Adapting TCP for the Bridge Architecture

Kashyap, Shruthi; Rao, Vijay; Venkatesha Prasad, Ranga Rao; Staring, Toine

DOI
10.1007/978-3-030-85836-0_5

Publication date
2021

Document Version
Final published version

Published in
SpringerBriefs in Applied Sciences and Technology

Chapter 5
Adapting TCP for the Bridge Architecture

Some experiments have been designed to understand what parameters of the TCP/IP protocol affect the performance of the cordless kitchen system. These parameters are recognized and adapted to the system appropriately in order to boost the performance of Internet applications.

5.1 Experimental Setup

The experimental setup consists of three Linux-based systems that behave as the cordless appliance, PTx and the end-user device. The Lightweight IP (LwIP) stack [1] is installed on these, where only the required layers of the stack are utilized, as shown in Fig. 3.3. An Ethernet connection is used between the PTx and the end-user device. An NFC communication channel is set up between the PTx and the cordless appliance. The block diagram of the NFC module used in this experiment is illustrated in Fig. 5.1 and the actual hardware setup is shown in Fig. 5.2. It consists of an NFC Reader/Writer (RW) device, an NFC Card Emulator (CE) device and micro-controllers (MCU) connected to each of them as shown in the figures. The NFC devices have the following characteristics.

- They operate at 13.56 MHz and use the ISO/IEC 14443-4 half-duplex transmission protocol.
- They support bit rates of 212 and 424 kbps.
- They are capable of transferring a chunk of 14 bytes (at 212 kbps) and 30 bytes (at 424 kbps) in one time slot of 1.5 ms. So the bandwidth in the NFC time-slotted mode would be 11.2 kbps (at 212 kbps) and 24 kbps (at 424 kbps).
- They require the data chunk to be available at least 2 ms before the occurrence of a communication time slot.
- The distance of about 3 cm is used between the NFC RW and CE devices (see Fig. 5.3).

S. Kashyap et al., *Cook Over IP*,
SpringerBriefs in Applied Sciences and Technology,
https://doi.org/10.1007/978-3-030-85836-0_5

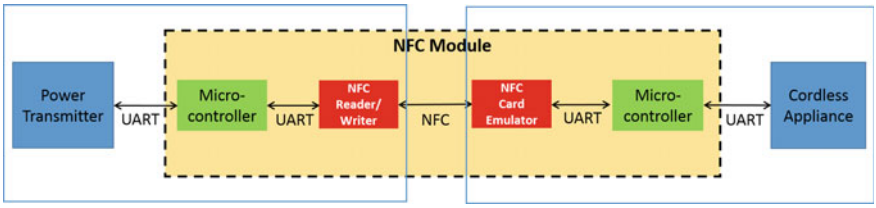


Fig. 5.1 Block diagram of the NFC module

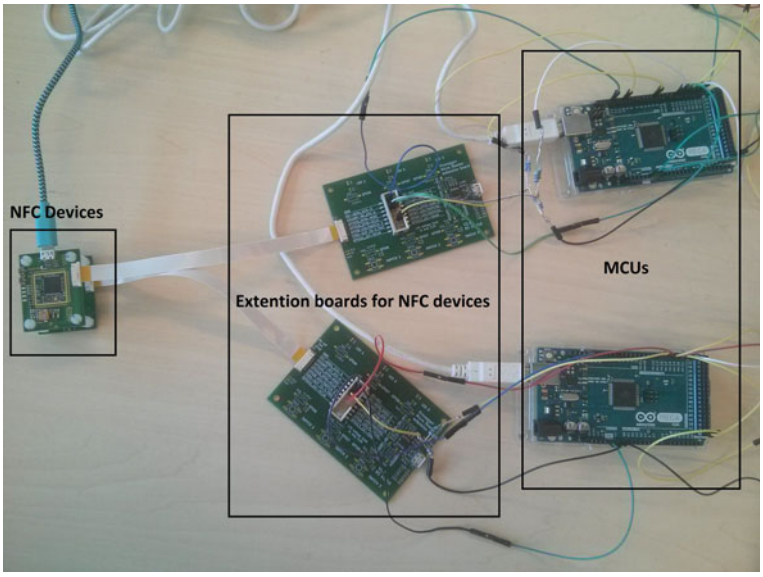


Fig. 5.2 NFC module used for the experiments

- In a kitchen scenario, one may not place the appliance exactly on top of a PTx always. The WPC standard allows a leeway of up to 10 cm and hence requires that no bit errors occur up to this radius from the center of the PTx. Therefore, error correction techniques are not used. The NFC RW device terminates the connection with an NFC CE device when bit errors are detected with the assumption that the appliance is in an unsafe position.

The MCUs used in the module are responsible for the fragmentation of the incoming packets from the TCP/IP stack. The defragmentation is done in the PTx and appliance stacks. The MCUs are also responsible for synchronizing the data transfer with the NFC communication time slots. A serial communication (UART) is used between the MCUs and the NFC devices. According to the cordless kitchen specification, the PTx needs to behave as the NFC RW device and the appliance as the CE device, so the connections are made by interfacing the PTx and the appliance to the appropriate MCUs using UART communication. The MCUs have an incoming packet buffer of

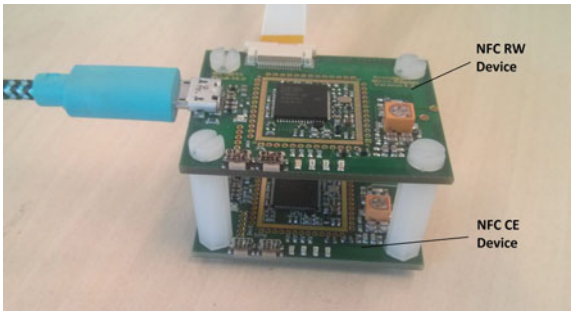


Fig. 5.3 NFC RW and CE modules used in the experiments

Table 5.1 LwIP stack configuration used in the experiments

Configuration type	Value
Protocol Version	IPV4
TCP Maximum Segment Size (MSS)	1024 bytes
Initial Contention Window (CWND) size	4096 bytes
Send buffer size	4096 bytes
Maximum CWND size	8096 bytes
For every ACK received	CWND increases by 1024 bytes
TCP Retransmission Timeout (RTO)	1 s
TCP timer period	500 ms
TCP fast timer period	250 ms

2 kB. The Maximum Transmission Unit (MTU) over the Ethernet channel is 1500 bytes. So a packet buffer size of 2 kB is chosen considering the overheads from the UI protocol and packet processing. They have an interrupt-driven UART reception, and they store and process only one packet at a time.

The proprietary DICOMM UI protocol is used between the end-user device and the appliance. The TCP server and client applications are run on these two devices to exchange data using the UI protocol. The TCP/IP stack configuration used for the experiments is listed in Table 5.1. Table 5.2 summarizes the communication overhead in the cordless kitchen. In the experiments, the cordless appliance is assigned with an IP address of 192.168.1.102, and the end-user device with 192.168.1.202, as shown in Fig. 5.4. Note that for ease of implementation, the PTx is also given the same IP address of 192.168.1.102 as that of the appliance. Wireshark packet analyzer tool is used over the Ethernet link. The packets over the NFC channel cannot be captured by this tool, so logs from the NFC and the TCP/IP stacks are also used for analysis.

