1 Reliable data transfer with TCP (Level 1)

1.1 Introduction

TCP provides reliable data transfer service to the application processes even when the underlying network service (IP service) is unreliable (loses, corrupts, garbles or duplicates packets). TCP uses checksum, sequence numbers, acknowledgements, timers and retransmission to ensure correct and in order delivery of data to the application processes.

TCP views the data stream from the client application process as an ordered stream of bytes. TCP will grab chunks of this data (stored temporarily in the TCP send buffer), add its own header and pass it on to the network layer. A key field of the TCP header is the sequence number which indicates the position of the first byte of the TCP data segment in the data stream. The sequence number will allow the TCP receiver to identify segment losses, duplicate packets and to ensure correct delivery of the data stream to the server application process.

When a server receives a TCP segment, it acknowledges the same with an ACK segment (the segment carrying the acknowledgement has the ACK bit set to 1) and also conveys the sequence number of the first missing byte in the application data stream, in the acknowledgement number field of the TCP header. All acknowledgements are cumulative; hence, all missing, and out-of-order TCP segments will result in duplicate acknowledgements for the corresponding TCP segments.

TCP sender relies on sequence numbering and acknowledgements to ensure reliable transfer of the data stream. In the event of a timeout (no acknowledgement is received before the timer expires) or triple duplicate acknowledgements (multiple ACK segments indicate a lost or missing TCP segment) for a TCP segment, the TCP sender will retransmit the segment until the TCP segment is acknowledged (at least cumulatively). In Figure 1-1, we illustrate retransmission by the TCP sender after a timeout for acknowledgement.

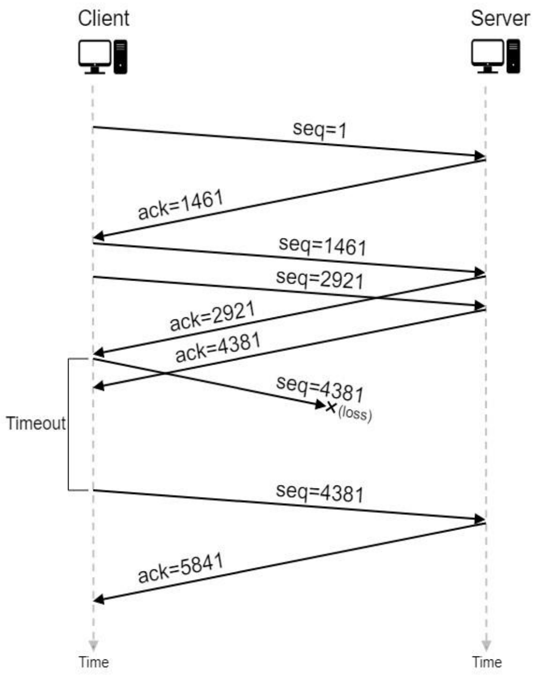
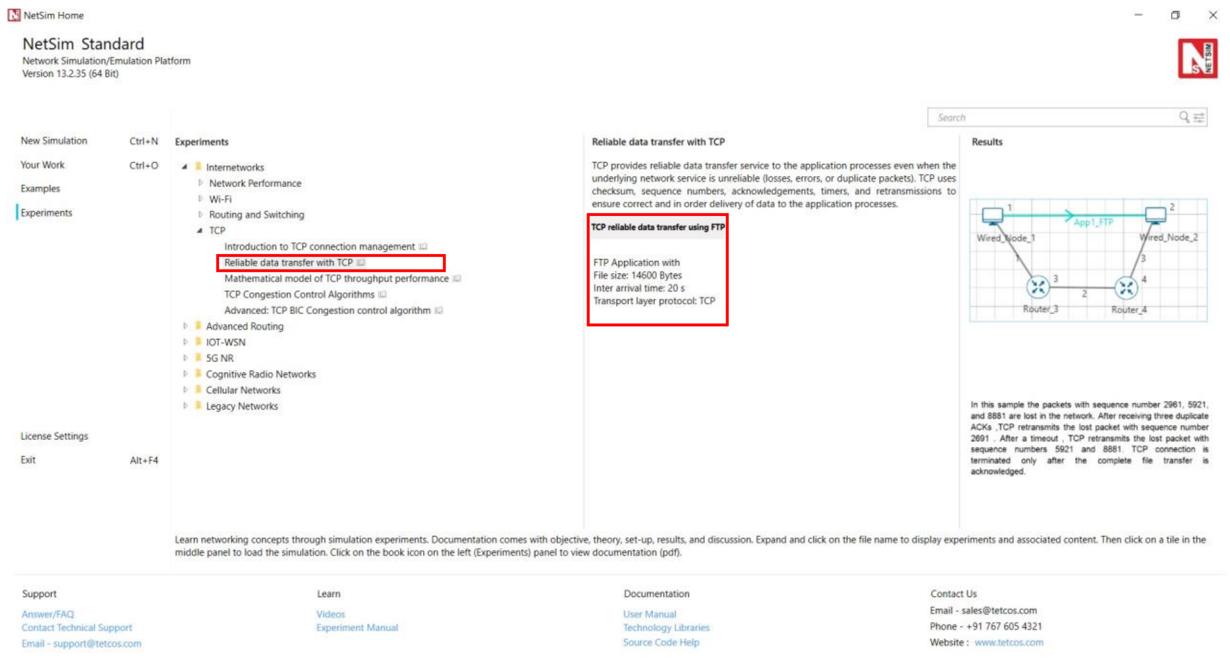


