%d",&nsimmax);

  fprintf(fp,"\nEnter the number of messages to simulate: %d\n", nsimmax);

  printf("Enter  packet loss probability [enter 0.0 for no loss]:");
  scanf("%f",&lossprob);

  fprintf(fp, "\nEnter  packet loss probability [enter 0.0 for no loss]: %f\n", lossprob);

  printf("Enter packet corruption probability [0.0 for no corruption]:");
  scanf("%f",&corruptprob);
```

```
fprintf(fp, "\nEnter packet corruption probability [0.0 for no corruption]: %f\n", corruptprob);

printf("Enter average time between messages from sender's layer5 [ > 0.0]:");
scanf("%f",&lambda);

fprintf(fp, "Enter average time between messages from sender's layer5 [ > 0.0]: %f \n", lambda);

printf("Enter TRACE:");
scanf("%d",&TRACE);

srand(9999);              /* init random number generator */
sum = 0.0;                /* test random number generator for students */
for (i=0; i<1000; i++)
   sum=sum+jimsrand();    /* jimsrand() should be uniform in [0,1] */


avg = sum/1000.0;
if (avg < 0.25 || avg > 0.75) {
 printf("It is likely that random number generation on your machine\n" );
 printf("is different from what this emulator expects.  Please take\n");
 printf("a look at the routine jimsrand() in the emulator code. Sorry. \n");
 exit(0);
 }

ntolayer3 = 0;
nlost = 0;
ncorrupt = 0;
```

```
  time=0.0;                  /* initialize time to 0.0 */
  generate_next_arrival();    /* initialize event list */
}

/*************************************************************************/
/* jimsrand(): return a float in range [0,1].  The routine below is used to */
/* isolate all random number generation in one location.  We assume that the*/
/* system-supplied rand() function return an int in therange [0,mmm]        */
/*************************************************************************/
float jimsrand()
{
 double mmm = 10000;
 float x;
 int i;
 i = rand();
 if (i >= mmm )
        x = fmod (i,mmm)/mmm;
 else
        x = i/mmm;     /* x should be uniform in [0,1] */

 return(x);
}

/******************** EVENT HANDLINE ROUTINES *******/
/*  The next set of routines handle the event list   */
/***************************************************/

generate_next_arrival()
{
```

```
  double x,log(),ceil();
  struct event *evptr;

  if (TRACE>2)
     fprintf(fp, "          GENERATE NEXT ARRIVAL: creating new arrival\n");

  x = lambda*jimsrand()*2;  /* x is uniform on [0,2*lambda] */
                            /* having mean of lambda        */
  evptr = (struct event *)malloc(sizeof(struct event));
  evptr->evtime =  time + x;
  evptr->evtype =  FROM_LAYER5;
  if (BIDIRECTIONAL && (jimsrand()>0.5) )
     evptr->eventity = B;
   else
     evptr->eventity = A;
  insertevent(evptr);
  ++evnum;
}



insertevent(p)
  struct event *p;
{
  struct event *q,*qold;

  if (TRACE>2) {
     fprintf(fp, "          INSERTEVENT: time is %lf\n",time);
     fprintf(fp, "          INSERTEVENT: future time will be %lf\n",p->evtime);
     }
```

```
  q = evlist;     /* q points to header of list in which p struct inserted */
  if (q==NULL) {   /* list is empty */
      evlist=p;
      p->next=NULL;
      p->prev=NULL;
      }
    else {
      for (qold = q; q !=NULL && p->evtime > q->evtime; q=q->next)
           qold=q;
      if (q==NULL) {   /* end of list */
          qold->next = p;
          p->prev = qold;
          p->next = NULL;
          }
         else if (q==evlist) { /* front of list */
          p->next=evlist;
          p->prev=NULL;
          p->next->prev=p;
          evlist = p;
          }
        else {     /* middle of list */
          p->next=q;
          p->prev=q->prev;
          q->prev->next=p;
          q->prev=p;
          }
       }
}
```