Table 5.2 Communication overhead in the cordless kitchen system

Overhead type	Size (Bytes)
IPV4	20
TCP	20
UI protocol	8
Packet handling	8
NFC protocol	4 per time slot
Total	**56 + (4 * No. of time slots per packet)**

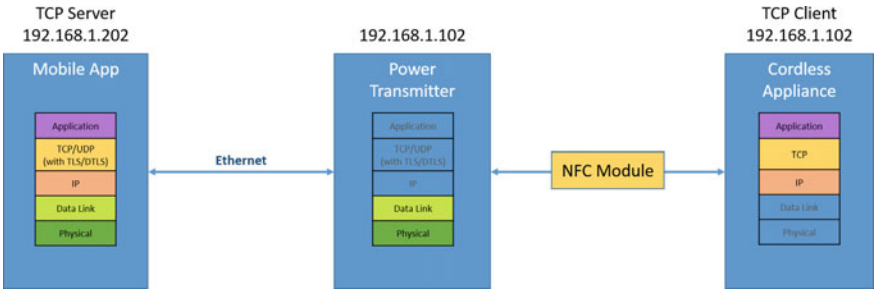


Fig. 5.4 TCP client as the cordless appliance and TCP server as the mobile app

5.2 Challenges in Adapting TCP

TCP is a transport layer protocol which is responsible for ensuring reliable transmission of data across Internet-connected networks. It is called a connection-oriented protocol as it establishes a virtual connection between two hosts using a series of request and reply messages. It divides the messages or files to be transmitted into segments that are encapsulated into the body of the IP packets. Upon reaching the destination, these segments are reassembled to form the complete message or file. TCP defines a parameter known as Maximum Segment Size (MSS) which represents the maximum payload size a TCP segment can hold excluding the TCP header. It is basically the application data size that can be sent in a single TCP/IP packet. TCP executes a three-way handshake sequence for connection establishment between two hosts, as shown in Fig. 5.5. During the handshake, the hosts agree upon the MSS value that will be used during the data transfer. Once the hosts finish exchanging data, the TCP session will be terminated using the connection termination procedure.

While the connection is established and the data transfer is in progress, TCP uses several mechanisms such as congestion control and flow control to provide a reliable connection. The congestion control includes the slow start, congestion avoidance, fast retransmit and fast recovery mechanisms [2]. TCP maintains a retransmission timer to detect and retransmit lost segments. Each time a segment is sent, TCP starts the retransmission timer which begins at a predetermined value called Retransmission

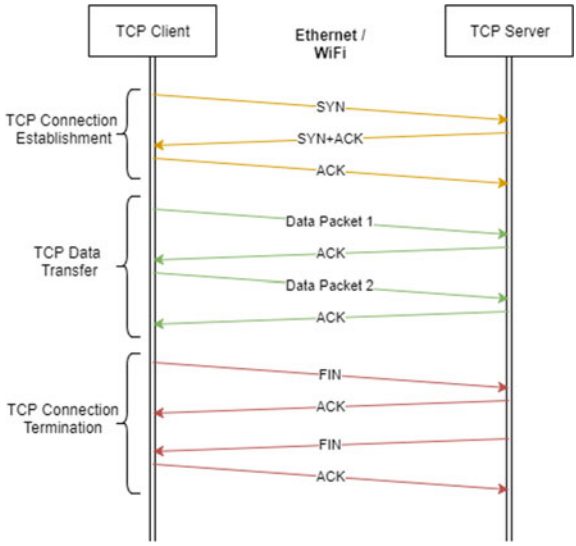


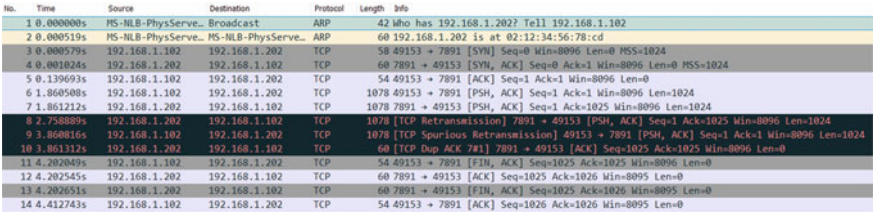
Fig. 5.5 TCP connection establishment, data transfer and connection termination procedures

Timeout (RTO) and counts down over time. If this timer expires before an acknowledgment is received for the segment, TCP retransmits the segment assuming that the packet is lost. The RTO value for segments is set dynamically by measuring the Round Trip Time (RTT) of the previous segments. This helps in setting appropriate RTO values by understanding the current delay on the channel.

The flow control determines the rate at which data is transmitted between the sender and receiver in a TCP session. TCP uses a sliding window mechanism for flow control. Due to the limited buffer space, the sender and receiver maintain a congestion window (CWND) and receive window which represent the amount of unacknowledged data that can be in transit at any given time. (Note: Please refer to [2] for a detailed explanation on the working of the TCP/IP protocol).

In this book, the TCP MSS, RTO, RTT and CWND parameters are considered while adapting TCP/IP for the slotted NFC channel as they are the fundamental factors that affect the performance of the cordless kitchen. This chapter mainly concentrates on adapting the TCP RTO and RTT parameters to the given system. The effects of TCP MSS and CWND sizes on performance are discussed in Chap. 7.

To analyze the performance of tunneling TCP/IP over the time-slotted NFC channel, a TCP session is established over NFC at a bit rate of 11.2 kbps and an initial TCP RTO value of 1 s. The PTx and appliance are configured to run the TCP server (192.168.1.202) and client (192.168.1.102) applications, respectively. A payload size equal to TCP MSS of 1024 bytes is exchanged in the session, which generates an NFC payload size of 1080 bytes, including all the overheads mentioned in Table 5.2. The result obtained is depicted in Fig. 5.6. It shows the output from the Wireshark tool taken over the Ethernet link. The packets over the NFC channel are not visible in the capture.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000519s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000579s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.001024s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.139693s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	1.860508s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	1.861212s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	2.758889s	192.168.1.202	192.168.1.102	TCP	1078	[TCP Retransmission] 7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
9	3.860816s	192.168.1.102	192.168.1.202	TCP	1078	[TCP Spurious Retransmission] 49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
10	3.861312s	192.168.1.202	192.168.1.102	TCP	60	[TCP Dup ACK 7#1] 7891 → 49153 [ACK] Seq=1025 Ack=1025 Win=8096 Len=0
11	4.202049s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
12	4.202545s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
13	4.202651s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
14	4.412743s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.6 TCP session with a data exchange of 1080 bytes at 11.2 kbps

In Fig. 5.6, it can be noticed that there are some retransmitted and duplicate ACK (Dup ACK) packets in the TCP session (highlighted in black). The Dup ACKs are transmitted when the receiver sees a gap in the sequence number of received packets. The logs from the TCP/IP stacks show that there are two retransmissions from the appliance, and one retransmission from the PTx followed by a Dup ACK sent in response to the retransmission from the appliance. The presence of such retransmissions has a large impact on the latency of the TCP session. This is because the latency of the system is already in the order of seconds due to the constrained bandwidth of the NFC channel, and transmitting these extra packets would increase the latency even further, impacting the end-user experience. It is therefore important to eliminate these retransmissions by identifying the cause of their occurrence. The subsequent sections discuss in detail the classification of these retransmissions and their elimination techniques.