Figure 1-1: An illustration of TCP retransmission with timeout. The segment with sequence number 4381 is lost in the network. The TCP client retransmits the segment after a timeout event.

1.2 Network Setup

Open NetSim and click on **Experiments > Internetworks> TCP> Reliable data transfer with TCP** then click on the tile in the middle panel to load the example as shown in below Figure 1-2.



TCP provides reliable data transfer service to the application processes even when the underlying network service is unreliable (losses, errors, or duplicate packets). TCP uses checksum, sequence numbers, acknowledgements, timers, and retransmissions to ensure correct and in order delivery of data to the application processes.

TCP reliable data transfer using FTP

FTP Application with
File size: 14600 Bytes
Inter arrival time: 20 s
Transport layer protocol: TCP

In this sample the packets with sequence number 2961, 5921, and 8881 are lost in the network. After receiving three duplicate ACKs ,TCP retransmits the lost packet with sequence number 2691 . After a timeout , TCP retransmits the lost packet with sequence numbers 5921 and 8881. TCP connection is terminated only after the complete file transfer is acknowledged.

Learn networking concepts through simulation experiments. Documentation comes with objective, theory, set-up, results, and discussion. Expand and click on the file name to display experiments and associated content. Then click on a tile in the middle panel to load the simulation. Click on the book icon on the left (Experiments) panel to view documentation (pdf).

Figure 1-2: List of scenarios for the example of Reliable data transfer with TCP

NetSim UI displays the configuration file corresponding to this experiment as shown below Figure 1-3.