```
printevlist()
{
  struct event *q;
//  int i;
  fprintf(fp, "--------------\nEvent List Follows:\n");
  for(q = evlist; q!=NULL; q=q->next) {
    fprintf(fp, "Event time: %f, type: %d entity: %d\n",q->evtime,q->evtype,q->eventity);
    }
  fprintf(fp, "--------------\n");
}



stoptimer(AorB)
int AorB;  /* A or B is trying to stop timer */
{
 struct event *q,*qold;

 if (TRACE>2)
    fprintf(fp, "          STOP TIMER: stopping timer at %f\n",time);
 for (q=evlist; q!=NULL ; q = q->next)
    if ( (q->evtype==TIMER_INTERRUPT  && q->eventity==AorB) ) {
      /* remove this event */
      if (q->next==NULL && q->prev==NULL)
            evlist=NULL;         /* remove first and only event on list */
          else if (q->next==NULL) /* end of list - there is one in front */
            q->prev->next = NULL;
          else if (q==evlist) { /* front of list - there must be event after */
            q->next->prev=NULL;
```

```
            evlist = q->next;
            }
           else {     /* middle of list */
            q->next->prev = q->prev;
            q->prev->next =  q->next;
            }
       free(q);
       return;
     }
  fprintf(fp, "Warning: unable to cancel your timer. It wasn't running.\n");
}



starttimer(AorB,increment)
int AorB;  /* A or B is trying to stop timer */
float increment;
{

 struct event *q;
 struct event *evptr;
 //char *malloc();

 if (TRACE>2)
    fprintf(fp, "          START TIMER: starting timer at %f for the packet %d\n",time, nMinUnACKed);
  for (q=evlist; q!=NULL ; q = q->next)
    if ( (q->evtype==TIMER_INTERRUPT  && q->eventity==AorB) ) {
      fprintf(fp, "Warning: attempt to start a timer that is already started\n");
      return;
      }
```

```
/* create future event for when timer goes off */
   evptr = (struct event *)malloc(sizeof(struct event));
   evptr->evtime =  time + increment;
   evptr->evtype =  TIMER_INTERRUPT;
   evptr->eventity = AorB;
   insertevent(evptr);
   ++evnum;
}



/************************ TOLAYER3 ***************/
tolayer3(AorB,packet)
int AorB;  /* A or B is trying to stop timer */
struct pkt packet;
{
 struct pkt *mypktptr;
 struct event *evptr,*q;
 //char *malloc();
 float lastime, x, jimsrand();
 int i;



 ntolayer3++;

 /* simulate losses: */
 if (jimsrand() < lossprob)  {
      nlost++;
      if (TRACE>0)
```

```
            fprintf(fp, "          TOLAYER3: packet being lost\n");
      return;
    }

/* make a copy of the packet */
 mypktptr = (struct pkt *)malloc(sizeof(struct pkt));
 mypktptr->seqnum = packet.seqnum;
 mypktptr->acknum = packet.acknum;
 mypktptr->checksum = packet.checksum;
 for (i=0; i<20; i++)
    mypktptr->payload[i] = packet.payload[i];
 if (TRACE>2)  {
   fprintf(fp, "          TOLAYER3: seq: %d, ack %d, check: %d ", mypktptr->seqnum,
                mypktptr->acknum,  mypktptr->checksum);
    for (i=0; i<20; i++)
        fprintf(fp, "%c",mypktptr->payload[i]);
    fprintf(fp, "\n");
   }

/* create future event for arrival of packet at the other side */
  evptr = (struct event *)malloc(sizeof(struct event));
  evptr->evtype =  FROM_LAYER3;   /* packet will pop out from layer3 */
  evptr->eventity = (AorB+1) % 2; /* event occurs at other entity */
  evptr->pktptr = mypktptr;       /* save ptr to my copy of packet */
/* finally, compute the arrival time of packet at the other end.
   medium can not reorder, so make sure packet arrives between 1 and 10
   time units after the latest arrival time of packets
   currently in the medium on their way to the destination */
 lastime = time;
```

```
 for (q=evlist; q!=NULL ; q = q->next)
   if ( (q->evtype==FROM_LAYER3  && q->eventity==evptr->eventity) )
     lastime = q->evtime;
 //evptr->evtime =  lastime + 1 + 9*jimsrand();
 evptr->evtime =  lastime + lambda*2*jimsrand();