5.2.1 TCP Spurious Retransmissions

The packets 8 and 9 in Fig. 5.6 are spurious retransmissions from the client and server stacks, respectively. Spurious retransmissions occur when the sender thinks that its packet is lost and sends it again, even though the receiver sent an acknowledgment for it. This happens when the sender experiences a timeout before the ACK is received due to the TCP RTO value being very small compared to that of the packet RTT. Figure 5.7 depicts a case where a spurious retransmission problem occurs. Here, the appliance stack does not wait long enough to receive the ACK from the end-user device, which leads to a series of unnecessary transmissions.

To confirm if some or all of these are spurious retransmissions, the experiment is repeated with smaller NFC payload sizes. Table 5.3 gives an overview of the number of retransmissions and Dup ACKs observed for different payload sizes at a bit rate of 11.2 kbps. It can be noticed that as the data size decreases, the number of retransmissions also decreases. If these are spurious retransmissions, this behavior makes sense because smaller data sizes will have smaller RTT. So the chances of the RTO timer of 1 s getting triggered will be less which would result in fewer or no spurious retransmissions. At 11.2 kbps, the RTT of a 500-byte packet is about 1.1 s,

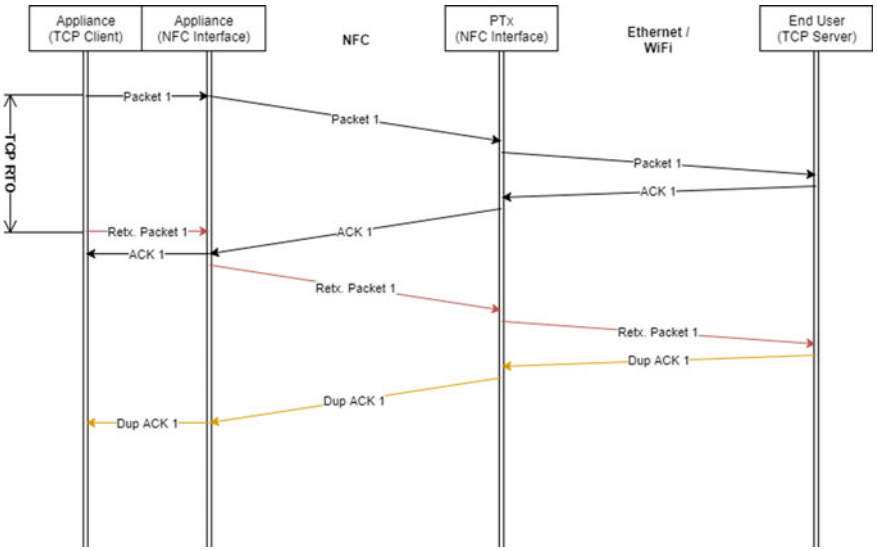


Fig. 5.7 Spurious retransmissions in a TCP session

Table 5.3 Number of retransmissions and Dup ACKs in TCP sessions for varying payload sizes at 11.2 kbps

Payload on NFC (Bytes)	Appliance		PTx	
	Retxs.	Dup ACKs	Retxs.	Dup ACKs
250	1	0	0	0
500	1	0	1	0
1000	2	0	1	1
1080	2	0	1	1

resulting in a total of two retransmissions, and the RTT of a 250-byte packet is about 0.6 s, which results in only a single retransmission.

The experiment is repeated at an NFC bit rate of 24 kbps for a payload size of 1080 bytes. The result is depicted in Fig. 5.8. It can be seen that there is one retransmission from both PTx and appliance, and two Dup ACKs only from the appliance stack. At higher bit rates, the RTT of the packets over NFC will be even less. This would further reduce the number of spurious retransmissions. Table 5.4 summarizes the results for different NFC payload sizes exchanged in the TCP session at 24 kbps. It can be observed that fewer retransmissions are observed compared to that in 11.2 kbps.

These experiments confirm that the TCP RTO value is very small for the given system which makes the stacks timeout sooner than the expected arrival time of the acknowledgment, leading to spurious retransmissions. To overcome this, the TCP packet size could be reduced such that its RTT will be less than the RTO value that is set by default. For this, however, the TCP MSS value will have to be reduced, which

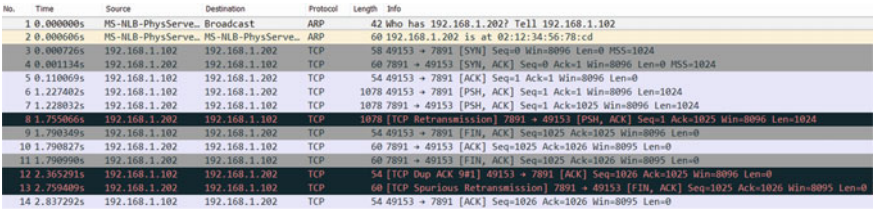


Fig. 5.8 TCP session with a data exchange of 1080 bytes at 24 kbps

Table 5.4 Number of retransmissions and Dup ACKs in TCP sessions for varying payload sizes at 24 kbps

Payload on NFC (Bytes)	Appliance		PTx	
	Retxs.	DUP ACKs	Retxs.	DUP ACKs
250	1	0	0	0
500	1	0	0	0
1000	1	0	0	0
1080	1	2	1	0

would lower the goodput of the system. Therefore, it makes more sense to adjust the RTO value appropriately to suit the system.

The TCP/IP stack updates the RTO for its packets dynamically by constantly measuring the RTT of its data packets. The authors of [3, 4] propose methods of avoiding spurious retransmissions in wireless networks that have high delay variability by injecting delays into the network. These delays increase the RTT of the packets and hence the calculated TCP RTO values. Leung et al. [5] present another technique to increase the TCP RTO value by increasing the mean deviation of the measured packet RTT. Although these solutions promise to reduce spurious retransmissions, they will not be very useful in the cordless kitchen system because TCP timeout occurs for the first data packet of the TCP session, for which the RTT measurement has not been made yet. The stack therefore ends up using the initial TCP RTO that is set at compile time, for this packet. Moreover, the TCP sessions in this system can be short, so there will not be enough time to adapt to the dynamically calculated RTO values. Furthermore, this system uses low bandwidth and large delay channel; it is therefore necessary to remove the retransmissions as much as possible, right from the beginning of the TCP session, to ensure a good end-user experience.

