Cook over IP: Adapting TCP for Cordless Kitchen Appliances

Shruthi Kashyap*§, Vijay S. Rao*, R. Venkatesha Prasad*, Toine Staring§
*Embedded Software, Delft University of Technology, The Netherlands
§Philips Research, Eindhoven, The Netherlands
Email: shruthi.kashyap@gmail.com, {V.Rao, R.R.VenkateshaPrasad}@tudelft.nl, toine.staring@philips.com

***Abstract*—Cordless kitchens are the next big step in Smart Kitchens that are enabled by the Internet of Things (IoT) paradigm. The appliances in a cordless kitchen are powered by inductive power sources (PTx) that are integrated into kitchen counter-tops. The appliance and the PTx exchange control information using a near-field communication (NFC) channel.**

These appliances currently do not have Internet connectivity to enable smart cooking and control of the appliance from smartphones. Embedding a WiFi radio powered by batteries on the appliance is undesirable as batteries require recharging or replacement, and also increase the cost of the appliance. Therefore, we propose to connect the PTx to Internet and exploit the NFC channel for tunneling Internet traffic to the appliances.

Due to the heavy magnetic fields induced by the PTx, this NFC channel has to be time-slotted, which is unique to the cordless kitchen appliances. This introduces many challenges on the communication, as the low data rates and high latencies of the NFC channel are aggravated by the slotting of the NFC channel. We focus on the TCP protocol as it is the most widely used transport protocol on the Internet. The performance of TCP is severely affected due to the time-slotted NFC channel. We identify two major problems that occur when TCP/IP is tunneled over the time-slotted NFC channel, namely spurious retransmissions of the TCP packets and packet drops at the NFC interface. Since most of the TCP/IP sessions in this environment are short, relying on TCP's natural course to adapt to long delays is not viable.

To solve these, we propose a method to determine optimal TCP retransmission timeout values, and a channel sensing mechanism to avoid packet drops. In addition, we perform a detailed analysis to study the influence of parameters such as the contention window size, maximum segment size and NFC bit error rates. We implement and evaluate the solutions on a cordless kitchen testbed. We find that the proposed solutions almost completely eliminate the spurious retransmissions and packet drops. Furthermore, we achieve up to 53 % lower end-to-end latency at 24 kbps in the NFC time-slotted mode.

1. Introduction

With the emergence of the Internet of Things (IoT) technologies, the concept of *Smart Kitchen* [1] is being developed. This concept has brought a wave of intelligent and connected devices that has transformed the way we cook and interact with our kitchen appliances, such as enabling the appliances to be controlled from smartphones and cooking by uploading recipes from a remote location. These cater to the busy lifestyles of the current generation. An imminent technological development in the smart kitchen domain is the concept of 'Cordless Kitchen' [2]. In this concept, introduced by Wireless Power Consortium (WPC) [3], the appliances do not need power cords or batteries to operate. Instead, they are powered by inductive power sources that may be built into a kitchen counter, cooktop, or a table. The appliance needs to be simply placed on top of a power transmitter (PTx) and the user should be able to cook, interact and control the appliance remotely.

As shown in Figure 1, the PTx transfers power via electromagnetic induction to another coil placed in the appliance. The power is then converted back into electrical energy and/or heat for cooking within the appliance. Figure 2 shows the power coils and NFC antennas in the PTx and appliance. Unlike the traditional kitchen appliances, the cordless kitchen appliances are made intelligent. They communicate with the PTx to ensure that the amount of power received remains within the limits of the appliance and according to the input from the user. The communication between the appliance and the PTx takes place using an NFC channel. This makes cooking much more precise, responsive and repeatable with the cordless appliances.

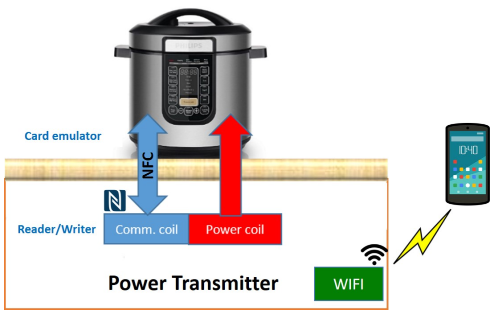
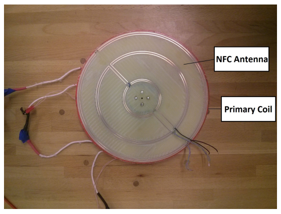
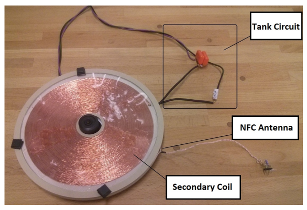


Figure 1: The cordless kitchen concept.

An important requirement of the cordless appliances is that the users must be able to upload recipes, software updates and control them from a smartphone. Currently, these



(a)



(b)

Figure 2: Power coils and NFC antennas. (a) PTx with primary coil and NFC antenna, and (b) Appliance with secondary coil and NFC antenna.

appliances lack Internet connectivity, and this paper attempts to solve this issue. A straightforward way of providing Internet to the appliances would be to install a WiFi module. Using batteries in the appliances is undesirable as batteries require recharging or replacement, and will also increase the cost of the appliances. Furthermore, the appliances need not support connectivity when they are away from the PTx. An alternative is to power the WiFi module by PTx. However, when the PTx goes into the standby mode, the appliance will be switched off. Therefore the WiFi module will not be awake at all times leading to loss of messages and eventually, connectivity. Another alternative is to use the existing NFC channel, which we propose to exploit in this paper. Here, the PTx is connected to the Internet (maybe through WiFi/Ethernet), and tunnels the packets to the appliance through the NFC channel. When a message arrives onto the PTx in the standby mode, it can power up the appliance, if placed on the top of the PTx, and establish communications. This approach makes the appliance cost effective too.

