Part 3: Flow Control

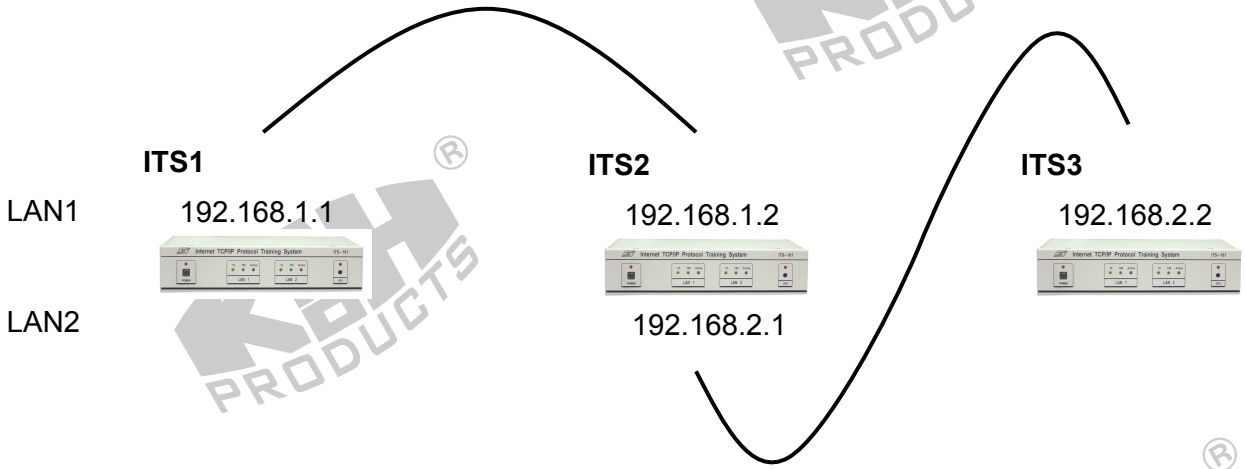
Exp 10. Network Disturbance for IP

OBJECTIVE：To measure the performance of router in various network conditions which are changed by some network disturbance implemented by this experiment. The created network conditions will be facilitated to study the next Error Control experiment.

BRIEF DESCRIPTION： In the real networking, packet lost, delay, and errors are measured as the most common problems in communication, so the following experiments, we will generate some network disturbance to measure them as in the actual internet environment.

DURATION：3 hrs

TOPOLOGY



TECHNICAL BACKGROUND

This experiment uses the following statistics to measure a router:

a. *Packet Lost* is a measure of losing packets along the data path.

b. *Packet Delay* is a measure of the difference in time between arrivals of the same packet at different point of observation.

c. *Packet Error* is a measure of corrupting packets along the data path.

d. *Bandwidth* is a measure of how much information can be sent over a connection at one time.

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 10.1.

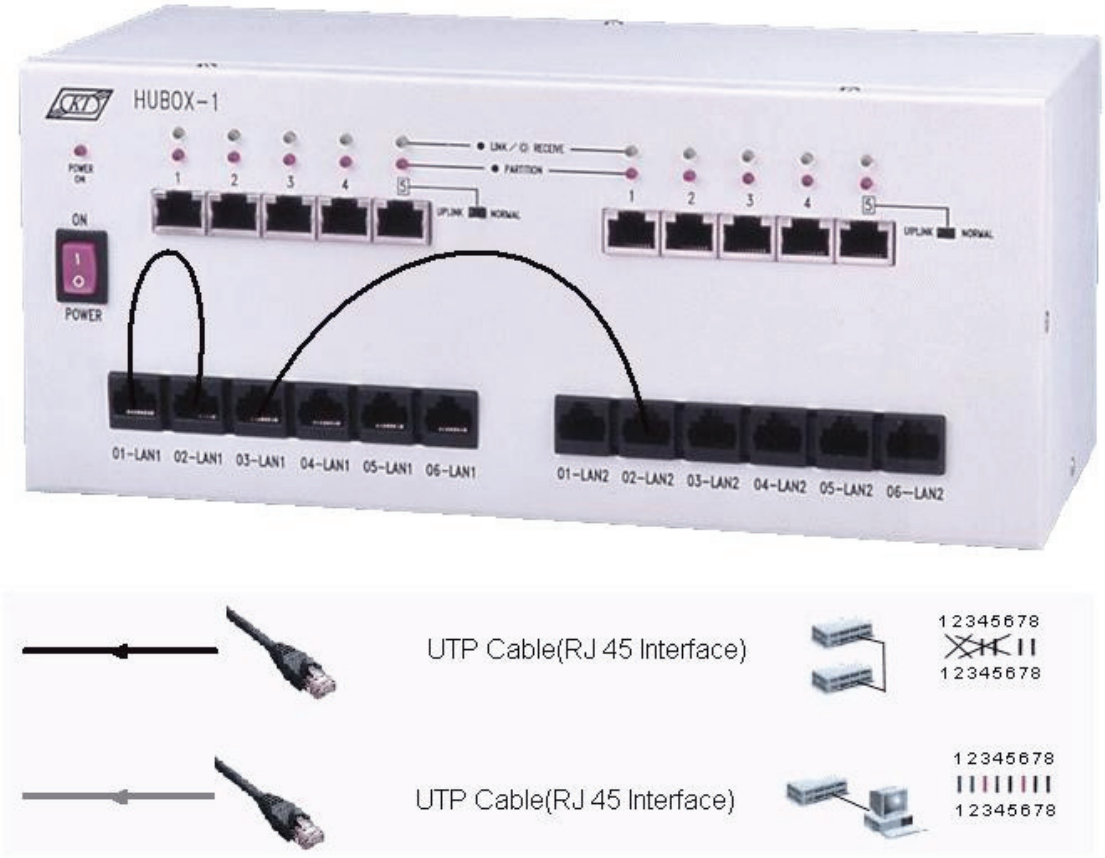


Figure 10.1

Generating Network Disturbance by MDDL Platform

A. Set Host and Gateway

1. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
2. Open the Network Configuration dialog box by selecting **Network Configuration** from the Tool menu.

ITS1 (Host)

3. Refer to Topology. Type "**192.168.1.1**" into IP Address of Interface 1 as shown in Figure 10.2. Then click the **Add new routing entry** button.

4. See Figure 10.3. Type "**192.168.2.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.1.2**" into Gateway. Click the **Update** button.

5. Choose **Host** and click the **Set & Close** button.

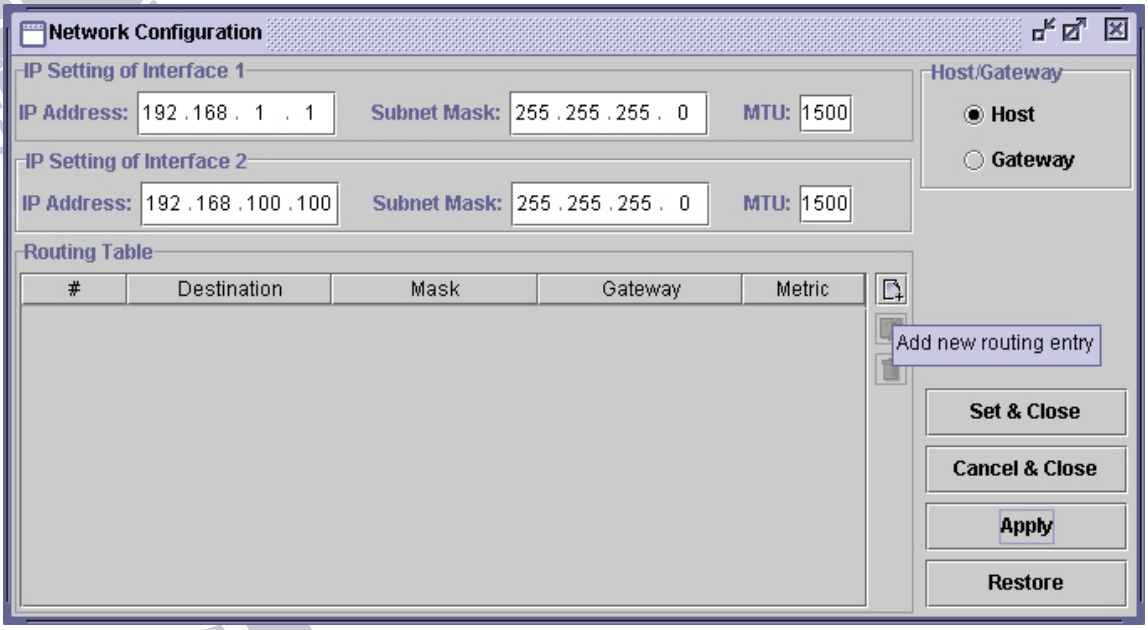


Figure 10.2

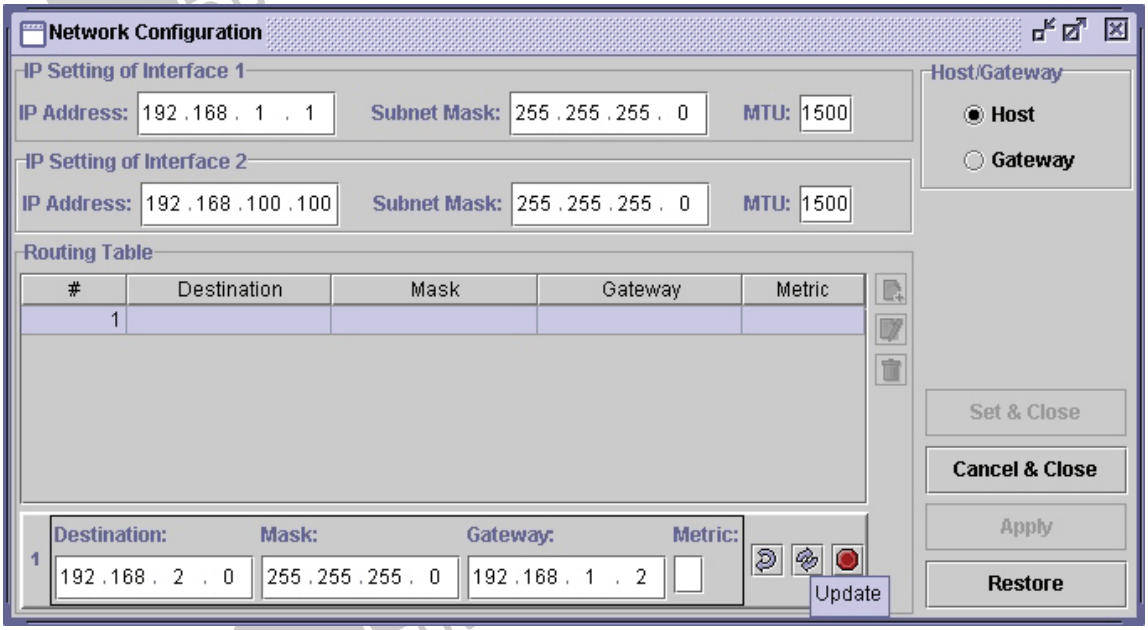


Figure 10.3

ITS3 (Host)

6. In the Network Configuration dialog, type "**192.168.2.2**" into IP Address of Interface 1. Click the **Add new routing entry** button.
7. Type "**192.168.1.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.2.1**" into Gateway. Click the **Update** button.
8. Choose **Host** and click the **Set & Close** button.

ITS2 (Gateway)

9. Refer to Topology A. Type "**192.168.1.2**" into IP Address of Interface 1 and "**192.168.2.1**" into IP Address of Interface 2. (See Figure 10.4.)
10. Choose **Gateway** and click the **Set & Close** button. Now, we already set up our routing table in ITS.

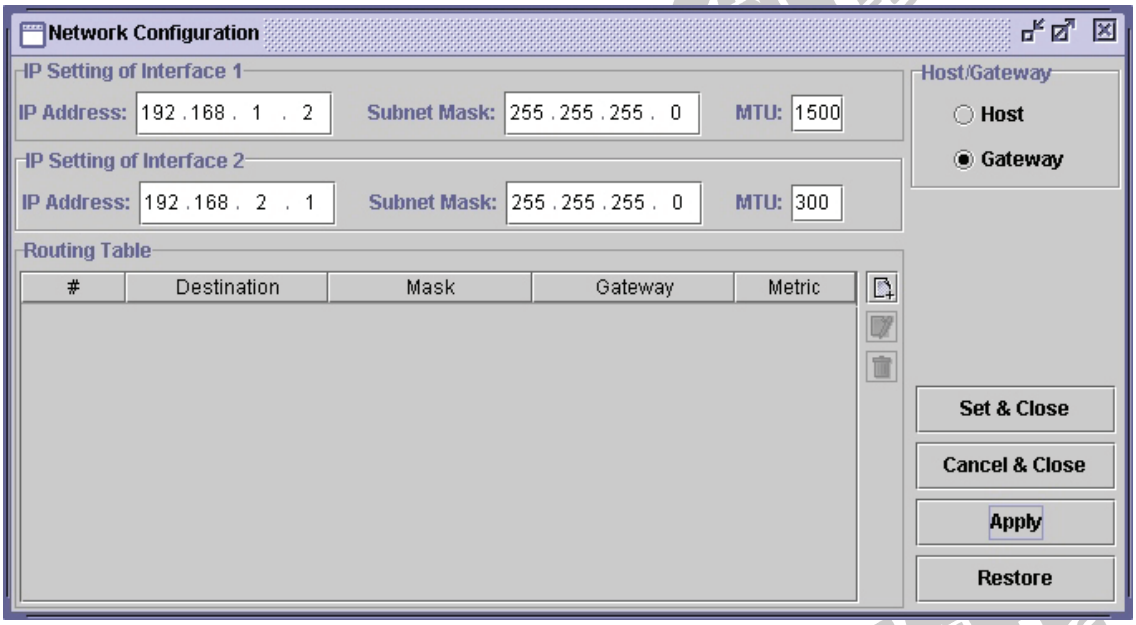


Figure 10.4

B. Making Random Packet Lost

ITS2

11. Open the Network Message Browser window. Check **Listening On**.
12. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
13. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktLost.mddl, and click the **UpId** button. (See Figure 10.5.)

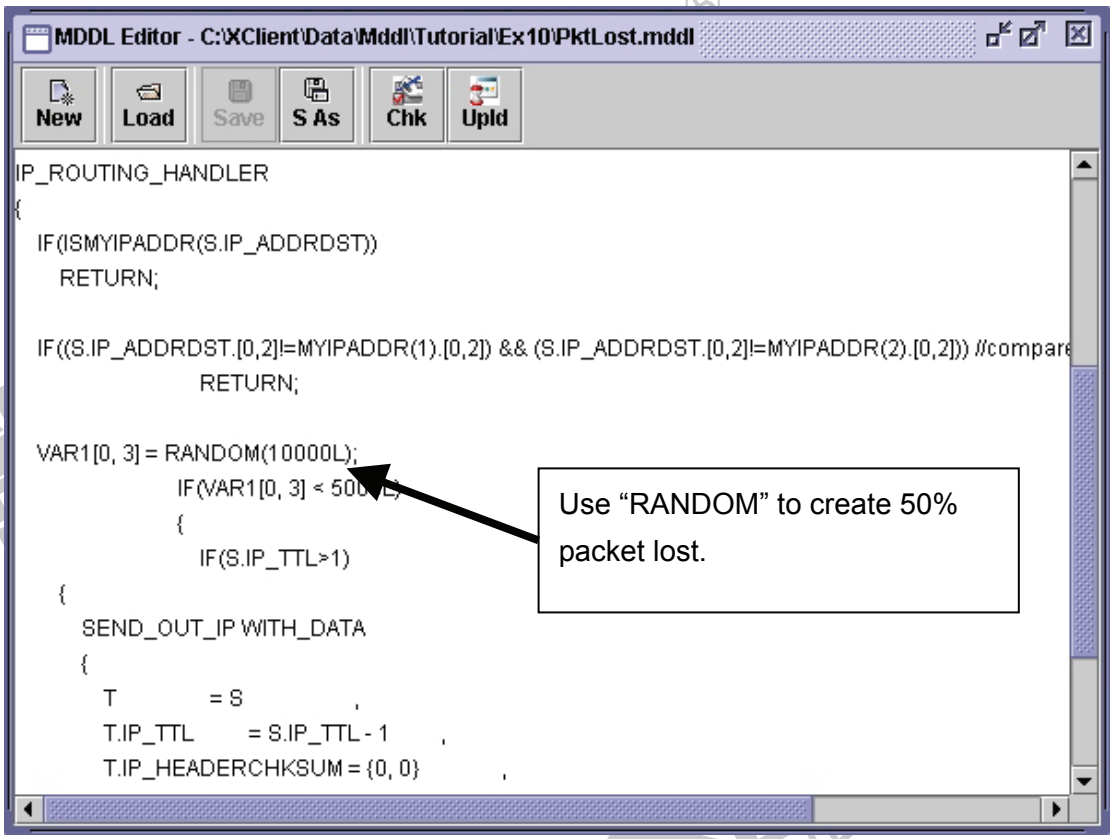


Figure 10.5

ITS3

14. Open the Network Message Browser window. Check **Listening On**.

ITS1

15. Open the Network Message Browser window. Check **Listening On**.
16. Referring to the previous experiments, send ICMP Echo Request to ITS3. You will find that 50% packets have been discarded.

C. Making Packet Delay

ITS2

17. Reset the Network Message Browser window.
18. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.

19. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktDelay.mddl, and click the **UpId** button. This program provides a routing policy that has 50% delay packets.

ITS1 and ITS3

20. Reset the Network Message Browser window.
21. Referring to the previous experiments, send ICMP Echo Request from ITS1 to ITS3. Observe data transmission and packet delay from the Network Message Browser.

D. Making Specific Packet Lost

ITS2

22. Reset the Network Message Browser window.
23. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
24. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktLost4.mddl, and click the **UpId** button. This program provides a routing policy that MDDL will discard the fourth packet every 5 packets that are transferred from ITS1 to ITS3.

ITS1 and ITS3

25. Reset the Network Message Browser window.
26. Referring to the previous experiments, send ICMP Echo Request from ITS1 to ITS3. Observe data transmission and packet lost from the Network Message Browser.

DISCUSSIONS

1. Write a Subnet-Routing Reactor program which has 20% probability of packet lost.
2. In case of Making Packet Delay, try to ping ITS2 and observe the delay from the Command Prompt.

REACTOR PROGRAMS

1. **PktLost.mddl**

```
IP_ROUTING_HANDLER
{
    IF(ISMYIPADDR(S.IP_ADDRDST))
        RETURN;

    IF((S.IP_ADDRDST.[0,2]!=MYIPADDR(1).[0,2]) && (S.IP_ADDRDST.[0,2]!=MYIPADDR(2).[0,2]))
                                                    //compare with netmask 255.255.255.0
     RETURN;

    VAR1[0, 3] = RANDOM(10000L);
   IF(VAR1[0, 3] < 5000L)
   {
      IF(S.IP_TTL>1)
       {
          SEND_OUT_IP WITH_DATA
          {
              T                       = S                       ,
              T.IP_TTL                = S.IP_TTL - 1            ,
              T.IP_HEADERCHKSUM = {0, 0}                        ,
              T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
          }
       }
    }
    DISCARD_MESSAGE;
}
```

2. **PktDelay.mddl**

```
IP_ROUTING_HANDLER
{
  IF(ISMYIPADDR(S.IP_ADDRDST))
  RETURN;

  IF((S.IP_ADDRDST.[0,2]!= MYIPADDR(1).[0,2])&&(S.IP_ADDRDST.[0,2]!=MYIPADDR(2).[0,2]))
                                                    //compare with netmask 255.255.255.0
   RETURN;

  VAR1[0, 3] = RANDOM(10000L);
  IF(VAR1[0, 3] < 5000L)
  {
       IF(S.IP_TTL>1)
      {
          SEND_OUT_IP WITH_DATA
          {
              T                       = S                       ,
              T.IP_TTL                = S.IP_TTL - 1            ,
              T.IP_HEADERCHKSUM     = {0, 0}                    ,
              T.IP_HEADERCHKSUM     = CHECKSUM(T.IP_HEADER)
          }
      }
  }
  ELSE IF(VAR1[0, 3] < 10000L)
  {
       ADD_TO_POOL 22 WITH_LIFETIME 20000 WITH_DATA
       {
```

```
            T.[0]      = 10                    ,
            T.[6, ]    = SOURCE
        }
    }
    DISCARD_MESSAGE;
}

TIMER_WITH_PERIOD 100
{
    FOR_EVERY_ELEMENT_IN_POOL 22
    {
      PE[0] = PE[0] - 1;
      IF(PE[0] == 0)
      {
          SEND_OUT_IP WITH_DATA
          {
                TARGET                  = PE[6, ],
                T.IP_TTL                = PE.IP_TTL - 1,
                T.IP_LEN                = LENGTH(T),
                T.IP_HEADERCHKSUM       = {0, 0}     ,
                T.IP_HEADERCHKSUM       = CHECKSUM(T.IP_HEADER)
          }
           REMOVE_CURRENT_POOL_ELEMENT;
      }
    }
}
```

3. **PktLost4.mddl**

```
VAR1[0] = 0;


IP_ROUTING_HANDLER
{
  IF(ISMYIPADDR(S.IP_ADDRDST))
   RETURN;

   IF(S.IP_ADDRDST.[0,2]!= MYIPADDR(2).[0,2])    //compare with netmask 255.255.255.0
   RETURN;

   VAR1[0] = VAR1[0]+1 ;

   GENERATE_USER_MSG WITH_DATA
   {
        TARGET = VAR1[0]
   }

  IF(VAR1[0] !=4)
  {
     IF (VAR1[0] == 5)
     {
        VAR1[0] = 0 ;
     }
     IF(S.IP_TTL>1)
     {
        SEND_OUT_IP WITH_DATA
        {
          T                 = S                     ,
          T.IP_TTL          = S.IP_TTL - 1
```

```
            T.IP_HEADERCHKSUM = {0, 0}                    ,
            T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
            }
          }

  }
   DISCARD_MESSAGE;
}
```

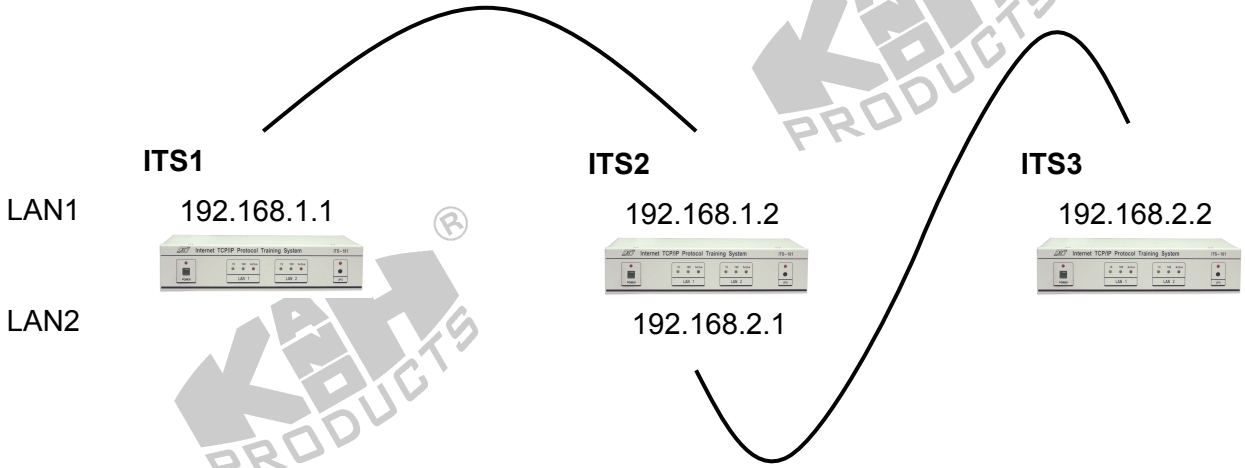
Exp 11. Error Control

OBJECTIVE：To understand the error control protocol how to solve the problems of packets lost and delay.

BRIEF DESCRIPTION：This experiment examines the error control mechanism that is used to ensure the reliable TCP connection. By using MDDL, students can learn how to implement the mechanism.

DURATION：6 hrs

TOPOLOGY



TECHNICAL BACKGROUND

The error control is to manage the packet with error detection/correction. The most used error control protocol in data link layer is automatic repeated request (ARQ). When the receiver detects an error in a packet, it automatically requests the transmitter to resend the packet. This process is repeated until the packet is error free or the error continues beyond a predetermined number of transmissions.

Types of ARQ:

1. Idle RQ
    a. Implicit retransmit request
    b. Explicit retransmit request

2. Continuous RQ
    a. Selective repeat
    b. Go-back-N

In this experiment, we will only consider the implementation of implicit retransmit request for Idle RQ and selective repeat Continuous RQ.

1. Idle RQ (Implicit retransmit request)
    a. Sender transmits an I-frame (information bearing frame) to the receiver.
    b. Sender waits for an ACK from the receiver.
    c. After receiving an ACK, the sender transmits a new I-frame.
    d. Note: Both I-frames and ACKs may be lost or corrupted.
    e. Operates at half duplex transmission mode (regardless of the actual connection).

2. Continuous RQ- Selective repeat (Implicit retransmit request)
    a. The sender sends frames continuously without waiting for ACKs.
    b. The receiver transmits an ACK for each correctly received I-frame.
    c. The sender maintains a retransmission list.
    d. The receiver maintains a receive list.
    e. A corrupted frame is detected when a frame with the next sequence number is received.
    f. The ACK for frame N acknowledges all frames in the retransmission list up to and including frame N.

To implement the idle RQ for IP communication, we define a new protocol over IP packet as shown in Figure 11.1.

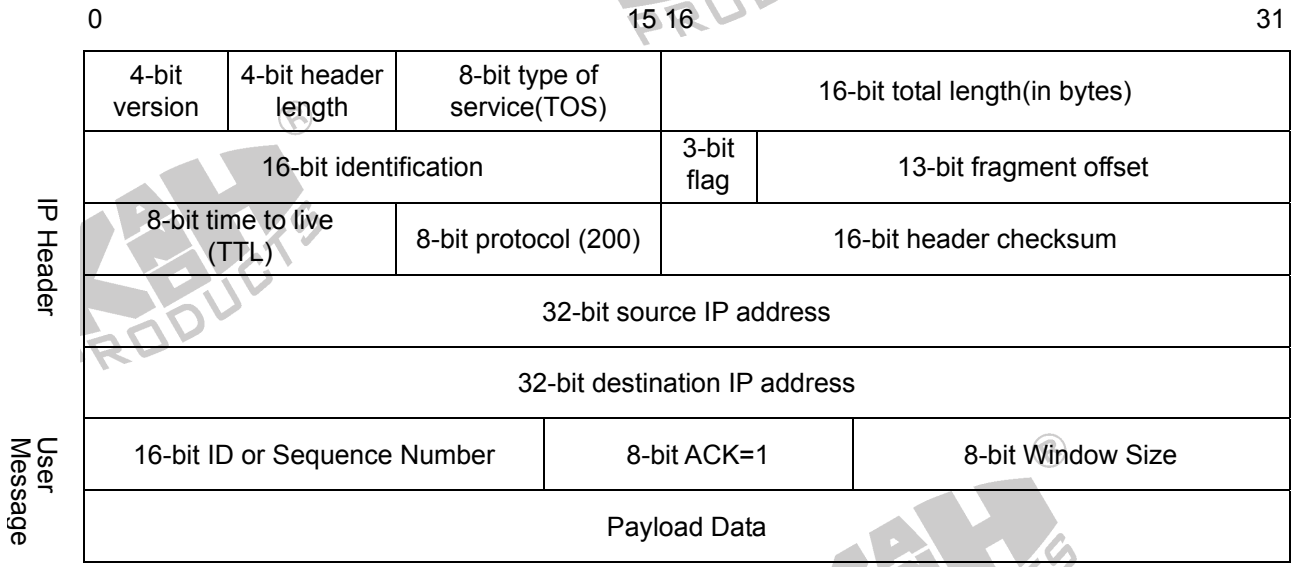


Figure 11.1

The protocol type field in IP protocol header is set to 200, identifying the protocol of this experiment, 16-bit ID is set to identify the packet in Idle RQ or 16-bit Sequence Number is set to sequentially identify the packet in Continuous RQ, 8-bit ACK is set to 1 indicating the Acknowledge packet, 8-bit Window Size is not used here, and Payload Data field is used to carry the user data. Now we discuss the two types of idle RQ:

a. Idle RQ without packet identification (see Figure 11.2): A simplified idle repeat request (RQ) error control scheme which does not contain information of packet identification in protocol.

b. Idle RQ (see Figure 11.3): a standard RQ implemented by implicit retransmit request approach.

Idle RQ without Packet Identification

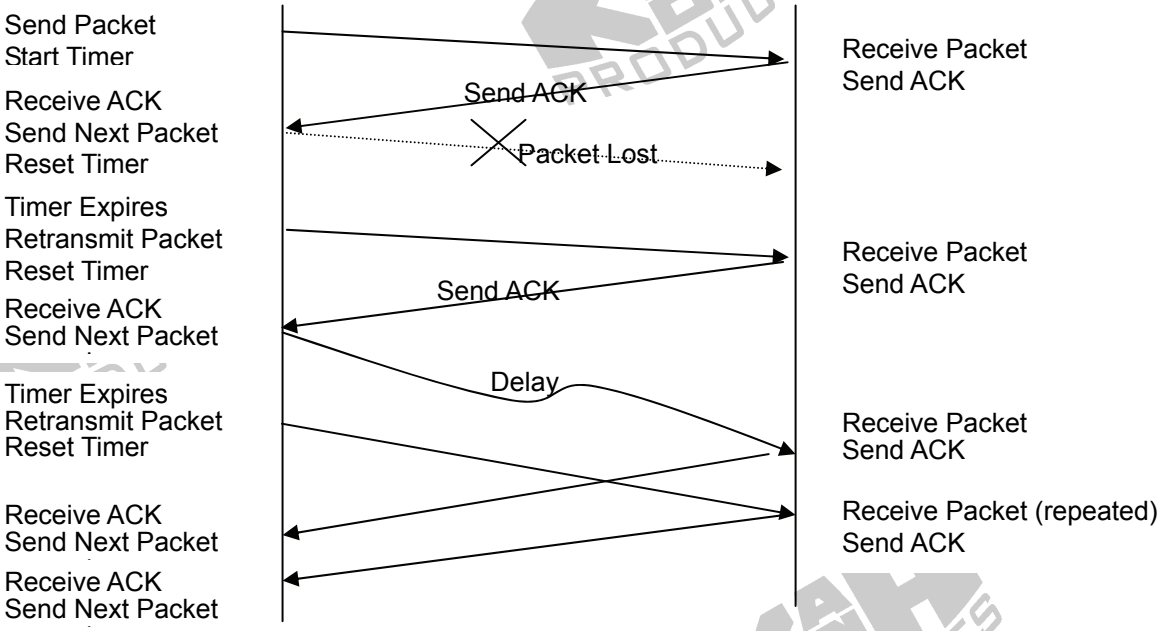


Figure 11.2

Idle RQ

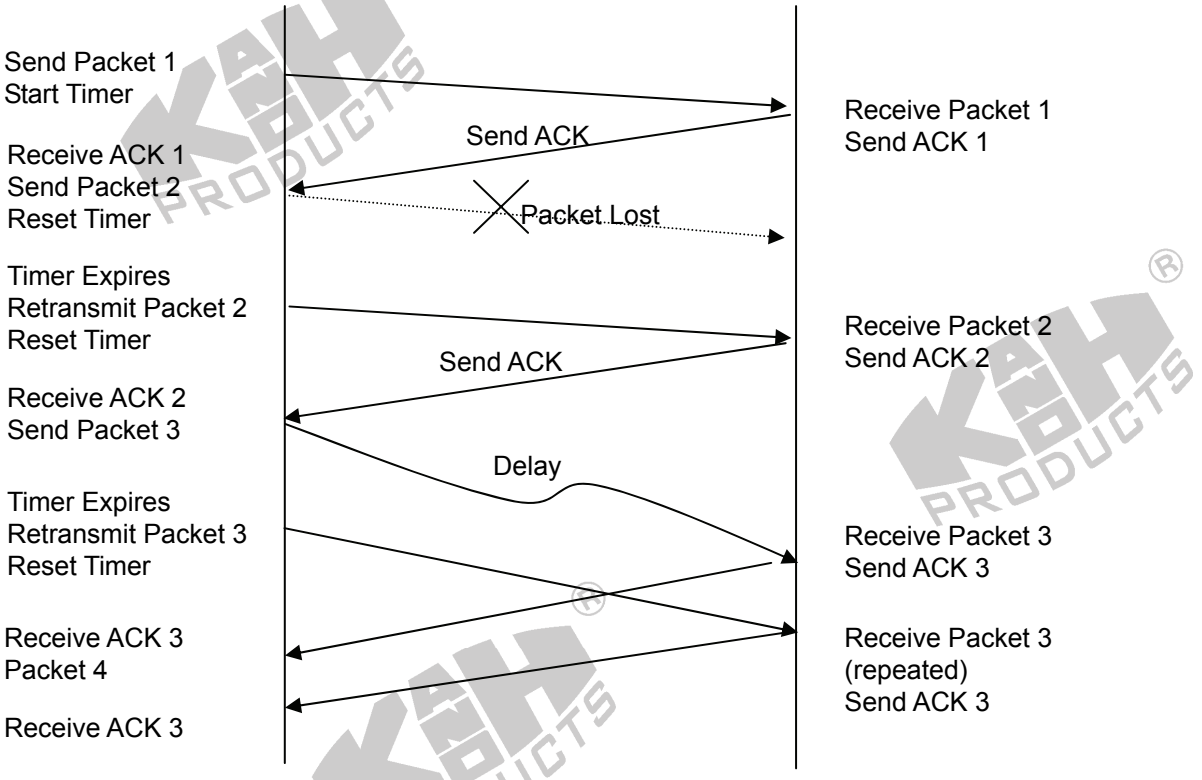


Figure 11.3

Continuous RQ

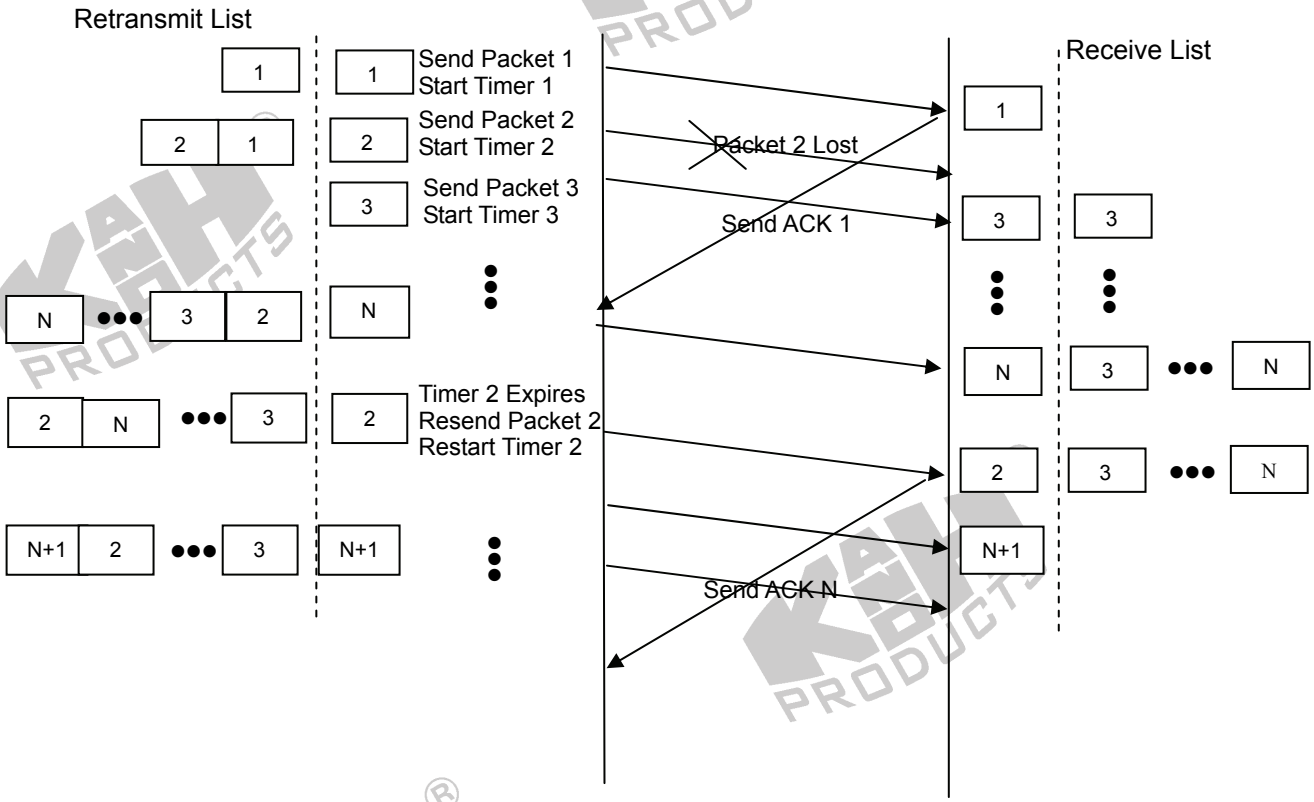


Figure 11.4

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 11.5.