 /* simulate corruption: */
 if (jimsrand() < corruptprob)  {
   ncorrupt++;
   if ( (x = jimsrand()) < .75)
      mypktptr->payload[0]='Z';   /* corrupt payload */
     else if (x < .875)
      mypktptr->seqnum = 999999;
     else
      mypktptr->acknum = 999999;
   if (TRACE>0)
            fprintf(fp, "          TOLAYER3: packet being corrupted\n");
   }


  if (TRACE>2)
     fprintf(fp, "          TOLAYER3: scheduling arrival on other side\n");
  insertevent(evptr);
}



tolayer5(AorB,datasent)
  int AorB;
```

```
 char datasent[20];
{
 int i;
 if (TRACE>2) {
    fprintf(fp, "          TOLAYER5: data received: ");
    for (i=0; i<20; i++)
       fprintf(fp, "%c",datasent[i]);
    fprintf(fp, "\n");
  }
 // sort all the packets in the buffer

}

/* called from layer 5, passed the data to be sent to other side */
A_output(struct msg message);
B_output(message)  /* need be completed only for bidirectional transfer */
 struct msg message;
{
            fprintf(fp, "It's in B_output\n");
}

/* called from layer 3, when a packet arrives for layer 4 */
A_input(struct pkt packet);

/* called when A's timer goes off */
A_timerinterrupt()
{
                        tolayer3(A, buffer_A[nMinUnACKed]);
                        starttimer(A, lambda*2);
```

```
                runingTimer = 1;
                //--nsim;
                fprintf(fp, "It's in A_timerinterupt, resending number %d packet!\n",
buffer_A[nMinUnACKed].seqnum);
                fprintf(fp, "nsim = %d\n",nsim);

}

A_init()
{
        runingTimer = 0;
        //getACK = 0;
        ntolayer4 = 0;

        sendbase = 0;
        nextseqnum = 0;
        numACKed = 0;
        nMinUnACKed = 0;
        winpktnum = 0;

        nBrecvd = 0;
        bufend = 0;
}

/* Note that with simplex transfer from A-to-B, there is no B_output() */

/* called from layer 3, when a packet arrives for layer 4 at B*/
B_input(struct pkt packet);
```

```
/* called when B's timer goes off */
B_timerinterrupt()
{
fprintf(fp, "It's in B_output\n");

}

B_init()
{
        int i;
        //for(i=0; i<5000; i++)
        //          buffer_B[i].seqnum = 987654321;
}

//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
A_output(struct msg message)

{
        //struct pkt packet;
        int i;

        --nsim; //in order to deal with nsim in mainloop, make sure nsim only increase when gets an ACK


        if (bufend < nsimmax)      {
        buffer_A[bufend].seqnum = bufend;  //start from 0
        buffer_A[bufend].acknum = 0;
```

```
        buffer_A[bufend].checksum = 0;

        for (i=0; i<20; i++)
                buffer_A[bufend].payload[i] = message.data[i];

        ++bufend;

        //window controls the transmission to the lower layer
        //if (winpktnum < winSize_A)        {
        if (nextseqnum < (sendbase + winSize_A))   {
                //sendbase = nMinUnACKed;
                tolayer3(A, buffer_A[nextseqnum]);
                ++winpktnum;
                if (nextseqnum < bufend) ++nextseqnum;
                if (runingTimer ==0) {
                        starttimer(A, lambda*2);
                        runingTimer = 1;
                }
        }
        }
        fprintf(fp, "\nA_output\n");
}
A_input(struct pkt packet)
{
        int i,j;

        if (packet.seqnum < nMinUnACKed)
                return;
```

```
if (packet.acknum == 1)    {

        if (runingTimer == 1)       {

                runingTimer = 0;
                stoptimer(0);
        }

//++nsim; //including the packet which has been marked ACKed, when timeout, resending,--nsim

for(i = sendbase; i<nextseqnum; i++)          {
        if (buffer_A[i].seqnum == packet.seqnum)    {
                if (buffer_A[i].acknum == 0)          {
                        ++nsim;;
                        buffer_A[i].acknum = 1; //mark the packet in the buffer is ACKed
                break;
                }
        }
}

fprintf(fp, "\n     Got ACK%d from B!\n", packet.seqnum);

if (i == nMinUnACKed)    {
        for (j=i; j<nextseqnum; j++)          {
                if (buffer_A[j].acknum == 0)
                        break;
        }

        nMinUnACKed = j;
```

```
                sendbase = j;
                //winpktnum = winpktnum - (j-i);
                if ((nextseqnum < (sendbase + winSize_A)) && (nextseqnum < bufend) &&
(nextseqnum < nsimmax)) {
                        tolayer3(A, buffer_A[nextseqnum]);  //start sending when window advances,
keep evlist not empty!
                        ++nextseqnum;
                        //++winpktnum;
                }
        } //window size is keeping the same, but win_base advances one more.



