Implementation of Sliding Window Protocol

CZ3006 – Net Centric Computing

Instructor: Prof. Sun Chengzheng

Zhang Danyang
Matric. No. U1122983C
Solo Project

Lab Group: SSP4

1 OBJECTIVE

The objective of the CZ3006 Net Centric Computing laboratory, Implementation of Sliding Window Protocol, is to understand protocols in Data Link Layer in a fine details. Follow control, piggybagging, buffering, circular window, and error control are appreciated and examined.

In this laboratory, the protocol 6, Sliding Window Protocol, is implemented. It is simulated over communication network environment of NetSim connecting two virtual machines VMach 1 and VMach 2.

2 SUMMARY OF COMPLETENESS

Features	Completeness	Author
Full-duplex data communication (Sender is the Receiver and vice versa over single communication channel)	Completed	Zhang Danyang
In-order delivery of packets to the network-layer (Out-of-order receive from Physical Layer, and in-order deliver to Network Layer)	Completed	Zhang Danyang
Selective repeat retransmission strategy	Completed	Zhang Danyang
Synchronization with the network-layer (Granting Credit)	Completed	Zhang Danyang
Negative acknowledgement (NACK)	Completed	Zhang Danyang
Separate acknowledgment when the reverse traffic is light or none (individual ACK)	Completed	Zhang Danyang
Ability to withstand quality level 0, 1, 2, and 3 of the simulator component.	Completed	Zhang Danyang

3 APPROACHES

3.1 FULL-DUPLEX DATA COMMUNICATION

Full-duplex data communication allows data to transmit in two directions simultaneously with single channel, instead of having two separate communication. The sender is the receiver at the same time and vice versa.

```
while(true) {
    wait_for_event(event);
     switch(event.type) {
         case (PEvent.NETWORK_LAYER_READY): // sending out
           …

         case (PEvent.FRAME_ARRIVAL ): // receiving

           …
```

Piggybacking is developed so that when a data frame arrives, the acknowledgement would not be sent immediately. Instead, the acknowledgement would be piggybacked onto the next outgoing data frame to better utilize the available channel.

```
private void send_frame(int frame_kind, int frame_number, int frame_expected,
Packet out_buffer[]) {
    ...
    frame.ack = (frame_expected+MAX_SEQ)%(MAX_SEQ+1);
    ...

}
```

3.2 IN-ORDER DELIVERY OF PACKETS TO THE NETWORK-LAYER

Sliding Window Protocol allows the frames to be received out of order while the packets would be passed to the network layer in order, provided that the sequence number of incoming frame is within the window. The frame which has higher sequence number will not be delivered to the network layer until the lower sequence number frame has been delivered.

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;

    while(this.arrived[frame_expected%NR_BUFS]){
        /*
        Notice: Network Layer must receive the packet in order
         */
        this.to_network_layer(this.in_buf[frame_expected%NR_BUFS]);
        this.no_nak = true;
        this.arrived[frame_expected%NR_BUFS] = false;
        frame_expected = this.inc(frame_expected);
        too_far = this.inc(too_far);
        this.start_ack_timer();
    }

}
```

The *between* method is used to check if the sequence number of the frame falls into the receiver expected frames.

Figure 3-1 Two situtaions of *between*: a<c or a>c

```
private boolean between(int a, int b, int c) {
    // normal situation
    if(a<c) {
        return a<=b&&b<c;
    }
    else if (c<a) {
        return a<=b || b<c;
    }
    return false;

}
```

3.3   SYNCHRONIZATION WITH THE NETWORK-LAYER BY GRANTING CREDITS

In initialization, the credits granted to the Network Layer equals to the receiver's window size.

```
this.enable_network_layer(NR_BUFS);
```

When sending one frame out, one credit is deducted. When the frame in the sending window is confirmed by acknowledgment, one credit is restored.

```
while(between(ack_expected, frame_received.ack, next_frame_to_send)) {
    stop_timer(ack_expected%NR_BUFS);
    ack_expected = this.inc(ack_expected);
    this.enable_network_layer(1); // grant credit to network_layer

}
```

3.4   SELECTIVE REPEAT RETRANSMISSION STRATEGY

Selective repeat retransmission strategy only required the lost/damaged frames to be retransmitted instead of *go back n*, and the subsequent frames will be accepted and buffered in the receiving buffer in the receiver, provided the frame numbers are within the window.

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;
    …

}
```

For requesting for retransmission of lost/damaged frame, a negative acknowledgment for that frame is sent as discussed in the following section.

3.5   NEGATIVE ACKNOWLEDGEMENT

For the receiver to notify the sender the incoming frame is received as an error or unexpectedly, the receiver will send a negative acknowledgement to the sender.