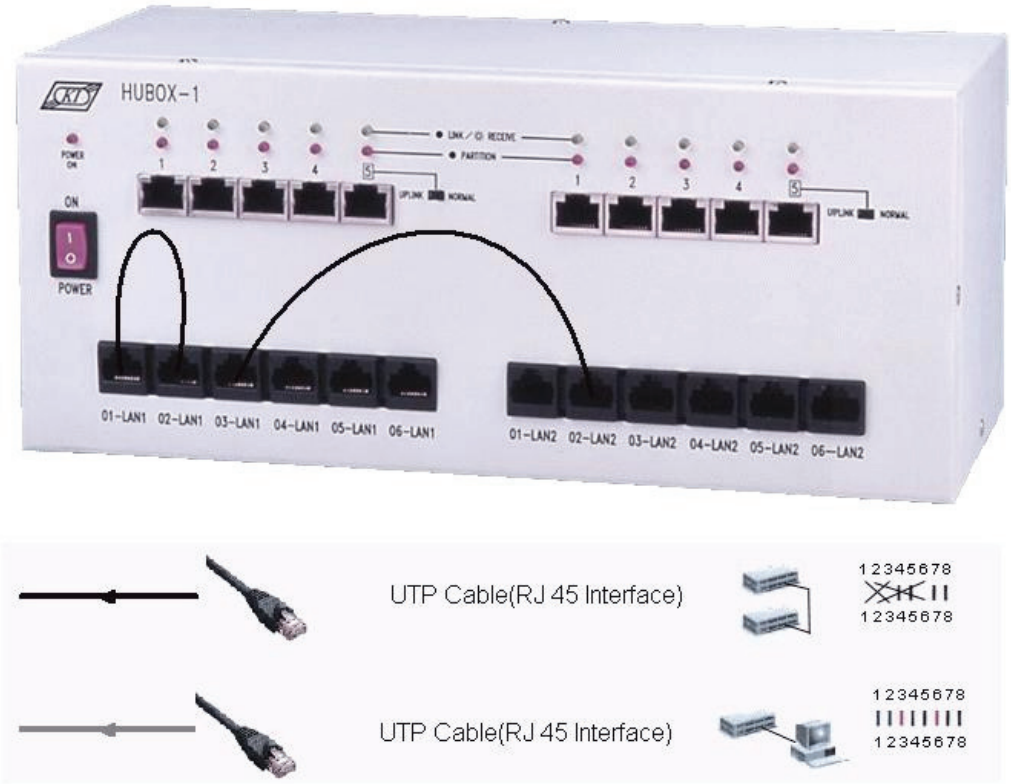


Figure 11.5

Setting Host and Gateway

2. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
3. Open the Network Configuration dialog box by selecting **Network Configuration** from the Tool menu.

ITS1 (Host)

4. Refer to Topology. Type "**192.168.1.1**" into IP Address of Interface 1 (see Figure 11.6), and click the **Add new routing entry** button.
5. Type "**192.168.2.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.1.2**" into Gateway (see Figure 11.7), and click the **Update** button.
6. Choose **Host** and click the **Set & Close** button.

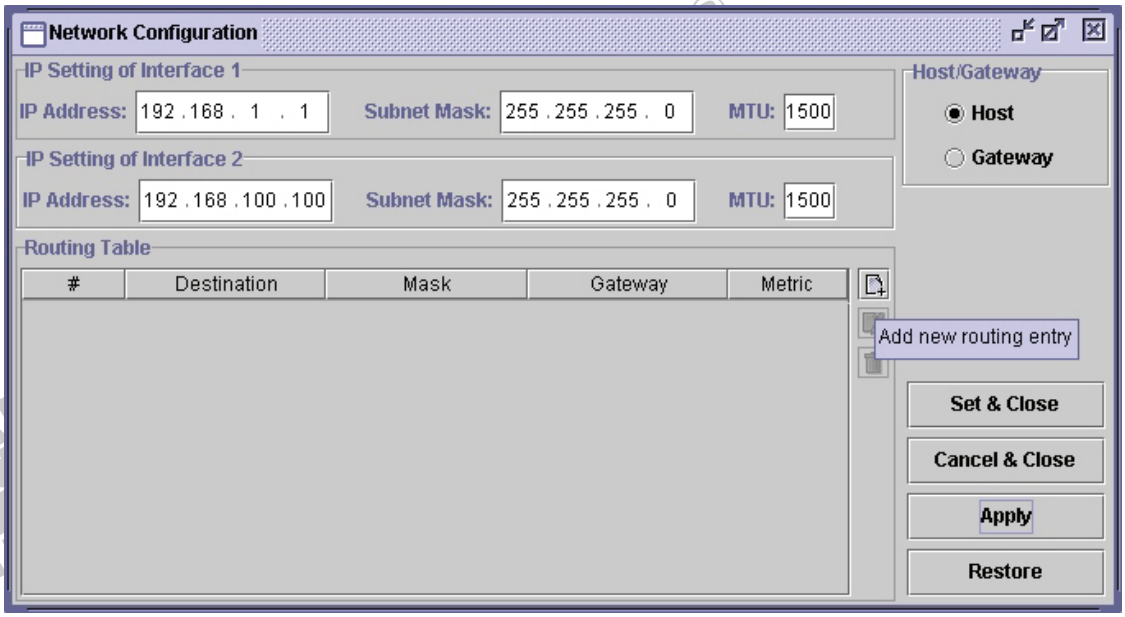


Figure 11.6

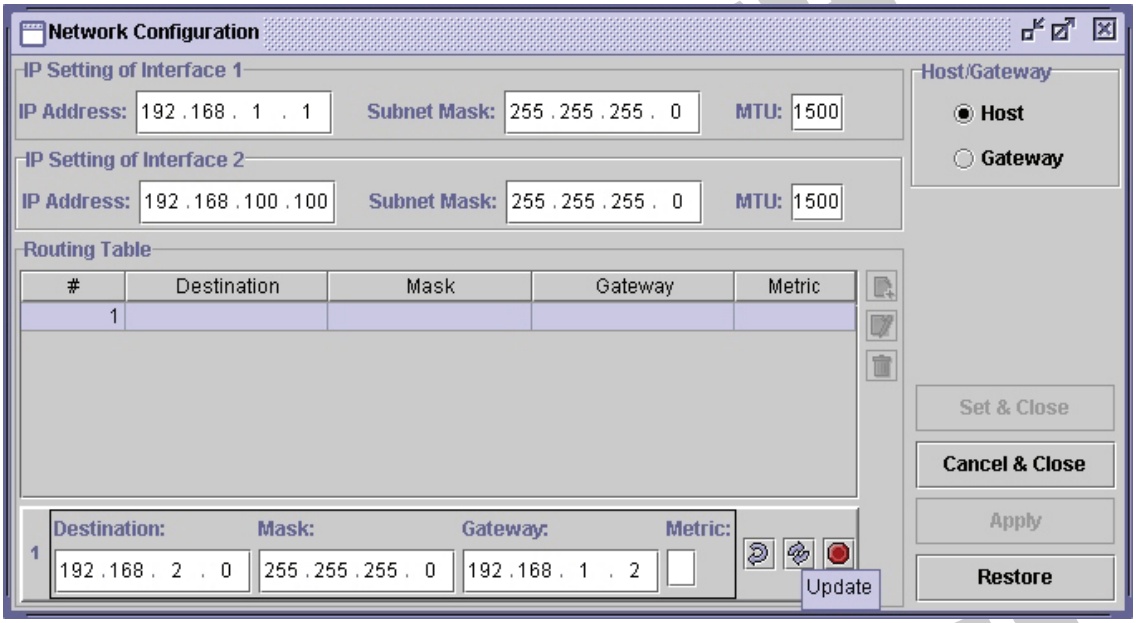


Figure 11.7

ITS3 (Host)

7. Type "**192.168.2.2**" into IP Address of Interface 1. Click the **Add new routing entry** button.
8. Type "**192.168.1.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.2.1**" into Gateway. Then click the **Update** button.
9. Choose **Host** and click the **Set & Close** button.

ITS2 (Gateway)

10. Refer to Topology. Type "**192.168.1.2**" into IP Address of Interface 1 and "**192.168.2.1**" into IP Address of Interface 2. (See Figure 11.8.)
11. Choose **Gateway** and click the **Set & Close** button. Now, we already set up our routing table in ITS.

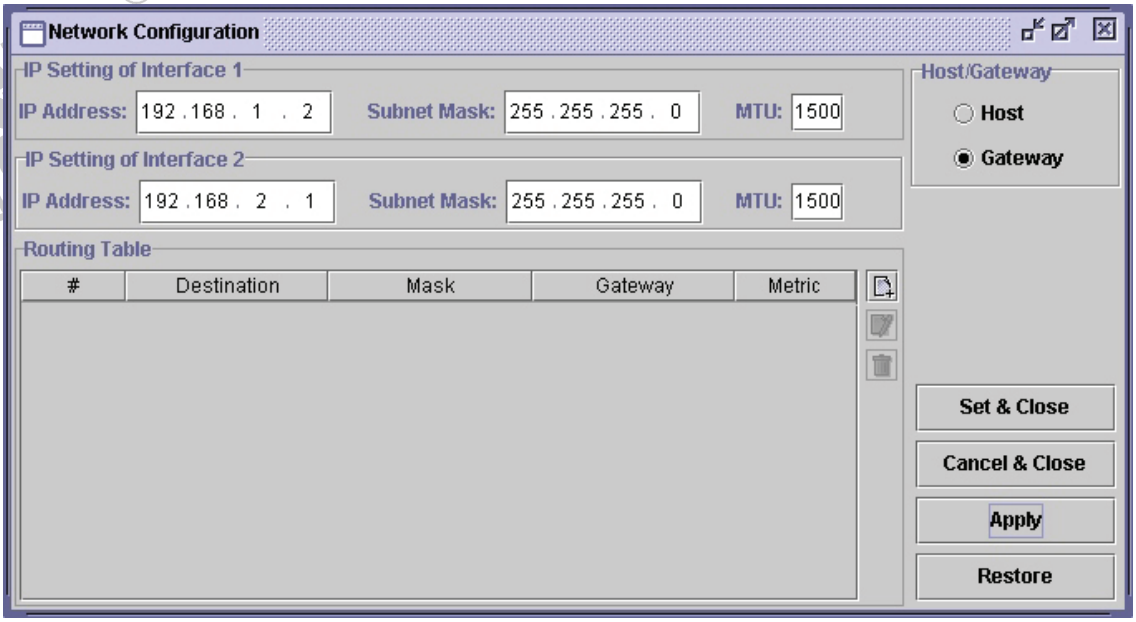


Figure 11.8

Retransmission and RTT (Round Trip Time)

ITS2

12. Open the Network Message Browser window. Check **Listening On**.
13. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
14. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \PktLostDelay-11-1.mddl, and click the **UpId** button. In this program, when ITS1's packets is sent over ITS2, the first packet of every 5 packets will pass without incident. The second packet of every 5 packets will delay 4 seconds. The third packet of every 5 packets will delay 7 seconds. The fourth packet of every 5 packets will pass without incident and the fifth packet of every 5 packets will be discarded.

ITS3

15. Open the Network Message Browser window. Check **Listening On**.
16. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
17. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \NoIDIdleRQReceiver.mddl, and click the **UpId** button.

ITS1

18. Open the Network Message Browser window. Check **Listening On**.
19. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
20. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \NoIDIdleRQSender.mddl, and click the **UpId** button.
21. Select **Send IP Packet** from the Send menu to open the IP Datagram Sender.
22. Type "**7**" into Protocol and "**192.168.2.2**" into Destination IP Address. Enter "**check**" into Data as shown in Figure 11.9.
23. Finally click the **Send** button. ITS1 will send an IP datagram to ITS3 then receive ACK from ITS3. Try to send more IP datagrams. We will see the difference between packet lost and packet delay.

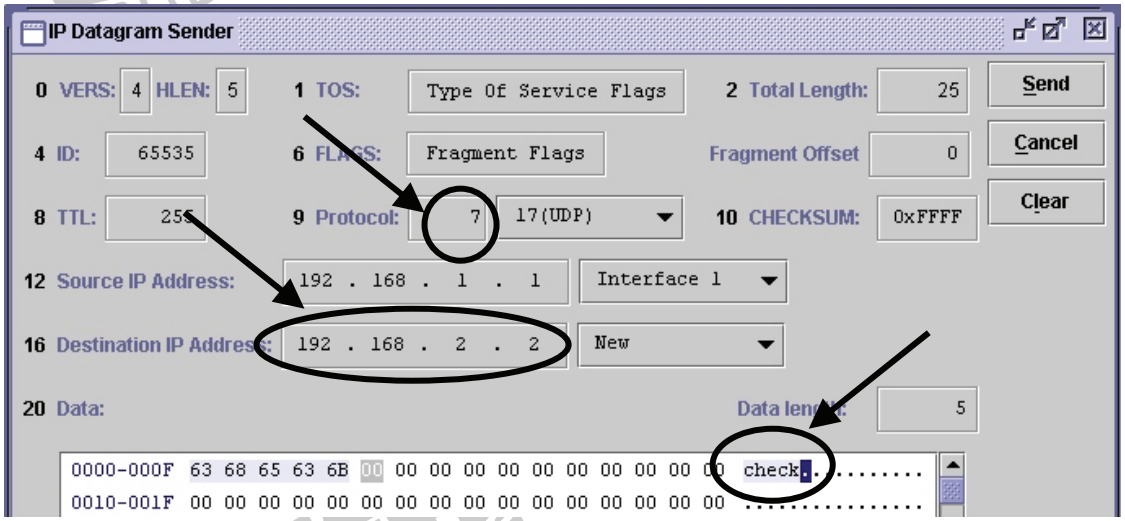
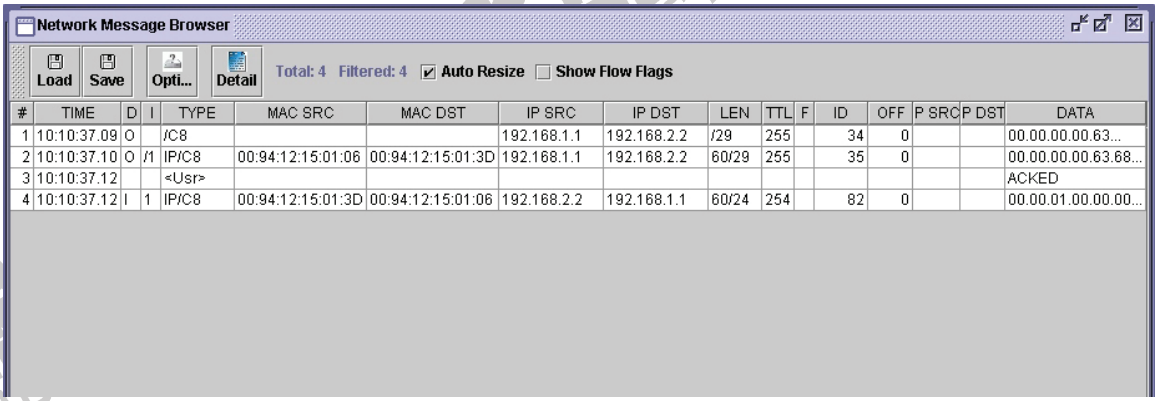


Figure 11.9

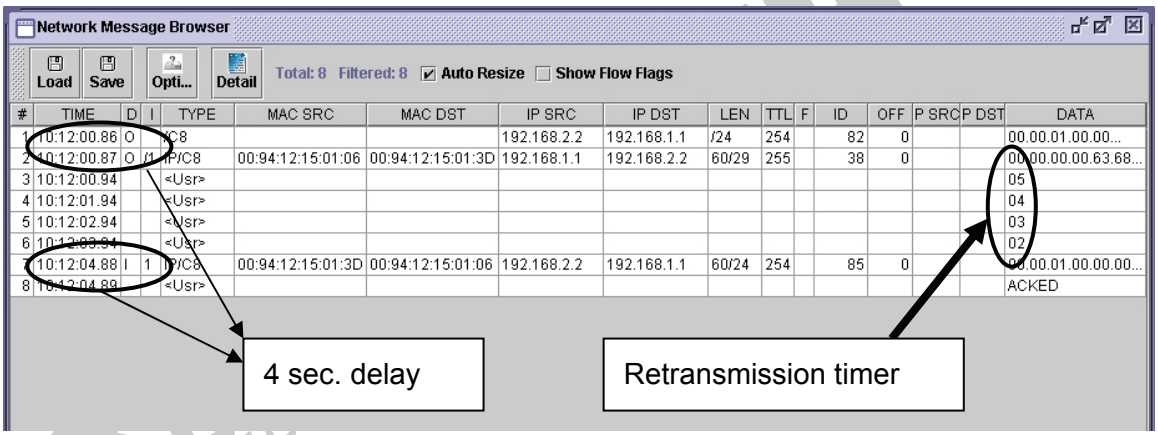
Figure 11.10 shows that ITS1 sends an IP datagram to ITS3 and receives an ACK from ITS3 without incident.



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:10:37.09	O		/C8			192.168.1.1	192.168.2.2	/29	255		34	0			00.00.00.00.63...
2	10:10:37.10	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		35	0			00.00.00.00.63.68...
3	10:10:37.12			<Usr>												ACKED
4	10:10:37.12	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		82	0			00.00.01.00.00.00...

Figure 11.10

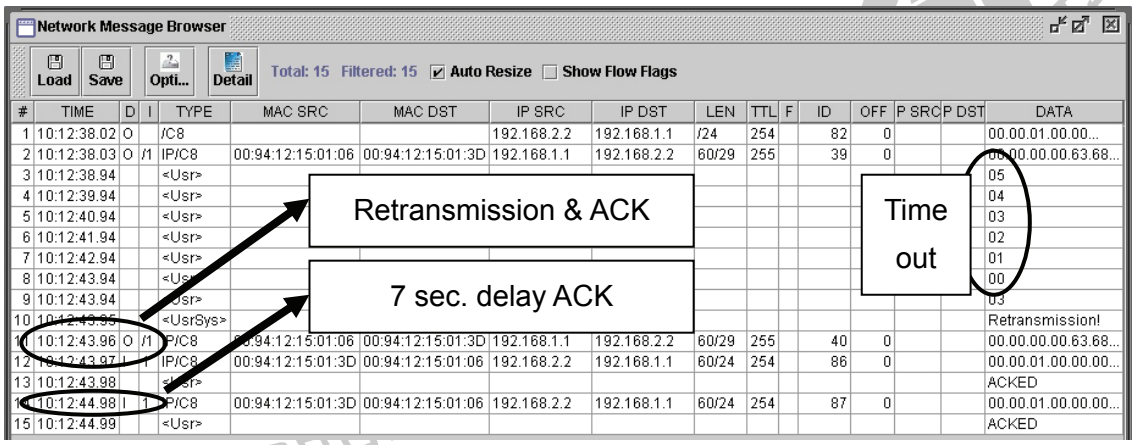
Figure 11.11 shows that ITS1 sends an IP datagram to ITS3 with a delay of 4 seconds.



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:12:00.86	O		/C8			192.168.2.2	192.168.1.1	/24	254		82	0			00.00.01.00.00...
2	10:12:00.87	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		38	0			00.00.00.00.63.68...
3	10:12:00.94			<Usr>												05
4	10:12:01.94			<Usr>												04
5	10:12:02.94			<Usr>												03
6	10:12:03.94			<Usr>												02
7	10:12:04.88	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		85	0			00.00.01.00.00.00...
8	10:12:04.89			<Usr>												ACKED

Figure 11.11

Figure 11.12 shows that ITS1 sends an IP datagram to ITS3 with a delay of 7 seconds and into retransmission.



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:12:38.02	O		/C8			192.168.2.2	192.168.1.1	/24	254		82	0			00.00.01.00.00...
2	10:12:38.03	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		39	0			00.00.00.00.63.68...
3	10:12:38.94			<Usr>												05
4	10:12:39.94			<Usr>												04
5	10:12:40.94			<Usr>												03
6	10:12:41.94			<Usr>												02
7	10:12:42.94			<Usr>												01
8	10:12:43.94			<Usr>												00
9	10:12:43.94			<Usr>												03
10	10:12:43.95			<UsrSys>												Retransmission!
11	10:12:43.96	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		40	0			00.00.00.00.63.68...
12	10:12:43.97	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		86	0			00.00.01.00.00.00...
13	10:12:43.98			<Usr>												ACKED
14	10:12:44.98	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		87	0			00.00.01.00.00.00...
15	10:12:44.99			<Usr>												ACKED

Figure 11.12

Figure 11.13 shows that ITS1 sends an IP datagram to ITS3, but that packet has lost and into retransmission.

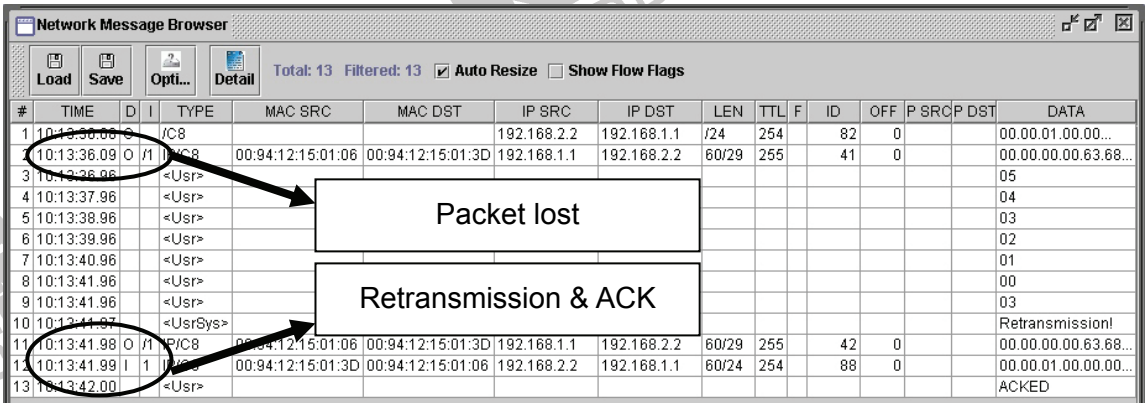


Figure 11.13

Error Control

A. Idle RQ without Packet Identification

ITS2

24. Open the Network Message Browser window. Check **Listening On**.
25. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
26. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktLost4.mddl, and click the **UpId** button.

ITS3

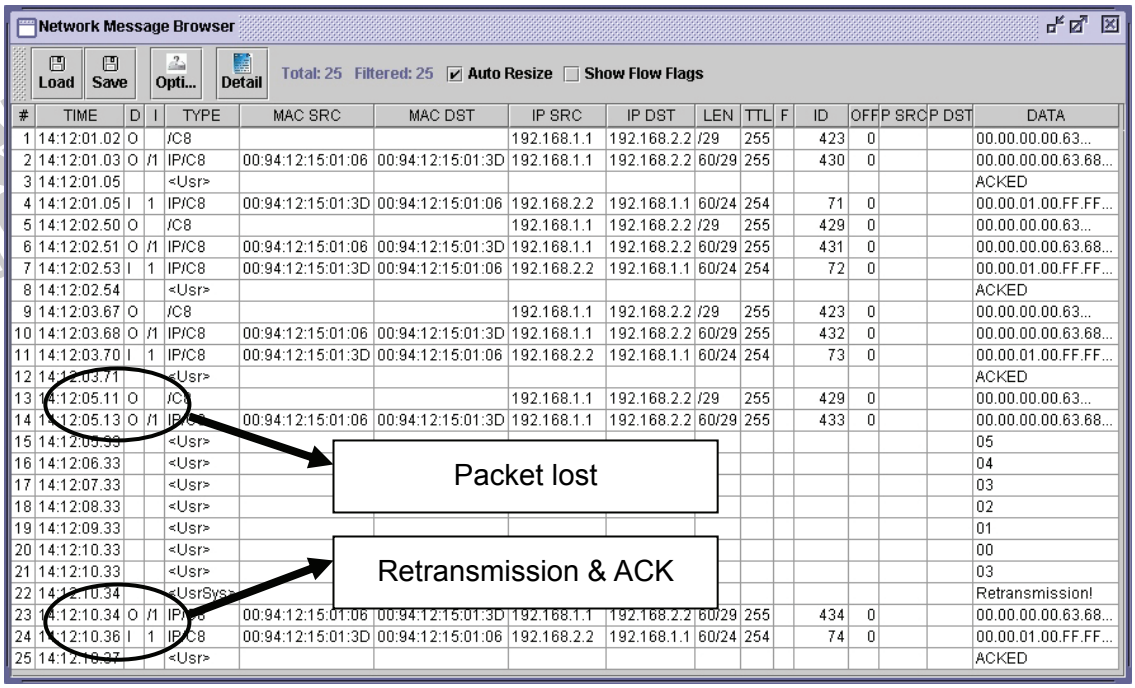
27. Open the Network Message Browser window. Check **Listening On**.
28. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
29. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \NoIDIdleRQReceiver.mddl, and click the **UpId** button.

ITS1

30. Open the Network Message Browser window. Check **Listening On**.
31. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.

32. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \NoIDIdleRQSender.mddl, and click the **UpId** button.

33. Referring to the previous experiments. Send IP datagrams to ITS3 then receive ACK from ITS3. Observe the packet lost of transmission. (See Figure 11.14)



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	14:12:01.02	O		/C8			192.168.1.1	192.168.2.2	/29	255		423	0			00.00.00.00.63...
2	14:12:01.03	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		430	0			00.00.00.00.63.68...
3	14:12:01.05			<Usr>												ACKED
4	14:12:01.05	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		71	0			00.00.01.00.FF.FF...
5	14:12:02.50	O		/C8			192.168.1.1	192.168.2.2	/29	255		429	0			00.00.00.00.63...
6	14:12:02.51	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		431	0			00.00.00.00.63.68...
7	14:12:02.53	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		72	0			00.00.01.00.FF.FF...
8	14:12:02.54			<Usr>												ACKED
9	14:12:03.67	O		/C8			192.168.1.1	192.168.2.2	/29	255		423	0			00.00.00.00.63...
10	14:12:03.68	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		432	0			00.00.00.00.63.68...
11	14:12:03.70	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		73	0			00.00.01.00.FF.FF...
12	14:12:03.71			<Usr>												ACKED
13	14:12:05.11	O		/C8			192.168.1.1	192.168.2.2	/29	255		429	0			00.00.00.00.63...
14	14:12:05.13	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		433	0			00.00.00.00.63.68...
15	14:12:05.33			<Usr>												05
16	14:12:06.33			<Usr>												04
17	14:12:07.33			<Usr>												03
18	14:12:08.33			<Usr>												02
19	14:12:09.33			<Usr>												01
20	14:12:10.33			<Usr>												00
21	14:12:10.33			<Usr>												03
22	14:12:10.34			<UsrSys>												Retransmission!
23	14:12:10.34	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		434	0			00.00.00.00.63.68...
24	14:12:10.36	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		74	0			00.00.01.00.FF.FF...
25	14:12:10.37			<Usr>												ACKED

Figure 11.14

B. Idle RQ

ITS2

34. Open the Network Message Browser window. Check **Listening On**.

35. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.

36. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktLost4.mddl, and click the **UpId** button.

ITS3

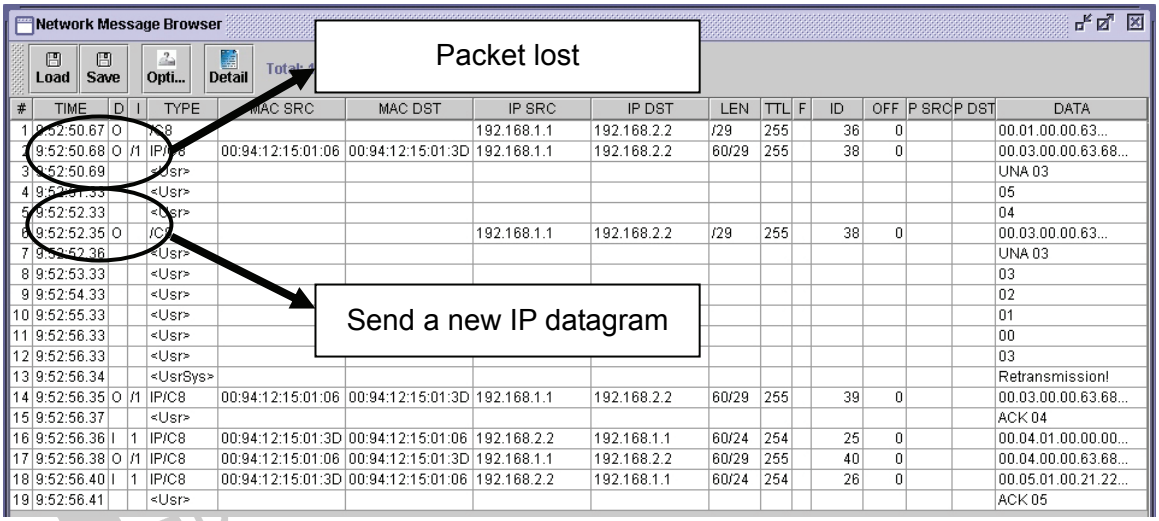
37. Open the Network Message Browser window. Check **Listening On**.

38. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.

39. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \SIDIdleRQReceiver.mddl, and click the **UpId** button.

ITS1

40. Open the Network Message Browser window. Check **Listening On**.
41. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
42. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \SIDIdleRQSender.mddl, and click the **UpId** button.
43. Send IP datagrams to ITS3 by referring to the previous experiments. When the packet lost occurs, ITS1 resend a new IP datagram immediately. We can see that the new datagram hold in sending buffer until the retransmission finishes. This is the standard RQ approach. (See Figure 11.15)



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	9:52:50.67	O		/C8			192.168.1.1	192.168.2.2	/29	255		36	0			00.01.00.00.63...
2	9:52:50.68	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		38	0			00.03.00.00.63.68...
3	9:52:50.69			<Usr>												UNA 03
4	9:52:51.33			<Usr>												05
5	9:52:52.33			<Usr>												04
6	9:52:52.35	O		/C8			192.168.1.1	192.168.2.2	/29	255		38	0			00.03.00.00.63...
7	9:52:52.36			<Usr>												UNA 03
8	9:52:53.33			<Usr>												03
9	9:52:54.33			<Usr>												02
10	9:52:55.33			<Usr>												01
11	9:52:56.33			<Usr>												00
12	9:52:56.33			<Usr>												03
13	9:52:56.34			<UsrSys>												Retransmission!
14	9:52:56.35	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		39	0			00.03.00.00.63.68...
15	9:52:56.37			<Usr>												ACK 04
16	9:52:56.36	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		25	0			00.04.01.00.00.00...
17	9:52:56.38	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		40	0			00.04.00.00.63.68...
18	9:52:56.40	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		26	0			00.05.01.00.21.22...
19	9:52:56.41			<Usr>												ACK 05

Figure 11.15

C. Continuous RQ

ITS2

44. Open the Network Message Browser window. Check **Listening On**.
45. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
46. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PktLost4.mddl, and click the **UpId** button.

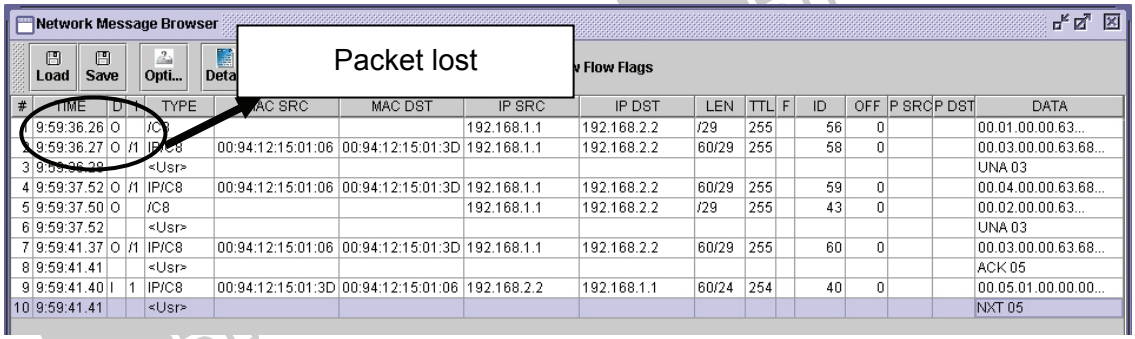
ITS3

47. Open the Network Message Browser window. Check **Listening On**.

48. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the **Reactor** menu.
49. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \SIDCRQReceiver.mddl, and click the **UpId** button.

ITS1

47. Open the Network Message Browser window. Check **Listening On**.
48. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the **Reactor** menu.
49. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex11 \SIDCRQSender.mddl , and click the **UpId** button.
50. Send IP datagrams to ITS3 by referring to the previous experiments. When the packet lost occurs, ITS1 resend a new IP datagram immediately. We can see that the new datagram is sent directly and needn't to wait a finish of transmission. (See Figure 11.16.)