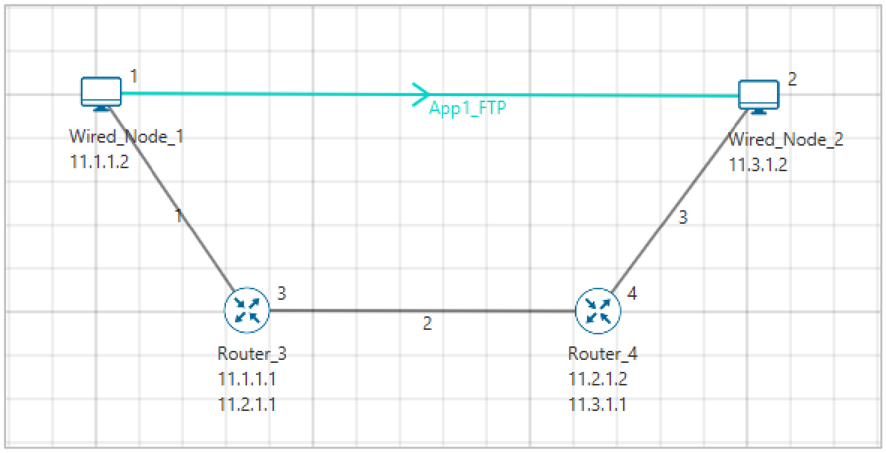


Figure 1-3: Network set up for studying the Reliable data transfer with TCP

We will seek a simple file transfer with TCP over a lossy link to study reliable data transfer with TCP. We will simulate the network setup illustrated in Figure 1-3 with the configuration parameters listed in detail to study reliable data transfer with TCP connection.

1.3 Procedure

The following set of procedures were done to generate this sample.

Step 1: A network scenario is designed in NetSim GUI comprising of 2 Wired Nodes and 2 Routers in the **"Internetworks"** Network Library.

Step 2: In the General Properties of Wired Node 1 i.e., Source and Wired Node 2 i.e., Destination, Wireshark Capture is set to Online. In Transport Layer ->**Congestion plot enabled** is set to **True** for **Wired_Node_1** and **False** for **Wired Node_2**.

Transport Layer properties Congestion plot is set to true for Wired_Node_1.

***NOTE*: Accept default properties for Routers as well as the Links**.

Step 3: Right-click on the link ID (of a wired link) and select Properties to access the link's properties. Set Max Uplink Speed and Max Downlink Speed to **10** Mbps. Set Uplink BER and Downlink BER to **0**. Set Uplink Propagation Delay and Downlink Propagation Delay as **100** microseconds for the links 1 and 3 (between the Wired Node's and the routers). Set Uplink Propagation Delay and Downlink Propagation Delay as **50000** microseconds and Uplink BER and Downlink BER to **0.00001** for the backbone link connecting the routers, i.e., 2.

Step 4: Right click on the Application Flow **App1 FTP** and select Properties or click on the Application icon present in the top ribbon/toolbar.

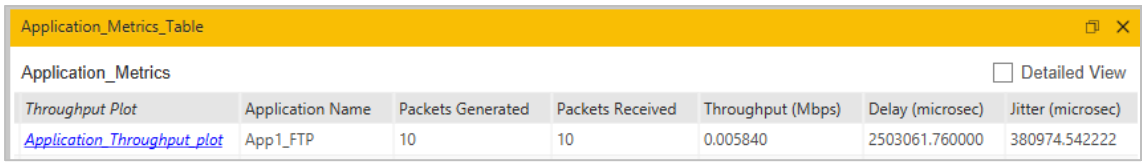
An FTP Application is generated from Wired_Node_1 i.e., Source to Wired_Node_2 i.e., Destination with File Size set to 14600 Bytes and File Inter Arrival Time set to 20 Seconds.

Step 5: Click on Display Settings > Device IP check box in the NetSim GUI to view the network topology along with the IP address.

Step 6: Enable the plots and click on Run simulation. The simulation time is set to 20 seconds.