To avoid spurious retransmissions, the initial TCP RTO needs to be greater than the RTT of the maximum packet size traveling through the NFC channel. This value gets automatically updated after TCP starts making RTT measurements. It is recommended to set the RTO slightly higher than the RTT of the data packet. This guarantees that there are no spurious retransmissions and also ensures quick retransmission in case of packet loss. Figure 5.9 shows a scenario where the appliance stack waits sufficiently to receive an ACK for the transmitted data packet. It can be seen

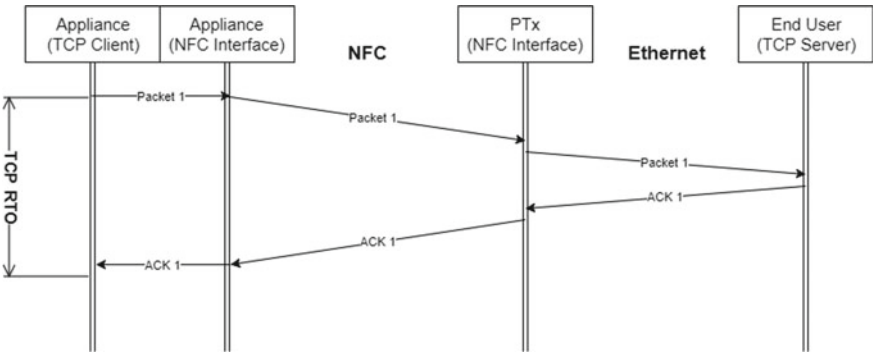


Fig. 5.9 Spurious retransmissions solved by increasing the initial RTO value

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000513s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000630s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.001072s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.139762s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	5.872002s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	5.872726s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	7.126788s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
9	7.127341s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
10	7.127436s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
11	7.337531s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.10 TCP session with a data exchange of 1080 bytes at 11.2 kbps with initial RTO of 5 s

that eliminating the retransmissions and Dup ACKs saves a lot of time by reducing the number of packets transmitted over the constrained NFC channel, which in turn improves the overall responsiveness of the system.

In the presence of so many retransmissions, it is difficult to estimate and generalize the exact RTT of the packets. Therefore, initially a high RTO value greater than the total TCP session duration is chosen so that all the spurious retransmissions are eliminated, which would make the analysis of the packet RTT easier. At an NFC bit rate of 11.2 kbps, the average TCP session duration with 1080 bytes of data exchange is about 4.56 s, so an initial RTO of 5 s is chosen for both the client and server stacks to make sure that TCP does not timeout before the first data acknowledgment is received.

Figure 5.10 shows the result after updating the initial TCP RTO to 5 s. No spurious retransmissions are observed in the TCP session, and both server and client stacks wait sufficiently to receive an acknowledgment. However, there is one retransmission at the appliance as indicated by the stack logs. It can be seen that the time difference between packets 5 and 6 is about 5 s, which is equal to the RTO set. This implies that packet 6 is a retransmitted packet. The fact that this was not removed by setting a high RTO suggests that this is not a spurious retransmission. Further analysis on this will be discussed in Sect. 5.2.2.

Similar results are obtained at other data sizes (refer Table 5.5), where one retransmission exists from the appliance stack. The experiment is repeated at 24 kbps by

Table 5.5 Number of retransmissions and Dup ACKs in TCP sessions for varying payload sizes at 11.2 kbps with an initial TCP RTO of 5 s

Payload on NFC (Bytes)	Appliance		PTx	
	Retxs.	DUP ACKs	Retxs.	DUP ACKs
250	1	0	0	0
500	1	0	0	0
1000	1	0	0	0
1080	1	0	0	0

setting an initial RTO of 3 s, as the average TCP session duration is around 2.45 s. It is again observed that although all spurious retransmissions are removed, one retransmission from the appliance stack still exists.

It is very important to set an optimum TCP RTO value for every packet to avoid spurious retransmissions. For this, it is necessary to understand the channel TCP is dealing with. Using RTT of the previous packets cannot be the only factor that should be considered for estimating the RTO for the data packets. The estimation needs to be done by analyzing the following parameters as well.

- NFC bit rate being used;
- Speed/bandwidth of the channel between PTx and the end-user device;
- Total packet size, as the RTT depends on the size of the packet.

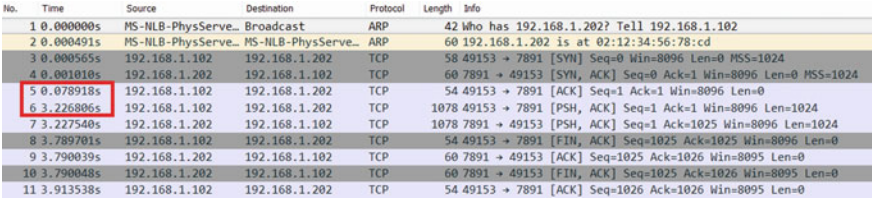
Note: For the initial TCP RTO, the RTT of the maximum possible packet size that can be transferred over the NFC channel should be used.

Generalizing the RTT estimation procedure would be more precise when the TCP session is free from all kinds of retransmissions and Dup ACKs. Therefore, it is necessary to first eliminate the remaining retransmissions before proceeding to a generalized approach for setting an optimum TCP RTO, which is explained in Sect. 5.3.2.1.

5.2.2 Packet Drops Due to Small Inter-Packet Delay

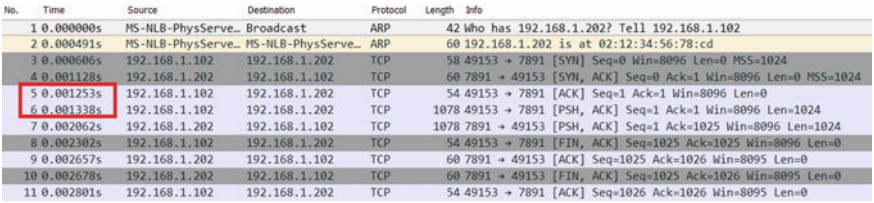
In Fig. 5.10, although the spurious retransmissions and duplicate ACKs are removed, the total time of the TCP connection has considerably increased to about 7.4 s compared to the one with an initial RTO of 1 s, which was 4.56 s on an average. This sudden increase takes place between packets 5 and 6 (highlighted in Fig. 5.10). The time difference of about 5 s between these packets, which is equal to the initial RTO set, suggests that packet 6 is a retransmitted packet from the appliance.

The result of the experiment at a bit rate of 24 kbps with an exchange of 1080 bytes of NFC payload and an initial RTO of 3 s is represented in Fig. 5.11. Again,



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000491s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000565s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.001010s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.078918s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	3.226806s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	3.227540s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	3.789701s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
9	3.790039s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
10	3.790048s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
11	3.913538s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.11 TCP session with a data exchange of 1080 bytes at 24 kbps with initial RTO of 3 s



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000491s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000606s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.001128s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.001253s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	0.001338s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	0.002062s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	0.002302s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
9	0.002657s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
10	0.002678s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
11	0.002801s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.12 TCP session with a data exchange of 1080 bytes without the NFC channel

the time difference of about 3 s between packets 5 and 6, equal to the initial RTO, suggests that packet 6 has been retransmitted by the appliance stack, just like the previous case. The NFC interface and appliance stack logs reveal that the first data packet (packet 6) which was sent right after packet 5 was dropped at the interface by the NFC module. This is the reason that it cannot be seen on the Wireshark capture.

To understand why the packet was dropped, it is important to study the time delay between TCP/IP packets exchanged between two devices in normal situations, i.e. without the NFC channel. This would give an idea of what the ideal delay between packets 5 and 6 should have been. Figure 5.12 shows the packet capture taken between two devices connected via Ethernet. The TCP client is at 192.168.1.102 and the TCP server is at 192.168.1.202.