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	9:59:36.26	O		/C8			192.168.1.1	192.168.2.2	/29	255		56	0			00.01.00.00.63...
2	9:59:36.27	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		58	0			00.03.00.00.63.68...
3	9:59:36.28			<Usr>												UNA 03
4	9:59:37.52	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		59	0			00.04.00.00.63.68...
5	9:59:37.50	O		/C8			192.168.1.1	192.168.2.2	/29	255		43	0			00.02.00.00.63...
6	9:59:37.52			<Usr>												UNA 03
7	9:59:41.37	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		60	0			00.03.00.00.63.68...
8	9:59:41.41			<Usr>												ACK 05
9	9:59:41.40	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		40	0			00.05.01.00.00.00...
10	9:59:41.41			<Usr>												NXT 05

Figure 11.16

DISCUSSIONS

1. RTT (Round Trip Time) is the total time for a packet sent by a node to reach its destination. We usually set the retransmission time double the RTT. What is the effect if we change retransmission time to be longer or shorter? Try to rewrite the reactor program of sender then discuss the effect.
2. Is Continuous RQ more efficient than Idle RQ? How many IP datagrams can be held in sending buffer in Idle RQ? Try to send more IP datagrams when retransmission happened and observe it.

REACTOR PROGRAMS

1. **PktLostDelay-11-1.mddl**

```
VAR1[0] =0 ;

IP_ROUTING_HANDLER
{
  IF(ISMYIPADDR(S.IP_ADDRDST))
        RETURN;

  IF(S.IP_ADDRDST.[0,2]!= MYIPADDR(2).[0,2]) //compare with netmask 255.255.255.0
      RETURN;


  VAR1[0] = VAR1[0]+1 ;

  GENERATE_USER_MSG WITH_DATA
  {
      TARGET = VAR1[0]
  }

       IF(VAR1[0] == 1)
       {
       IF(S.IP_TTL>1)
        {
            SEND_OUT_IP WITH_DATA
            {
                T                       = S                         ,
                T.IP_TTL                = S.IP_TTL - 1              ,
                T.IP_HEADERCHKSUM = {0, 0}                          ,
                T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
            }
       }

  }
  ELSE IF(VAR1[0] == 2)
 {
     ADD_TO_POOL 22 WITH_LIFETIME 20000 WITH_DATA
     {
```

```
            T.[0]      = 40                    ,
            T.[6, ]    = SOURCE
        }

    }
    ELSE IF(VAR1[0] == 3)
    {
        ADD_TO_POOL 22 WITH_LIFETIME 20000 WITH_DATA
        {
            T.[0]      = 70                    ,
            T.[6, ]    = SOURCE
        }

    }
    ELSE IF(VAR1[0] == 4)
    {
        IF(S.IP_TTL>1)
        {
            SEND_OUT_IP WITH_DATA
            {
                T                       = S                         ,
                T.IP_TTL                = S.IP_TTL - 1              ,
                T.IP_HEADERCHKSUM = {0, 0}                      ,
                T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
            }
        }

    }
    // else if(VAR1[0] == 5)
    // {
    //          // LOST
    // }
    ELSE IF(VAR1[0] == 6)
    {
            IF(S.IP_TTL>1)
        {
            SEND_OUT_IP WITH_DATA
            {
                T                       = S                         ,
                T.IP_TTL                = S.IP_TTL - 1              ,
                T.IP_HEADERCHKSUM = {0, 0}                      ,
                T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
            }
        }
        VAR1[0] = 0 ;
    }

    DISCARD_MESSAGE;
}

TIMER_WITH_PERIOD 100
{
    FOR_EVERY_ELEMENT_IN_POOL 22
    {
      PE[0] = PE[0] - 1;
      IF(PE[0] == 0)
      {
          SEND_OUT_IP WITH_DATA
          {
              TARGET                  = PE[6, ],
```

```
                T.IP_TTL                = PE.IP_TTL - 1,
                T.IP_LEN                = LENGTH(T),
                T.IP_HEADERCHKSUM       = {0, 0}    ,
                T.IP_HEADERCHKSUM       = CHECKSUM(T.IP_HEADER)
            }
             REMOVE_CURRENT_POOL_ELEMENT;
        }
      }
}
```

2. **NoIDIdleRQSender.mddl**

```
VAR2[0, 3]      = {192, 168, 1, 1};     // SRC Address.
VAR2[4, 7]      = {192, 168, 2, 2};     // DST Address.
VAR2[8]          = 0;                    // No output message pending.

IP_OUT_HANDLER
{
    IF( S.IP_ADDRDST != VAR2[4, 7] || S.IP_PROT == CNST_IP_PROT_KDP )
        RETURN;

    DISCARD_MESSAGE;

    IF(VAR2[8]==1)
    {
        ADD_TO_POOL 19 WITH_LIFETIME 1800000 WITH_DATA
        {
            T = S
        }
    }
    ELSE
    {
        ASSIGN_VARIABLE 3 WITH_DATA
        {
            T.[0]       = 6                     ,       // Retry after 6 seconds
            T.[1]       = 4                     ,       // Retry at maximum 3 times
            T.[2, 3]    = LENGTH(S.IP_DATA) ,
            T.[4,]      = S.IP_DATA
        }
        SEND_OUT_IP WITH_DATA
        {
          T.IP_PROT                     = CNST_IP_PROT_KDP    ,
            T.IP_ADDRDST                    = VAR2[4, 7]            ,
          T.IP_DATA.KDP_ID              = 0W                    ,
            T.IP_DATA.KDP_ACK               = 0                     ,
            T.IP_DATA.KDP_WINDOW_SIZE = 0                       ,
            T.IP_DATA.KDP_DATA              = S.IP_DATA
        }
        VAR2[8] = 1;                                    // Output message pending
    }
}

TIMER_WITH_PERIOD 1000
{
  IF(VAR2[8] != 1)                                      // Output message pending
      RETURN;
```

```
        VAR3[0] = VAR3[0] - 1;
    GENERATE_USER_MSG WITH_DATA
    {
        TARGET = VAR3[0]
    }

        IF(VAR3[0] == 0)
        {
            VAR3[1] = VAR3[1] - 1;
          GENERATE_USER_MSG WITH_DATA
          {
            TARGET = VAR3[1]
          }

            IF(VAR3[1] == 0)
            {
            GENERATE_USER_MSG WITH_DATA
            {
                TARGET = "Communication Aborted!"
            }
            VAR2[8] = 0;
            }
            ELSE
            {
            GENERATE_USER_SYSMSG WITH_DATA
            {
                TARGET = "Retransmission!"
            }
            VAR3[0] = 6;                                        // Retry after 6 seconds
            SEND_OUT_IP WITH_DATA
            {
              T.IP_PROT                 = CNST_IP_PROT_KDP    ,
                T.IP_ADDRDST              = VAR2[4, 7]          ,
              T.IP_DATA.KDP_ID          = 0W                  ,
                T.IP_DATA.KDP_ACK         = 0                   ,
                T.IP_DATA.KDP_WINDOW_SIZE = 0                   ,
                T.IP_DATA.KDP_DATA        = VAR3[4, 4 + VAR3[2, 3] - 1]
            }
            }
        }
}

IP_IN_HANDLER
{
  IF(S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT !=CNST_IP_PROT_KDP || S.IP_DATA.KDP_ACK !=1)
      RETURN;

    GENERATE_USER_MSG WITH_DATA
    {
        TARGET = "ACKED"
    }

    DISCARD_MESSAGE;

    VAR2[8] = 0;
    LOOK_FOR_ONE_ELEMENT_IN_POOL 19
    {
        ASSIGN_VARIABLE 3 WITH_DATA
        {
            T.[0]       = 6                 ,                   // Retry after 6 seconds
```

```
            T.[1]          = 4                          ,          // Retry at maximum 3 times
            T.[2, 3]       = LENGTH(PE.IP_DATA),
            T.[4,]         = PE.IP_DATA
        }
        SEND_OUT_IP WITH_DATA
        {
          T.IP_PROT                       = CNST_IP_PROT_KDP    ,
            T.IP_ADDRDST                    = VAR2[4, 7]          ,
          T.IP_DATA.KDP_ID                = 0W                  ,
            T.IP_DATA.KDP_ACK               = 0                   ,
            T.IP_DATA.KDP_WINDOW_SIZE = 0                         ,
            T.IP_DATA.KDP_DATA              = PE.IP_DATA
        }
        VAR8[2] = 1;                                              // Output message pending
        REMOVE_CURRENT_POOL_ELEMENT;
    }
}
```

3. **NoIDIdleRQReceiver.mddl**

```
VAR2[0, 3]        = {192, 168, 2, 2};       // SRC Address.
VAR2[4, 7]        = {192, 168, 1, 1};       // DST Address.

IP_RECEIVED_HANDLER
{
    IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP ||
        S.IP_DATA.KDP_ACK != 0W )
        RETURN;

    DISCARD_MESSAGE;

  SEND_OUT_IP WITH_DATA
  {
        T.IP_PROT                       = CNST_IP_PROT_KDP,
        T.IP_ADDRDST                      = VAR2[4, 7]        ,
        T.IP_DATA.KDP_ID                = 0W                  ,
        T.IP_DATA.KDP_ACK                 = 1                   ,
        T.IP_DATA.KDP_WINDOW_SIZE     = 0
    }

  GENERATE_USER_MSG WITH_DATA
    {
        TARGET = S[24,]
    }
}
```

4. **SIDIdleRQSender.mddl**

```
VAR1.SND_UNA      = 0W;                       // SND_UNA initialization.

VAR2[0, 3]        = {192, 168, 1, 1};       // SRC Address.
VAR2[4, 7]        = {192, 168, 2, 2};       // DST Address.
VAR2[8]            = 0;                       // No output message pending.

IP_OUT_HANDLER
{
    IF( S.IP_ADDRDST != VAR2[4, 7] || S.IP_PROT == CNST_IP_PROT_KDP )
```

```
        RETURN;

    DISCARD_MESSAGE;

    IF(VAR2[8]==1)
    {
        ADD_TO_POOL 19 WITH_LIFETIME 1800000 WITH_DATA
      {
            T = S
      }
    }
    ELSE
    {
        ASSIGN_VARIABLE 3 WITH_DATA
        {
            T.[0]          = 6                          ,          // Retry after 6 seconds
            T.[1]          = 4                          ,          // Retry at maximum 3 times
            T.[2, 3]       = LENGTH(S.IP_DATA) ,
            T.[4,]         = S.IP_DATA
        }
        SEND_OUT_IP WITH_DATA
        {
          T.IP_PROT                       = CNST_IP_PROT_KDP    ,
            T.IP_ADDRDST                    = VAR2[4, 7]          ,
          T.IP_DATA.KDP_ID                = VAR1.SND_UNA        ,
            T.IP_DATA.KDP_ACK               = 0                   ,
            T.IP_DATA.KDP_WINDOW_SIZE = 0                         ,
            T.IP_DATA.KDP_DATA              = S.IP_DATA
        }
        VAR2[8] = 1;                        // Output message pending
    }
   GENERATE_USER_MSG WITH_DATA
    {
        T.[4] = ((VAR1.SND_UNA)/10)+0X30,
        T.[5] = ((VAR1.SND_UNA)%10)+0X30,
        TARGET = "UNA "

    }

}

TIMER_WITH_PERIOD 1000
{
  IF(VAR2[8] != 1)                      // Output message pending
      RETURN;

      VAR3[0] = VAR3[0] - 1;
    GENERATE_USER_MSG WITH_DATA
    {
        TARGET = VAR3[0]

    }
      IF(VAR3[0] == 0)
      {
          VAR3[1] = VAR3[1] - 1;
        GENERATE_USER_MSG WITH_DATA
        {
            TARGET = VAR3[1]
        }
```

```
            IF(VAR3[1] == 0)
            {
            GENERATE_USER_MSG WITH_DATA
            {
                  TARGET = "Communication Aborted!"
            }
            VAR2[8] = 0;
            }
            ELSE
            {
            GENERATE_USER_SYSMSG WITH_DATA
            {
                  TARGET = "Retransmission!"
            }
            VAR3[0] = 6;                        // Retry after 6 seconds
            SEND_OUT_IP WITH_DATA
            {
               T.IP_PROT                        = CNST_IP_PROT_KDP    ,
                 T.IP_ADDRDST                     = VAR2[4, 7]          ,
               T.IP_DATA.KDP_ID                 = VAR1.SND_UNA        ,
                 T.IP_DATA.KDP_ACK                = 0                   ,
                 T.IP_DATA.KDP_WINDOW_SIZE = 0                          ,
                 T.IP_DATA.KDP_DATA               = VAR3[4, 4 + VAR3[2, 3] - 1]
            }
            }
        }

}

IP_IN_HANDLER
{
  IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP || S.IP_DATA.KDP_ACK !=
      1 )
       RETURN;

      GENERATE_USER_MSG WITH_DATA
      {
          T.[4] = ((S.IP_DATA.KDP_ID)/10)+0X30,
          T.[5] = ((S.IP_DATA.KDP_ID)%10)+0X30,
          TARGET = "ACK "
      }



  IF(VAR1.SND_UNA + 1W != S.IP_DATA.KDP_ID)
       RETURN;

      DISCARD_MESSAGE;

      VAR1.SND_UNA = S.IP_DATA.KDP_ID;

      VAR2[8] = 0;
      LOOK_FOR_ONE_ELEMENT_IN_POOL 19
      {
          ASSIGN_VARIABLE 3 WITH_DATA
          {
              T.[0]          = 6                  ,          // Retry after 6 seconds
              T.[1]          = 4                  ,          // Retry at maximum 3 times
              T.[2, 3]       = LENGTH(PE.IP_DATA),
```

```
            T.[4,]          = PE.IP_DATA
        }
        SEND_OUT_IP WITH_DATA
        {
          T.IP_PROT                    = CNST_IP_PROT_KDP   ,
            T.IP_ADDRDST               = VAR2[4, 7]         ,
          T.IP_DATA.KDP_ID             = VAR1.SND_UNA       ,
            T.IP_DATA.KDP_ACK          = 0                  ,
            T.IP_DATA.KDP_WINDOW_SIZE = 0                   ,
            T.IP_DATA.KDP_DATA         = PE.IP_DATA
        }
        VAR8[2] = 1;                        // Output message pending
        REMOVE_CURRENT_POOL_ELEMENT;
    }
}
```

5. **SIDIdleRQReceiver.mddl**

```
VAR1.RCV_NXT      = 0W;                     // RCV_NXT initialization.

VAR2[0, 3]        = {192, 168, 2, 2};       // SRC Address.
VAR2[4, 7]        = {192, 168, 1, 1};       // DST Address.

IP_IN_HANDLER
{
    IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP ||
S.IP_DATA.KDP_ACK != 0W )
        RETURN;

    DISCARD_MESSAGE;

    IF(S.IP_DATA.KDP_ID!=VAR1.RCV_NXT)
        RETURN;

    VAR1.RCV_NXT = VAR1.RCV_NXT + 1W;

  SEND_OUT_IP WITH_DATA
  {
        T.IP_PROT                    = CNST_IP_PROT_KDP,
        T.IP_ADDRDST                 = VAR2[4, 7]       ,
        T.IP_DATA.KDP_ID             = VAR1.RCV_NXT     ,
        T.IP_DATA.KDP_ACK            = 1                ,
        T.IP_DATA.KDP_WINDOW_SIZE    = 0
    }

  GENERATE_USER_MSG WITH_DATA
    {
        TARGET = S[24,]
    }
}
```

6. **SIDCRQSender.mddl**

```
VAR1.SND_UNA      = 0W;                     // SND_UNA initialization.
VAR1.SND_NXT      = VAR1.SND_UNA;           // SND_NXT initialization.
```

```
VAR2[0, 3]        = {192, 168, 1, 1};       // SRC Address.
VAR2[4, 7]        = {192, 168, 2, 2};       // DST Address.

IP_OUT_HANDLER
{
    IF( S.IP_ADDRDST != VAR2[4, 7] || S.IP_PROT == CNST_IP_PROT_KDP )
        RETURN;

    DISCARD_MESSAGE;

    IF(VAR1.SND_NXT - VAR1.SND_UNA >= 32768W )
        RETURN;

    ADD_TO_POOL 20 WITH_DATA
    {
        T.[0]                          = 6                    ,
        T.[1]                          = 5                    ,
        T.[2,].KDP_ID                  = VAR1.SND_NXT         ,
        T.[2,].KDP_ACK                 = 0                    ,
        T.[2,].KDP_WINDOW_SIZE         = 0                    ,
        T.[2,].KDP_DATA                = S.IP_DATA
 }

    SEND_OUT_IP WITH_DATA
    {
      T.IP_PROT                        = CNST_IP_PROT_KDP     ,
      T.IP_ADDRDST                     = VAR2[4, 7]           ,
      T.IP_DATA.KDP_ID                 = VAR1.SND_NXT         ,
        T.IP_DATA.KDP_ACK              = 0                    ,
        T.IP_DATA.KDP_WINDOW_SIZE = 0                         ,
        T.IP_DATA.KDP_DATA             = S.IP_DATA
    }

    VAR1.SND_NXT = VAR1.SND_NXT + 1W;

    GENERATE_USER_MSG WITH_DATA
    {
        T.[4] = ((VAR1.SND_UNA)/10)+0X30,
        T.[5] =((VAR1.SND_UNA)%10)+0X30,
        TARGET = "UNA "
    }
}

TIMER_WITH_PERIOD 1000
{
  FOR_EVERY_ELEMENT_IN_POOL 20
  {
      PE[0] = PE[0] - 1;
        IF(PE[0] == 0)
        {
            PE[1] = PE[1] - 1;
            IF(PE[1] == 0)
            {
                GENERATE_USER_SYSMSG WITH_DATA
                {
                    TARGET = "Communication Aborted!"
                }
                REMOVE_CURRENT_POOL_ELEMENT;
            }
            ELSE
```

```
                {
                    PE[0] = 6;
                    SEND_OUT_IP WITH_DATA
                    {
                        T.IP_PROT                   = CNST_IP_PROT_KDP      ,
                        T.IP_ADDRDST                = VAR2[4, 7]            ,
                        T.IP_DATA                   = PE.[2,]
                    }
                }
            }
        }
    }

    IP_IN_HANDLER
    {
     IF(S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT !=CNST_IP_PROT_KDP || S.IP_DATA.KDP_ACK !=1)
        RETURN;

        GENERATE_USER_MSG WITH_DATA
        {
            T.[4] =((S.IP_DATA.KDP_ID)/10)+0X30,
            T.[5] = ((S.IP_DATA.KDP_ID)%10)+0X30,
            TARGET = "ACK "
        }
        GENERATE_USER_MSG WITH_DATA
        {
            T.[4] =((VAR1.SND_NXT)/10)+0X30,
            T.[5] = ((VAR1.SND_NXT)%10)+0X30,
            TARGET = "NXT "
        }


      IF(VAR1.SND_UNA - S.IP_DATA.KDP_ID <   32768W)
          RETURN;

      IF(VAR1.SND_NXT - S.IP_DATA.KDP_ID >= 32768W)
          RETURN;

      DISCARD_MESSAGE;

      FOR_EVERY_ELEMENT_IN_POOL 20
        {
            IF(PE[2,].IP_DATA.KDP_ID - S.IP_DATA.KDP_ID >= 32768W)
                REMOVE_CURRENT_POOL_ELEMENT;
      }

      VAR1.SND_UNA = S.IP_DATA.KDP_ID;
    }
```

7. SIDCRQReceiver.mddl

```
VAR1.RCV_NXT      = 0W;                     // RCV_NXT initialization.

VAR2[0, 3]        = {192, 168, 2, 2};       // SRC Address.
VAR2[4, 7]        = {192, 168, 1, 1};       // DST Address.

VAR3[4, 5] = 0W;                            // Some pointer.

IP_IN_HANDLER
```

```
{
    IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP ||
    S.IP_DATA.KDP_ACK != 0W )
        RETURN;

    DISCARD_MESSAGE;

    IF(S.IP_DATA.KDP_ID - VAR1.RCV_NXT >= 32768W)
        RETURN;

    LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
    S.IP_DATA.KDP_ID)
        RETURN;

    ADD_TO_POOL 21 WITH_CONDITION (S.IP_DATA.KDP_ID - PE.IP_DATA.KDP_ID < 32768W)
    WITH_DATA
    {
        T = S
    }

    FOR(VAR3[4, 5] = VAR1.RCV_NXT;;VAR3[4, 5] = VAR3[4, 5] + 1W)
    {
        LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
        VAR3[4, 5])
            CONTINUE;
        ELSE
            BREAK;
    }

    IF(VAR3[4, 5] == VAR1.RCV_NXT)
        RETURN;

    VAR1.RCV_NXT = VAR3[4, 5];

    FOR_EVERY_ELEMENT_IN_POOL 21 WITH_CONDITION(PE.IP_DATA.KDP_ID –
    VAR1.RCV_NXT >= 32768W)
        REMOVE_CURRENT_POOL_ELEMENT;

  SEND_OUT_IP WITH_DATA
  {
        T.IP_PROT                       = CNST_IP_PROT_KDP,
        T.IP_ADDRDST                    = VAR2[4, 7]       ,
        T.IP_DATA.KDP_ID                = VAR1.RCV_NXT     ,
        T.IP_DATA.KDP_ACK               = 1                ,
        T.IP_DATA.KDP_WINDOW_SIZE       = 0
    }
}
```

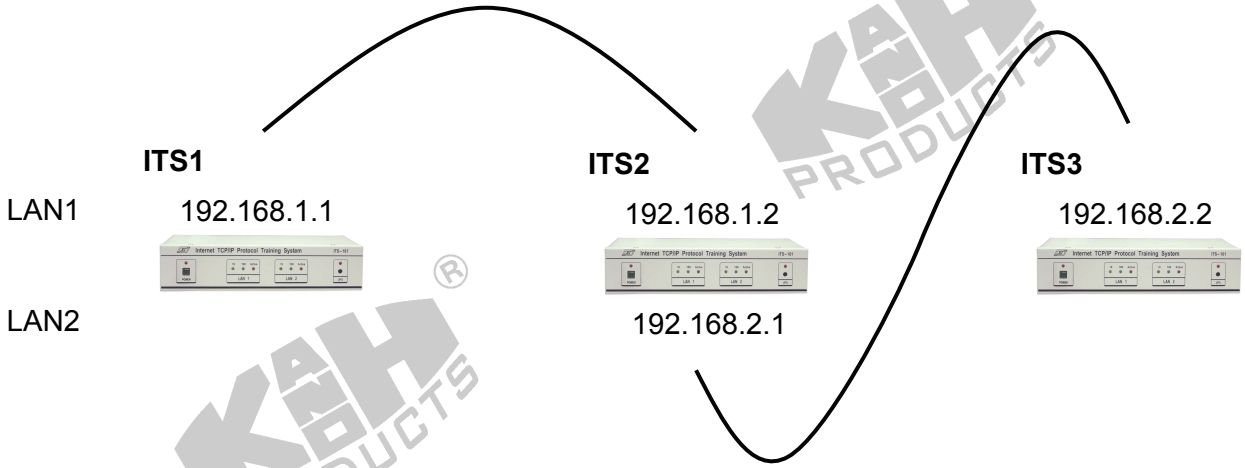
Exp 12. Sliding Window

OBJECTIVE：To understand the sliding window control in TCP.

BRIEF DESCRIPTION：This experiment examines the sliding window control mechanism that is used to improve the performance in TCP traffic. By using MDDL, students can learn how to implement the mechanism.

DURATION：3 hrs

TOPOLOGY



TECHNICAL BACKGROUND

Flow Control

1. Idle RQ
   a. It is inefficiency.
2. Continuous RQ
   a. The sender continuously sends frames to the receiver.
   b. So you must have a mechanism such that the sender stops sending after a finite number of unacknowledged frames.

To solve the above problems use sliding window flow control.

Sliding Window Control

a. Set a maximum number of unacknowledged frames - The Send Window Size.

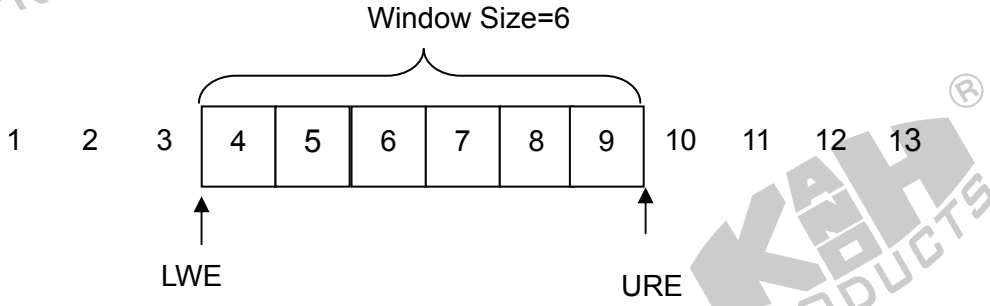
b. Keep Track of the UWE (Upper Window Edge) and LWE (Lower Window Edge)

c. UWE is incremented (by one) each time a frame is transmitted.

d. LWE is incremented (by one) each time a frame is acknowledged.

e. The sender sets (UWE - LWE) ≤ Send Window Size.

f. If (UWE - LWE) = Send Window Size then the sender must stop transmitting frames.



Sliding Window Control with Variable Window Size

The receiver sends acknowledgement, contains a *window size advertisement* that specifies how many additional octets of data the receiver is prepared to accept. We think of the window advertisement as specifying the receiver's current buffer size. In response to an increased window advertisement, the sender increases the size of its sliding window and proceeds to send octets that have not been acknowledged. In response to a decreased window advertisement, the sender decreases the size of its window and stops sending octets beyond the boundary.

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 12.1.

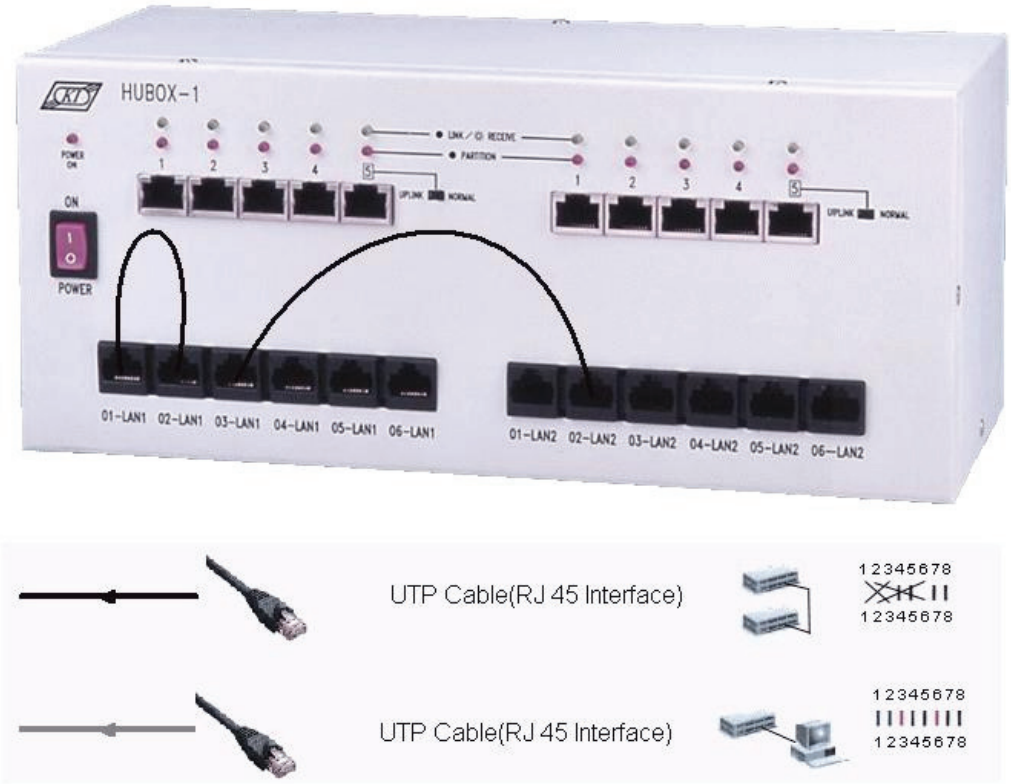


Figure 12.1

Setting Host and Gateway

2. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
3. Open the Network Configuration dialog box by selecting **Network Configuration** from the Tool menu.

ITS1 (Host)

4. Refer to Topology, type "**192.168.1.1**" into IP Address of Interface 1 (see Figure 12.2), and click the **Add new routing entry** button.
5. Type "**192.168.2.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.1.2**" into Gateway (see Figure 12.3), and click the **Update** button.
6. Choose **Host** and click the **Set & Close** button.

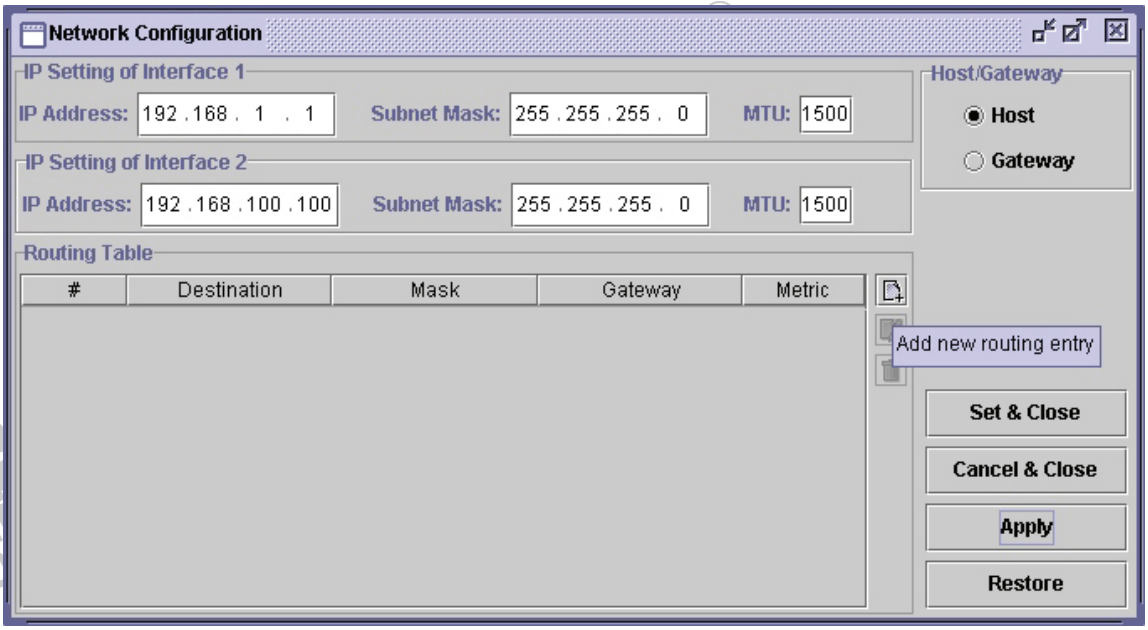


Figure 12.2

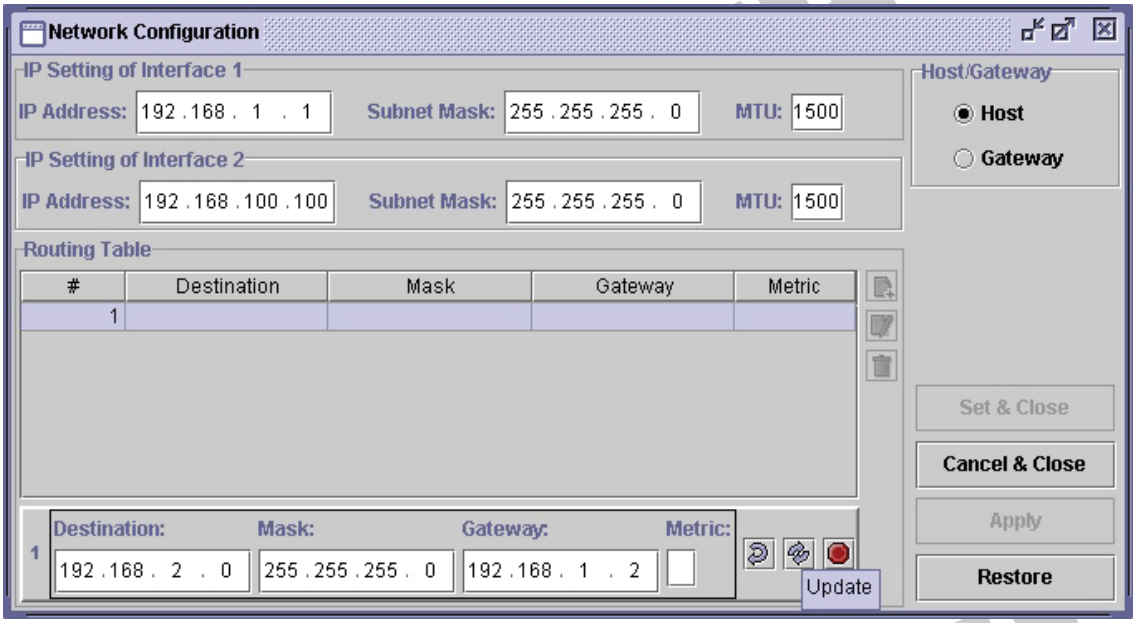


Figure 12.3

ITS3 (Host)

7. Refer to Topology. Type "**192.168.2.2**" into IP Address of Interface 1 and click the **Add new routing entry** button.
8. Type "**192.168.1.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.2.1**" into Gateway. Then click the **Update** button.
9. Choose **Host** and click the **Set & Close** button.

ITS2 (Gateway)

10. Refer to Topology A. Type "**192.168.1.2**" into IP Address of Interface 1 and "**192.168.2.1**" into IP Address of Interface 2. (See Figure 12.4.)
11. Choose **Gateway** and click the **Set & Close** button. Now, we already set up our routing table in ITS.