Out-of-order receiving is allowed but negative acknowledgment is sent, because selective repeat is designed and implemented. In the case of unexpected frame

```
if(frame_received.seq!=frame_expected&&this.no_nak) {
    this.send_frame(PFrame.NAK, 0, frame_expected, this.out_buf);

}
```

In the case of check sum error where the data in the frame is damaged:

```
case (PEvent.CKSUM_ERR):
    if(this.no_nak) {
        this.send_frame(PFrame.NAK, 0, frame_expected, this.out_buf);
    }
    break;
```

3.6   SEPARATE ACKNOWLEDGMENT WHEN REVERSE TRAFFIC IS LIGHT OR NONE

It is not uncommon that the acknowledgement will be waiting for an extended period of time. To resolve this issue, an acknowledgement timer is introduced where a separate acknowledgement will be sent indicating the last frame which is received successfully.

Start timer when passing the received frame into the Network Layer (last line of the code):

```
if(this.between(frame_expected, frame_received.seq,
too_far)&&!arrived[frame_received.seq%NR_BUFS]) {
    // frames received may be in any order
    this.arrived[frame_received.seq%NR_BUFS] = true;
    this.in_buf[frame_received.seq%NR_BUFS] = frame_received.info;

    while(this.arrived[frame_expected%NR_BUFS]){
        /*
        Notice: Network Layer must receive the packet in order
         */
        this.to_network_layer(this.in_buf[frame_expected%NR_BUFS]);
        this.no_nak = true;
        this.arrived[frame_expected%NR_BUFS] = false;
        frame_expected = this.inc(frame_expected);
        too_far = this.inc(too_far);
        this.start_ack_timer();
    }

}
```

In case of time out, transmit the acknowledgment in a separate frame:

```
case (PEvent.TIMEOUT):
    this.send_frame(PFrame.DATA, this.oldest_frame, frame_expected, this.out_buf);
    break;
```

4   TESTING

According to different simulated network environment, different levels of intensity were used.

- java NetSim 0: 5 times
- java NetSim 1: 5 times
- java NetSim 2: 5 times
- java NetSim 3: 25+ times

In all tests, the receiver's files were identical with the sender's files.

5   SUGGESTIONS

Students should write batch files in Windows OS or bash files in UNIX-like OS to automatically execute commands of "javac SWP.java", "java NetSim 3", "java VMach 1", "java VMach 2":

Compile.bat
NetSim.bat
VMach1.bat
VMach2.bat
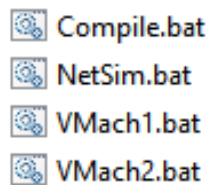
Such batch/ bash files will avoid manual input of instructions in command line.

6 SOURCE CODE LIST

CLASS File (17)

EventQueue.class	6/25/2005 7:19	CLASS File	2 KB
Forwarder.class	6/25/2005 7:19	CLASS File	3 KB
FrameHandler.class	6/25/2005 7:19	CLASS File	2 KB
NetSim.class	6/25/2005 7:19	CLASS File	3 KB
NetworkReceiver.class	6/25/2005 7:19	CLASS File	2 KB
NetworkSender.class	6/25/2005 7:19	CLASS File	2 KB
Packet.class	6/25/2005 7:19	CLASS File	1 KB
PacketQueue.class	6/25/2005 7:19	CLASS File	1 KB
PEvent.class	6/25/2005 7:19	CLASS File	1 KB
PFrame.class	2/11/2014 13:24	CLASS File	1 KB
PFrameMsg.class	6/25/2005 7:19	CLASS File	1 KB
SWE.class	6/25/2005 7:19	CLASS File	5 KB
SWP$1.class	2/21/2014 21:29	CLASS File	1 KB
SWP$AckTimerTask.class	2/21/2014 21:29	CLASS File	1 KB
SWP$NormalTimerTask.class	2/21/2014 21:29	CLASS File	1 KB
SWP.class	2/21/2014 21:29	CLASS File	5 KB
VMach.class	6/25/2005 7:19	CLASS File	2 KB

File folder (3)

.git	2/21/2014 21:38	File folder	
.idea	2/21/2014 21:38	File folder	
out	2/11/2014 14:19	File folder	

IML File (1)

SlidingWindowProtocol.iml	2/11/2014 14:59	IML File	1 KB

JAVA File (2)

PFrame.java	9/30/2005 18:00	JAVA File	2 KB
SWP.java	2/21/2014 22:01	JAVA File	15 KB

MD File (1)

readme.md	2/21/2014 0:03	MD File	1 KB

Text Document (1)

.gitignore	2/11/2014 14:22	Text Document	3 KB

TXT File (4)

receive_file_1.txt	2/21/2014 21:29	TXT File	1 KB
receive_file_2.txt	2/21/2014 21:30	TXT File	1 KB
send_file_1.txt	6/25/2005 7:19	TXT File	1 KB
send_file_2.txt	6/25/2005 7:19	TXT File	1 KB

Noticeable source files and class files:

- SWP.java: Sliding Window Protocol file.
- SWP.class: Complied class file of SWP.java.
- SWP$NormalTimerTask.class: complied class file for SWP's internal class NormalTimerTask.
- SWP$AckTimerTask.class: complied class file for internal class AckTimerTask.