The average delay between packets 5 and 6, representing the ACK of the TCP handshake and the first data packet, respectively, is around 50.6 µs. This means that when the NFC channel is being used, the TCP stack generates packet 6 50.6 µs after packet 5 and pushes it to the NFC channel. This inter-packet delay between consecutive packets would be too small for the NFC channel as it is half-duplex and can only transmit packets one at a time. Moreover, the NFC module used in this setup can store and process only a single packet at a time. It discards all the packets that it receives while it is transmitting. In this case, packet 5 takes around 69.04 ms to travel through the NFC channel at 11.2 kbps and around 39.76 ms at 24 kbps. So when the appliance stack sends packet 6 only 50.6 µs after sending packet 5, the NFC module discards it as it will be busy transmitting packet 5. Figure 5.13 summarizes this problem. It shows how a small inter-packet delay between two consecutive packets causes packet loss, which impacts the overall latency of the TCP session.

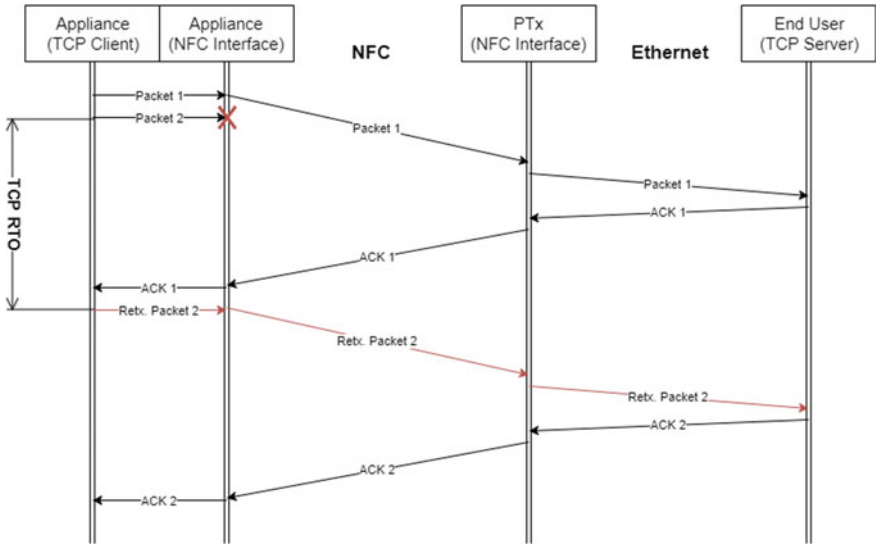


Fig. 5.13 Packet drop at the NFC interface due to small inter-packet delay

5.3 Addressing the Challenges

Both the outlined problems are due to the low data rate, time-slotted NFC channel. The packet drops at the NFC interface causes retransmissions, and spurious retransmissions can cause further packet drops at the NFC interfaces. We break this tie by first solving the packet drops issue, and then address the spurious retransmissions problem.

5.3.1 Avoiding Packet Drops Due to Small Inter-Packet Delay

To avoid packet drops at the NFC interface caused due to small inter-packet delay between consecutive packets, there must be a way for the stack to sense the NFC channel before sending packets to it. An NFC channel sensing mechanism is implemented where the NFC channel notifies the stack when it is busy or free. The stack keeps track of this and sends the packets only when the channel is free. By implementing this mechanism on both the ends of the NFC channel, i.e. in the NFC-appliance and the NFC-PTx interfaces, it can be ensured that packet drops are not caused due to sending the packets too soon into the channel. This way the processing speed of the TCP/IP stack can be brought down to match the speed of the NFC channel so that they function in sync. Figure 5.14 shows how the packet drop problem is solved by implementing the NFC channel sensing mechanism.

Figures 5.15 and 5.16 show the result after implementing the mechanism at 11.2 kbps and 24 kbps, respectively. The TCP sessions are free from retransmis-

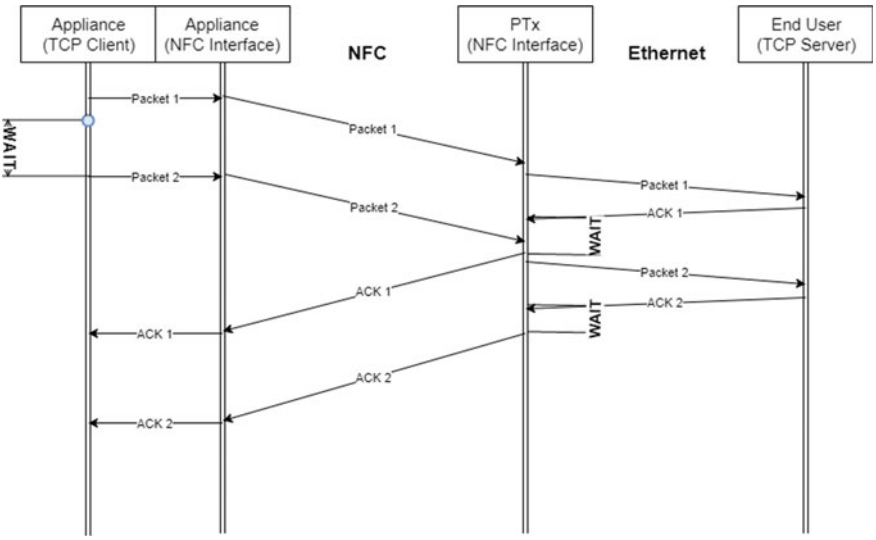


Fig. 5.14 NFC channel sensing mechanism

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000403s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000449s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.000803s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.139703s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	1.325267s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	1.325804s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	2.581868s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
9	2.582253s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
10	2.582361s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
11	2.792432s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.15 TCP session with a data exchange of 1080 bytes with NFC channel sensing mechanism at 11.2 kbps and initial RTO of 5 s

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000s	MS-NLB-PhysServe…	Broadcast	ARP	42	Who has 192.168.1.202? Tell 192.168.1.102
2	0.000493s	MS-NLB-PhysServe…	MS-NLB-PhysServe…	ARP	60	192.168.1.202 is at 02:12:34:56:78:cd
3	0.000536s	192.168.1.102	192.168.1.202	TCP	58	49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
4	0.000853s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=8096 Len=0 MSS=1024
5	0.079878s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
6	0.603941s	192.168.1.102	192.168.1.202	TCP	1078	49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
7	0.604630s	192.168.1.202	192.168.1.102	TCP	1078	7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
8	1.165876s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
9	1.166360s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
10	1.166529s	192.168.1.202	192.168.1.102	TCP	60	7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
11	1.286891s	192.168.1.102	192.168.1.202	TCP	54	49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0

Fig. 5.16 TCP session with a data exchange of 1080 bytes with NFC channel sensing mechanism at 24 kbps and initial RTO of 3 s

sions and Dup ACKs which results in the reduction of latency. With a payload size of 1080 bytes, the TCP session latency is about 2.87 s at 11.2 kbps, which was 4.56 s before solving the retransmission problems. At 24 kbps the latency is around 1.33 s which was initially 2.45 s.