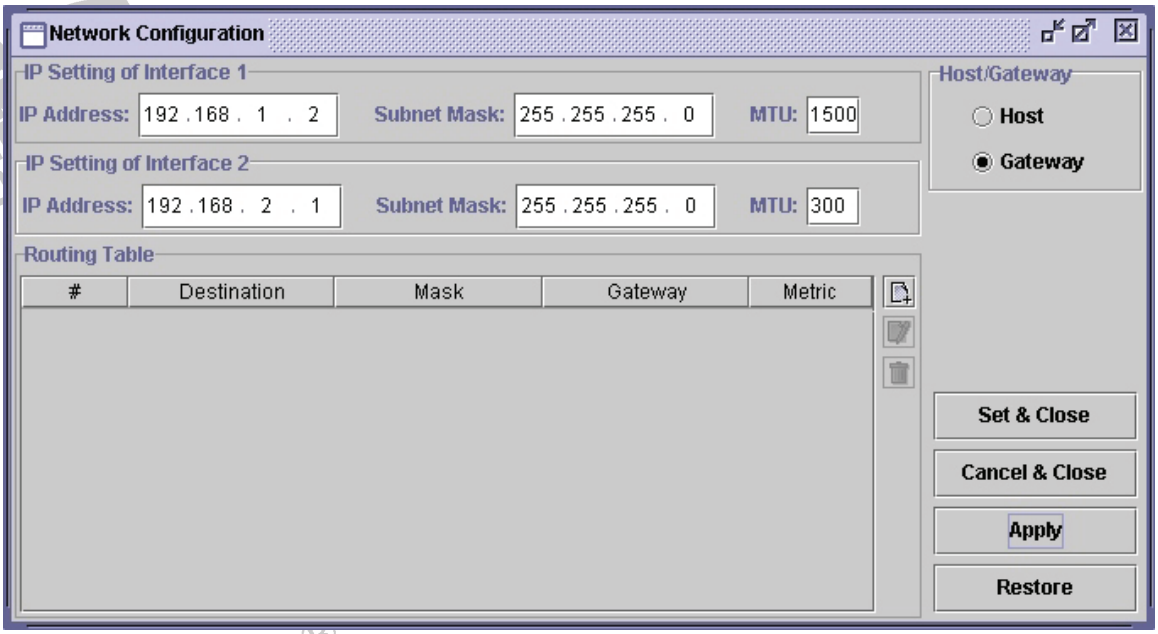


Figure 12.4

Usable Window Size of Sliding Window

ITS2

12. Open the Network Message Browser window. Check **Listening On**.
13. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
14. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex10 \PkLost4.mddl, and click the **UpId** button.

ITS3

15. Open the Network Message Browser window. Check **Listening On**.
16. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
17. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex12 \SlidingWindowReceiver.mddl, and click the **UpId** button.

ITS1

18. Open the Network Message Browser window. Check **Listening On**.
19. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
20. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex12 \SlidingWindowSender.mddl, and click the **UpId** button.
21. Select **Send IP Packet** from the Send menu to open the IP Datagram Sender. Type "**7**" into Protocol, type "**192.168.2.2**" into Destination IP Address and enter "**check**" into Data. (See Figure 12.5.)

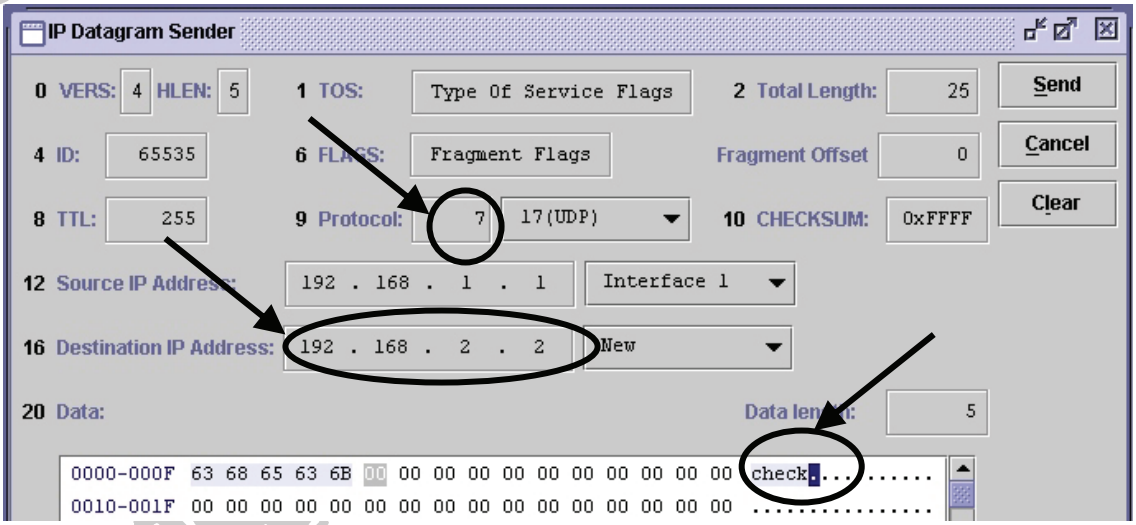
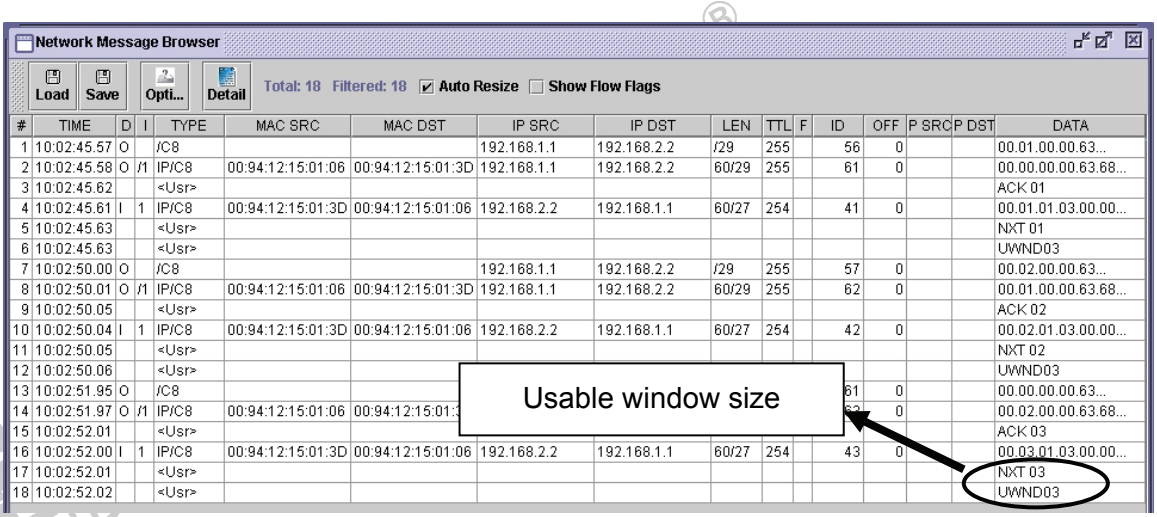


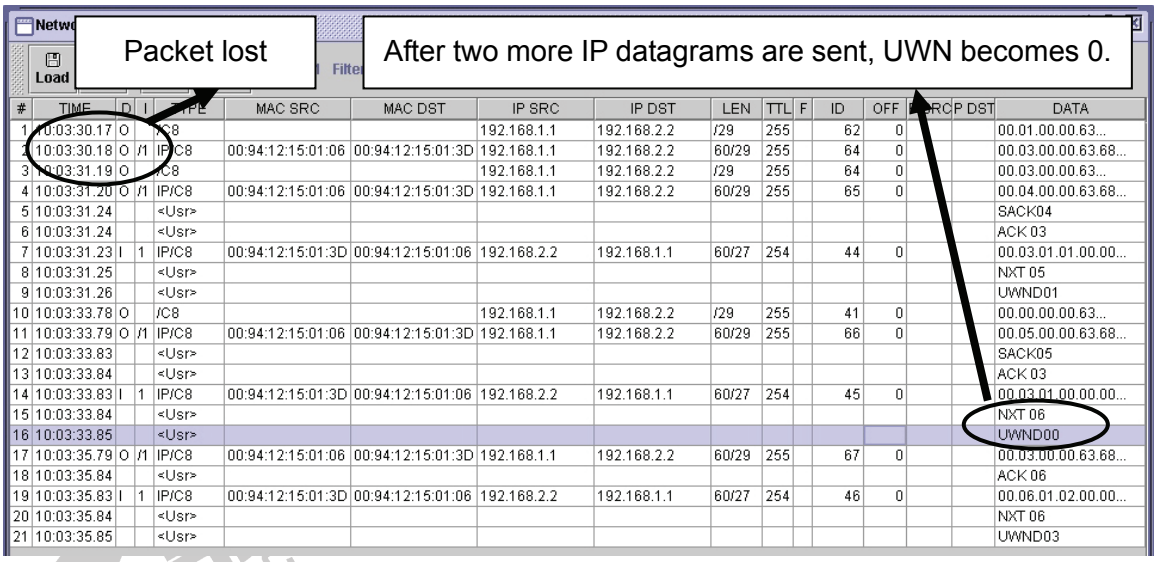
Figure 12.5

22. Finally click the **Send** button. ITS1 will send an IP datagram to ITS3 then receive ACK from ITS3. Observing the Network Message Browser shown in Figure 12.6, we will see the total widow size of 3. Try to send more IP datagrams. When IP datagrams enter into retransmission, send one more IP datagram. We will see that the widow size becomes 1. Send two more IP datagrams, we will see that the widow size becomes 0 indicating that the usable window is full, as shown in Figure 12.7. (Usable Window Size = SND.UNA + SND.WND – SND.NXT)



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:02:45.57	O		/C8			192.168.1.1	192.168.2.2	/29	255		56	0			00.01.00.00.63...
2	10:02:45.58	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		61	0			00.00.00.00.63.68...
3	10:02:45.62			<Usr>												ACK 01
4	10:02:45.61	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		41	0			00.01.01.03.00.00...
5	10:02:45.63			<Usr>												NXT 01
6	10:02:45.63			<Usr>												UWND03
7	10:02:50.00	O		/C8			192.168.1.1	192.168.2.2	/29	255		57	0			00.02.00.00.63...
8	10:02:50.01	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		62	0			00.01.00.00.63.68...
9	10:02:50.05			<Usr>												ACK 02
10	10:02:50.04	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		42	0			00.02.01.03.00.00...
11	10:02:50.05			<Usr>												NXT 02
12	10:02:50.06			<Usr>												UWND03
13	10:02:51.95	O		/C8								61	0			00.00.00.00.63...
14	10:02:51.97	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3						62	0			00.02.00.00.63.68...
15	10:02:52.01			<Usr>												ACK 03
16	10:02:52.00	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		43	0			00.03.01.03.00.00...
17	10:02:52.01			<Usr>												NXT 03
18	10:02:52.02			<Usr>												UWND03

Figure 12.6



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:03:30.17	O		/C8			192.168.1.1	192.168.2.2	/29	255		62	0			00.01.00.00.63...
2	10:03:30.18	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		64	0			00.03.00.00.63.68...
3	10:03:31.19	O		/C8			192.168.1.1	192.168.2.2	/29	255		64	0			00.03.00.00.63...
4	10:03:31.20	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		65	0			00.04.00.00.63.68...
5	10:03:31.24			<Usr>												SACK04
6	10:03:31.24			<Usr>												ACK 03
7	10:03:31.23	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		44	0			00.03.01.01.00.00...
8	10:03:31.25			<Usr>												NXT 05
9	10:03:31.26			<Usr>												UWND01
10	10:03:33.78	O		/C8			192.168.1.1	192.168.2.2	/29	255		41	0			00.00.00.00.63...
11	10:03:33.79	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		66	0			00.05.00.00.63.68...
12	10:03:33.83			<Usr>												SACK05
13	10:03:33.84			<Usr>												ACK 03
14	10:03:33.83	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		45	0			00.03.01.00.00.00...
15	10:03:33.84			<Usr>												NXT 06
16	10:03:33.85			<Usr>												UWND00
17	10:03:35.79	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		67	0			00.03.00.00.63.68...
18	10:03:35.84			<Usr>												ACK 06
19	10:03:35.83	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/27	254		46	0			00.06.01.02.00.00...
20	10:03:35.84			<Usr>												NXT 06
21	10:03:35.85			<Usr>												UWND03

Figure 12.7

DISCUSSIONS

1. What will happen if we remove the cable from ITS while transmitting?
2. Make two IP datagrams into retransmission, then send one more IP datagram. What's the widow size?

REACTOR PROGRAMS

1. **SlidingWindowSender.mddl**

```
VAR1.SND_UNA      = 0W;                    // SND_UNA initialization.
VAR1.SND_NXT      = VAR1.SND_UNA;          // SND_NXT initialization.
VAR1.SND_WND      = 3W;                    // SND_WND initialization.

VAR2[0, 3]        = {192, 168, 1, 1};      // SRC Address.
VAR2[4, 7]        = {192, 168, 2, 2};      // DST Address.

VAR4[0,1]         = 0W;            // SACK   (Selective Acknowledgment)

IP_OUT_HANDLER
{
    IF( S.IP_ADDRDST != VAR2[4, 7] || S.IP_PROT == CNST_IP_PROT_KDP )
        RETURN;

    DISCARD_MESSAGE;
//
    IF(VAR1.SND_NXT - (VAR1.SND_UNA + VAR1.SND_WND) <   32768W )
    IF(VAR1.SND_NXT - (VAR1.SND_UNA + 3) <   32768W )
        RETURN;

    ADD_TO_POOL 20 WITH_DATA
 {
        T.[0]                          = 6                    ,
        T.[1]                          = 5                    ,
        T.[2,].KDP_ID                  = VAR1.SND_NXT         ,
        T.[2,].KDP_ACK                 = 0                    ,
        T.[2,].KDP_WINDOW_SIZE         = 0                    ,
        T.[2,].KDP_DATA                = S.IP_DATA
 }

    SEND_OUT_IP WITH_DATA
    {
      T.IP_PROT                        = CNST_IP_PROT_KDP     ,
      T.IP_ADDRDST                     = VAR2[4, 7]           ,
      T.IP_DATA.KDP_ID                 = VAR1.SND_NXT         ,
        T.IP_DATA.KDP_ACK              = 0                    ,
        T.IP_DATA.KDP_WINDOW_SIZE = 0                         ,
        T.IP_DATA.KDP_DATA             = S.IP_DATA
    }

    VAR1.SND_NXT = VAR1.SND_NXT + 1W;
}

TIMER_WITH_PERIOD 1000
{
  FOR_EVERY_ELEMENT_IN_POOL 20
  {
      PE[0] = PE[0] - 1;
        IF(PE[0] == 0)
        {
            PE[1] = PE[1] - 1;
            IF(PE[1] == 0)
            {
                GENERATE_USER_SYSMSG WITH_DATA
                {
                    TARGET = "Communication Aborted!"
```

```
                }
                REMOVE_CURRENT_POOL_ELEMENT;
            }
            ELSE
            {
                PE[0] = 6;
                SEND_OUT_IP WITH_DATA
                {
                    T.IP_PROT                   = CNST_IP_PROT_KDP      ,
                    T.IP_ADDRDST                = VAR2[4, 7]            ,
                    T.IP_DATA                   = PE.[2,]
                }

            }
        }
    }
}

IP_IN_HANDLER
{
IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP || S.IP_DATA.KDP_ACK != 1)
 RETURN;

IF (S.IP_DATA.[5,6] != VAR1.SND_UNA)
{

GENERATE_USER_MSG WITH_DATA
    {
        T.[4] =((S.IP_DATA.[5,6])/10)+0X30 ,
        T.[5] =((S.IP_DATA.[5,6])%10)+0X30 ,
        TARGET = "SACK"

    }
     VAR4[0,1] = S.IP_DATA.[5,6] ;
}

    GENERATE_USER_MSG WITH_DATA
    {
        T.[4] = ((S.IP_DATA.KDP_ID)/10)+0X30,
        T.[5] = ((S.IP_DATA.KDP_ID)%10)+0X30,
        TARGET = "ACK "
    }

    GENERATE_USER_MSG WITH_DATA
    {
         T.[4] = ((VAR1.SND_NXT)/10)+0X30,
         T.[5] = ((VAR1.SND_NXT)%10)+0X30,
         TARGET = "NXT "
    }


 IF(S.IP_DATA.KDP_ID - VAR1.SND_UNA >=   32768W)
     RETURN;

 IF(VAR1.SND_NXT - S.IP_DATA.KDP_ID >= 32768W)
     RETURN;

 DISCARD_MESSAGE;
```

```
FOR_EVERY_ELEMENT_IN_POOL 20
    {
        IF(PE[2,].IP_DATA.KDP_ID - S.IP_DATA.KDP_ID >= 32768W)
            REMOVE_CURRENT_POOL_ELEMENT;
}

VAR1.SND_UNA = S.IP_DATA.KDP_ID;
VAR1.SND_WND = S.IP_DATA.KDP_WINDOW_SIZE;

    GENERATE_USER_MSG WITH_DATA
    {
        T.[4] = ((VAR1.SND_UNA + 3 - VAR1.SND_NXT)/10)+0X30,
        T.[5] = ((VAR1.SND_UNA + 3 - VAR1.SND_NXT)%10)+0X30,
        TARGET = "UWND"
    }
}
```

2. **SlidingWindowReceiver.mddl**

```
VAR1.RCV_NXT      = 0W;                    // RCV_NXT initialization.
VAR1.RCV_WND      = 4W;                    // RCV_WND initialization.

VAR2[0, 3]        = {192, 168, 2, 2};      // SRC Address.
VAR2[4, 7]        = {192, 168, 1, 1};      // DST Address.

VAR3[4, 5] = 0W;                           // Some pointer.

IP_IN_HANDLER
{
    IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP ||
    S.IP_DATA.KDP_ACK != 0W )
        RETURN;

    DISCARD_MESSAGE;

    IF(S.IP_DATA.KDP_ID - VAR1.RCV_NXT >= 32768W)
        RETURN;

    IF(S.IP_DATA.KDP_ID - (VAR1.RCV_NXT + VAR1.RCV_WND) < 32768W)
        RETURN;

    LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
     S.IP_DATA.KDP_ID)
        RETURN;

    VAR1.RCV_WND = VAR1.RCV_WND - 1W;

     GENERATE_USER_MSG WITH_DATA
    {
        TARGET = VAR1.RCV_WND
    }

    ADD_TO_POOL 21 WITH_CONDITION (S.IP_DATA.KDP_ID - PE.IP_DATA.KDP_ID < 32768W)
     WITH_DATA
 {
        T = S
```

```
}

    FOR(VAR3[4, 5] = VAR1.RCV_NXT;;VAR3[4, 5] = VAR3[4, 5] + 1W)
    {

        LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
          VAR3[4, 5])
        {
            VAR1.RCV_WND = VAR1.RCV_WND + 1W;
            CONTINUE;
        }
        ELSE
            BREAK;
    }

    VAR1.RCV_NXT = VAR3[4, 5];

    FOR_EVERY_ELEMENT_IN_POOL 21 WITH_CONDITION(PE.IP_DATA.KDP_ID –
      VAR1.RCV_NXT >= 32768W)
        REMOVE_CURRENT_POOL_ELEMENT;


    SEND_OUT_IP WITH_DATA
 {
        T.IP_PROT                       = CNST_IP_PROT_KDP,
        T.IP_ADDRDST                     = VAR2[4, 7]       ,
        T.IP_DATA.KDP_ID                 = VAR1.RCV_NXT     ,
        T.IP_DATA.KDP_ACK                = 1                ,
        T.IP_DATA.KDP_WINDOW_SIZE     = VAR1.RCV_WND
    }

}
```

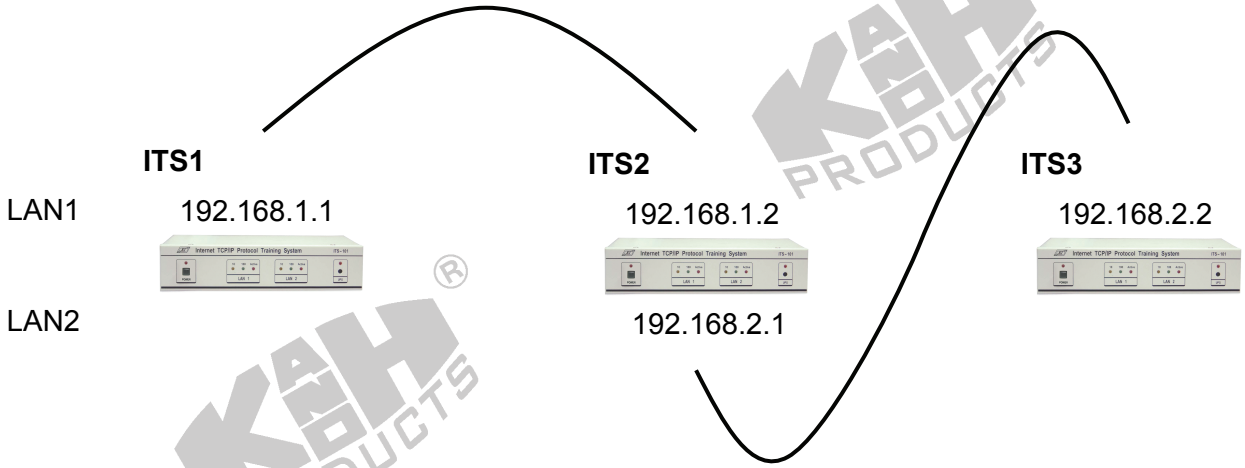
Exp 13. Congestion Avoidance

OBJECTIVE：To understand the congestion algorithm in TCP.

BRIEF DESCRIPTION： This experiment examines the congestion algorithm that is used to solve the problems of congestion. By using MDDL, students can learn how to implement the algorithm.

DURATION：3 hrs

TOPOLOGY



TECHNICAL BACKGROUND

Congestion is a condition of severe delay caused by an overload of datagrams at one or more switching points (e.g., at routers). When congestion occurs, delays increase and the router begins to enqueue datagrams until it can route them. To avoid congestion, the TCP standard now recommends using two techniques: *slow-start* and *multiplicative decrease*. They are related and can be implemented easily. We said that for each connection, TCP must remember the size of the receiver's window (i.e., the buffer size advertised in acknowledgements). To control congestion TCP maintains a second limit, called the *congestion window limit* or *congestion window* that it uses

to restrict data flow to less than the receiver's buffer size when congestion occurs. Allowed_window = min(receiver_advertisement, congestion_window).

Multiplicative Decrease Congestion Avoidance: Upon loss of a segment, reduce the congestion window by half (down to a minimum of at least one segment). For those segments that remain in the allowed window, back off the retransmission timer exponentially.

Slow-Start (Additive) Recovery: Whenever starting traffic on a new connection or increasing traffic after a period of congestion, start the congestion window at the size of a single segment and increase the congestion window by one segment each time an acknowledgement arrives.

To avoid increasing the window size too quickly and causing additional congestion, TCP adds one additional restriction. Once the congestion window reaches one half of its original size before congestion, TCP enters a *congestion avoidance* phase and slows down the rate of increment. During congestion avoidance, it increases the congestion window by *1* only if all segments in the window have been acknowledged.

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 13.1.

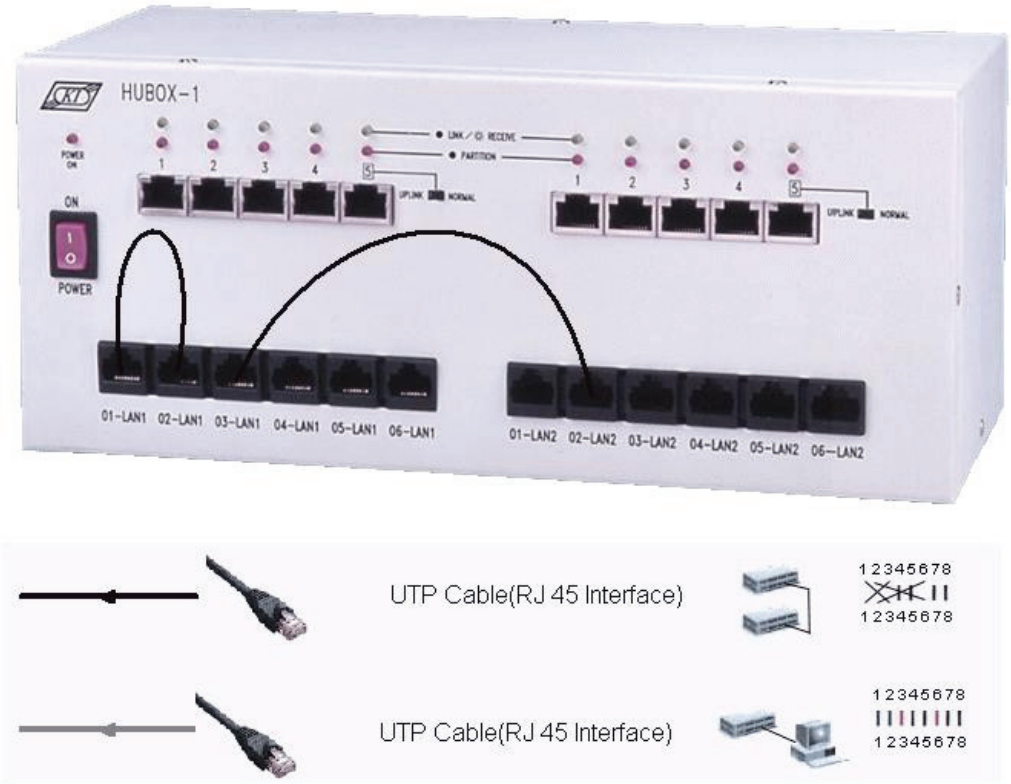


Figure 13.1

Setting Host and Gateway

2. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
3. Open the Network Configuration dialog box by selecting **Network Configuration** from the Tool menu.

ITS1 (Host)

4. Refer to Topology. Type "**192.168.1.1**" into IP Address of Interface 1 as shown in Figure 13.2. Click the **Add new routing entry** button.
5. Type "**192.168.2.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.1.2**" into Gateway (see Figure 13.3). Then click the **Update** button.
6. Choose **Host** and click the **Set & Close** button.

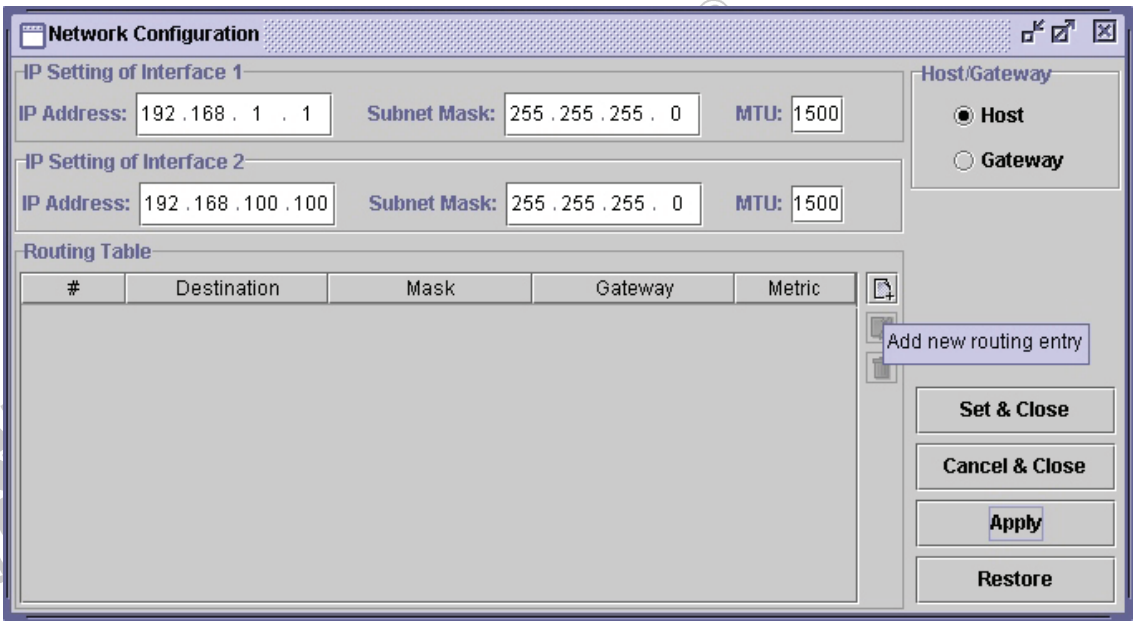


Figure 13.2

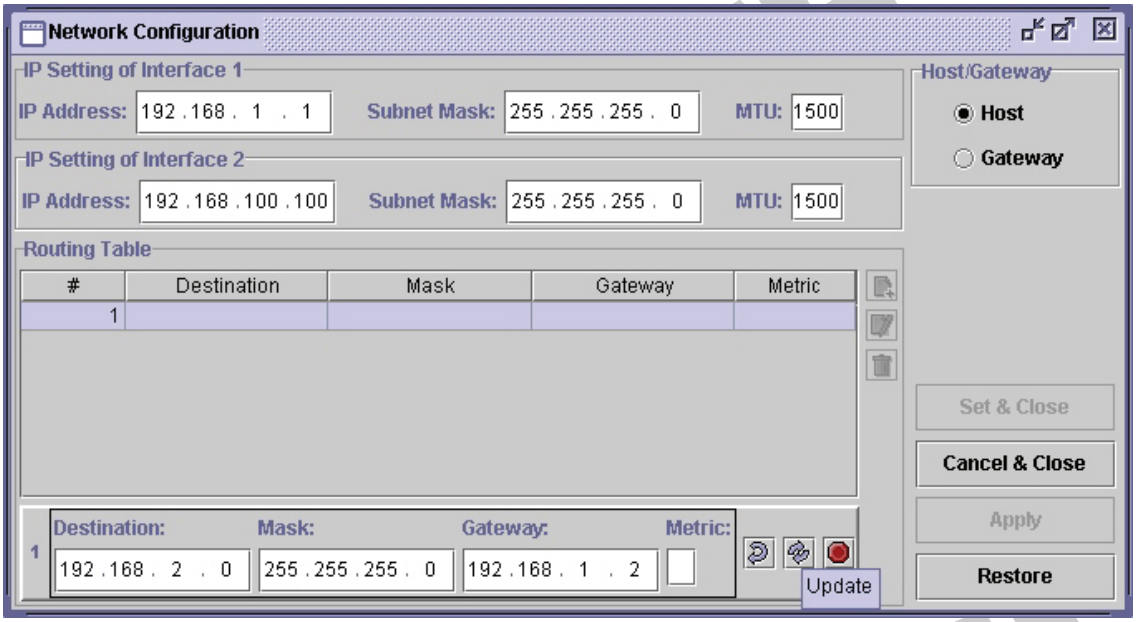


Figure 13.3

ITS3 (Host)

7. Refer to Topology. Type "**192.168.2.2**" into IP Address of Interface 1. Then click the **Add new routing entry** button.
8. Type "**192.168.1.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.2.1**" into Gateway. Then click the **Update** button.
9. Choose **Host** and click the **Set & Close** button.

ITS2 (Gateway)

10. Refer to Topology. Type "**192.168.1.2**" into IP Address of Interface 1 and "**192.168.2.1**" into IP Address of Interface 2 as shown in Figure 13.4.
11. Choose **Gateway** and click the **Set & Close** button. Now, we already set up our routing table in ITS.

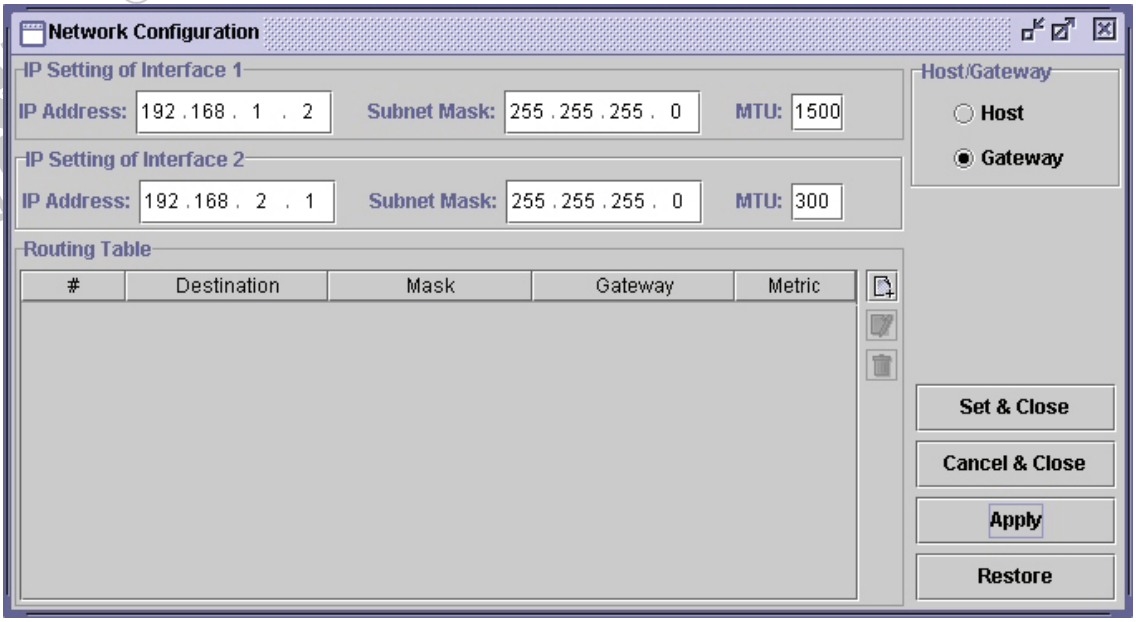


Figure 13.4

Slow-Start and Multiplicative Decrease

ITS2

12. Open the Network Message Browser window. Check **Listening On**.
13. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.

ITS3

14. Open the Network Message Browser window. Check **Listening On**.
15. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
16. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex13 \CongestionWindowReceiver.mddl, and click the **UpId** button.

ITS1

17. Open the Network Message Browser window. Check **Listening On**.

18. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
19. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex13 \CongestionWindowSender.mddl, and click the **UpId** button.
20. Select **Send IP Packet** from the Send menu to open the IP Datagram Sender. Type "**7**" into Protocol, type "**192.168.2.2**" into Destination IP Address and enter "**check**" into Data. (See Figure 13.5.)