1.4 Output

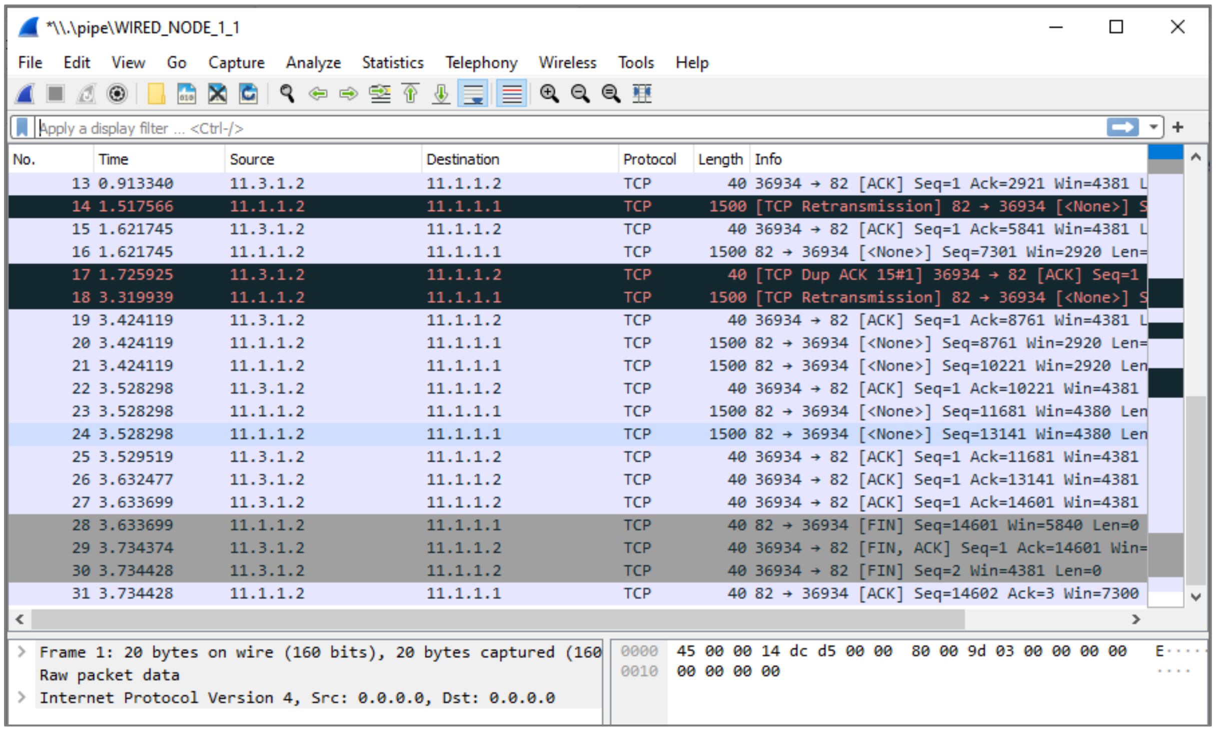
We aimed to transfer a file of size 14600 bytes (i.e., 10 packets, each of size 1460 bytes) with TCP over a lossy link. In Figure 1-4, we report the application metrics data for FTP which indicates that the complete file was transferred.



Throughput Plot	Application Name	Packets Generated	Packets Received	Throughput (Mbps)	Delay (microsec)	Jitter (microsec)
Application_Throughput_plot	App1_FTP	10	10	0.005840	2503061.760000	380974.542222

Figure 1-4: Application Metrics table for FTP

We have enabled Wireshark Capture in Wired_Node 1 and Wired Node_2. The PCAP files are generated at the end of the simulation and are shown in Figure 1-5 and Figure 1-6.