        starttimer(A, lambda*2);
        runingTimer = 1;
        }
        fprintf(fp, "\nA_input\n");
}

B_input(struct pkt packet)
{
        struct msg message;
        int i;

        //if (nBwinpkt < winSize_B)          {

                //after checking it's well received
                //will add later
        if ((packet.checksum == 0) && (packet.payload[0]!='Z') && (packet.seqnum != 999999) &&
(packet.acknum != 999999))
```

```
        {
                packet.acknum = 1;
                for (i=0; i<20; i++)
                        message.data[i] = packet.payload[i];

                tolayer3(B, packet);  //sending ACK

            //fill in the buffer, after unpack the packet
            //and sort
            buffer_B[nBrecvd].seqnum = packet.seqnum;  //start from 0
            buffer_B[nBrecvd].acknum = packet.seqnum;
            buffer_B[nBrecvd].checksum = packet.checksum;

            for (i=0; i<20; i++)
                    buffer_B[nBrecvd].payload[i] = message.data[i];

            ++nBrecvd;
            //sorting code, skip for now

            tolayer5(B,message.data);
            if (++nBwinpkt == winSize_B)
                    nBwinpkt = 0;     //reset window
        }
        fprintf(fp, "It's in B_input\n");
}
```

Appendix 3

```
main()
{
```

```
struct event *eventptr;
struct msg  msg2give;
struct pkt  pkt2give;

int i,j,loopCounter;
//char c;
        ft_start("d:\\temp\\StressTest");
        //  initialize the critical memory
        FtMemOpen(0);
        //  declare the critical variables
        critical((char *) &nsimmax,sizeof(nsimmax),0);
        critical((char *) &lossprob,sizeof(lossprob),0);
        critical((char *) &corruptprob,sizeof(corruptprob),0);
        critical((char *) &lambda,sizeof(lambda),0);
        critical((char *) &TRACE,sizeof(TRACE),0);
        critical((char *) &nsim,sizeof(nsim),0);
        critical((char *) &ntolayer3,sizeof(ntolayer3),0);
        critical((char *) &nlost,sizeof(nlost),0);
        critical((char *) &ncorrupt,sizeof(ncorrupt),0);
        critical((char *) &time,sizeof(time),0);
        critical((char *) &evlist,sizeof(evlist),0);
        int loopnum;
        for (loopnum = 0;loopnum<10; loopnum++) {

init();
A_init();
B_init();

loopCounter = 0;
```

```
//////////////////////////////////////////////////////////////////

///////////Fault tolerance ////////////////////////////////////////

                                        //  set the checkpoint file name

while (1)

{

    eventptr = evlist;           /* get next event to simulate */

    if (eventptr==NULL)

                                goto terminate;

    evlist = evlist->next;       /* remove this event from event list */

    if (evlist!=NULL)

      evlist->prev=NULL;

                    if (TRACE>=2) {

      fprintf(fp, "\nEVENT time: %f,",eventptr->evtime);

      fprintf(fp, "  type: %d",eventptr->evtype);

      if (eventptr->evtype==0)

                                  fprintf(fp, ", timerinterrupt  ");

      else if (eventptr->evtype==1)

                                fprintf(fp, ", fromlayer5 ");

                                        else

                                                fprintf(fp, ", fromlayer3 ");

                        fprintf(fp, " entity: %d\n",eventptr->eventity);

    }

    time = eventptr->evtime;       /* update time to next event time */

    if (nsim==nsimmax)

                    //if( numACKed == nsimmax)

                    break;                    /* all done with simulation */

    if (eventptr->evtype == FROM_LAYER5 )

                    {
```

```
  generate_next_arrival();   /* set up future arrival */
  /* fill in msg to give with string of same letter */
  //j = nsim % 26;
                        j = loopCounter % 26;
  for (i=0; i<20; i++)
    msg2give.data[i] = 97 + j;
  if (TRACE>2)
                        {
    fprintf(fp, "          MAINLOOP: data to be transmitting: ");
    for (i=0; i<20; i++)
      fprintf(fp, "%c", msg2give.data[i]);
    fprintf(fp, "\n");
                        }
  nsim++;
  if (eventptr->eventity == A)
                        {
                                //++ntolayer4;
                                A_output(msg2give);
                                printevlist();
                                //fprintf(fp, "ntolayer4 = %d", ntolayer4);
                        }
  else
                        {
                                B_output(msg2give);
                           printevlist();
                        }
}
else if (eventptr->evtype ==  FROM_LAYER3)
                {
```