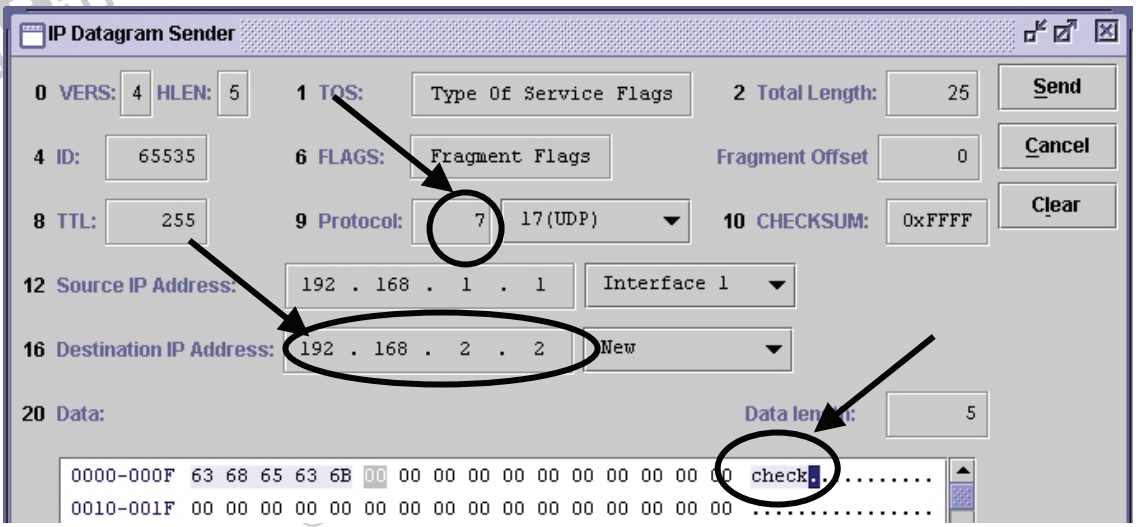
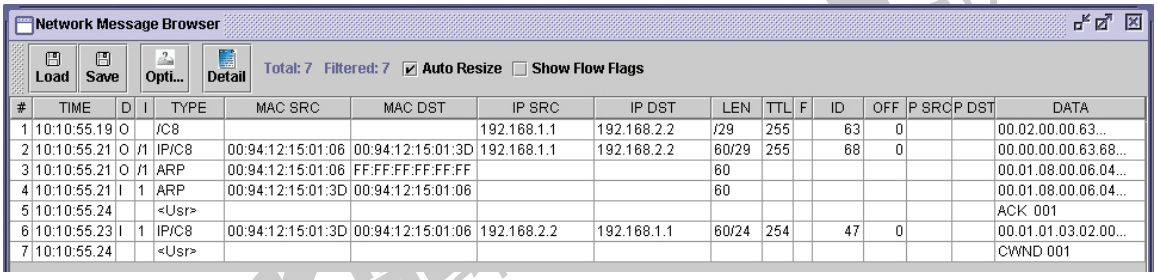


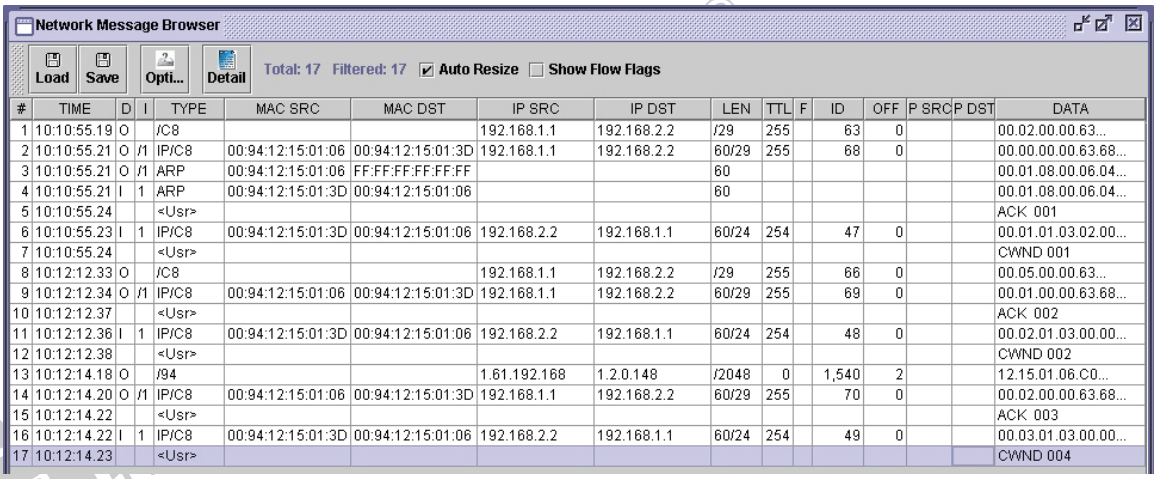
Figure 13.5

21. Finally click the **Send** button. ITS1 will send an IP datagram to ITS3 and receive an ACK from ITS3 as shown in Figure 13.6. We can see that the congestion window size (CWND) is '001'. If data transmission without incidents, the congestion window size will exponentially increase to '004' then linearly increase as shown in Figure 13.7.



#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:10:55.19	O		/C8			192.168.1.1	192.168.2.2	/29	255		63	0			00.02.00.00.63...
2	10:10:55.21	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		68	0			00.00.00.00.63.68...
3	10:10:55.21	O	/1	ARP	00:94:12:15:01:06	FF:FF:FF:FF:FF:FF			60							00.01.08.00.06.04...
4	10:10:55.21	I	1	ARP	00:94:12:15:01:3D	00:94:12:15:01:06			60							00.01.08.00.06.04...
5	10:10:55.24			<Usr>												ACK 001
6	10:10:55.23	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		47	0			00.01.01.03.02.00...
7	10:10:55.24			<Usr>												CWND 001

Figure 13.6

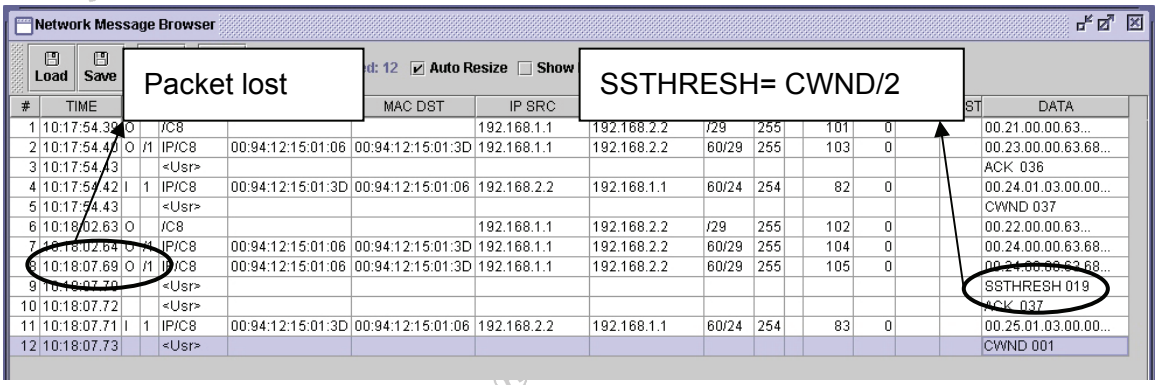


#	TIME	D	I	TYPE	MAC SRC	MAC DST	IP SRC	IP DST	LEN	TTL	F	ID	OFF	P SRC	P DST	DATA
1	10:10:55.19	O		/C8			192.168.1.1	192.168.2.2	/29	255		63	0			00.02.00.00.63...
2	10:10:55.21	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		68	0			00.00.00.00.63.68...
3	10:10:55.21	O	/1	ARP	00:94:12:15:01:06	FF:FF:FF:FF:FF:FF			60							00.01.08.00.06.04...
4	10:10:55.21	I	1	ARP	00:94:12:15:01:3D	00:94:12:15:01:06			60							00.01.08.00.06.04...
5	10:10:55.24			<Usr>												ACK 001
6	10:10:55.23	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		47	0			00.01.01.03.02.00...
7	10:10:55.24			<Usr>												CWND 001
8	10:12:12.33	O		/C8			192.168.1.1	192.168.2.2	/29	255		66	0			00.05.00.00.63...
9	10:12:12.34	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		69	0			00.01.00.00.63.68...
10	10:12:12.37			<Usr>												ACK 002
11	10:12:12.36	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		48	0			00.02.01.03.00.00...
12	10:12:12.38			<Usr>												CWND 002
13	10:12:14.18	O		/94			1.61.192.168	1.2.0.148	/2048	0		1,540	2			12.15.01.06.C0...
14	10:12:14.20	O	/1	IP/C8	00:94:12:15:01:06	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255		70	0			00.02.00.00.63.68...
15	10:12:14.22			<Usr>												ACK 003
16	10:12:14.22	I	1	IP/C8	00:94:12:15:01:3D	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254		49	0			00.03.01.03.00.00...
17	10:12:14.23			<Usr>												CWND 004

Figure 13.7

22. ITS2 make a congestion as packet-lost (Referring to the previous experiments) while transmission. Congestion window size would reduce to “001” as shown in Figure 13.8. If transmission is reconnected, we will see that the congestion window size increases exponentially until it reaches one half of its original size (SSTHRESH) as shown in Figure 13.9.

23. Repeat this experiment. Every time we should have a bit different value of SSTHRESH if we make congestions at different value of congestion windows size.



#	TIME				MAC DST	IP SRC							DATA
1	10:17:54.39	O		/C8		192.168.1.1	192.168.2.2	/29	255	101	0		00.21.00.00.63...
2	10:17:54.40	O	/1	IP/C8	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255	103	0		00.23.00.00.63.68...
3	10:17:54.43			<Usr>									ACK 036
4	10:17:54.42	I	1	IP/C8	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254	82	0		00.24.01.03.00.00...
5	10:17:54.43			<Usr>									CWND 037
6	10:18:02.63	O		/C8		192.168.1.1	192.168.2.2	/29	255	102	0		00.22.00.00.63...
7	10:18:02.64	O	/1	IP/C8	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255	104	0		00.24.00.00.63.68...
8	10:18:07.69	O	/1	IP/C8	00:94:12:15:01:3D	192.168.1.1	192.168.2.2	60/29	255	105	0		00.24.00.00.63.68...
9	10:18:07.70			<Usr>									SSTHRESH 019
10	10:18:07.72			<Usr>									ACK 037
11	10:18:07.71	I	1	IP/C8	00:94:12:15:01:06	192.168.2.2	192.168.1.1	60/24	254	83	0		00.25.01.03.00.00...
12	10:18:07.73			<Usr>									CWND 001

Figure 13.8

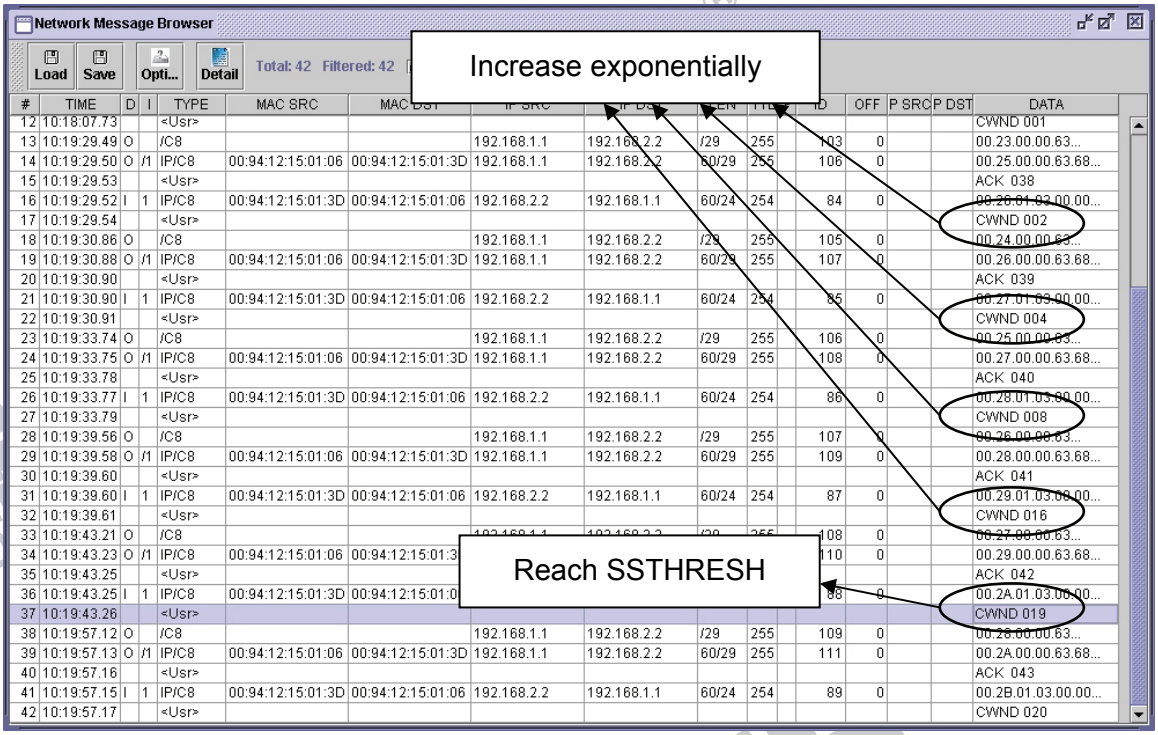

Network Message Browser
Load
Save
Opti...
Detail
Total: 42 Filtered: 42
Increase exponentially
Reach SSTHRESH
TIME D I TYPE MAC SRC MAC DST IP SRC IP DST LEN TTL ID OFF P SRC P DST DATA
12 10:18:07.73 <Usr> CWND 001
13 10:19:29.49 O /C8 192.168.1.1 192.168.2.2 /29 255 103 0 00.23.00.00.63...
14 10:19:29.50 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3D 192.168.1.1 192.168.2.2 60/29 255 106 0 00.25.00.00.63.68...
15 10:19:29.53 <Usr> ACK 038
16 10:19:29.52 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:06 192.168.2.2 192.168.1.1 60/24 254 84 0 00.26.01.03.00.00...
17 10:19:29.54 <Usr> CWND 002
18 10:19:30.86 O /C8 192.168.1.1 192.168.2.2 /29 255 105 0 00.24.00.00.63...
19 10:19:30.88 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3D 192.168.1.1 192.168.2.2 60/29 255 107 0 00.26.00.00.63.68...
20 10:19:30.90 <Usr> ACK 039
21 10:19:30.90 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:06 192.168.2.2 192.168.1.1 60/24 254 85 0 00.27.01.03.00.00...
22 10:19:30.91 <Usr> CWND 004
23 10:19:33.74 O /C8 192.168.1.1 192.168.2.2 /29 255 106 0 00.25.00.00.63...
24 10:19:33.75 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3D 192.168.1.1 192.168.2.2 60/29 255 108 0 00.27.00.00.63.68...
25 10:19:33.78 <Usr> ACK 040
26 10:19:33.77 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:06 192.168.2.2 192.168.1.1 60/24 254 86 0 00.28.01.03.00.00...
27 10:19:33.79 <Usr> CWND 008
28 10:19:39.56 O /C8 192.168.1.1 192.168.2.2 /29 255 107 0 00.26.00.00.63...
29 10:19:39.58 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3D 192.168.1.1 192.168.2.2 60/29 255 109 0 00.28.00.00.63.68...
30 10:19:39.60 <Usr> ACK 041
31 10:19:39.60 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:06 192.168.2.2 192.168.1.1 60/24 254 87 0 00.29.01.03.00.00...
32 10:19:39.61 <Usr> CWND 016
33 10:19:43.21 O /C8 108 0 00.27.00.00.63...
34 10:19:43.23 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3 110 0 00.29.00.00.63.68...
35 10:19:43.25 <Usr> ACK 042
36 10:19:43.25 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:0 88 0 00.2A.01.03.00.00...
37 10:19:43.26 <Usr> CWND 019
38 10:19:57.12 O /C8 192.168.1.1 192.168.2.2 /29 255 109 0 00.28.00.00.63...
39 10:19:57.13 O /1 IP/C8 00:94:12:15:01:06 00:94:12:15:01:3D 192.168.1.1 192.168.2.2 60/29 255 111 0 00.2A.00.00.63.68...
40 10:19:57.16 <Usr> ACK 043
41 10:19:57.15 I 1 IP/C8 00:94:12:15:01:3D 00:94:12:15:01:06 192.168.2.2 192.168.1.1 60/24 254 89 0 00.2B.01.03.00.00...
42 10:19:57.17 <Usr> CWND 020

Figure 13.9

DISCUSSION

1. Send 20 IP datagrams and make a packet lost. After that, send 10 more IP datagrams. Complete Figure 13.10 by referring to the CWND values and packet number.

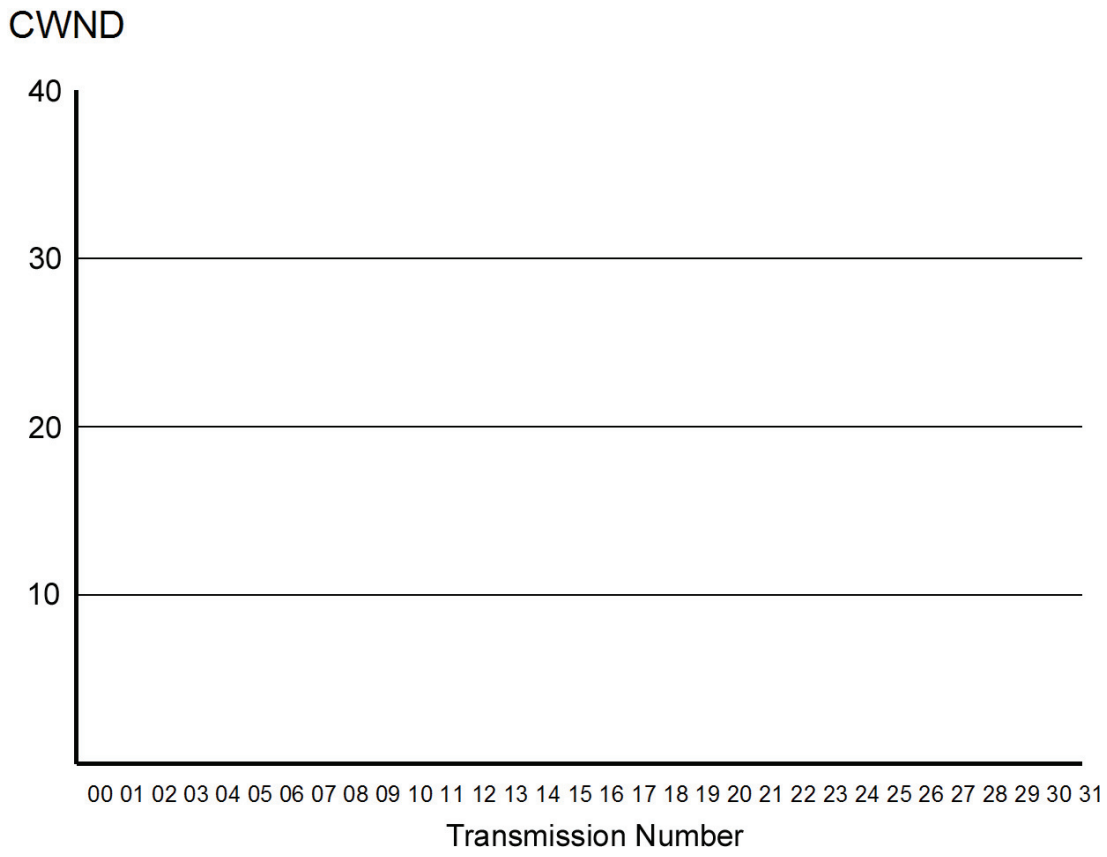


Figure 13.10

REACTOR PROGRAMS

1. CongestionWindowSender.mddl

```
VAR1.SND_UNA          = 0W;                  // SND_UNA       initialization.
VAR1.SND_NXT          = VAR1.SND_UNA;        // SND_NXT       initialization.
VAR1.SND_WND          = 16W;                 // SND_WND       initialization.
VAR1.SND_CWND         = 1W;                  // SND_CWND      initialization.
VAR1.SND_SSTHRESH     = 4W;                  // SND_SSTHRESH initialization.

VAR2[0, 3]      = {192, 168, 1, 1};      // SRC Address.
VAR2[4, 7]      = {192, 168, 2, 2};      // DST Address.

IP_OUT_HANDLER
{
    IF( S.IP_ADDRDST != VAR2[4, 7] || S.IP_PROT == CNST_IP_PROT_KDP )
        RETURN;

    DISCARD_MESSAGE;

    IF(VAR1.SND_NXT - (VAR1.SND_UNA + VAR1.SND_WND) < 32768W )
        RETURN;

    IF(VAR1.SND_NXT - (VAR1.SND_UNA + VAR1.SND_CWND) < 32768W )
        RETURN;

    ADD_TO_POOL 20 WITH_DATA
 {
        T.[0]                       = 6                    ,
        T.[1]                       = 5                    ,
        T.[2,].KDP_ID               = VAR1.SND_NXT         ,
        T.[2,].KDP_ACK              = 0                    ,
        T.[2,].KDP_WINDOW_SIZE      = 0                    ,
        T.[2,].KDP_DATA             = S.IP_DATA
 }

    SEND_OUT_IP WITH_DATA
    {
      T.IP_PROT                     = CNST_IP_PROT_KDP    ,
      T.IP_ADDRDST                  = VAR2[4, 7]          ,
      T.IP_DATA.KDP_ID              = VAR1.SND_NXT        ,
        T.IP_DATA.KDP_ACK           = 0                   ,
        T.IP_DATA.KDP_WINDOW_SIZE = 0                     ,
        T.IP_DATA.KDP_DATA          = S.IP_DATA
    }

    VAR1.SND_NXT = VAR1.SND_NXT + 1W;
}

TIMER_WITH_PERIOD 1000
{
    FOR_EVERY_ELEMENT_IN_POOL 20
    {
    PE[0] = PE[0] - 1;
        IF(PE[0] == 0)
        {
            PE[1] = PE[1] - 1;
            IF(PE[1] == 0)
            {
            GENERATE_USER_SYSMSG WITH_DATA
```

```
                {
                    TARGET = "Communication Aborted!"
                }
                REMOVE_CURRENT_POOL_ELEMENT;
            }
            ELSE IF (PE[1] == 4)
            {
                PE[0] = 6;
                SEND_OUT_IP WITH_DATA
                {
                    T.IP_PROT                   = CNST_IP_PROT_KDP      ,
                    T.IP_ADDRDST                = VAR2[4, 7]            ,
                    T.IP_DATA                   = PE.[2,]
                }
                VAR1.SND_SSTHRESH =   VAR1.SND_CWND/2 ;
                 VAR1.SND_CWND = 1W;

                 GENERATE_USER_MSG WITH_DATA
                {
                    T.[9] = ((VAR1.SND_SSTHRESH)/100)+0X30,
                    T.[10] = (((VAR1.SND_SSTHRESH)%100)/10)+0X30,
                    T.[11] = ((VAR1.SND_SSTHRESH)%10)+0X30,
                    TARGET = "SSTHRESH "
                }
            }
           ELSE
            {
                PE[0] = 6;
                SEND_OUT_IP WITH_DATA
                {
                    T.IP_PROT                   = CNST_IP_PROT_KDP      ,
                    T.IP_ADDRDST                = VAR2[4, 7]            ,
                    T.IP_DATA                   = PE.[2,]
                }
            }
        }
    }
}

IP_IN_HANDLER
{
  IF(S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP || S.IP_DATA.KDP_ACK !=1)
      RETURN;


    GENERATE_USER_MSG WITH_DATA
    {
        T.[5] = ((S.IP_DATA.KDP_ID)/100)+0X30,
        T.[6] = (((S.IP_DATA.KDP_ID)%100)/10)+0X30,
        T.[7] = ((S.IP_DATA.KDP_ID)%10)+0X30,
        TARGET = "ACK    "
    }

  IF(S.IP_DATA.KDP_ID - VAR1.SND_UNA >=   32768W)
      RETURN;

  IF(VAR1.SND_NXT - S.IP_DATA.KDP_ID >= 32768W)
      RETURN;

  DISCARD_MESSAGE;
```

```
FOR_EVERY_ELEMENT_IN_POOL 20
   {
        IF(PE[2,].IP_DATA.KDP_ID - S.IP_DATA.KDP_ID >= 32768W)
             REMOVE_CURRENT_POOL_ELEMENT;
}

VAR1.SND_UNA = S.IP_DATA.KDP_ID;
VAR1.SND_WND = S.IP_DATA.KDP_WINDOW_SIZE;

     GENERATE_USER_MSG WITH_DATA
   {
        T.[5] = ((VAR1.SND_CWND)/100)+0X30,
        T.[6] = (((VAR1.SND_CWND)%100)/10)+0X30,
        T.[7] = ((VAR1.SND_CWND)%10)+0X30,
        TARGET = "CWND "
   }

IF(VAR1.SND_CWND - VAR1.SND_SSTHRESH < 32768W)
     VAR1.SND_CWND = VAR1.SND_CWND + 1W;

                                IF(VAR1.SND_CWND - VAR1.SND_SSTHRESH >= 32768W)
{
                                    IF(VAR1.SND_SSTHRESH - (VAR1.SND_CWND*2) < 32768W)
                                     VAR1.SND_CWND = VAR1.SND_CWND +VAR1.SND_CWND;
                                    ELSE
                                     VAR1.SND_CWND =VAR1.SND_SSTHRESH ;
                                 }
}
```

2. **CongestionWindowReceiver.mddl**

```
VAR1.RCV_NXT      = 0W;                    // RCV_NXT initialization.
VAR1.RCV_WND      = 16W;                     // RCV_WND initialization.

VAR2[0, 3]        = {192, 168, 2, 2};      // SRC Address.
VAR2[4, 7]        = {192, 168, 1, 1};      // DST Address.

VAR3[4, 5] = 0W;                           // Some pointer.

IP_IN_HANDLER
{
    IF( S.IP_ADDRSRC != VAR2[4, 7] || S.IP_PROT != CNST_IP_PROT_KDP ||
    S.IP_DATA.KDP_ACK != 0W )
        RETURN;

    DISCARD_MESSAGE;

    IF(S.IP_DATA.KDP_ID - VAR1.RCV_NXT >= 32768W)
        RETURN;

    IF(S.IP_DATA.KDP_ID - (VAR1.RCV_NXT + VAR1.RCV_WND) < 32768W)
        RETURN;

    LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
    S.IP_DATA.KDP_ID)
        RETURN;
```

```
    VAR1.RCV_WND = VAR1.RCV_WND - 1W;

     GENERATE_USER_MSG WITH_DATA
    {
        TARGET = VAR1.RCV_WND
    }

    ADD_TO_POOL 21 WITH_CONDITION (S.IP_DATA.KDP_ID - PE.IP_DATA.KDP_ID < 32768W)
    WITH_DATA
 {
        T = S
 }

    FOR(VAR3[4, 5] = VAR1.RCV_NXT;;VAR3[4, 5] = VAR3[4, 5] + 1W)
    {
        LOOK_FOR_ONE_ELEMENT_IN_POOL 21 WITH_CONDITION (PE.IP_DATA.KDP_ID ==
        VAR3[4, 5])
        {
            VAR1.RCV_WND = VAR1.RCV_WND + 1W;
            CONTINUE;
        }
        ELSE
            BREAK;
    }

    VAR1.RCV_NXT = VAR3[4, 5];

    FOR_EVERY_ELEMENT_IN_POOL 21 WITH_CONDITION(PE.IP_DATA.KDP_ID –
    VAR1.RCV_NXT >= 32768W)
        REMOVE_CURRENT_POOL_ELEMENT;


  SEND_OUT_IP WITH_DATA
  {
        T.IP_PROT                    = CNST_IP_PROT_KDP,
        T.IP_ADDRDST                  = VAR2[4, 7]       ,
        T.IP_DATA.KDP_ID              = VAR1.RCV_NXT     ,
        T.IP_DATA.KDP_ACK             = 1                ,
        T.IP_DATA.KDP_WINDOW_SIZE     = VAR1.RCV_WND
    }
}
```

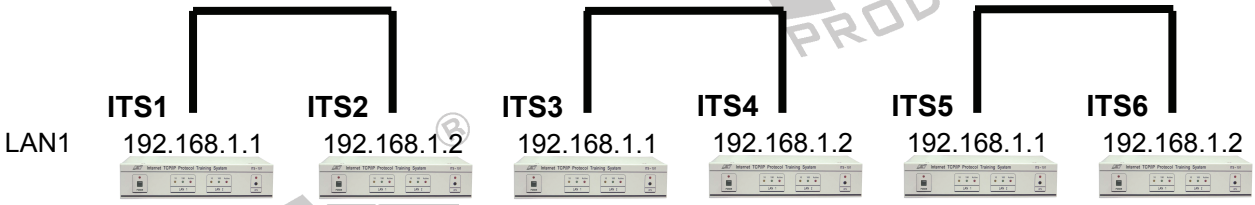
Exp 14. Full TCP Implementation

OBJECTIVE：To understand how to implement TCP protocol.

BRIEF DESCRIPTION：This experiment examines the implementation of the most important and well-known network-level service, reliable stream delivery, *Transmission Control Protocol* (*TCP*). By using MDDL language, students can learn how to implement TCP protocol.

DURATION：9 hrs

TOPOLOGY



TECHNICAL BACKGROUND

The unit of transfer between the TCP software on two machines is called a *segment.* Segments are exchanged to establish connections, transfer data, send acknowledgements, advertise window sizes, and close connections. Furthermore, TCP uses piggybacking such that ACK can along with data. Figure 14.1 shows the TCP segment format.

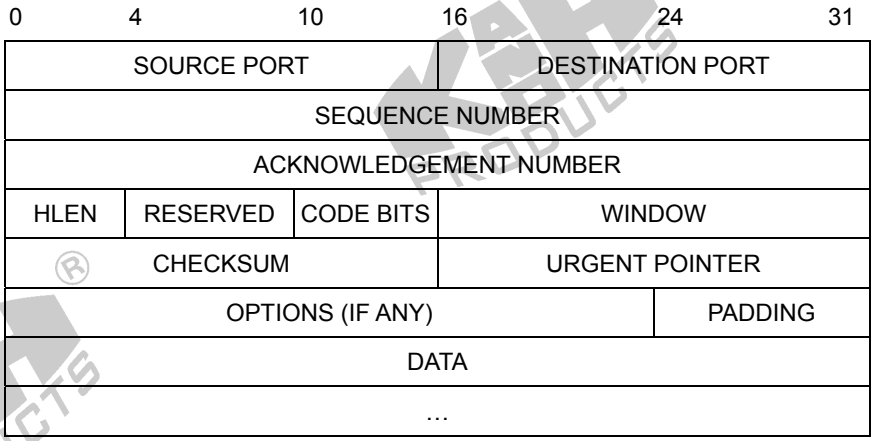


Figure 14.1

SOURCE PORT and *DESTINATION PORT:* the TCP port numbers that identify the application programs at the ends of the connection.

SEQUENCE NUMBER: the position in the sender's byte stream of the data.

ACKNOWLEDGEMENT NUMBER: the number of the octet that the source expects to receive next.

CODE BITS (6-bit): to determine the purpose and contents of the segment. The six bits tell how to interpret other fields in the header according to Figure 14.2.

Bit (left to right)	Meaning if bit set to 1.
URG	Urgent pointer field is valid.
ACK	Acknowledgement field is valid.
PSH	This segment requests a push.
RST	Reset the connection.
SYN	Synchronize sequence numbers.
FIN	Sender has reached end of its byte stream.

Figure 14.2

WINDOW (16-bit): to advertise how much data it is willing to accept every time.

On the other hand, TCP follows the finite state machine (FSM) of Figure 14.3.

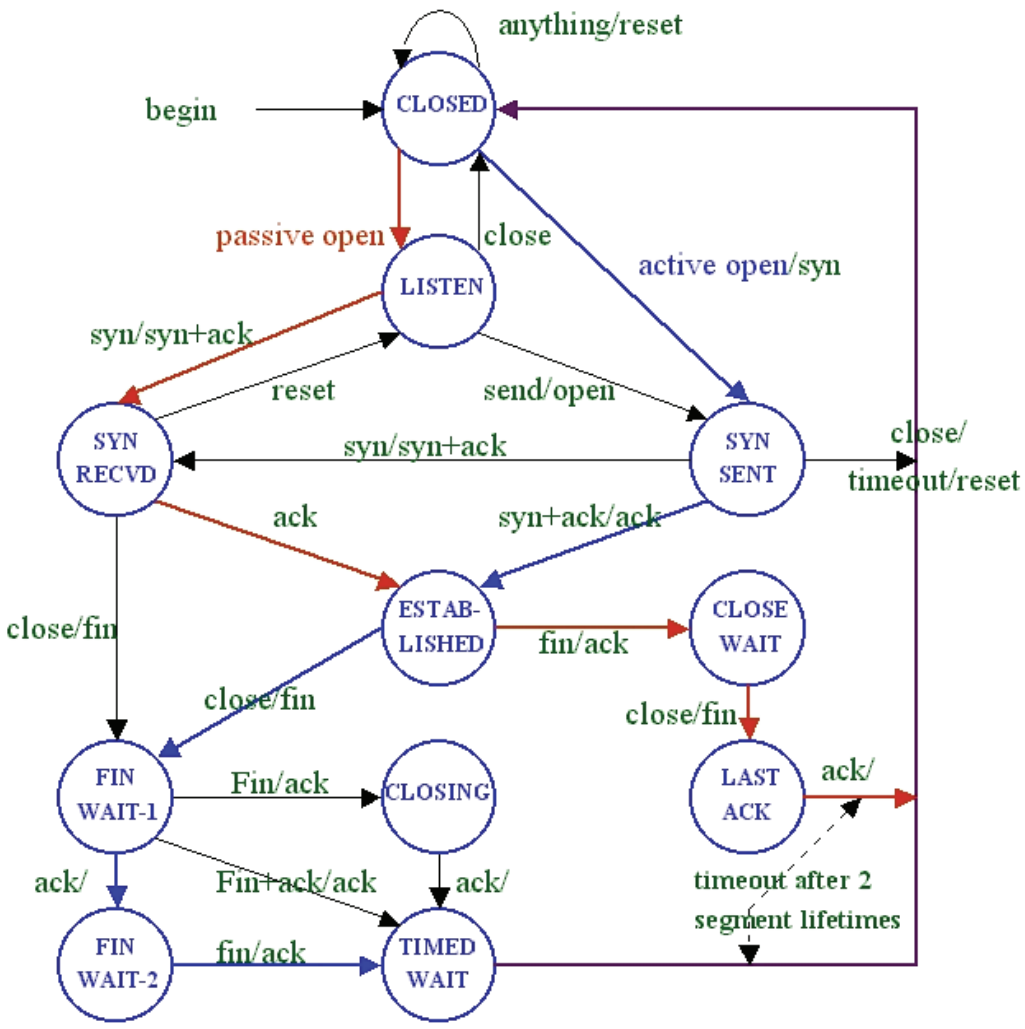


Figure 14.3

By following FSM, TCP can achieve the following tasks:

a. Establish a TCP connection

b. Lose a TCP connection

c. Reset TCP connection

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 14.4.

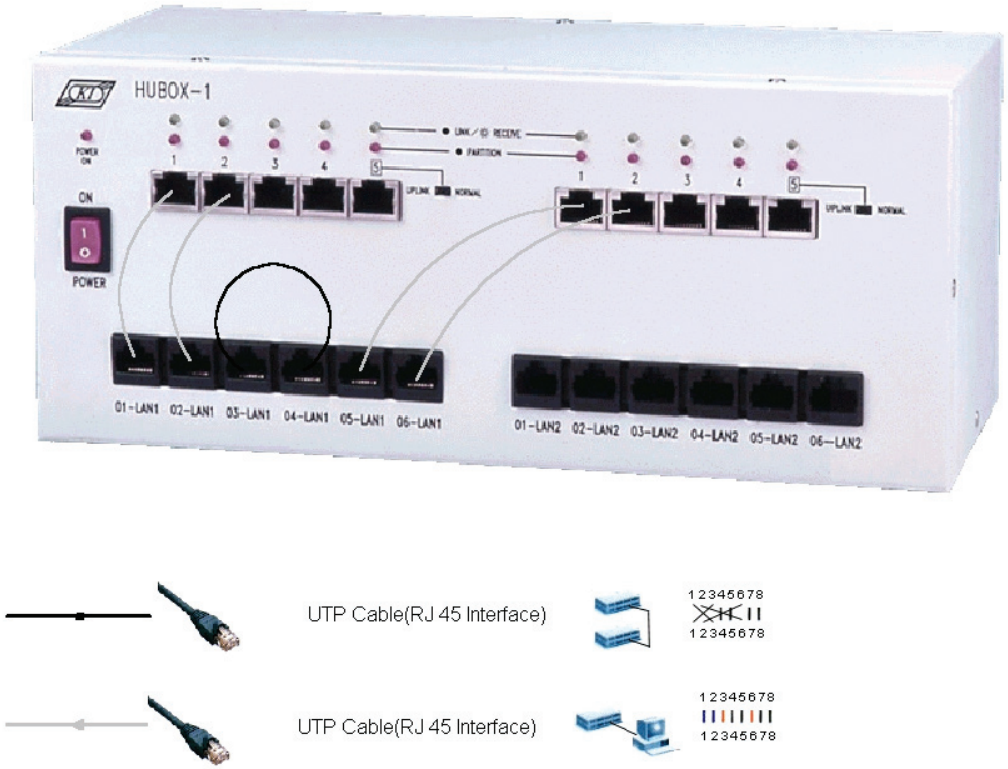


Figure 14.4

Setting Host

ITS1 and ITS2

2. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
3. Open the Network Configuration dialog box by selecting **Network Configuration** from the Tool menu.