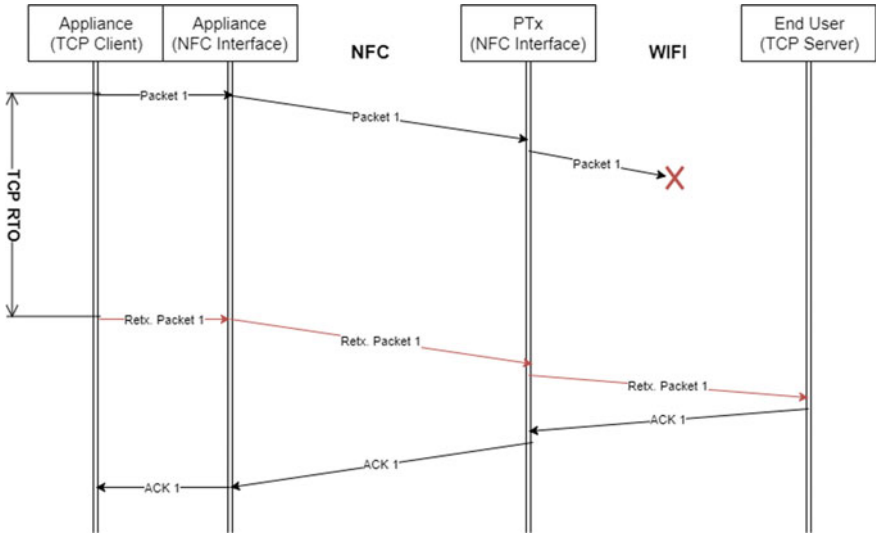


Fig. 5.17 TCP session with a very large TCP RTO

5.3.2 Avoiding TCP Spurious Retransmissions

5.3.2.1 Generalized Approach for TCP RTO Estimation

Setting a high initial TCP RTO will avoid spurious retransmissions for sure, however, it may also delay the retransmission when a packet is really lost. Figure 5.17 shows how the latency of the TCP session increases when a large initial TCP RTO is set and when packet loss occurs. It is therefore recommended to set the RTO slightly higher (at least one timer period) than the RTT of the data packet. This is because when the TCP stacks have coarse timers, there will be a tendency of timing out around one timer period sooner than what is estimated.

Now that all the retransmissions are removed, the RTT of the TCP packets can be estimated by analyzing the transmission conditions of the system in detail. The TCP/IP packet from the appliance travels through the NFC and Ethernet/Wi-Fi channels before reaching the end-user device. So the packet RTT can be broadly defined as

$$RTT = RTT_{NFC} + RTT_{WiFi} \text{ (ms)} \tag{5.1}$$

where RTT is the total packet round trip time, RTT_{NFC} is the round trip time over the NFC channel and RTT_{WiFi} is the round trip time over the Wi-Fi channel.

The initial RTO set at compile time for standard wireless (or Ethernet) channels should be used as RTT_{WiFi}. Paxson et al. [6] recommend a minimum value of 1 s as

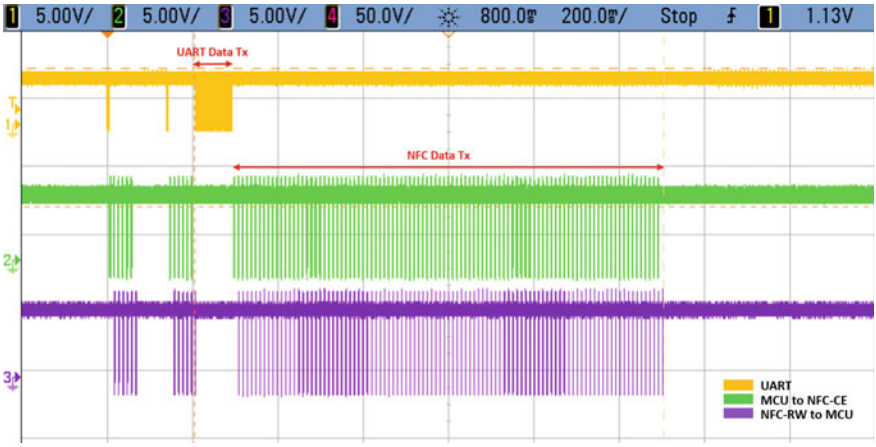


Fig. 5.18 TCP session capture in the direction from the appliance through the NFC-CE and NFC-RW modules at 11.2 kbps

the TCP RTO for wireless channels. The measured RTT of the previous packet can later be used to vary this value dynamically, as explained in the next section.

The RTT_{NFC} is the critical component which consumes most of the time. When the appliance stack transmits a packet, it first travels over the UART channel to reach the NFC module, as shown in Fig. 5.1. The NFC module then fragments the packet into chunks and transmits it over the NFC channel to the PTx stack. Figure 5.18 shows an oscilloscope output of a TCP session with 1080 bytes of data exchange at 11.2 kbps, captured between the ends of the NFC module. It depicts the packet transmissions in the direction from the appliance through the NFC-CE and NFC-RW modules. Different signals seen in the capture are explained below.

1. Yellow signal: it represents the transmission of data packets from the appliance stack to the MCU over UART.
2. Green signal: it represents the transmission of data chunks from the MCU to the NFC-CE module over UART.
3. Purple signal: it represents the transmission of data chunks from the NFC-RW module to the MCU over UART.

The time to transmit a fragmented 1080-byte data packet over the NFC channel is depicted as NFC Data Tx in Fig. 5.18. This value is equal to the theoretical time to transmit the data over the NFC channel, unless some time slots are missed in between. The UART Data Tx in the figure depicts the time needed to transfer the data packet from the appliance to the NFC module. This value adds to the packet processing time.

Apart from these, it is also important to take the waiting time for a time slot into account. From Sect. 5.1, it is clear that the packet chunks need to be present in the NFC module at least 2 ms before the arrival of the time slot. When the TCP/IP stack sends a packet, the packet can arrive at the NFC module at any point between two

time slots. So the maximum amount of time a packet would need to wait for a time slot would be 12 ms.

When the chunks are received at the other end of the NFC channel, they are transmitted over the UART to the PTx stack. This transmission will be done in parallel to the transmission on the NFC channel, so they do not add to the RTT of the packet. However, the transmission of the last chunk needs to be taken into account. Considering all these, the RTT_{NFC} will be

$$RTT_{NFC} = 2 * (t_{UART} + t_{maxslotwait} + t_{NFC} + t_{UARTchunk}) \text{ (ms)} \tag{5.2}$$

where t_{UART} is the packet transmission time over UART. It depends on the baud of the UART being used. $t_{maxslotwait}$ is taken as 12 ms, as explained above. t_{NFC} is the theoretical transmission time over the slotted NFC channel, and $t_{UARTchunk}$ is the transmission time of the last chunk over UART.

The t_{UART} is given by the following equation:

$$t_{UART} = \frac{size_{pckt}}{baud_{UART}} \text{ (ms)} \tag{5.3}$$

where $size_{pckt}$ is the total packet size sent to the NFC module in bytes and $baud_{UART}$ is the baud of the UART in bytes per millisecond.