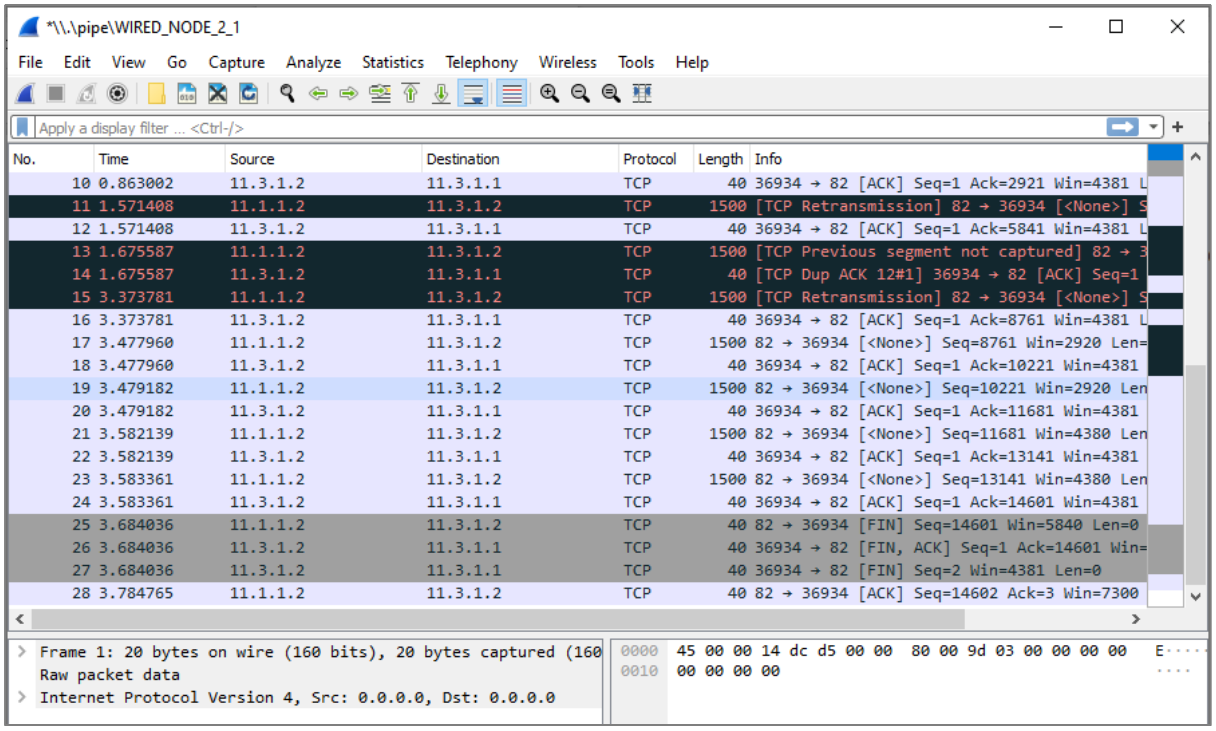

*\\.\pipe\WIRED_NODE_1_1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source    Destination Protocol Length Info
  13 0.913340  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=2921 Win=4381 L
  14 1.517566  11.1.1.2  11.1.1.1    TCP      1500 [TCP Retransmission] 82 → 36934 [<None>] S
  15 1.621745  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=5841 Win=4381 L
  16 1.621745  11.1.1.2  11.1.1.1    TCP      1500 82 → 36934 [<None>] Seq=7301 Win=2920 Len=
  17 1.725925  11.3.1.2  11.1.1.2    TCP        40 [TCP Dup ACK 15#1] 36934 → 82 [ACK] Seq=1
  18 3.319939  11.1.1.2  11.1.1.1    TCP      1500 [TCP Retransmission] 82 → 36934 [<None>] S
  19 3.424119  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=8761 Win=4381 L
  20 3.424119  11.1.1.2  11.1.1.1    TCP      1500 82 → 36934 [<None>] Seq=8761 Win=2920 Len=
  21 3.424119  11.1.1.2  11.1.1.1    TCP      1500 82 → 36934 [<None>] Seq=10221 Win=2920 Len
  22 3.528298  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=10221 Win=4381
  23 3.528298  11.1.1.2  11.1.1.1    TCP      1500 82 → 36934 [<None>] Seq=11681 Win=4380 Len
  24 3.528298  11.1.1.2  11.1.1.1    TCP      1500 82 → 36934 [<None>] Seq=13141 Win=4380 Len
  25 3.529519  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=11681 Win=4381
  26 3.632477  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=13141 Win=4381
  27 3.633699  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [ACK] Seq=1 Ack=14601 Win=4381
  28 3.633699  11.1.1.2  11.1.1.1    TCP        40 82 → 36934 [FIN] Seq=14601 Win=5840 Len=0
  29 3.734374  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [FIN, ACK] Seq=1 Ack=14601 Win=
  30 3.734428  11.3.1.2  11.1.1.2    TCP        40 36934 → 82 [FIN] Seq=2 Win=4381 Len=0
  31 3.734428  11.1.1.2  11.1.1.1    TCP        40 82 → 36934 [ACK] Seq=14602 Ack=3 Win=7300