ITS1

4. Refer to Topology. Type "**192.168.1.1**" into IP Address of Interface 1 as shown in Figure 14.5.
5. Choose **Host** and click the **Set & Close** button.

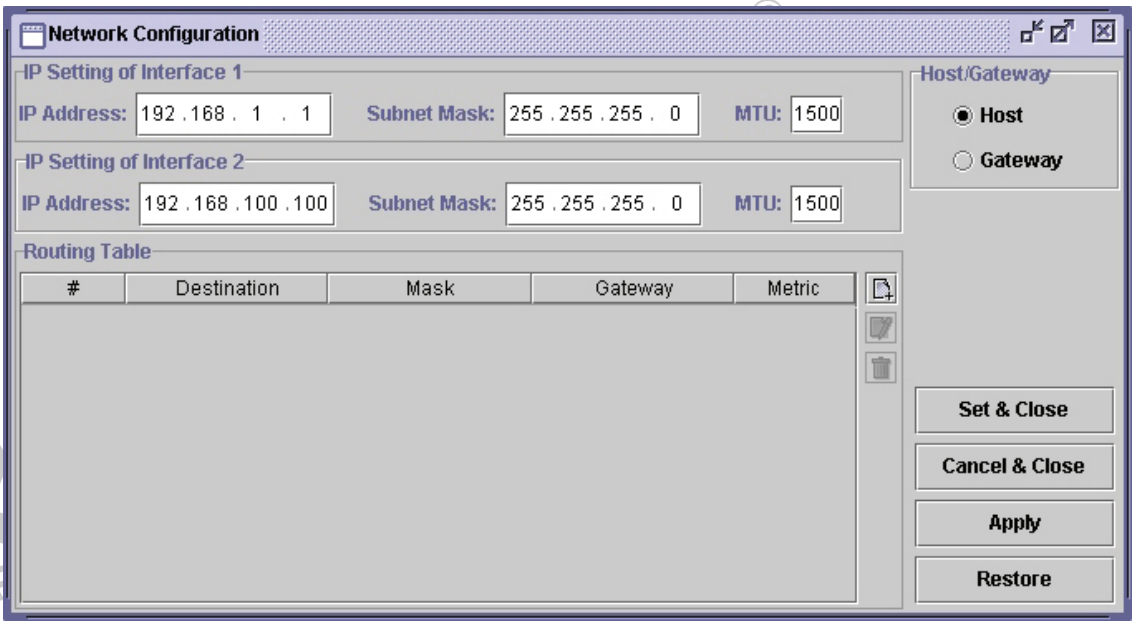


Figure 14.5

ITS2

6. Type "**192.168.1.2**" into IP Address of Interface 1.
7. Choose **Host** and click the **Set & Close** button.

TCP Session

ITS1

8. Open the Network Message Browser window. Check **Listening On**.
9. Select **New TCP Session** from the TCP menu to open the New TCP Session dialog box.
10. Select the **System Default TCP** option. Type "**192.168.1.1**" into Source IP Address, choose **HTTP (80)** from Source Port. (See Figure 14.6.)
11. Click the **Listen** button.

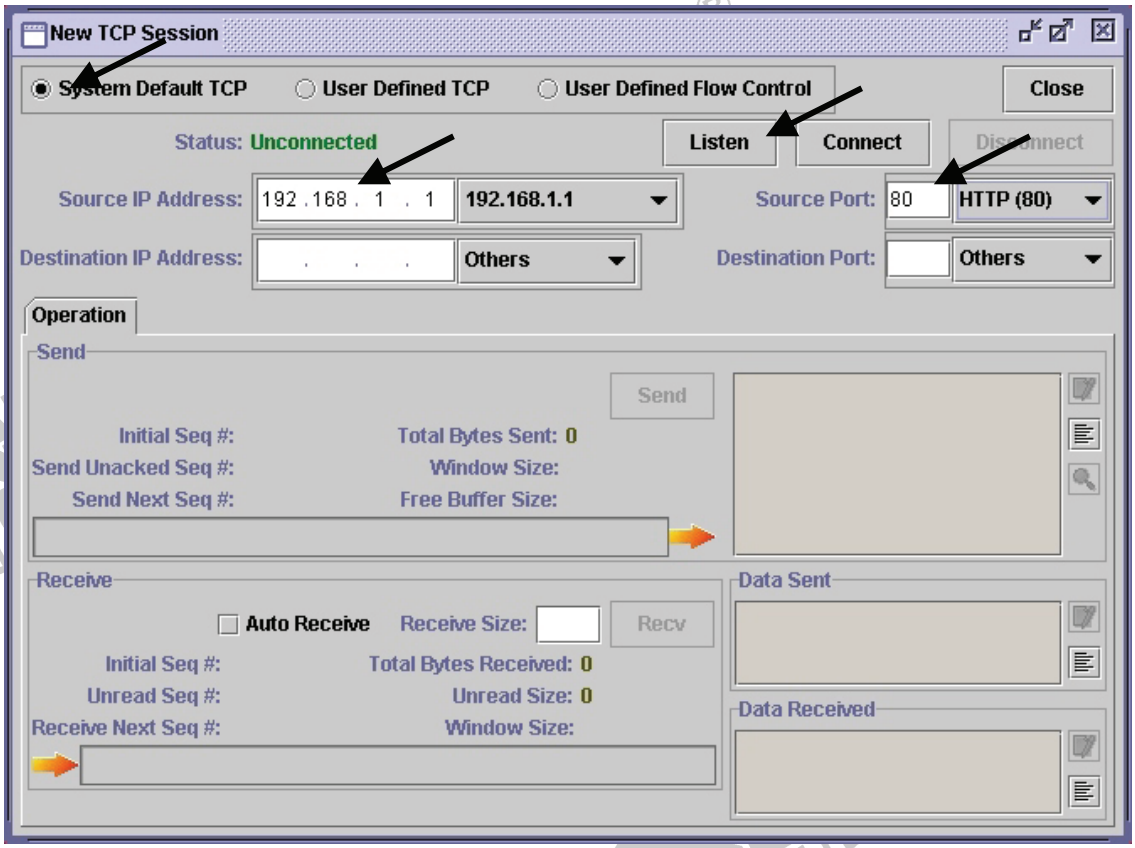


Figure 14.6 New TCP Session dialog

ITS2

12. Open the Network Message Browser window. Check **Listening On**.
13. Click **New TCP Session** from the TCP menu to open the New TCP Session dialog box.
14. Select **System Default TCP**. Type "**192.168.1.1**" into Destination IP Address, choose **HTTP (80)** from Destination Port. (See Figure 14.7.)
15. Click the **Connect** button. Now we have already connected ITS1 to ITS2 using TCP. We will see the 'Three way handshakes' from the message browser as shown in Figure 14.8

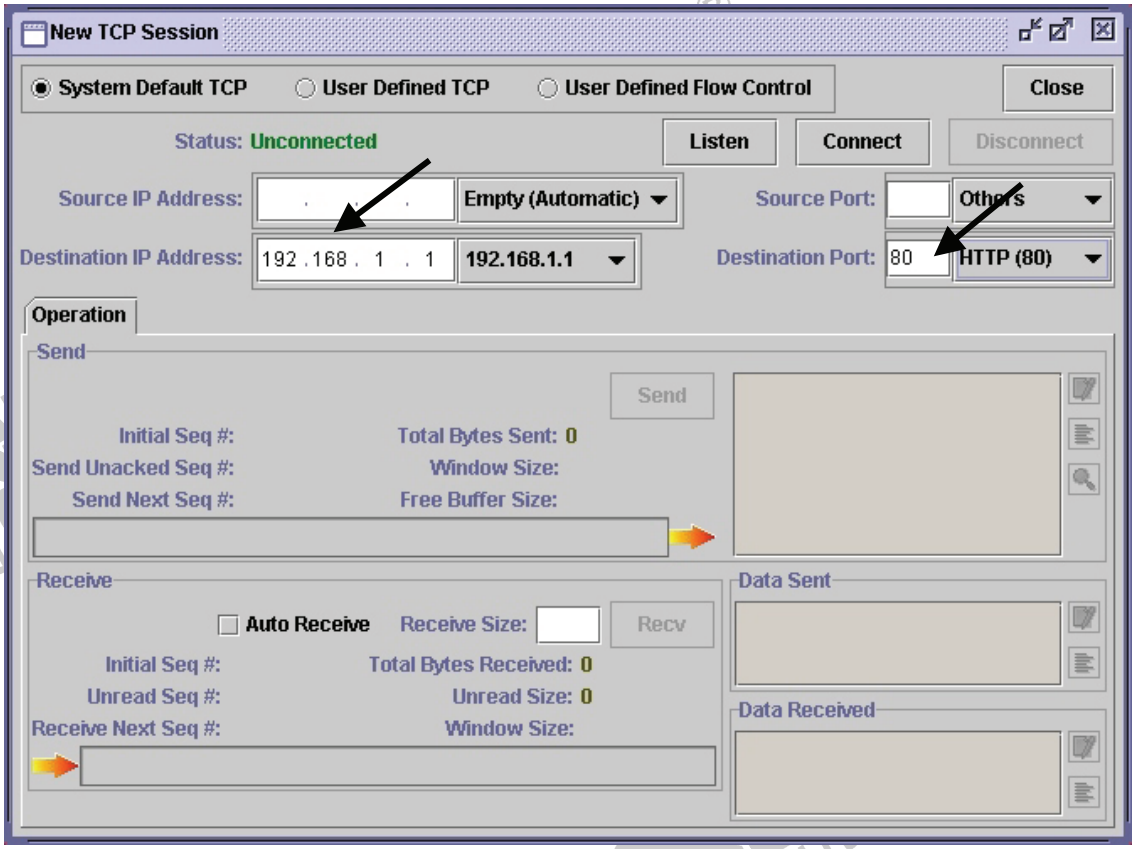


Figure 14.7

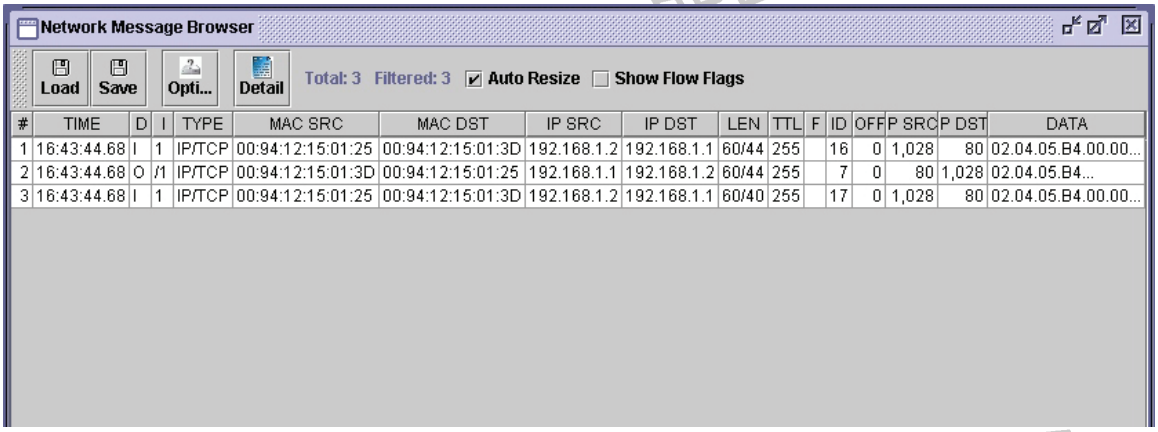


Figure 14.8

User Defined TCP

We can define TCP using MDDL as well. At first, we need to reset our ITS and perform the following steps:

ITS1 and ITS2

16. Open the Network Message Browser window. Check **Listening On**.
17. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
18. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex14 \TCP.mddl and click the **UpId** button.

ITS1

19. Open the New TCP Session dialog box. Select **User Defined TCP**, type "**192.168.1.1**" into Source IP Address, and choose **HTTP (80)** from **Source Port**. Finally click the **Listen** button.

ITS2

20. Open the New TCP Session dialog box. Select **User Defined TCP**, type "**192.168.1.1**" into Destination IP Address, and choose **HTTP (80)** from Destination Port. Finally click the **Listen** button. ITS1 and ITS2 are now connected together by user defined TCP.

Sending Data by User Defined TCP

ITS2

21. In the New TCP Session dialog box, enter "**test**" in the edit box. Then click the **Send** button. (See Figure 14.9.)

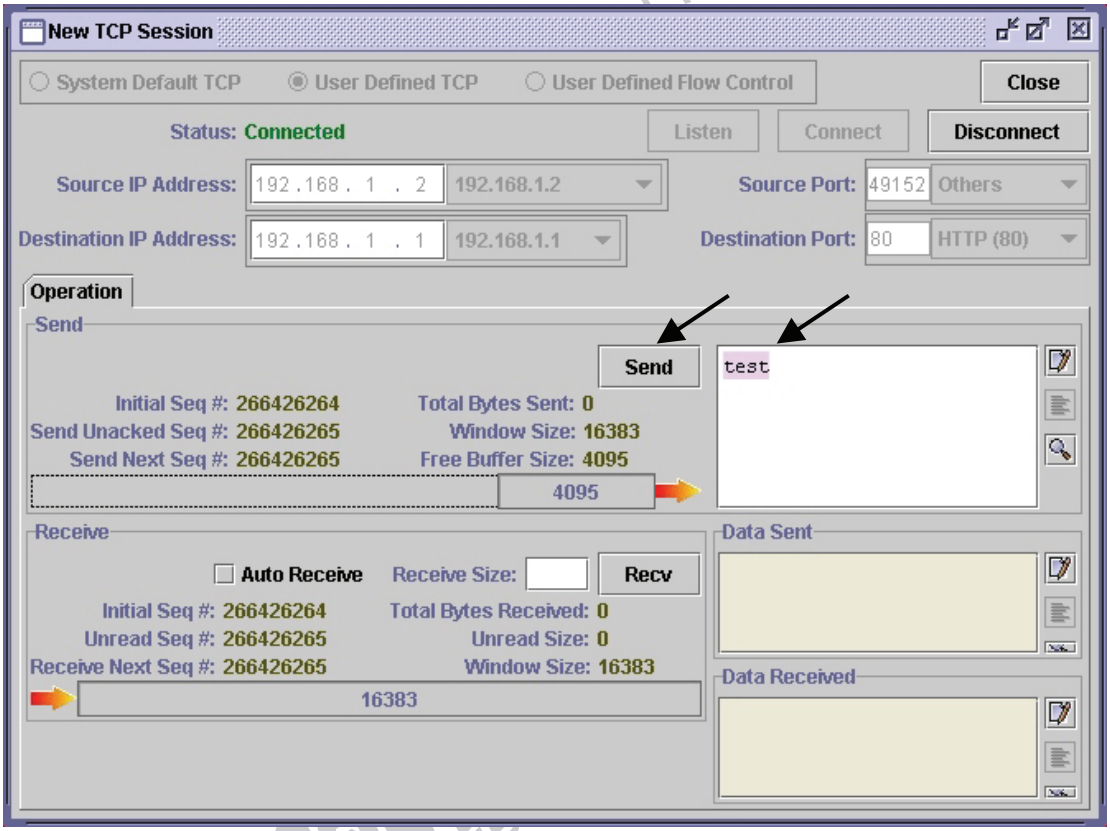


Figure 14.9

ITS1

22. Once data is sent by ITS2, you should receive data and store it in buffer as shown in Figure 14.10.

23. Click the **Recv** button. You should find the received data in the Data Received mailbox as shown in Figure 14.11.

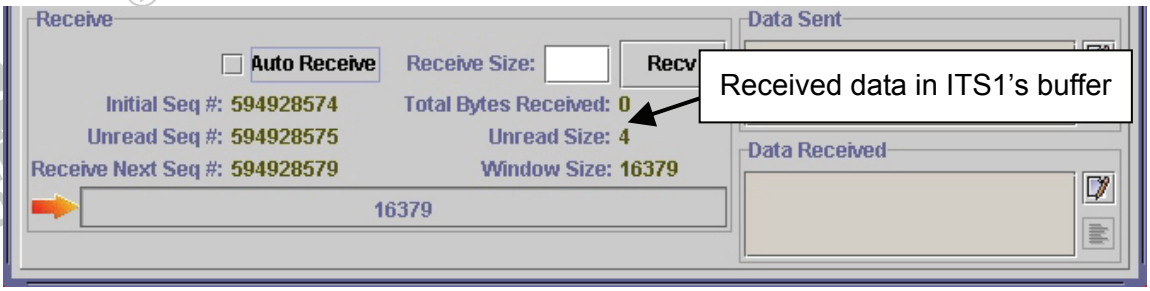


Figure 14.10

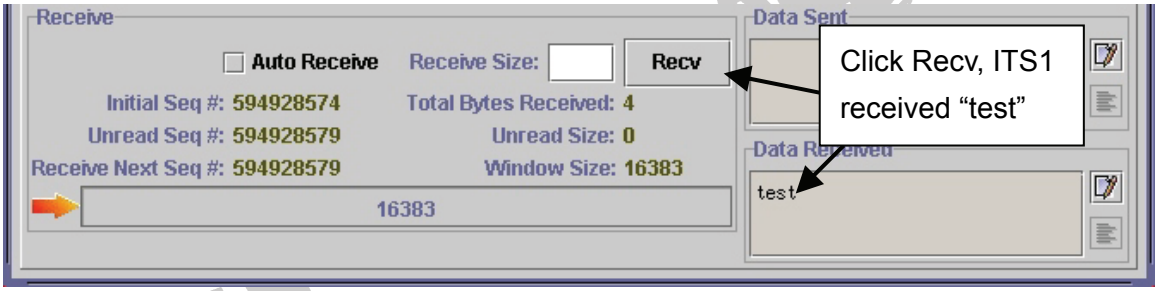


Figure 14.11

DISCUSSIONS

1. What will happen if we click the Disconnect button in the New TCP Session dialog box while data is transferring from ITS1 to ITS2? Try to observe it from the message browser.