```
   pkt2give.seqnum = eventptr->pktptr->seqnum;
   pkt2give.acknum = eventptr->pktptr->acknum;
   pkt2give.checksum = eventptr->pktptr->checksum;
   for (i=0; i<20; i++)
      pkt2give.payload[i] = eventptr->pktptr->payload[i];
                        if (eventptr->eventity ==A)      /* deliver packet by calling */
                        {
                                A_input(pkt2give);           /* appropriate entity */
                                printevlist();
                        }
                        else
                        {
                                B_input(pkt2give);
                                printevlist();
                        }
                        free(eventptr->pktptr);          /* free the memory for packet */
}
                         else if (eventptr->evtype ==  TIMER_INTERRUPT)
                         {
                                if (eventptr->eventity == A)
                                        A_timerinterrupt();
                                else
                                        B_timerinterrupt();
                         }
                                else
                                {
                                        fprintf(fp, "INTERNAL PANIC: unknown event type \n");
                                }
free(eventptr);
```

```
                ++loopCounter;
        }

  terminate:
  fprintf(fp, " Simulator terminated at time %f\n after sending %d msgs from layer5\n",time,nsim);
  printevlist();
  for ( i=0;i<nsimmax;i++)
        fprintf(fp, "the buffer_A is %d, %c. \n", buffer_A[i].seqnum, buffer_A[i].payload[0]);
  for ( i=0;i<nBrecvd;i++)
        fprintf(fp, "the buffer_B is %d, %c. \n", buffer_B[i].seqnum, buffer_B[i].payload[0]);
        //*evlist = NULL;

  //fclose(fp);
        ////////////////////////////////////////////////////////////////
  ////////FT else end////////////////////////////////////////////////////
        printf("end?");
        } //ft for end
        FtMemClose(0);//close ft memory
        fclose(fp);

}



init()                  /* initialize the simulator */
{
  int i;
  float sum, avg;
  float jimsrand();
```

```
if (criticalsize(0)>0) {
        // if it has checkpointed data, recover the data;
        recover(INFILE);
        printf("recovered nsim = %d\n", nsim);
} else {
        printf("to be stored nsim = %d\n",nsim);

 if ((fp = fopen("testresult.txt", "w")) == NULL)       {
          fprintf(fp, "Cannot open file.\n");
          exit(1);
 }
  fprintf(fp, "*****************************************************");
  fprintf(fp, "*************************************************************");
  fprintf(fp, "********************New Test Result************************");

  printf("Enter the number of messages to simulate: ");
  scanf("%d",&nsimmax);

  fprintf(fp,"\nEnter the number of messages to simulate: %d\n", nsimmax);

  printf("Enter  packet loss probability [enter 0.0 for no loss]:");
  scanf("%f",&lossprob);

  fprintf(fp, "\nEnter  packet loss probability [enter 0.0 for no loss]: %f\n", lossprob);

  printf("Enter packet corruption probability [0.0 for no corruption]:");
  scanf("%f",&corruptprob);
```

```
  fprintf(fp, "\nEnter packet corruption probability [0.0 for no corruption]: %f\n", corruptprob);

  printf("Enter average time between messages from sender's layer5 [ > 0.0]:");
  scanf("%f",&lambda);

  fprintf(fp, "Enter average time between messages from sender's layer5 [ > 0.0]: %f \n", lambda);

  printf("Enter TRACE:");
  scanf("%d",&TRACE);

        checkpoint(INFILE);

  srand(9999);              /* init random number generator */
  sum = 0.0;                /* test random number generator for students */
  for (i=0; i<1000; i++)
    sum=sum+jimsrand();    /* jimsrand() should be uniform in [0,1] */
  avg = sum/1000.0;
  if (avg < 0.25 || avg > 0.75) {
   printf("It is likely that random number generation on your machine\n" );
   printf("is different from what this emulator expects.  Please take\n");
   printf("a look at the routine jimsrand() in the emulator code. Sorry. \n");
   exit(0);
   }
}//ft else end

  ntolayer3 = 0;
  nlost = 0;
  ncorrupt = 0;
```

```
  time=0.0;                 /* initialize time to 0.0 */
  generate_next_arrival();    /* initialize event list */
}
```

Appendix 4