> Frame 1: 20 bytes on wire (160 bits), 20 bytes captured (160
  Raw packet data
> Internet Protocol Version 4, Src: 0.0.0.0, Dst: 0.0.0.0

0000  45 00 00 14 dc d5 00 00  80 00 9d 03 00 00 00 00   E·····
0010  00 00 00 00                                        ····
```

Figure 1-5: PCAP file at Wired Node 1. TCP ensures reliable data transfer using timeout, duplicate ACKs and retransmissions.


*\\.\pipe\WIRED_NODE_2_1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.  Time      Source    Destination Protocol Length Info
  10 0.863002  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=2921 Win=4381 L
  11 1.571408  11.1.1.2  11.3.1.2    TCP      1500 [TCP Retransmission] 82 → 36934 [<None>] S
  12 1.571408  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=5841 Win=4381 L
  13 1.675587  11.1.1.2  11.3.1.2    TCP      1500 [TCP Previous segment not captured] 82 → 3
  14 1.675587  11.3.1.2  11.3.1.1    TCP        40 [TCP Dup ACK 12#1] 36934 → 82 [ACK] Seq=1
  15 3.373781  11.1.1.2  11.3.1.2    TCP      1500 [TCP Retransmission] 82 → 36934 [<None>] S
  16 3.373781  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=8761 Win=4381 L
  17 3.477960  11.1.1.2  11.3.1.2    TCP      1500 82 → 36934 [<None>] Seq=8761 Win=2920 Len=
  18 3.477960  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=10221 Win=4381
  19 3.479182  11.1.1.2  11.3.1.2    TCP      1500 82 → 36934 [<None>] Seq=10221 Win=2920 Len
  20 3.479182  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=11681 Win=4381
  21 3.582139  11.1.1.2  11.3.1.2    TCP      1500 82 → 36934 [<None>] Seq=11681 Win=4380 Len
  22 3.582139  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=13141 Win=4381
  23 3.583361  11.1.1.2  11.3.1.2    TCP      1500 82 → 36934 [<None>] Seq=13141 Win=4380 Len
  24 3.583361  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [ACK] Seq=1 Ack=14601 Win=4381
  25 3.684036  11.1.1.2  11.3.1.2    TCP        40 82 → 36934 [FIN] Seq=14601 Win=5840 Len=0
  26 3.684036  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [FIN, ACK] Seq=1 Ack=14601 Win=
  27 3.684036  11.3.1.2  11.3.1.1    TCP        40 36934 → 82 [FIN] Seq=2 Win=4381 Len=0
  28 3.784765  11.1.1.2  11.3.1.2    TCP        40 82 → 36934 [ACK] Seq=14602 Ack=3 Win=7300

> Frame 1: 20 bytes on wire (160 bits), 20 bytes captured (160
  Raw packet data
> Internet Protocol Version 4, Src: 0.0.0.0, Dst: 0.0.0.0

0000  45 00 00 14 dc d5 00 00  80 00 9d 03 00 00 00 00   E·····
0010  00 00 00 00                                        ····
```

Figure 1-6: PCAP file at Wired Node 2

1.5 Inference

1. From Figure 1-5 and Figure 1-6, we note that the packets with sequence number 1461 and 5841 are errored in the network, which can also observe in Packet Trace.
2. After receiving three duplicate ACKs (in lines 13, 14 of Figure 1-5), TCP retransmits the errored packet. (In line 15 of Figure 1-5).
3. TCP connection is terminated only after the complete file transfer is acknowledged which can be observed in **Error! Reference source not found.** (Line 25 and 26).