2. What will happen if we let ITS2 connect to ITS1 before ITS1 starts listening?

REACTOR PROGRAM

1. **TCP.mddl**

```
VAR6.TCB                                = 0                                 ;
VAR6.TCB_SOCKET_ID                      = 5                                 ;
VAR6.TCB_STATE                          = CNST_TCB_STATE_CLOSED             ;

VAR6.TCB_SND_BUF_SEM                    = 15                                ;
VAR6.TCB_SND_BUF_SIZE                   = 4096W                             ;

VAR6.TCB_RCV_BUF_SEM                    = 25                                ;
VAR6.TCB_RCV_BUF_SIZE                   = 16384W                              ;

VAR6.TCB_WAS_LISTENING                  = FALSE                             ;
VAR6.TCB_POOL_RETRANSMISSION            = 15                                ;
VAR6.TCB_POOL_REASSEMBLY                = 25                                ;

SERVICE_TCP_OPEN
{
    IF( VAR6.TCB_STATE != CNST_TCB_STATE_CLOSED )
    {
        GENERATE_USER_MSG WITH_DATA
        {
            TARGET = "TCP STATE IS NOT CLOSED!"
        }
        GENERATE_USER_MSG WITH_DATA
        {
            TARGET = VAR6.TCB_STATE
        }
        RETURN;
    }

    VAR6.TCB_SND_BUF_L                  = 0W                                ;
    VAR6.TCB_SND_BUF_H                  = 0W                                ;
    VAR6.TCB_RCV_BUF_L                  = 0W                                ;
    VAR6.TCB_RCV_BUF_H                  = 0W                                ;
    VAR6.TCB_RCV_WND = ( VAR6.TCB_RCV_BUF_L +   VAR6.TCB_RCV_BUF_SIZE - 1 -
    VAR6.TCB_RCV_BUF_H ) % VAR6.TCB_RCV_BUF_SIZE;
    VAR6.TCB_TIMER_2MSL                 = 0W                                ;

    VAR6.TCB_STATE                  = CNST_TCB_STATE_SYN_SENT               ;

    IF(PARA_IPADDR_SRC() == CNST_IP_ADDR_BROADCAST)
        VAR6.TCB_IP_ADDRSRC             = MYIP(1)                           ;
    ELSE
        VAR6.TCB_IP_ADDRSRC         = PARA_IPADDR_SRC()                     ;

    VAR6.TCB_IP_ADDRDST             = PARA_IPADDR_DST()                     ;

    IF(PARA_PORT_SRC() == 0W)
        VAR6.TCB_PORTSRC                = 49152W                            ;
    ELSE
        VAR6.TCB_PORTSRC            = PARA_PORT_SRC()                       ;

    VAR6.TCB_PORTDST                = PARA_PORT_DST()                       ;
    VAR6.TCB_ISS                    = RANDOM()                              ;
    VAR6.TCB_SND_UNA                = VAR6.TCB_ISS                          ;
    VAR6.TCB_SND_NXT                = VAR6.TCB_SND_UNA                      ;
```

```
VAR7.TCP_PSEUDO_IP_ADDRSRC          = VAR6.TCB_IP_ADDRSRC                    ;
VAR7.TCP_PSEUDO_IP_ADDRDST          = VAR6.TCB_IP_ADDRDST                    ;
VAR7.TCP_PSEUDO_ZERO                = 0                                      ;
VAR7.TCP_PSEUDO_PROT                = CNST_IP_PROT_TCP                       ;
VAR7.TCP_PSEUDO_LEN                 = 24W                                    ;

VAR7.TCP_PSEUDO_DATA.TCP_PORTSRC        = VAR6.TCB_PORTSRC                   ;
VAR7.TCP_PSEUDO_DATA.TCP_PORTDST        = VAR6.TCB_PORTDST                   ;
VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        = VAR6.TCB_SND_NXT                   ;
VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = 0L                                 ;
VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET    = VAR7.TCP_PSEUDO_LEN << 2           ;
VAR7.TCP_PSEUDO_DATA.TCP_FLAGS          = CNST_TCP_FLAG_SYN                  ;
VAR7.TCP_PSEUDO_DATA.TCP_WINDOW         = VAR6.TCB_RCV_WND                   ;
VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = 0W                                 ;
VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR        = 0W                                 ;
VAR7.TCP_PSEUDO_DATA.TCP_OPTION         = {0X02, 0X04, 0X01, 0XCC}           ;
VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = CHECKSUM(VAR7[0,
                                            VAR7.TCP_PSEUDO_LEN + 11])   ;

VAR6.TCB_SND_NXT                    += 1                                     ;
SEND_OUT_IP WITH_DATA
{
    T.IP_PROT                        = CNST_IP_PROT_TCP                      ,
    T.IP_ADDRDST                     = VAR6.TCB_IP_ADDRDST                   ,
    T.IP_DATA                        = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
}

WAIT_SIGNAL VAR6.TCB_SOCKET_ID;

IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED )
{
    VAR7.TCP_PSEUDO_LEN              = 20W                                   ;
    VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        = VAR6.TCB_SND_NXT              ;
    VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = VAR6.TCB_RCV_NXT              ;
    VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET    = VAR7.TCP_PSEUDO_LEN << 2      ;
    VAR7.TCP_PSEUDO_DATA.TCP_FLAGS          = CNST_TCP_FLAG_ACK             ;
    VAR7.TCP_PSEUDO_DATA.TCP_WINDOW         = VAR6.TCB_RCV_WND              ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = 0W                            ;
    VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR        = 0W                            ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = CHECKSUM(VAR7[0,
                                                VAR7.TCP_PSEUDO_LEN + 11])  ;

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                    = CNST_IP_PROT_TCP                     ,
        T.IP_ADDRDST                 = VAR6.TCB_IP_ADDRDST                  ,
        T.IP_DATA                    = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
    }

    RETVAL_ID_SEND_INI               = VAR6.TCB_ISS                         ;
    RETVAL_ID_RECV_INIT              = VAR6.TCB_IRS                         ;
    RETVAL_SEND_BUFFER_SIZE          = ( VAR6.TCB_SND_BUF_L +
                                         VAR6.TCB_SND_BUF_SIZE - 1 -
                                         VAR6.TCB_SND_BUF_H ) %
                                         VAR6.TCB_SND_BUF_SIZE;
    RETVAL_WIN_SIZE_SEND_INIT        = VAR6.TCB_SND_WND                     ;
    RETVAL_WIN_SIZE_RECV_INIT        = ( VAR6.TCB_RCV_BUF_L +
                                         VAR6.TCB_RCV_BUF_SIZE - 1 -
                                         VAR6.TCB_RCV_BUF_H ) %
                                         VAR6.TCB_RCV_BUF_SIZE;
```

```
        RETVAL_SOCKET_ID                = VAR6.TCB_SOCKET_ID                ;
        RETVAL_IPADDR_SRC               = VAR6.TCB_IP_ADDRSRC               ;
        RETVAL_PORT_SRC                 = VAR6.TCB_PORTSRC                  ;
        RETVAL_ERRORCODE                = CNST_TCP_NO_ERROR                 ;
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_SYN_RECEIVED )
    {
        VAR7.TCP_PSEUDO_LEN             = 20W                               ;
        VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM      = VAR6.TCB_ISS                ;
        VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM      = VAR6.TCB_RCV_NXT            ;
        VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET  = VAR7.TCP_PSEUDO_LEN << 2    ;
        VAR7.TCP_PSEUDO_DATA.TCP_FLAGS        = CNST_TCP_FLAG_ACK |
                                                CNST_TCP_FLAG_SYN           ;
        VAR7.TCP_PSEUDO_DATA.TCP_WINDOW       = VAR6.TCB_RCV_WND            ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM       = 0W                          ;
        VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR      = 0W                          ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM       = CHECKSUM(VAR7[0,
                                                VAR7.TCP_PSEUDO_LEN + 11])  ;

        SEND_OUT_IP WITH_DATA
        {
            T.IP_PROT                   = CNST_IP_PROT_TCP                  ,
            T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST               ,
            T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
        }
    }
    ELSE
        RETVAL_ERRORCODE            = CNST_TCP_ERROR_OPEN                   ;
}

SERVICE_TCP_LISTEN
{
    IF( VAR6.TCB_STATE != CNST_TCB_STATE_CLOSED )
    {
        GENERATE_USER_MSG WITH_DATA
        {
            TARGET = "TCP STATE IS NOT CLOSED!"
        }
        GENERATE_USER_MSG WITH_DATA
        {
            TARGET = VAR6.TCB_STATE
        }
        RETURN;
    }

    VAR6.TCB_SND_BUF_L                  = 0W                                ;
    VAR6.TCB_SND_BUF_H                  = 0W                                ;
    VAR6.TCB_RCV_BUF_L                  = 0W                                ;
    VAR6.TCB_RCV_BUF_H                  = 0W                                ;
    VAR6.TCB_RCV_WND = ( VAR6.TCB_RCV_BUF_L +   VAR6.TCB_RCV_BUF_SIZE - 1 -
                         VAR6.TCB_RCV_BUF_H ) % VAR6.TCB_RCV_BUF_SIZE;
    VAR6.TCB_TIMER_2MSL                 = 0W                                ;

    VAR6.TCB_STATE                      = CNST_TCB_STATE_LISTEN             ;
    VAR6.TCB_IP_ADDRSRC                 = PARA_IPADDR_SRC()                 ;
    VAR6.TCB_IP_ADDRDST                 = PARA_IPADDR_DST()                 ;
    VAR6.TCB_PORTSRC                    = PARA_PORT_SRC()                   ;
    VAR6.TCB_PORTDST                    = PARA_PORT_DST()                   ;
    VAR6.TCB_WAS_LISTENING              = TRUE                              ;
```

```
WAIT_SIGNAL VAR6.TCB_SOCKET_ID;

IF(VAR6.TCB_STATE != CNST_TCB_STATE_SYN_RECEIVED)
{
    VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
    RETURN;
}

VAR6.TCB_ISS                            = RANDOM()                                  ;
VAR6.TCB_SND_UNA                        = VAR6.TCB_ISS                              ;
VAR6.TCB_SND_NXT                        = R6.TCB_SND_UNA                            ;

VAR7.TCP_PSEUDO_IP_ADDRSRC      = VAR6.TCB_IP_ADDRSRC                               ;
VAR7.TCP_PSEUDO_IP_ADDRDST      = VAR6.TCB_IP_ADDRDST                               ;
VAR7.TCP_PSEUDO_ZERO            = 0                                                 ;
VAR7.TCP_PSEUDO_PROT            = CNST_IP_PROT_TCP                                  ;
VAR7.TCP_PSEUDO_LEN             = 24W                                               ;

VAR7.TCP_PSEUDO_DATA.TCP_PORTSRC        = VAR6.TCB_PORTSRC                          ;
VAR7.TCP_PSEUDO_DATA.TCP_PORTDST        = VAR6.TCB_PORTDST                          ;
VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        = VAR6.TCB_SND_NXT                          ;
VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = VAR6.TCB_RCV_NXT                          ;
VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET    = VAR7.TCP_PSEUDO_LEN << 2                  ;
VAR7.TCP_PSEUDO_DATA.TCP_FLAGS          = CNST_TCP_FLAG_SYN |
                                          CNST_TCP_FLAG_ACK         ;
VAR7.TCP_PSEUDO_DATA.TCP_WINDOW         = VAR6.TCB_RCV_WND                          ;
VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = 0W                                        ;
VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR        = 0W                                        ;
VAR7.TCP_PSEUDO_DATA.TCP_OPTION         = {0X02, 0X04, 0X05, 0XB4}                  ;
VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = CHECKSUM(VAR7[0,
                                          VAR7.TCP_PSEUDO_LEN + 11])    ;

VAR6.TCB_SND_NXT += 1;
SEND_OUT_IP WITH_DATA
{
    T.IP_PROT                   = CNST_IP_PROT_TCP                                  ,
    T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                               ,
    T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
}

WAIT_SIGNAL VAR6.TCB_SOCKET_ID;

IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED )
{
    RETVAL_ID_SEND_INIT         = VAR6.TCB_ISS                                      ;
    RETVAL_ID_RECV_INIT         = VAR6.TCB_IRS                                      ;
    RETVAL_SEND_BUFFER_SIZE     = ( VAR6.TCB_SND_BUF_L +
                                    VAR6.TCB_SND_BUF_SIZE - 1 -
                                    VAR6.TCB_SND_BUF_H ) %
                                    VAR6.TCB_SND_BUF_SIZE;;
    RETVAL_WIN_SIZE_SEND_INIT   = VAR6.TCB_SND_WND                                  ;
    RETVAL_WIN_SIZE_RECV_INIT   = ( VAR6.TCB_RCV_BUF_L +
                                    VAR6.TCB_RCV_BUF_SIZE - 1 -
                                    VAR6.TCB_RCV_BUF_H ) %
                                    VAR6.TCB_RCV_BUF_SIZE;
    RETVAL_SOCKET_ID            = VAR6.TCB_SOCKET_ID                                ;
    RETVAL_IPADDR_SRC           = VAR6.TCB_IP_ADDRSRC                               ;
    RETVAL_IPADDR_DST           = VAR6.TCB_IP_ADDRDST                               ;
    RETVAL_PORT_DST             = VAR6.TCB_PORTDST                                  ;
    RETVAL_ERRORCODE            = CNST_TCP_NO_ERROR                                 ;
```

```
    }
    ELSE
        RETVAL_ERRORCODE            = CNST_TCP_ERROR_OPEN                   ;
}

SERVICE_TCP_CLOSE
{
    IF( VAR6.TCB_SOCKET_ID != PARA_SOCKET_ID() )
        RETURN;

    IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED )
    {
        VAR6.TCB_STATE                  = CNST_TCB_STATE_FIN_WAIT_1             ;

        VAR7.TCP_PSEUDO_LEN             = 20W                                   ;
        VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM      = VAR6.TCB_SND_NXT                ;
        VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM      = VAR6.TCB_RCV_NXT                ;
        VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET  = VAR7.TCP_PSEUDO_LEN << 2        ;
        VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_FIN |
                                              CNST_TCP_FLAG_ACK   ;
        VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND                ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                              ;
        VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                              ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = CHECKSUM(VAR7[0,
                                              VAR7.TCP_PSEUDO_LEN + 11])    ;

        VAR6.TCB_SND_NXT                  += 1                                  ;
        SEND_OUT_IP WITH_DATA
        {
            T.IP_PROT                   = CNST_IP_PROT_TCP                  ,
            T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST               ,
            T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
        }
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_CLOSE_WAIT )
    {
        VAR6.TCB_STATE                  = CNST_TCB_STATE_CLOSING            ;

        VAR7.TCP_PSEUDO_LEN             = 20W                                   ;
        VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM      = VAR6.TCB_SND_NXT                ;
        VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM      = VAR6.TCB_RCV_NXT                ;
        VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET  = VAR7.TCP_PSEUDO_LEN << 2        ;
        VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_FIN |
                                              CNST_TCP_FLAG_ACK             ;
        VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND                ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                              ;
        VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                              ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = CHECKSUM(VAR7[0,
                                              VAR7.TCP_PSEUDO_LEN + 11])    ;

        VAR6.TCB_SND_NXT                += 1                                    ;
        SEND_OUT_IP WITH_DATA
        {
            T.IP_PROT                   = CNST_IP_PROT_TCP                  ,
            T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST               ,
            T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
        }
    }
    ELSE
        RETURN;
```

```
}

SERVICE_TCP_SEND
{
    IF( PARA_SOCKET_BUFFER_LEN() == 0 || VAR6.TCB_SOCKET_ID != PARA_SOCKET_ID() ||
        (VAR6.TCB_STATE != CNST_TCB_STATE_ESTABLISHED && VAR6.TCB_STATE !=
        CNST_TCB_STATE_CLOSE_WAIT))
    {
        RETVAL_DATA_LEN = 0;
        RETURN;
    }

    VAR7.TCP_PSEUDO_LEN             = 20W + PARA_SOCKET_BUFFER_LEN()            ;
    VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        = VAR6.TCB_SND_NXT                  ;
    VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = VAR6.TCB_RCV_NXT                  ;
    VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET        = 80                            ;
    VAR7.TCP_PSEUDO_DATA.TCP_FLAGS        = CNST_TCP_FLAG_ACK |
                                              CNST_TCP_FLAG_PSH                 ;
    VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND                      ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                                    ;
    VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                                    ;

    VAR7.TCP_PSEUDO_DATA.TCP_DATA         = PARA_DATA()                         ;

    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = CHECKSUM(VAR7[0, 11 +
                                             VAR7.TCP_PSEUDO_LEN]);

    ADD_TO_POOL VAR6.TCB_POOL_RETRANSMISSION WITH_DATA
    {
        T[   0,   3]                    = 6                                     ,
        T[   4,   7]                    = 5                                     ,
        T[   8, 11]                     = VAR6.TCB_SND_NXT                      ,
        T[ 12, 15]                      = VAR6.TCB_SND_NXT + PARA_SOCKET_BUFFER_LEN()    ,
        T[ 16,    ]                     = VAR7.TCP_PSEUDO_DATA.[0, VAR7.TCP_PSEUDO_LEN - 1]
    }

    VAR6.TCB_SND_NXT            += PARA_SOCKET_BUFFER_LEN()                     ;
    IF( VAR6.TCB_SND_MAX - VAR6.TCB_SND_NXT >= 0X80000000L )
        VAR6.TCB_SND_MAX                = VAR6.TCB_SND_NXT                      ;

    GENERATE_SEND_BUFFER_PARAMETERS_CHANGED( VAR6.TCB_SOCKET_ID,
    VAR6.TCB_SND_UNA, VAR6.TCB_SND_NXT, VAR6.TCB_SND_WND, ( VAR6.TCB_SND_BUF_L
    +   VAR6.TCB_SND_BUF_SIZE - 1 - VAR6.TCB_SND_BUF_H ) % VAR6.TCB_SND_BUF_SIZE );

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                   = CNST_IP_PROT_TCP                          ,
        T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                       ,
        T.IP_DATA                    = VAR7.TCP_PSEUDO_DATA.[0, VAR7.TCP_PSEUDO_LEN - 1]
    }

    RETVAL_DATA_LEN = PARA_SOCKET_BUFFER_LEN();
    RETVAL_DATA = PARA_DATA();
}

SERVICE_TCP_RECEIVE
{
    IF(VAR6.TCB_RCV_BUF_L == VAR6.TCB_RCV_BUF_H || PARA_SOCKET_BUFFER_LEN() == 0)
        RETURN;
```

```
IF(VAR6.TCB_SOCKET_ID == PARA_SOCKET_ID() && (VAR6.TCB_STATE ==
  CNST_TCB_STATE_ESTABLISHED ||VAR6.TCB_STATE == CNST_TCB_STATE_CLOSE_WAIT
  ||VAR6.TCB_STATE == CNST_TCB_STATE_FIN_WAIT_1 ||VAR6.TCB_STATE ==
  CNST_TCB_STATE_FIN_WAIT_2 ))
{
    IF(PARA_SOCKET_BUFFER_LEN() > (VAR6.TCB_RCV_BUF_H +
       VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_L ) % VAR6.TCB_RCV_BUF_SIZE)
    {
        IF(VAR6.TCB_RCV_BUF_L < VAR6.TCB_RCV_BUF_H)
        {
            RETVAL_DATA.[0, ] = VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_L,
            VAR6.TCB_RCV_BUF_H - 1];
        }
        ELSE
        {
            RETVAL_DATA.[0, ] =   VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_L,
            VAR6.TCB_RCV_BUF_SIZE - 1];
            RETVAL_DATA.[VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_L, ] =
            VAR6.TCB_RCV_BUF.[0, VAR6.TCB_RCV_BUF_H - 1];
        }

        RETVAL_DATA_LEN = (VAR6.TCB_RCV_BUF_H + VAR6.TCB_RCV_BUF_SIZE -
        VAR6.TCB_RCV_BUF_L ) % VAR6.TCB_RCV_BUF_SIZE;

        VAR6.TCB_RCV_BUF_L = VAR6.TCB_RCV_BUF_H;
    }
    ELSE
    {
        IF(VAR6.TCB_RCV_BUF_L < VAR6.TCB_RCV_BUF_H)
        {
            RETVAL_DATA.[0, ] = VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_L,
            VAR6.TCB_RCV_BUF_L + PARA_SOCKET_BUFFER_LEN()- 1];
        }
        ELSE
        {
            IF(PARA_SOCKET_BUFFER_LEN() <= VAR6.TCB_RCV_BUF_SIZE -
            VAR6.TCB_RCV_BUF_L)
            {
                RETVAL_DATA.[0, ] = VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_L,
                VAR6.TCB_RCV_BUF_L + PARA_SOCKET_BUFFER_LEN()- 1];
            }
            ELSE
            {
                RETVAL_DATA.[0, ] =   VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_L,
                VAR6.TCB_RCV_BUF_SIZE - 1];
                RETVAL_DATA.[VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_L, ] =
                VAR6.TCB_RCV_BUF.[0, PARA_SOCKET_BUFFER_LEN() -
                (VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_L) - 1];
            }
        }

        RETVAL_DATA_LEN = PARA_SOCKET_BUFFER_LEN();

        VAR6.TCB_RCV_BUF_L = (VAR6.TCB_RCV_BUF_L + PARA_SOCKET_BUFFER_LEN())
        % VAR6.TCB_RCV_BUF_SIZE;
    }

    VAR6.TCB_RCV_WND = ( VAR6.TCB_RCV_BUF_L +   VAR6.TCB_RCV_BUF_SIZE - 1 -
    VAR6.TCB_RCV_BUF_H ) % VAR6.TCB_RCV_BUF_SIZE;
```

```
            GENERATE_RECEIVE_BUFFER_PARAMETERS_CHANGED(VAR6.TCB_SOCKET_ID,
            VAR6.TCB_RCV_NXT - ( VAR6.TCB_RCV_BUF_H + VAR6.TCB_RCV_BUF_SIZE -
            VAR6.TCB_RCV_BUF_L ) % VAR6.TCB_RCV_BUF_SIZE, VAR6.TCB_RCV_NXT,
            VAR6.TCB_RCV_WND);

            VAR7.TCP_PSEUDO_LEN                 = 20W                                   ;
            VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM    = VAR6.TCB_SND_NXT                      ;
            VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM    = VAR6.TCB_RCV_NXT                      ;
            VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET  = VAR7.TCP_PSEUDO_LEN << 2    ;
            VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_ACK                     ;
            VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND                      ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                                    ;
            VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                                    ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = CHECKSUM(VAR7[0,
                                                    VAR7.TCP_PSEUDO_LEN + 11])    ;

            SEND_OUT_IP WITH_DATA
            {
                T.IP_PROT                   = CNST_IP_PROT_TCP                      ,
                T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                   ,
                T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
            }
        }
        ELSE
        {
            RETVAL_DATA_LEN = 0;
        }
    }

    IP_IN_HANDLER
    {
        IF(S.IP_PROT!=CNST_IP_PROT_TCP)
            RETURN;

        DISCARD_MESSAGE;

        IF( VAR6.TCB_STATE > CNST_TCB_STATE_SYN_SENT && VAR6.TCB_IP_ADDRSRC ==
            S.IP_ADDRDST && VAR6.TCB_IP_ADDRDST == S.IP_ADDRSRC && VAR6.TCB_PORTSRC
            == S.IP_DATA.TCP_PORTDST && VAR6.TCB_PORTDST == S.IP_DATA.TCP_PORTSRC )
        {
            VAR9[0, 3] = S.IP_LEN - ((S.IP_VERHEADERLEN & 0X0F) << 2) -
            (S.IP_DATA.TCP_DATA_OFFSET >> 2);

            VAR9[4] = FALSE;

            IF      ( VAR9[0, 3] == 0 && VAR6.TCB_RCV_WND == 0 )
            {
                IF( S.IP_DATA.TCP_SEQ_NUM == VAR6.TCB_RCV_NXT )
                    VAR9[4] = TRUE;
            }
            ELSE IF( VAR9[0, 3] == 0 && VAR6.TCB_RCV_WND >   0 )
            {
                IF( ( S.IP_DATA.TCP_SEQ_NUM - VAR6.TCB_RCV_NXT < 0X80000000L ) &&
                  ( S.IP_DATA.TCP_SEQ_NUM - ( VAR6.TCB_RCV_NXT + VAR6.TCB_RCV_WND ) ) >=
                  0X80000000L )
                    VAR9[4] = TRUE;
            }
            ELSE IF( VAR9[0, 3] >   0 && VAR6.TCB_RCV_WND == 0 )
            {
                VAR9[4] = FALSE;
```

```
}
ELSE
{
    IF( ( ( S.IP_DATA.TCP_SEQ_NUM - VAR6.TCB_RCV_NXT < 0X80000000L) &&
      ( S.IP_DATA.TCP_SEQ_NUM - ( VAR6.TCB_RCV_NXT + VAR6.TCB_RCV_WND ) >=
      0X80000000L ) ) || (   (   ( S.IP_DATA.TCP_SEQ_NUM + VAR9[0, 3] - 1 ) -
      VAR6.TCB_RCV_NXT < 0X80000000L ) && ( ( S.IP_DATA.TCP_SEQ_NUM + VAR9[0, 3]
      - 1 ) - ( VAR6.TCB_RCV_NXT + VAR6.TCB_RCV_WND ) >= 0X80000000L ) ) )
        VAR9[4] = TRUE;
}

IF(VAR9[4] == FALSE)
{
    IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) == CNST_TCP_FLAG_RST)
        RETURN;

    VAR7.TCP_PSEUDO_LEN                    = 20W                                    ;
    VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM     = VAR6.TCB_SND_NXT                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM     = VAR6.TCB_RCV_NXT                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET   = VAR7.TCP_PSEUDO_LEN << 2 ;
    VAR7.TCP_PSEUDO_DATA.TCP_FLAGS     = CNST_TCP_FLAG_ACK                          ;
    VAR7.TCP_PSEUDO_DATA.TCP_WINDOW      = VAR6.TCB_RCV_WND                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = 0W                                       ;
    VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR     = 0W                                     ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = CHECKSUM(VAR7[0,
                                           VAR7.TCP_PSEUDO_LEN + 11]) ;

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                  = CNST_IP_PROT_TCP                             ,
        T.IP_ADDRDST               = VAR6.TCB_IP_ADDRDST                          ,
        T.IP_DATA                  = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
    }

    RETURN;
}

IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) == CNST_TCP_FLAG_RST )
{
    IF      ( VAR6.TCB_STATE == CNST_TCB_STATE_SYN_RECEIVED )
    {
        IF( VAR6.TCB_WAS_LISTENING == TRUE)
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_LISTEN;
            RETURN;
        }
        ELSE
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
            RETURN;
        }
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED &&
    VAR6.TCB_STATE == CNST_TCB_STATE_FIN_WAIT_1 && VAR6.TCB_STATE ==
    CNST_TCB_STATE_FIN_WAIT_2 && VAR6.TCB_STATE ==
    CNST_TCB_STATE_CLOSE_WAIT )
    {
        VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
        RETURN;
    }
```

```
    ELSE
    {
        VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
        RETURN;
    }
}

IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_SYN ) == CNST_TCP_FLAG_SYN )
{
    VAR7.TCP_PSEUDO_LEN              = 20W                                  ;
    VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        = S.IP_DATA.TCP_ACK_NUM      ;
    VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = 0L                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET   = VAR7.TCP_PSEUDO_LEN << 2 ;
    VAR7.TCP_PSEUDO_DATA.TCP_FLAGS           = NST_TCP_FLAG_RST           ;
    VAR7.TCP_PSEUDO_DATA.TCP_WINDOW          = VAR6.TCB_RCV_WND           ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM          = 0W                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR         = 0W                         ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM          = CHECKSUM(VAR7[0,
                                                 VAR7.TCP_PSEUDO_LEN +
                                                 11])    ;

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                   = CNST_IP_PROT_TCP                    ,
        T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                 ,
        T.IP_DATA                    = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
    }

    RETURN;
}

IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_ACK ) == CNST_TCP_FLAG_ACK )
{
    IF      ( VAR6.TCB_STATE == CNST_TCB_STATE_SYN_RECEIVED )
    {
        IF( ( S.IP_DATA.TCP_ACK_NUM - VAR6.TCB_SND_UNA < 0X80000000L ) &&
            ( VAR6.TCB_SND_NXT - S.IP_DATA.TCP_ACK_NUM < 0X80000000L ) )
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_ESTABLISHED;

            WAKEUP_SIGNAL VAR6.TCB_SOCKET_ID;
        }
        ELSE
        {
            VAR7.TCP_PSEUDO_LEN              = 20W                        ;
            VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM     =
            S.IP_DATA.TCP_ACK_NUM      ;
            VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM      = 0L                    ;
            VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET     =
            VAR7.TCP_PSEUDO_LEN << 2                                      ;
            VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_RST       ;
            VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND        ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                      ;
            VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR     = 0W                     ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = CHECKSUM(VAR7[0,
                                                     VAR7.TCP_PSEUDO_LEN +
                                                     11])    ;

            SEND_OUT_IP WITH_DATA
            {
```

```
                T.IP_PROT                   = CNST_IP_PROT_TCP              ,
                T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST           ,
                T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
            }

            WAKEUP_SIGNAL VAR6.TCB_SOCKET_ID;

            RETURN;
        }
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED || VAR6.TCB_STATE
    == CNST_TCB_STATE_FIN_WAIT_1 || VAR6.TCB_STATE ==
    CNST_TCB_STATE_FIN_WAIT_2 || VAR6.TCB_STATE ==
    CNST_TCB_STATE_CLOSE_WAIT || VAR6.TCB_STATE ==
    CNST_TCB_STATE_CLOSING )
    {
        IF      ( S.IP_DATA.TCP_ACK_NUM - VAR6.TCB_SND_UNA >= 0X80000000L )
        {
            RETURN;
        }
        ELSE IF( VAR6.TCB_SND_NXT - S.IP_DATA.TCP_ACK_NUM >= 0X80000000L )
        {
            VAR6.TCB_FLAGS |= CNST_TF_DELACK;
            RETURN;
        }

        VAR6.TCB_SND_UNA = S.IP_DATA.TCP_ACK_NUM;

        FOR_EVERY_ELEMENT_IN_POOL VAR6.TCB_POOL_RETRANSMISSION
        WITH_CONDITION( VAR6.TCB_SND_UNA - PE[12, 15] < 0X80000000L )
            REMOVE_CURRENT_POOL_ELEMENT;

        IF( ( VAR6.TCB_SND_WL1 - S.IP_DATA.TCP_SEQ_NUM >= 0X80000000L )||
        ( VAR6.TCB_SND_WL1 == S.IP_DATA.TCP_SEQ_NUM &&
        ( S.IP_DATA.TCP_ACK_NUM - VAR6.TCB_SND_WL2 < 0X80000000L ) ) )
        {
            VAR6.TCB_SND_WND = S.IP_DATA.TCP_WINDOW;
            VAR6.TCB_SND_WL1 = S.IP_DATA.TCP_SEQ_NUM;
            VAR6.TCB_SND_WL2 = S.IP_DATA.TCP_ACK_NUM;
        }

        GENERATE_SEND_BUFFER_PARAMETERS_CHANGED( VAR6.TCB_SOCKET_ID,
        VAR6.TCB_SND_UNA, VAR6.TCB_SND_NXT, VAR6.TCB_SND_WND,
        ( VAR6.TCB_SND_BUF_L +    VAR6.TCB_SND_BUF_SIZE - 1 -
        VAR6.TCB_SND_BUF_H ) % VAR6.TCB_SND_BUF_SIZE );

        IF( VAR6.TCB_STATE == CNST_TCB_STATE_FIN_WAIT_1 )
            VAR6.TCB_STATE = CNST_TCB_STATE_FIN_WAIT_2;

        IF( VAR6.TCB_STATE == CNST_TCB_STATE_CLOSING )
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_TIME_WAIT;
            VAR6.TCB_TIMER_2MSL = 10;
        }
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_LAST_ACK )
    {
        VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
        RETURN;
    }
```

```
        ELSE
        {
            VAR6.TCB_FLAGS |= CNST_TF_DELACK;
            RETURN;
        }
    }
    ELSE
        RETURN;

    IF( VAR9[0, 3] > 0 )
    {
        IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED || VAR6.TCB_STATE ==
            CNST_TCB_STATE_FIN_WAIT_1 || VAR6.TCB_STATE ==
            CNST_TCB_STATE_FIN_WAIT_2 )
        {
            ADD_TO_POOL VAR6.TCB_POOL_REASSEMBLY WITH_CONDITION
            ( ( S.IP_DATA.TCP_SEQ_NUM - PE[0, 3] >= 0X80000000L ) || ( PE[0, 3] ==
            S.IP_DATA.TCP_SEQ_NUM && ( ( S.IP_DATA.TCP_SEQ_NUM + VAR9[0, 3] - PE[4,
            7] >= 0X80000000L ) ) ) ) WITH_DATA
            {
                T[0, 3] = S.IP_DATA.TCP_SEQ_NUM                    ,
                T[4, 7] = S.IP_DATA.TCP_SEQ_NUM + VAR9[0, 3]     ,
                T[8,   ] = S.IP_DATA.TCP_DATA
            }

            VAR9[4, 7] = VAR6.TCB_RCV_NXT;

            FOR_EVERY_ELEMENT_IN_POOL VAR6.TCB_POOL_REASSEMBLY
            {
                IF( ( VAR9[4, 7] - PE[0, 3] < 0X80000000L ) && ( PE[4, 7] - VAR6.TCB_RCV_NXT
                    <= ( VAR6.TCB_RCV_BUF_L +   VAR6.TCB_RCV_BUF_SIZE - 1 -
                    VAR6.TCB_RCV_BUF_H ) % VAR6.TCB_RCV_BUF_SIZE ) )
                {
                    IF( PE[4, 7] - VAR9[4, 7] < 0X80000000L )
                        VAR9[4, 7] = PE[4, 7];
                }
                ELSE
                {
                    BREAK;
                }
            }

            FOR_EVERY_ELEMENT_IN_POOL VAR6.TCB_POOL_REASSEMBLY
            WITH_CONDITION( VAR9[4, 7] - PE[4, 7] < 0X80000000L )
            {
                IF((PE[4, 7] - VAR6.TCB_RCV_NXT) > ( VAR6.TCB_RCV_BUF_L +
                VAR6.TCB_RCV_BUF_SIZE - 1 - VAR6.TCB_RCV_BUF_H ) %
                VAR6.TCB_RCV_BUF_SIZE)
                {
                    BREAK;
                }

                IF((VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_H) >= (PE[4, 7] -
                VAR6.TCB_RCV_NXT))
                {
                    VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_H, ] =
                    PE.[8, ].[VAR6.TCB_RCV_NXT - PE[0, 3], PE[4, 7] - PE[0, 3] - 1];
                }
                ELSE
                {
```

```
                    VAR6.TCB_RCV_BUF.[VAR6.TCB_RCV_BUF_H, ] =
                    PE.[8, ].[VAR6.TCB_RCV_NXT - PE[0, 3], VAR6.TCB_RCV_NXT - PE[0, 3] +
                    (VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_H) - 1];

                    VAR6.TCB_RCV_BUF.[0, ] = PE.[8, ].[VAR6.TCB_RCV_NXT - PE[0, 3] +
                    (VAR6.TCB_RCV_BUF_SIZE - VAR6.TCB_RCV_BUF_H), PE[4, 7] - PE[0, 3]
                    - 1];
                }

                VAR6.TCB_RCV_BUF_H += PE[4, 7] - VAR6.TCB_RCV_NXT;

                VAR6.TCB_RCV_BUF_H %= VAR6.TCB_RCV_BUF_SIZE;

                VAR6.TCB_RCV_NXT = PE[4, 7];

                REMOVE_CURRENT_POOL_ELEMENT;
            }

            VAR6.TCB_RCV_WND = ( VAR6.TCB_RCV_BUF_L +  VAR6.TCB_RCV_BUF_SIZE
            - 1 - VAR6.TCB_RCV_BUF_H ) % VAR6.TCB_RCV_BUF_SIZE;


GENERATE_RECEIVE_BUFFER_PARAMETERS_CHANGED(VAR6.TCB_SOCKET_ID,
VAR6.TCB_RCV_NXT - ( VAR6.TCB_RCV_BUF_H + VAR6.TCB_RCV_BUF_SIZE -
VAR6.TCB_RCV_BUF_L ) % VAR6.TCB_RCV_BUF_SIZE, VAR6.TCB_RCV_NXT,
VAR6.TCB_RCV_WND);

                VAR6.TCB_FLAGS |= CNST_TF_DELACK;
            }
        }

        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_FIN ) == CNST_TCP_FLAG_FIN )
        {
            VAR6.TCB_RCV_NXT += 1L;

            IF      ( VAR6.TCB_STATE == CNST_TCB_STATE_SYN_RECEIVED )
            {
                VAR6.TCB_STATE = CNST_TCB_STATE_CLOSE_WAIT;
            }
            ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_ESTABLISHED )
            {
                VAR6.TCB_STATE = CNST_TCB_STATE_CLOSE_WAIT;

                VAR7.TCP_PSEUDO_LEN          = 20W                                ;
                VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM       = VAR6.TCB_SND_NXT     ;
                VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM       = VAR6.TCB_RCV_NXT     ;
                VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET = VAR7.TCP_PSEUDO_LEN << 2;
                VAR7.TCP_PSEUDO_DATA.TCP_FLAGS       = CNST_TCP_FLAG_ACK          ;
                VAR7.TCP_PSEUDO_DATA.TCP_WINDOW   = VAR6.TCB_RCV_WND              ;
                VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM    = 0W                           ;
                VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR   = 0W                           ;
                VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM            = CHECKSUM(VAR7[0,
                                                             VAR7.TCP_PSEUDO_LEN
                                                             + 11])    ;

                SEND_OUT_IP WITH_DATA
                {
                    T.IP_PROT              = CNST_IP_PROT_TCP                    ,
                    T.IP_ADDRDS            = VAR6.TCB_IP_ADDRDST                 ,
                    T.IP_DATA               = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
```

```
        }

        VAR6.TCB_FLAGS &= ~(CNST_TF_DELACK);

        GENERATE_REMOTE_CLOSED(0, VAR6.TCB_SOCKET_ID);
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_FIN_WAIT_1 )
    {
        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_ACK ) ==
        CNST_TCP_FLAG_ACK )
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_TIME_WAIT;
            VAR6.TCB_TIMER_2MSL = 10;
        }
        ELSE
            VAR6.TCB_STATE = CNST_TCB_STATE_CLOSING;
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_FIN_WAIT_2 )
    {
        VAR6.TCB_STATE = CNST_TCB_STATE_TIME_WAIT;
        VAR6.TCB_TIMER_2MSL = 10;

        VAR7.TCP_PSEUDO_LEN              = 20W                                  ;
        VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM     = VAR6.TCB_SND_NXT       ;
        VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM     = VAR6.TCB_RCV_NXT       ;
        VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET = VAR7.TCP_PSEUDO_LEN << 2;
        VAR7.TCP_PSEUDO_DATA.TCP_FLAGS       = CNST_TCP_FLAG_ACK        ;
        VAR7.TCP_PSEUDO_DATA.TCP_WINDOW      = VAR6.TCB_RCV_WND         ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = 0W                                   ;
        VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR     = 0W                                   ;
        VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM      = CHECKSUM(VAR7[0,
                                               VAR7.TCP_PSEUDO_LEN +
                                               11])     ;

        SEND_OUT_IP WITH_DATA
        {
            T.IP_PROT            = CNST_IP_PROT_TCP                             ,
            T.IP_ADDRDST         = VAR6.TCB_IP_ADDRDST                          ,
            T.IP_DATA            = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
        }

        GENERATE_DISCONNECTED(0, VAR6.TCB_SOCKET_ID);
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_CLOSE_WAIT )
    {

    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_CLOSING )
    {

    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_LAST_ACK )
    {

    }
    ELSE
    {

    }
}
```

```
}
ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_SYN_SENT && VAR6.TCB_IP_ADDRSRC ==
S.IP_ADDRDST && VAR6.TCB_IP_ADDRDST == S.IP_ADDRSRC && VAR6.TCB_PORTSRC ==
S.IP_DATA.TCP_PORTDST && VAR6.TCB_PORTDST == S.IP_DATA.TCP_PORTSRC )
{
    IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_ACK ) == CNST_TCP_FLAG_ACK )
    {
        IF(S.IP_DATA.TCP_ACK_NUM!=VAR6.TCB_SND_NXT)
        {
            IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) ==
            CNST_TCP_FLAG_RST )
                RETURN;
            ELSE
            {
                VAR7.TCP_PSEUDO_LEN             = 20W                           ;
                VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM    =
                S.IP_DATA.TCP_ACK_NUM   ;
                VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM    = 0L                        ;
                VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET    =
                VAR7.TCP_PSEUDO_LEN << 2                        ;
                VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      =
                CNST_TCP_FLAG_RST                       ;
                VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     =
                VAR6.TCB_RCV_WND                        ;
                VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                        ;
                VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                        ;
                VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = CHECKSUM(VAR7[0,
                VAR7.TCP_PSEUDO_LEN + 11])  ;

                SEND_OUT_IP WITH_DATA
                {
                    T.IP_PROT           = CNST_IP_PROT_TCP                  ,
                    T.IP_ADDRDST        = VAR6.TCB_IP_ADDRDST               ,
                    T.IP_DATA           = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
                }

                RETURN;
            }
        }
        ELSE
        {
            IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) ==
            CNST_TCP_FLAG_RST )
            {
                VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
                RETURN;
            }

            IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_SYN ) ==
            CNST_TCP_FLAG_SYN )
            {
                VAR6.TCB_STATE      = CNST_TCB_STATE_ESTABLISHED    ;
                VAR6.TCB_SND_UNA    = S.IP_DATA.TCP_ACK_NUM         ;
                VAR6.TCB_IRS        = S.IP_DATA.TCP_SEQ_NUM         ;
                VAR6.TCB_RCV_NXT    = S.IP_DATA.TCP_SEQ_NUM  + 1L   ;
                VAR6.TCB_SND_WND    = S.IP_DATA.TCP_WINDOW          ;

                WAKEUP_SIGNAL VAR6.TCB_SOCKET_ID;
            }
        }
```

```
        }
        ELSE
        {
            IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) == CNST_TCP_FLAG_RST )
                RETURN;

            IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_SYN ) == CNST_TCP_FLAG_SYN )
            {
                VAR6.TCB_STATE          = CNST_TCB_STATE_SYN_RECEIVED       ;
                VAR6.TCB_IRS            = S.IP_DATA.TCP_SEQ_NUM             ;
                VAR6.TCB_RCV_NXT        = S.IP_DATA.TCP_SEQ_NUM   + 1L      ;
                VAR6.TCB_SND_WND        = S.IP_DATA.TCP_WINDOW              ;

                WAKEUP_SIGNAL VAR6.TCB_SOCKET_ID;
            }
        }
    }
    ELSE IF( VAR6.TCB_STATE == CNST_TCB_STATE_LISTEN && ( VAR6.TCB_IP_ADDRSRC ==
    CNST_IP_ADDR_BROADCAST || VAR6.TCB_IP_ADDRSRC == S.IP_ADDRDST ) &&
    ( VAR6.TCB_PORTSRC == S.IP_DATA.TCP_PORTDST ) )
    {
        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) == CNST_TCP_FLAG_RST )
            RETURN;

        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_ACK ) == CNST_TCP_FLAG_ACK )
        {
            VAR7.TCP_PSEUDO_LEN             = 20W                           ;
            VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        =
            S.IP_DATA.TCP_ACK_NUM                   ;
            VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = 0L                    ;
            VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET    = VAR7.TCP_PSEUDO_LEN << 2  ;
            VAR7.TCP_PSEUDO_DATA.TCP_FLAGS          = CNST_TCP_FLAG_RST         ;
            VAR7.TCP_PSEUDO_DATA.TCP_WINDOW         = VAR6.TCB_RCV_WND          ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = 0W                        ;
            VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR        = 0W                        ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM         = CHECKSUM(VAR7[0,
                                                        VAR7.TCP_PSEUDO_LEN + 11]) ;

            SEND_OUT_IP WITH_DATA
            {
                T.IP_PROT                   = CNST_IP_PROT_TCP              ,
                T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST           ,
                T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
            }

            RETURN;
        }

        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_SYN ) == CNST_TCP_FLAG_SYN )
        {
            VAR6.TCB_STATE              = CNST_TCB_STATE_SYN_RECEIVED   ;
            VAR6.TCB_IRS                = S.IP_DATA.TCP_SEQ_NUM         ;
            VAR6.TCB_RCV_NXT            = S.IP_DATA.TCP_SEQ_NUM   + 1L  ;

            IF(VAR6.TCB_IP_ADDRSRC == CNST_IP_ADDR_BROADCAST)
                VAR6.TCB_IP_ADDRSRC             = S.IP_ADDRDST;

            VAR6.TCB_IP_ADDRDST                 = S.IP_ADDRSRC;

            VAR6.TCB_PORTDST                    = S.IP_DATA.TCP_PORTSRC;
```

```
            WAKEUP_SIGNAL VAR6.TCB_SOCKET_ID;
        }

    }
    ELSE
    {
        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_RST ) == CNST_TCP_FLAG_RST )
            RETURN;

        IF( ( S.IP_DATA.TCP_FLAGS & CNST_TCP_FLAG_ACK ) == CNST_TCP_FLAG_ACK )
        {
            VAR7.TCP_PSEUDO_LEN            = 20W                                    ;
            VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM        =
            S.IP_DATA.TCP_ACK_NUM                                ;
            VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM        = 0L                          ;
            VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET = VAR7.TCP_PSEUDO_LEN << 2 ;
            VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_RST              ;
            VAR7.TCP_PSEUDO_DATA.TCP_WINDOW   = VAR6.TCB_RCV_WND                 ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM   = 0W                               ;
            VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR   = 0W                              ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = CHECKSUM(VAR7[0,
                                                    VAR7.TCP_PSEUDO_LEN + 11])   ;

            SEND_OUT_IP WITH_DATA
            {
                T.IP_PROT                   = CNST_IP_PROT_TCP                   ,
                T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                ,
                T.IP_DATA                    = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
            }

            RETURN;
        }
        ELSE
        {
            VAR7.TCP_PSEUDO_LEN            = 20W                                 ;
            VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM       = 0L;                         ;
            VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM          = S.IP_DATA.TCP_SEQ_NUM +
                                                        S.IP_LEN -
                                                        (S.IP_VERHEADERLEN & 0X0F)
                                                        << 2 -
                                                        S.IP_DATA.TCP_DATA_OFFSET
                                                        >> 2 ;
            VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET =   VAR7.TCP_PSEUDO_LEN << 2   ;
            VAR7.TCP_PSEUDO_DATA.TCP_FLAGS             = CNST_TCP_FLAG_RST |
                                                         CNST_TCP_FLAG_ACK
                                                         ;
            VAR7.TCP_PSEUDO_DATA.TCP_WINDOW        = VAR6.TCB_RCV_WND            ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM        = 0W                          ;
            VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR       = 0W                          ;
            VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM       = CHECKSUM(VAR7[0,
                                                      VAR7.TCP_PSEUDO_LEN + 11]) ;

            SEND_OUT_IP WITH_DATA
            {
                T.IP_PROT                  = CNST_IP_PROT_TCP                    ,
                T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                ,
                T.IP_DATA                  = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
            }
```

```
            RETURN;
        }
    }
}

TIMER_WITH_PERIOD 200
{
    IF( ( VAR6.TCB_FLAGS & CNST_TF_DELACK ) != CNST_TF_DELACK )
        RETURN;

    VAR7.TCP_PSEUDO_LEN                 = 20W                                   ;
    VAR7.TCP_PSEUDO_DATA.TCP_SEQ_NUM    = VAR6.TCB_SND_NXT                      ;
    VAR7.TCP_PSEUDO_DATA.TCP_ACK_NUM    = VAR6.TCB_RCV_NXT                      ;
    VAR7.TCP_PSEUDO_DATA.TCP_DATA_OFFSET = VAR7.TCP_PSEUDO_LEN << 2             ;
    VAR7.TCP_PSEUDO_DATA.TCP_FLAGS      = CNST_TCP_FLAG_ACK                     ;
    VAR7.TCP_PSEUDO_DATA.TCP_WINDOW     = VAR6.TCB_RCV_WND                      ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM     = 0W                                    ;
    VAR7.TCP_PSEUDO_DATA.TCP_URG_PTR    = 0W                                    ;
    VAR7.TCP_PSEUDO_DATA.TCP_CHKSUM             = CHECKSUM(VAR7[0, 11 +
                                                  VAR7.TCP_PSEUDO_LEN])     ;

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                   = CNST_IP_PROT_TCP                      ,
        T.IP_ADDRDST                = VAR6.TCB_IP_ADDRDST                   ,
        T.IP_DATA                   = VAR7.TCP_PSEUDO_DATA.TCP_HEADER
    }

    VAR6.TCB_FLAGS &= ~(CNST_TF_DELACK);
}

TIMER_WITH_PERIOD 500
{
    FOR_EVERY_ELEMENT_IN_POOL VAR6.TCB_POOL_RETRANSMISSION
    {
        PE[0, 3] -= 1;
        IF( PE[0, 3] > 0 )
            CONTINUE;

        PE[4, 7] -= 1;
        IF( PE[4, 7] == 0 )
        {
            VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
            RETURN;
        }

        PE[0, 3] = 6;

        SEND_OUT_IP WITH_DATA
        {
            T.IP_PROT               = CNST_IP_PROT_TCP                  ,
            T.IP_ADDRDST            = VAR6.TCB_IP_ADDRDST               ,
            T.IP_DATA               = PE.[16, ]
        }
    }

    IF(VAR6.TCB_TIMER_2MSL > 0)
    {
        VAR6.TCB_TIMER_2MSL -= 1;
        IF(VAR6.TCB_TIMER_2MSL==0)
```

```
                VAR6.TCB_STATE = CNST_TCB_STATE_CLOSED;
        }
    }
```

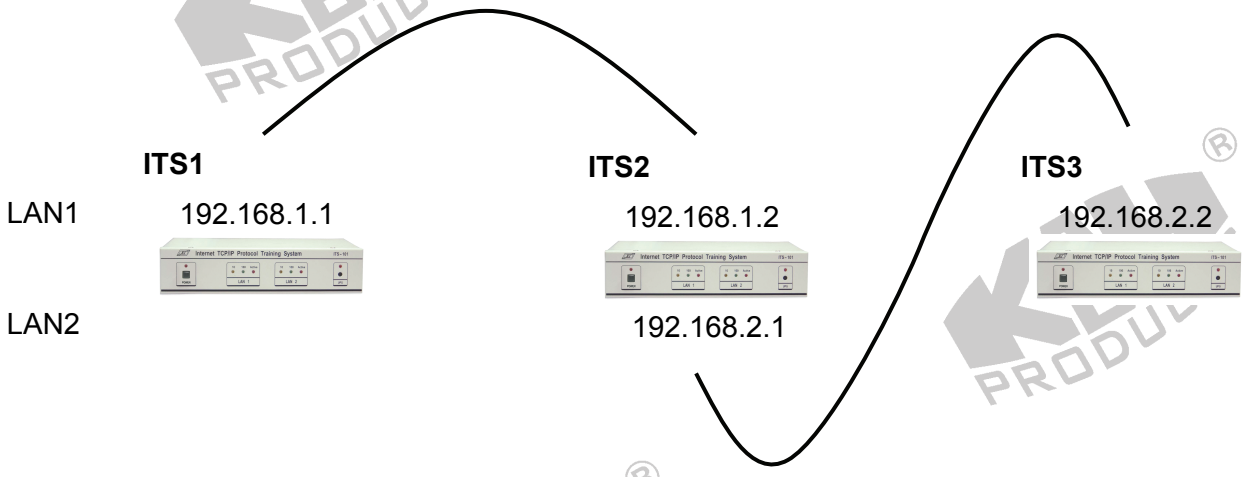
Exp 15. TCP vs. UDP

OBJECTIVE：To test the performance of TCP and UDP using different self-defined flow control.