```
A_input(struct pkt packet)
{
        int i,j;

        if (packet.seqnum < nMinUnACKed)
                return;

        if (packet.acknum == 1)    {

                if (runingTimer == 1)       {

                        runingTimer = 0;
                        stoptimer(0);
                }

        //++nsim; //including the packet which has been marked ACKed, when timeout, resending,--nsim

        for(i = sendbase; i<nextseqnum; i++)          {
                if (buffer_A[i].seqnum == packet.seqnum)   {
                        if (buffer_A[i].acknum == 0)         {
                                ++nsim;;
                                buffer_A[i].acknum = 1; //mark the packet in the buffer is ACKed
                        break;
                        }
```

```
            }
        }

        fprintf(fp, "\n     Got ACK%d from B!\n", packet.seqnum);

        if (i == nMinUnACKed)   {
            for (j=i; j<nextseqnum; j++)          {
                if (buffer_A[j].acknum == 0)
                    break;
            }

            nMinUnACKed = j;
            sendbase = j;
            //winpktnum = winpktnum - (j-i);
            if ((nextseqnum < (sendbase + winSize_A)) && (nextseqnum < bufend) &&
(nextseqnum < nsimmax)) {
                tolayer3(A, buffer_A[nextseqnum]);  //start sending when window advances,
keep evlist not empty!
                ++nextseqnum;
                //++winpktnum;
            }
        } //window size is keeping the same, but win_base advances one more.


        starttimer(A, lambda*2);
        runingTimer = 1;
        }
        fprintf(fp, "\nA_input\n");
}
```

Appendix 5

```
A_output(struct msg message)
{
        //struct pkt packet;
        int i;
        --nsim; //in order to deal with nsim in mainloop, make sure nsim only increase when gets an ACK

        if (bufend < nsimmax)       {
        buffer_A[bufend].seqnum = bufend;  //start from 0
        buffer_A[bufend].acknum = 0;
        buffer_A[bufend].checksum = 0;

        for (i=0; i<20; i++)
                buffer_A[bufend].payload[i] = message.data[i];
        ++bufend;
        //window controls the transmission to the lower layer
        //if (winpktnum < winSize_A)        {
        if (nextseqnum < (sendbase + winSize_A))   {
                //sendbase = nMinUnACKed;
                tolayer3(A, buffer_A[nextseqnum]);
                ++winpktnum;
                if (nextseqnum < bufend) ++nextseqnum;
                if (runingTimer ==0) {
                        starttimer(A, lambda*2);    //should check if the timer is still running
                        runingTimer = 1;
                }
        }
        }
```

```
        fprintf(fp, "\nA_output\n");
}
A_input(struct pkt packet)
{
        int i,j;

        if (packet.seqnum < nMinUnACKed)
                return;

        if (packet.acknum == 1)    {

                if (runingTimer == 1)       {

                        runingTimer = 0;
                        stoptimer(0);
                }

        //++nsim; //including the packet which has been marked ACKed, when timeout, resending,--nsim

        for(i = sendbase; i<nextseqnum; i++)          {
                if (buffer_A[i].seqnum == packet.seqnum)    {
                        if (buffer_A[i].acknum == 0)         {
                                ++nsim;;
                                buffer_A[i].acknum = 1; //mark the packet in the buffer is ACKed
                        break;
                        }
                }
        }
```

```
        fprintf(fp, "\n     Got ACK%d from B!\n", packet.seqnum);


        if (i == nMinUnACKed)   {
                for (j=i; j<nextseqnum; j++)          {
                        if (buffer_A[j].acknum == 0)
                                break;
                }


                nMinUnACKed = j;
                sendbase = j;
                //winpktnum = winpktnum - (j-i);
                if ((nextseqnum < (sendbase + winSize_A)) && (nextseqnum < bufend) &&
(nextseqnum < nsimmax)) {
                        tolayer3(A, buffer_A[nextseqnum]);  //start sending when window advances,
keep evlist not empty!
                        ++nextseqnum;
                        //++winpktnum;
                }
        } //window size is keeping the same, but win_base advances one more.
        starttimer(A, lambda*2);
        runingTimer = 1;
        }
        fprintf(fp, "\nA_input\n");
}
```