The t_{NFC} is given by the following equation:

$$t_{NFC} = slots_{pckt} * 10 \text{ (ms)} \tag{5.4}$$

$$slots_{pckt} = \frac{size_{pckt}}{size_{chunk}} \tag{5.5}$$

where $slots_{pckt}$ is the number of time slots needed to transmit the packet. $size_{chunk}$ is the size of the payload section of the NFC protocol (in bytes). This depends or varies with the bit rate of the NFC being used.

The $t_{UARTchunk}$ is given by the following equation:

$$t_{UARTchunk} = \frac{size_{chunk}}{baud_{UART}} \text{ (ms)} \tag{5.6}$$

The initial RTO to be set must be greater than the maximum packet size that is transmitted through the NFC channel. In this experiment, the TCP MSS is set as 1024 bytes, which gives a maximum packet size of 1080 bytes. The total RTT for this packet size is estimated using Eq. 5.1, and it is found to be 3373.24 ms. As the timer period of the LwIP stack is 500 ms, this RTT value needs to be rounded up to the nearest 500 ms. This results in an optimum initial RTO of 3500 ms (3.5 s) for an NFC bit rate of 11.2 kbps. For the bit rate of 24 kbps, the optimum initial RTO is found to be 2.5 s.

5.3.2.2 New Algorithm for Dynamic TCP RTO Estimation

TCP in LwIP calculates the RTO after measuring the RTT of the data packets using Van Jacobson's (VJ) RTT estimation algorithm [7]. VJ's algorithm uses the Smoothed RTT (SRTT) calculation for RTO prediction. It measures the RTT value of the data packets to estimate the RTO of the next packet to be sent. Therefore, the RTO which is assigned to a packet is based on the RTT of the previous packet, which is done irrespective of the packet size.

Consider situations where the TCP sessions are long and the initial TCP RTO is set to 3.5 s at compile time (NFC bit rate of 11.2 kbps). For example, if a user chooses to cook step by step by creating their own recipe instead of uploading a recipe in one go, the TCP session would last very long and it could comprise several TCP messages with randomly varying sizes. If the application sends very small data packets of less than 10 bytes for a long time, TCP would adjust the RTO to a smaller value of about 1 s. Now, if the application suddenly sends very large packets, like recipes greater than 1 kB, an RTO of 1 s would be too small. This would result in spurious retransmissions until TCP adjusts the RTO according to the new packet sizes. On the contrary, if the application sends very small data packets right after sending large packets, the RTO of the small packets would be large initially until it is gradually adjusted to an appropriate value. In the meanwhile, if one of these packets gets lost, the system would take longer to timeout resulting in delayed retransmission (see Fig. 5.17). This would increase the overall latency of the system.

Figure 5.19 shows the TCP stream diagram of the client stack in a long session with 68 data packets of varying sizes. Points with the same sequence number denote retransmissions. It can be noticed that every time a large packet (denoted by a large jump in sequence number and/or time) is sent after a series of small packets, spurious retransmissions occur. This is because TCP would have adjusted the RTO suitable for small packets, and when large packets are suddenly sent this RTO would become too small considering the RTT of large packets. In the diagram, this is represented by packets having the same sequence number being sent more than once at different times. There are eight spurious retransmissions and eight Dup ACKs resulting in a total session duration of 22.58 s. It is very important to eliminate these retransmissions because it increases the latency of the TCP session in the order of seconds, due to the constrained nature of the NFC channel.

To mitigate this, a new algorithm is introduced that sets the TCP RTO depending on the estimated RTT of the current packet to be sent, instead of completely relying on the RTT estimation of the previous data packet. This approach has been designed by taking the following problems into account.

1. When the RTT estimation is made before sending the packet, the delay variability of the channels should also be considered. In VJ's RTT estimation algorithm, the RTO is adapted to the changing delay of the channel. If this mechanism is removed, then the stack will always assume a constant delay which may affect the latency by either causing spurious retransmissions or delaying retransmissions. Therefore, the new algorithm must take the delay variability into account.

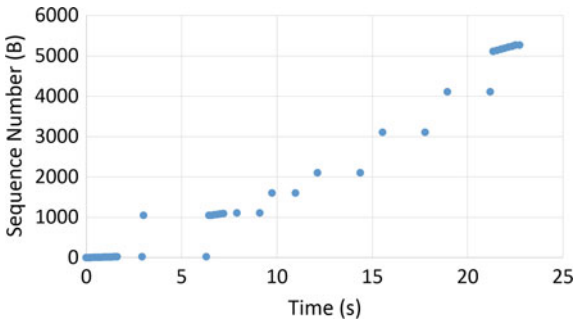


Fig. 5.19 Long TCP session with VJ's algorithm for setting the TCP RTO

Table 5.6 Problems due to delayed ACK and/or Nagle's algorithm

Packet size Sent/Received	Large	Small
Large	(Good) No spurious retx.	(Not bad) No spurious retx. but packet loss leads to delayed retx.
Small	(Bad) Spurious retx.	(Good) No spurious retx.

2. The Wi-Fi and the NFC channels could have variable delays. The NFC channel in the cordless kitchen would be used to send non-TCP/IP messages such as power control messages every now and then. This would affect the RTT of the TCP/IP messages and could delay their delivery. If the delay of the channel increases over time, it is difficult to identify if this increase is on the NFC channel or on the Wi-Fi channel. If the delay decreases from the theoretical value, it will be due to the reduced delay only on the Wi-Fi channel because the RTT on NFC will not go below the theoretical value (maximum reduction can be 10 ms, i.e. packet gets a time slot as soon as it arrives). So the RTO must be updated by closely observing the changing channel delay.
3. The RTT will be estimated considering the packet transmission time in both directions. The receiver may not always send back a packet of the same size. If only an ACK is received, the estimation will be larger than anticipated. But if the receiver replies with a bigger packet, for example, the delayed ACK algorithm does not send an ACK immediately, it waits for $<= 500$ ms [8] to check if the application has any further data to send so that it can piggyback the ACK with the next data packet. Another example is Nagle's algorithm which combines smaller packets to form a full-sized packet. In these cases, the estimated RTO will be smaller than the estimated value, which will lead to spurious retransmissions. Table 5.6 summarizes this problem. To solve this, the delayed ACK algorithm can be modified such that the stack sends an ACK for the received packet immediately, if the response packet is bigger than the received packet. This would avoid unnecessary spurious retransmissions.

4. If there is a packet loss, then that packet needs to be retransmitted using exponential backoff, where the RTO is doubled every time the same packet is retransmitted. For this, the estimated RTT of the packet needs to be used with the back-off multiplier.