BRIEF DESCRIPTION： TCP-based applications are popular in current Internet such as TELNET, FTP, since TCP provides reliable communication so that the applications need not to consider the problems of packet lost, error and out of sequence. However, TCP is not the best solution for some applications, for example, multimedia communication which requires real-time data transmission, however, the flow control of TCP will violate this. Based on OSI 7 layers, the application layer can only access TCP or UDP. Alternately, the applications using UDP to perform the user-defined flow control can solve the problems. In fact, some UDP-based application protocols are defined in RFC.

DURATION：4.5 hrs

TOPOLOGY



TECHNICAL BACKGROUND

In most of these experiments, we emphasize on data-link, network, and transport layers, however, they are built up for applications. For normal applications, they access network ability by calling network APIs, and normally, they can only choose to use TCP, UDP, or both.

In most applications, TCP is the most adequate choice, and it is designed in that way, mainly on easily-accessed API interface and self-contained flow-control. By accessing TCP, applications can focus on their main objective, there's no need to worry about those network data maintenance, such as data lost/re-transmission, data correctness, correct sequence of network data, and etc. However, for some special application domain, or under some special environment, TCP is not the best performance choice, and sometimes, TCP cannot satisfy the requirement of some application, for example, the real-time audio/video transmission. For those applications, the alternative choice is to use UDP to fill-in the special-purposed packets.

Except above comparison, there exists a distinct difference, TCP is a connection-wise protocol, UDP is not. Therefore, for some applications which need to send/receive broadcast, UDP is the only choice. However, in this situation, you can only choose UDP, thus it's not our objective to discuss it, and it is mentioned here for completeness.

TCP ensures that data is securely delivered to the destination and that the order of the packets arriving at the destination is correct. We can see that the rather involved program TCP.mddl used in Exp 14 sees to the realization that the data transfer is secure and the packet order is correct. The main mechanisms used in TCP.mddl is IP datagrams.

On the other hand, UDP is a very simple service which does not ensure secured packet delivery nor packet arrival order. The advantage of UDP over TCP is that it is very efficient. UDP does not have to deal with the involved retransmissions and acknowledgements.

But this cost saving by UDP does not obviate UDP from providing secured packet delivery or correct packet arrival order. If the user would like to have these good qualities while using UDP, all what the user has to do is do it himself or herself!

A good example of this is RTP, the Real-time Transport Protocol standardized by IETF RFC 1889. RTP is used very often by multimedia transmissions over the Internet. Transmitting audio or video information is bandwidth consuming, thus the packet transport has to be very efficient.

But oftentimes TCP and UDP are build into the operating systems, and the user cannot change. If the user chooses the transmit multimedia information over the Internet using TCP, the user is forced to use retransmissions and acknowledgements. These mechanisms require extra bandwidth for transmissions.

Now remember that for audio or video transmission, packet arrival order is much important than secured packet delivery. As a matter of fact, one can tolerate some loss of data during audio or video transmission without the overall fidelity of data compression. But one cannot tolerate that the audio or video information is not played back in the correct order.

In other words, TCP is an overkill for multimedia information. So one is left with UDP, but UDP does not provide correct packet arrival order. So one is left with no choice but to do it himself or herself.

PROCEDURE

Realizing Network Topology

1. Complete the network connections on HUBOX by referring to Figure 15.1.

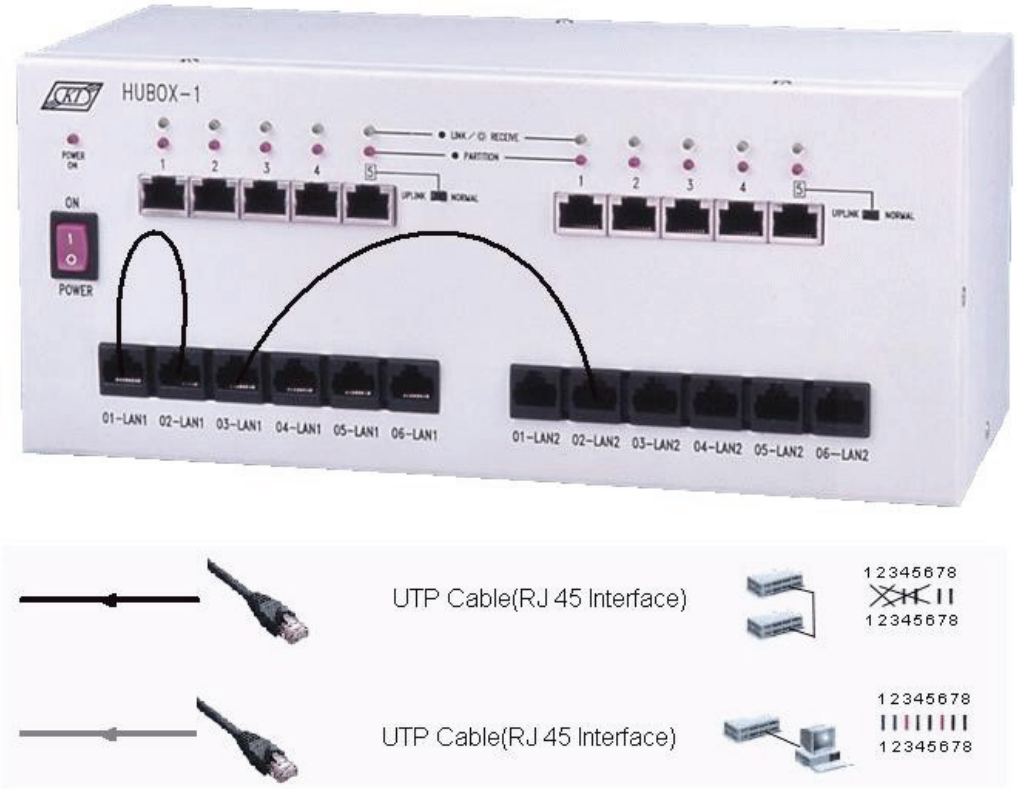


Figure 15.1

Setting Host and Gateway

2. Execute **XCLIENT.BAT** to open the KCodes Network Explorer for ITS window.
3. Open Network Configuration by selecting **Network Configuration** from the Tool menu.

ITS1 (Host)

4. Refer to Topology. Type "**192.168.1.1**" into IP Address of Interface 1 and click the **Add new routing entry** button. (See Figure 15.2.)
5. Type "**192.168.2.0**" into Destination, "**255.255.255.0**" into Mask, and "**192.168.1.2**" into Gateway. Then click the **Update** button. (See Figure 15.3.)
6. Choose **Host**, and click the **Set & Close** button.

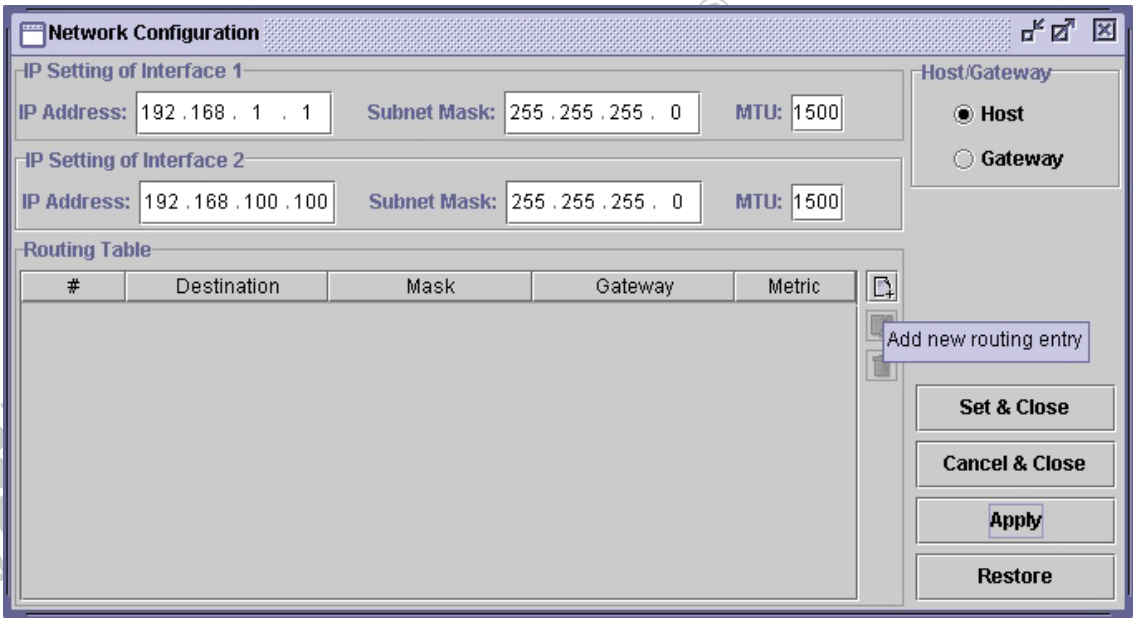


Figure 15.2

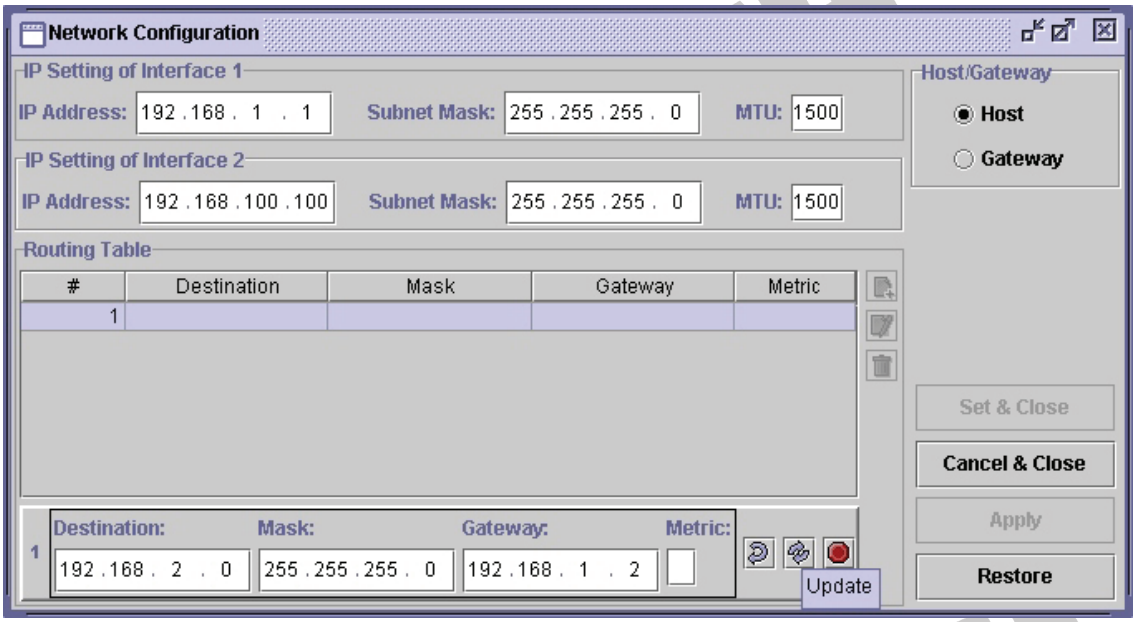


Figure 15.3

ITS3 (Host)

7. Type “**192.168.2.2**” into IP Address of Interface 1 and click the **Add new routing entry** button.
8. Type “**192.168.1.0**” into Destination, “**255.255.255.0**” into Mask, and “**192.168.2.1**” into Gateway. Then click the **Update** button.
9. Choose **Host** and click the **Set & Close** button.

ITS2 (Gateway)

10. Refer to Topology. Type "**192.168.1.2**" into IP Address of Interface 1, and "**192.168.2.1**" into IP Address of Interface 2. (See Figure 15.4.)
11. Choose **Gateway** and click the **Set & Close** button. Now, we already set up our routing table in ITS.

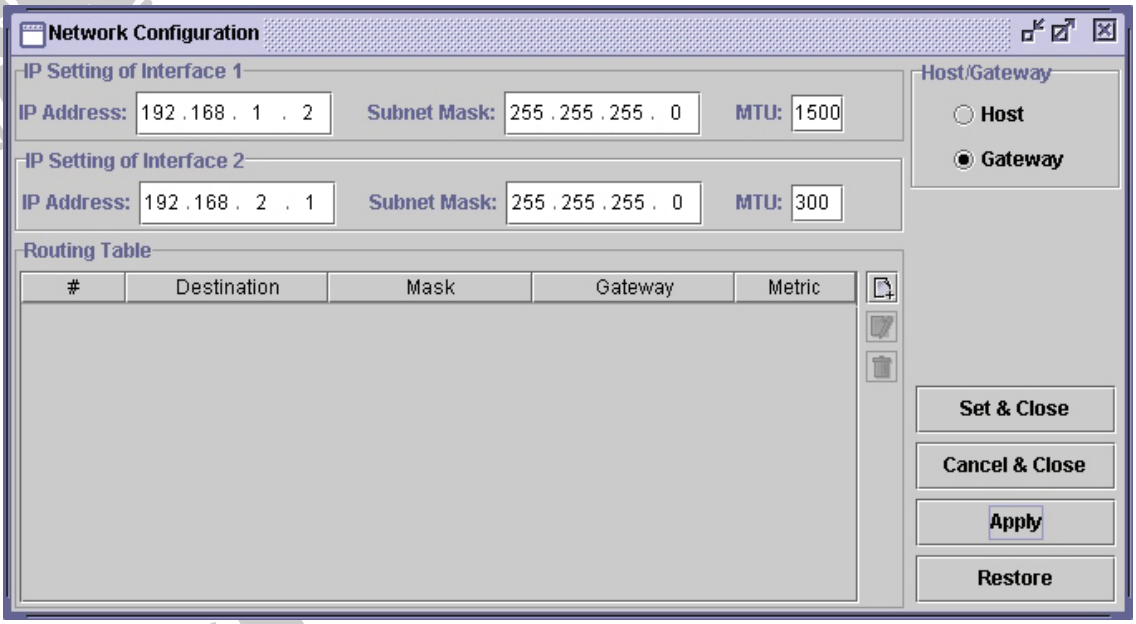


Figure 15.4

TCP File Transfer

ITS2

12. Open the Network Message Browser window. Check **Listening On**.
13. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
14. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex15 \PktDelay15.mddl, and click the **UpId** button.

ITS1

15. Select **File Transfer** from the Application menu to open the File Transfer dialog box.
16. Select **System Default TCP**, type "**192.168.1.1**" into Source IP Address, and choose **HTTP (80)** from Source Port. Finally click the **Listen** button. (See Figure 15.5.)

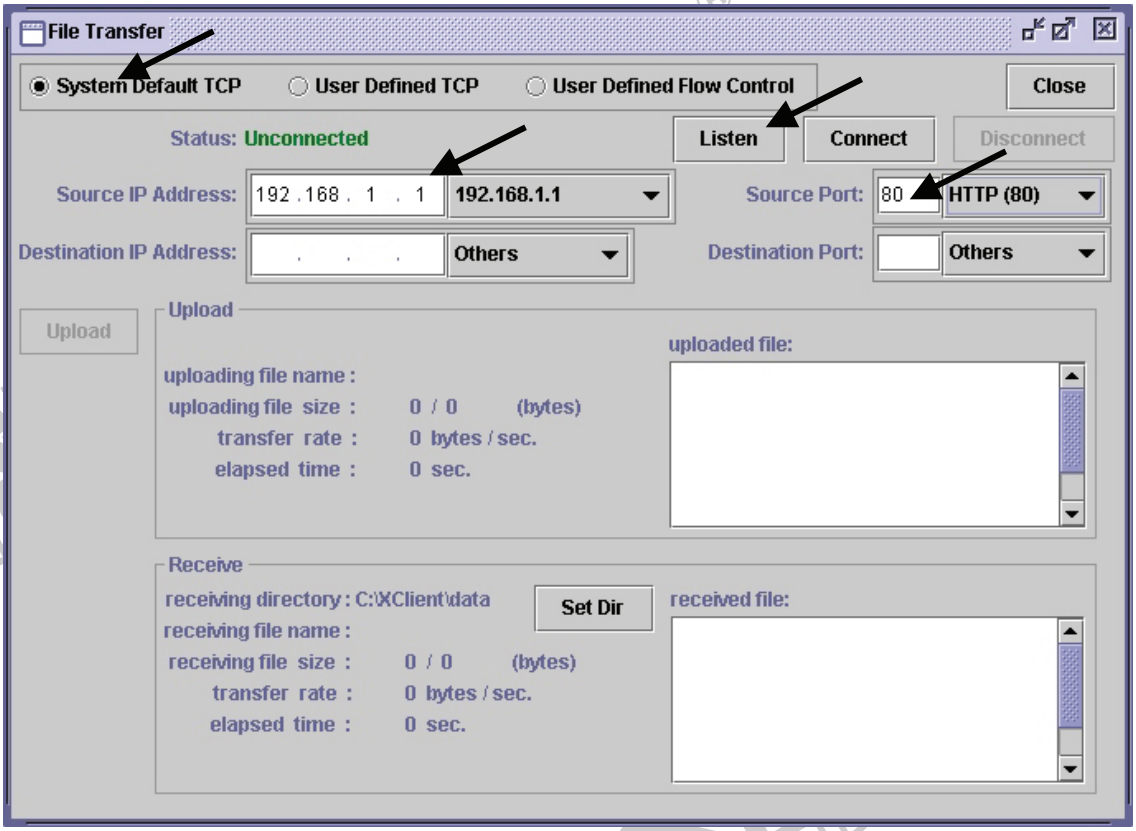


Figure 15.5

ITS3

17. Select **File Transfer** from the Application menu to open the File Transfer dialog box.
18. Select **System Default TCP**, type "**192.168.1.1**" into Destination IP Address, and choose **HTTP (80)** from Destination Port. Finally click the **Connect** button.
19. After the connection is established, click the **Upload** button and open the sample file C:\XClient \Data \9.70.ini (10 KB in size), as shown in Figure 15.6. ITS3 will send this file to ITS1. Observe the transfer rate and elapsed time.

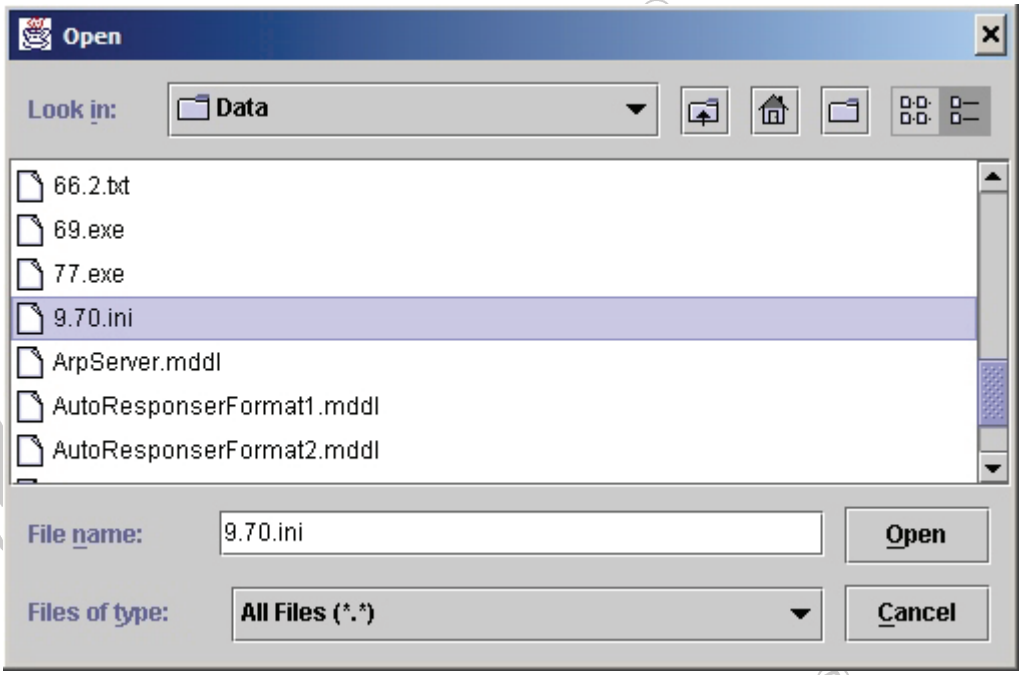


Figure 15.6

UDP File Transfer

ITS2

20. Open the Network Message Browser window. Check **Listening On**.
21. Open the MDDL Editor by selecting **MDDL Reactor Panel** from the Reactor menu.
22. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex15 \PktDelay15.mddl, and click the **UpId** button.

ITS1 and ITS3

23. Click the **Load** button. Open the file C: \XClient \Data \Mddl \Tutorial \Ex15 \UFC.mddl, and click the **UpId** button.
24. Select **File Transfer** from the Application menu to open the File Transfer dialog box as shown in Figure 15.7.

ITS1

25. Select **User Defined Flow Control**, type "**192.168.1.1**" into the Source IP Address textbox, choose **HTTP (80)** from **Source Port**, and press the **Listen** button.

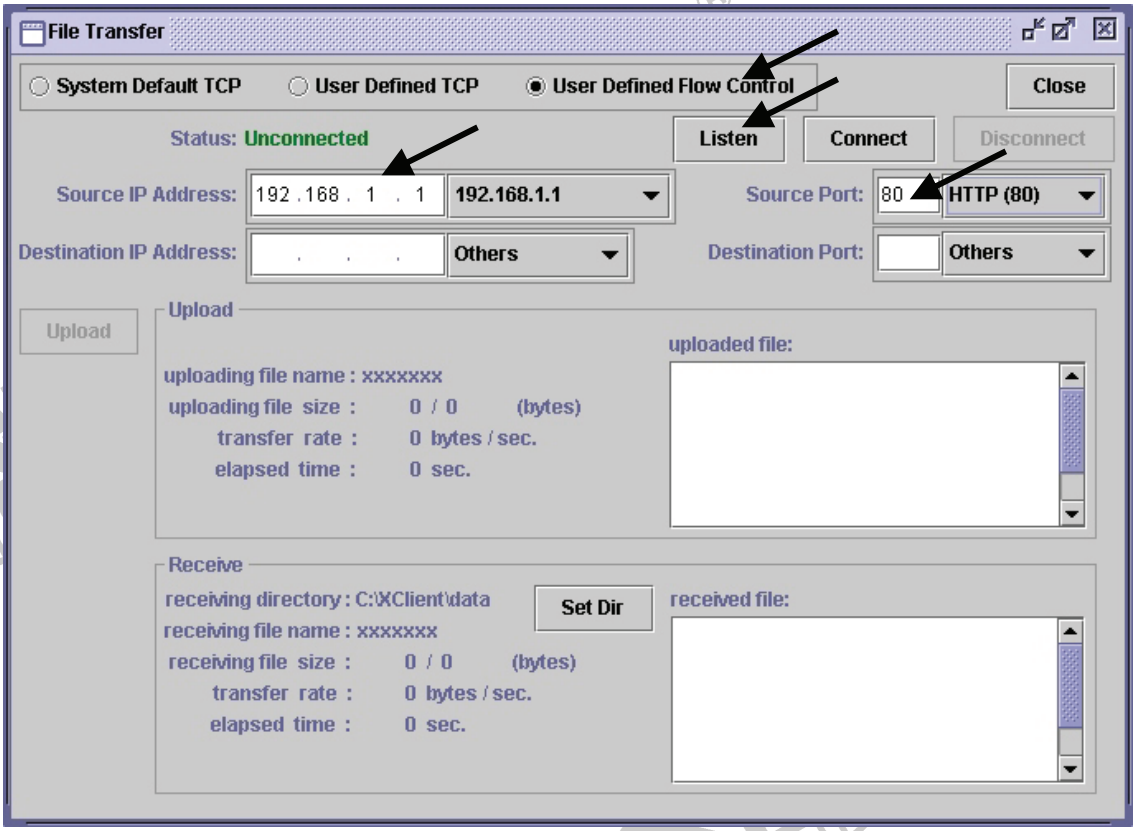


Figure 15.7 File Transfer dialog box

ITS3

26. Select **User Defined Flow Control**, type "**192.168.1.1**" into Destination IP Address, and choose **HTTP (80)** from Destination Port. Finally click the **Connect** button.

27. After the connection is established, click the **Upload** button and open the sample file C:\XClient \Data \9.70.ini (10 KB in size). ITS3 will send this file to ITS1. Compare the transfer rate and elapsed time between TCP and UDP.

DISCUSSIONS

1. Try to send files from ITS3 to ITS1 without loading PktDelay15.mddl to ITS2. Is UDP more faster than TCP?
2. Change the time delay of PktDelay15.mddl to 0.2sec, 0.5sec, 1sec, 1.5sec, 2sec, 3sec, 5sec respectively. Observe how many times the file "9.70.ini" is transferred and discuss the difference between TCP and UDP.

REACTOR PROGRAMS

1. **PktDelay15.mddl**

```
VAR1[0] =0 ;

IP_ROUTING_HANDLER
{
  IF(ISMYIPADDR(S.IP_ADDRDST))
        RETURN;

  IF((S.IP_ADDRDST.[0,2]!= MYIPADDR(2).[0,2])&&(S.IP_ADDRDST.[0,2]!= MYIPADDR(1).[0,2]))
                                                          //compare with netmask 255.255.255.0
     RETURN;

   VAR1[0, 3] = RANDOM(10000L);

    IF(VAR1[0, 3] < 5000L)
   {
       IF(S.IP_TTL>1)
        {
            SEND_OUT_IP WITH_DATA
            {
                T                       = S
                T.IP_TTL                = S.IP_TTL - 1
                T.IP_HEADERCHKSUM = {0, 0}
                T.IP_HEADERCHKSUM = CHECKSUM(T.IP_HEADER)
            }
        }

   }
   else if(VAR1[0, 3] < 10000L)
   {
       ADD_TO_POOL 22 WITH_LIFETIME 20000 WITH_DATA
        {
            T.[0]      = 10
            T.[6, ]   = SOURCE
        }
   }
     DISCARD_MESSAGE;
}

TIMER_WITH_PERIOD 100
{
    FOR_EVERY_ELEMENT_IN_POOL 22
    {
        PE[0] = PE[0] - 1;
        IF(PE[0] == 0)
        {
            SEND_OUT_IP WITH_DATA
            {
                TARGET                  = PE[6, ],
              T.IP_TTL                = PE.IP_TTL - 1,
              T.IP_LEN                = LENGTH(T),
              T.IP_HEADERCHKSUM    = {0, 0}     ,
              T.IP_HEADERCHKSUM    = CHECKSUM(T.IP_HEADER)
            }
             REMOVE_CURRENT_POOL_ELEMENT;
        }
    }
```

10 x 100 ms = 1000 ms =1 sec

Delay timer

```
}
```

2. **UFC.mddl**

```
VAR6.UCB_SOCKET_ID      = 5                         ;
VAR6.UCB_STATE          = CNST_UFC_STATE_CLOSED     ;

SERVICE_UFC_OPEN
{
    IF( VAR6.UCB_STATE != CNST_UFC_STATE_CLOSED )
        RETURN;

    VAR6.UCB_STATE                                  = CNST_UFC_STATE_ESTABLISHED;

    IF(PARA_IPADDR_SRC() == CNST_IP_ADDR_BROADCAST)
        VAR6.UCB_IP_ADDRSRC = MYIP(1);
    ELSE
        VAR6.UCB_IP_ADDRSRC = PARA_IPADDR_SRC();

    VAR6.UCB_IP_ADDRDST                             = PARA_IPADDR_DST()             ;

    IF(PARA_PORT_SRC() == 0W)
        VAR6.UCB_PORTSRC                            = 49152W;
    ELSE
        VAR6.UCB_PORTSRC                            = PARA_PORT_SRC()               ;

    VAR6.UCB_PORTDST                                = PARA_PORT_DST()               ;

    RETVAL_SOCKET_ID    = VAR6.UCB_SOCKET_ID;
    RETVAL_IPADDR_SRC   = VAR6.UCB_IP_ADDRSRC;
    RETVAL_PORT_SRC     = VAR6.UCB_PORTSRC;
    RETVAL_ERRORCODE    = CNST_UFC_NO_ERROR;
}

SERVICE_UFC_LISTEN
{
    IF( VAR6.UCB_STATE != CNST_UFC_STATE_CLOSED )
        RETURN;

    VAR6.UCB_STATE          = CNST_UFC_STATE_LISTEN ;
    VAR6.UCB_IP_ADDRSRC     = PARA_IPADDR_SRC()     ;
    VAR6.UCB_IP_ADDRDST     = PARA_IPADDR_DST()     ;
    VAR6.UCB_PORTSRC        = PARA_PORT_SRC()       ;
    VAR6.UCB_PORTDST        = PARA_PORT_DST()       ;

    WAIT_SIGNAL VAR6.UCB_SOCKET_ID;

    RETVAL_SOCKET_ID    = VAR6.UCB_SOCKET_ID;
    RETVAL_IPADDR_SRC   = VAR6.UCB_IP_ADDRSRC;
    RETVAL_IPADDR_DST   = VAR6.UCB_IP_ADDRDST;
    RETVAL_PORT_DST     = VAR6.UCB_PORTDST;
    RETVAL_ERRORCODE    = CNST_UFC_NO_ERROR;
}

SERVICE_UFC_CLOSE
{
    IF( VAR6.UCB_STATE != CNST_UFC_STATE_ESTABLISHED )
        RETURN;

    VAR6.UCB_STATE                                  = CNST_UFC_STATE_CLOSED;
```

```
    GENERATE_DISCONNECTED(1, VAR6.UCB_SOCKET_ID);
}

SERVICE_UFC_SEND
{
    VAR7.UFC_PSEUDO_IP_ADDRSRC          = VAR6.UCB_IP_ADDRSRC                ;
    VAR7.UFC_PSEUDO_IP_ADDRDST          = VAR6.UCB_IP_ADDRDST                ;
    VAR7.UFC_PSEUDO_ZERO                = 0                                  ;
    VAR7.UFC_PSEUDO_PROT                = CNST_IP_PROT_UFC                   ;
    VAR7.UFC_PSEUDO_LEN                 = 8W                                 ;

    VAR7.UFC_PSEUDO_DATA.UFC_PORTSRC    = VAR6.UCB_PORTSRC                   ;
    VAR7.UFC_PSEUDO_DATA.UFC_PORTDST    = VAR6.UCB_PORTDST                   ;
    VAR7.UFC_PSEUDO_DATA.UFC_LEN        = PARA_SOCKET_BUFFER_LEN() + 8       ;
    VAR7.UFC_PSEUDO_DATA.UFC_CHKSUM     = 0W                                 ;

    VAR7.UFC_PSEUDO_DATA.UFC_DATA       = PARA_DATA()                        ;

    VAR7.UFC_PSEUDO_DATA.UFC_CHKSUM     = CHECKSUM(VAR7[0, 12 +
                                            VAR7.UFC_PSEUDO_LEN +
                                            PARA_SOCKET_BUFFER_LEN() - 1 ])  ;

    SEND_OUT_IP WITH_DATA
    {
        T.IP_PROT                       = CNST_IP_PROT_UFC                   ,
        T.IP_ADDRDST                    = VAR6.UCB_IP_ADDRDST                ,
        T.IP_DATA                       = VAR7.UFC_PSEUDO_DATA.[0,
                                            VAR7.UFC_PSEUDO_LEN +
                                            PARA_SOCKET_BUFFER_LEN() - 1]
    }

    RETVAL_DATA_LEN = PARA_SOCKET_BUFFER_LEN();
    RETVAL_DATA = PARA_DATA();
}

SERVICE_UFC_RECEIVE
{
    GENERATE_USER_MSG WITH_DATA
    {
        TARGET = "UFC_RECEIVE"
    }
    RETVAL_DATA_LEN = 0;
    FOR_EVERY_ELEMENT_IN_POOL 30
    {
        RETVAL_DATA[ RETVAL_DATA_LEN, ] = PE[0, ];
        RETVAL_DATA_LEN += LENGTH(PE);
        REMOVE_CURRENT_POOL_ELEMENT;
    }
}

IP_IN_HANDLER
{
    IF(S.IP_PROT!=CNST_IP_PROT_UFC)
        RETURN;

    DISCARD_MESSAGE;
```

```
        IF( VAR6.UCB_STATE == CNST_UFC_STATE_ESTABLISHED && VAR6.UCB_IP_ADDRSRC ==
        S.IP_ADDRDST && VAR6.UCB_IP_ADDRDST == S.IP_ADDRSRC && VAR6.UCB_PORTSRC ==
        S.IP_DATA.UFC_PORTDST && VAR6.UCB_PORTDST == S.IP_DATA.UFC_PORTSRC )
        {
            ADD_TO_POOL 30 WITH_DATA
            {
                TARGET = S.IP_DATA.UFC_DATA
            }

            GENERATE_SEND_BUFFER_PARAMETERS_CHANGED(VAR6.UCB_SOCKET_ID, 0, 0, 0,
             0);
        }
        ELSE IF( VAR6.UCB_STATE == CNST_UFC_STATE_LISTEN && ( VAR6.UCB_IP_ADDRSRC ==
        CNST_IP_ADDR_BROADCAST || VAR6.UCB_IP_ADDRSRC == S.IP_ADDRDST ) &&
        ( VAR6.UCB_PORTSRC == S.IP_DATA.UFC_PORTDST ) )
        {
            IF(VAR6.UCB_IP_ADDRSRC == CNST_IP_ADDR_BROADCAST)
                VAR6.UCB_IP_ADDRSRC                     = S.IP_ADDRDST;

            VAR6.UCB_IP_ADDRDST                     = S.IP_ADDRSRC;

            VAR6.UCB_PORTDST                        = S.IP_DATA.UFC_PORTSRC;

            VAR6.UCB_STATE = CNST_UFC_STATE_ESTABLISHED;

            WAKEUP_SIGNAL VAR6.UCB_SOCKET_ID;

            ADD_TO_POOL 30 WITH_DATA
            {
                TARGET = S.IP_DATA.UFC_DATA
            }

            GENERATE_SEND_BUFFER_PARAMETERS_CHANGED(VAR6.UCB_SOCKET_ID, 0, 0, 0,
            0);
        }
    }
```