Algorithm 1 New RTO estimation algorithm

RTT_p: Theoretical RTT of the previous packet
RTT_{meas_p}: Measured RTT of the previous packet
RTO_c: RTO of the current packet
RTT_{N_c}: RTT_{NFC} of the current packet
RTT_{W_c}: RTT_{WiFi} of the current packet
r: Factor r
expBackoff(): computes binary exponential backoff based on retransmit count

procedure
    $r \leftarrow 1$ // Initialize r to 1
    while Packet queue is not empty **do**
        $RTT_{N\_c} \leftarrow$ Calculate theoretical RTT_{NFC} using Eq. 5.2
        $RTT_{W\_c} \leftarrow$ Use recommended initial RTO
        $RTT_p \leftarrow RTT_{N\_c} + RTT_{W\_c}$ // Store theoretical RTT to calculate r
        if $r >= 1$ **then**
            $RTT_{N\_c} \leftarrow r * RTT_{N\_c}$
            $RTT_{W\_c} \leftarrow r * RTT_{W\_c}$
        else
            $RTT_{W\_c} \leftarrow max(1000, r * RTT_{W\_c})$
        $RTO_c \leftarrow \lceil (RTT_{N\_c} + RTT_{W\_c})/500 \rceil * 500$ //Round-up to the next 500 ms
        if Retransmission = true **then**
            $RTO_c \leftarrow RTO_c *$ expBackoff() // Backoff procedure
        $RTT_{meas\_p} \leftarrow$ Measure and update RTT of the packet transmitted
        $r \leftarrow RTT_{meas\_p}/RTT_p$ // Compute r

Based on this analysis, the new algorithm is designed to dynamically estimate the optimum packet RTO value. The working of this algorithm is discussed in detail below.

- The theoretical RTO is calculated for each packet before its transmission, using Eq. 5.1. RTT_{WiFi} is set according to the initial RTO recommended for Wi-Fi (or Ethernet) channels. Furthermore, a minimum RTO value of 1 s is maintained for RTT_{WiFi} as recommended by [7]. The RTT_{NFC} is calculated as explained previously, using Eq. 5.2.
  (Note: The LwIP stack uses an initial RTO of 3 s. However, as the experiments are carried out on an Ethernet channel with <1 ms delay, an initial RTO of 1 s is used in the experiments.)
- The RTT of each data packet transmitted is dynamically measured to estimate the current delay in the NFC and Wi-Fi (or Ethernet) channels. The delay is estimated by comparing the theoretical RTT of the previous packet with the measured RTT

of the previous packet. The factor (r) by which the measured value varies from the theoretical value is calculated.

$$r = \frac{RTT_{measured\,Prev}}{RTT_{Prev}} \tag{5.7}$$

where r is the ratio of measured RTT to theoretical RTT of a packet, $RTT_{measured\,Prev}$ is measured RTT of the previous packet and RTT_{Prev} is the RTT of the previous packet calculated using Eq. 5.1.

- When the factor $r \geq 1$, the theoretical values of both RTT_{NFC} and RTT_{WiFi} are scaled up by this value. If r < 1, then only RTT_{WiFi} is scaled down. As explained earlier, this is due to the fact that the RTT of a packet over the NFC channel cannot go lower than its theoretical value. A minimum value of 1 s is maintained for RTT_{WiFi} as recommended by [7]. The new RTT is calculated with these scaled values using Eq. 5.1. For better estimation of the delay, a window of recent values of r can be maintained and the highest value in the window can be used for the current RTT estimation. The window size should be chosen depending on the type of applications being supported and the rate of packet transmission.

$$RTT_{NFC} := r * RTT_{NFC}\ (\text{ms})\ if\ r \geq 1 \tag{5.8}$$

$$RTT_{WiFi} := max(1000, r * RTT_{WiFi})\ (\text{ms}) \forall\ r \tag{5.9}$$

- The LwIP stack has a timer period of 500 ms to check for retransmission timeout. The RTO is therefore calculated as a multiple of 500 ms. So in the new RTO estimation algorithm, the estimated RTO of the current packet is rounded up to the nearest 500 ms.
- In case of packet loss, the exponential backoff algorithm is used with the estimated RTO of the lost packet. Using the estimated RTO for the backoff procedure would be more accurate than using the RTO of the most recently sent packet.
- To solve the spurious retransmission problem described in Table 5.6, the delayed ACK algorithm is modified such that an empty ACK will be sent if the size of the received packet is less than the size of the packet to be transmitted. A drawback of this solution is that the stack would send ACK packets even if the received packet size is slightly smaller than the packet to be sent. The RTO values are rounded up to the nearest 500 ms, so the packets of similar sizes may (but not necessarily) have the same RTO value. In this case, it would be unnecessary to send an extra ACK packet which could increase the latency of the system. It would be safe to use the modified algorithm even though it may not give the best result in the case discussed above.

Algorithm 5.1 summarizes the procedure for RTO estimation. It is tested on the TCP session shown in Fig. 5.19. The same experimental setup with the Ethernet channel is used for testing. Without the modification in the delayed ACK algorithm, a latency of 15.96 s is achieved as shown in Fig. 5.20, which is 6.62 s less than that with

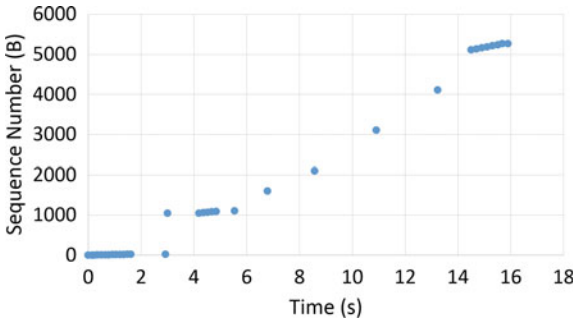


Fig. 5.20 Long TCP session with new algorithm for RTO estimation

the original algorithm. This gives a 29.32% reduction in the latency in this example. However, there is still one spurious retransmission and one Dup ACK caused due to the delayed ACK algorithm. When the modified delayed ACK algorithm is used, all of the retransmissions are removed but the overall latency will be 16.1 s, which is slightly higher than the previous case. This is due to the fact that the stack sends out an ACK even when the received packet is sightly smaller than the packet to be sent. Note that the percentage improvement in the case of the new RTO estimation algorithm solely depends on the data set that is in consideration. It varies with different data sets.

References

1. lwIP - A Lightweight TCP/IP stack - Summary [Savannah]. (n.d.). Savannah. https://savannah.nongnu.org/projects/lwip/. Accessed 6 June 2021
2. *TCP/IP Illustrated* (3 Volume Set) by W.R. Stevens, G.R. Wright (2001) Hardcover. (2021). Addison-Wesley Professional
3. T. Klein, K. Leung, R. Parkinson, L.G. Samuel, Avoiding spurious TCP timeouts in wireless networks by delay injection. IEEE Global Telecommunications Conference, 2004. GLOBECOM '04., 5, 2754-2759 vol.5 (2004)
4. G.I. Fotiadis, V. Siris, Improving TCP throughput in 802.11 WLANs with high delay variability, in *2005 2nd International Symposium on Wireless Communication Systems* (2005), pp. 555–559
5. K. Leung, T. Klein, C. Mooney, M. Haner, Methods to improve TCP throughput in wireless networks with high delay variability [3G network example], in *IEEE 60th Vehicular Technology Conference, 2004*. VTC2004-Fall, vol. 4 (2004), pp. 3015–3019
6. V. Paxson, M. Allman, H.K. Chu, M. Sargent, Computing TCP's Retransmission Timer. RFC **6298**, 1–11 (2011)
7. V. Jacobson, Congestion avoidance and control. SIGCOMM (1988)
8. M. Allman, V. Paxson, E. Blanton, TCP Congestion Control. RFC **5681**, 1–18 (2009)