Currently, the maximum data rate supported by the NFC channel is 848 kbps. However due to practical constraints, the NFC channel needs to be time-slotted and must operate at a 15% duty-cycle (see Section 2), which results in low data rates on the NFC channel. Transmission Control Protocol (TCP) over Internet Protocol (IP), the most widely used protocol on the Internet, is heavy as it is optimized for exchanging large amounts of data at much higher data rates. Therefore, tunneling TCP/IP over the constrained NFC channel will have reduced performance including increased latency. While cooking applications are firm and soft real-time and missing deadlines may not be hazardous, a high latency affects the cooking procedure, quality of the food and also the user-experience. Although TCP is designed to adapt to the channel delay and the available bandwidth, the adaptation does not work as intended in the cordless kitchen scenario. This is because the number of B exchanged in our scenario is quite low (1 to 5 kB), which is not enough for the adaptation to take place. Motivated by this, we focus on adapting the TCP/IP protocol to the time-multiplexed NFC channel in order to achieve an acceptable performance of TCP.

Two major problems arise while using TCP in the cordless kitchen system. (a) Spurious retransmissions of packets: the delay on the NFC channel causes TCP to timeout even though packets are not lost; and (b) packet drops at the NFC interface due to the processing speed mismatch between the TCP/IP stack and the NFC module. We propose solutions to these problems, and further analyze various parameters that influence the communication. To the best of our knowledge, this is the first work that deals with these challenges and provides implementable solutions with a good performance. Specifically, our contributions are as follows:

1) We demonstrate the challenges posed by the high delay and unique time-slotted NFC channel. Further, to eliminate spurious retransmissions, we provide a generalized solution to calculate optimal retransmission timeout (RTO) values depending on the packet size and the data rate of the NFC channel.
2) Although TCP is designed to adapt the RTO over time by estimating the delay on the channel, it does not consider the payload sizes in this estimation. This leads to choosing an incorrect RTO value for this system, resulting in spurious retransmissions when the TCP payload size varies over time. To avoid this, we propose a new algorithm for dynamic RTO estimation considering the channel delays. This algorithm ensures that optimum RTO values are set for each packet such that spurious retransmissions are eliminated, and delayed retransmissions are prevented in case of packet loss.
3) We propose an NFC channel sensing mechanism which slows down the TCP stack to match the transmission speed of the NFC channel, thereby achieving an optimum inter-packet delay.
4) We also do a parametric analysis of other factors that affect the performance such as the TCP contention window (CWND) size, maximum segment size (MSS), NFC BERs, non-TCP/IP messages.
5) We implement our solutions on an experimental cordless kitchen setup and evaluate the solutions with respect to a standard TCP implementation.

The rest of the paper is organized as follows. Section 2 gives an overview of the NFC interface. Section 3 explains the Internet connectivity architecture and the experimental setup used. Section 4 describes the challenges in adapting the TCP protocol and the proposed solutions. Section 5 provides the evaluation of our solutions. An analysis of various parameters is presented in Section 6. An overview of related work is discussed in Section 7. Section 8 provides concluding remarks.

2. Overview of the NFC Interface

Similar to the inductive power transfer, NFC technology is also based on the concept of electromagnetic induction. As the PTx generates very high magnetic fields that can disrupt the NFC communications, WPC [3] has proposed a solution where the wireless power transfer and the NFC communications operate in a time-multiplexed fashion as shown in Figure 3. The NFC communications take place only at the zero crossings of the power signal for a duration of T_{zero} = 1.5 ms. The power transfer takes place in the remaining time (8.5 ms).

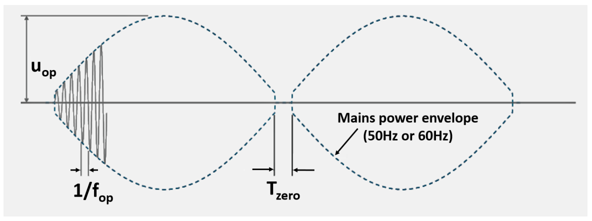


Figure 3: Time-multiplexing of the power and NFC signals

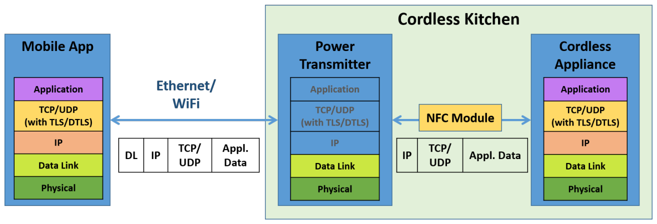


Figure 4: The bridge architecture for Internet connectivity

The NFC communications mostly comply with the NFC Forum specifications, operating at bit rates of 106, 212, 424, and 848 kbps. These rates reduce to 15% in the time-slotted mode. Table 1 shows the number of B that can be read using different NFC bit rates in the time-slotted mode. Two new NFC read and write commands have been introduced to reduce the communications overhead. The commands follow the ISO/IEC 14443 half-duplex transmission protocol. The new write command supports similar payload sizes. These commands carry messages containing measurement data, operating limits, control data and auxiliary data for Internet connectivity.

TABLE 1: Payload size in the new NFC read command

Bit rate (kbps)	Num. of B in payload
106	5
212	19
424	48
848	104

3. System Overview

3.1. Internet Connectivity Architecture

We propose an architecture to enable Internet connectivity to cordless kitchen appliances wherein the PTx has wired or wireless Internet connectivity and TCP/IP packets are tunneled over the NFC channel to the appliances. We call this the *bridge architecture* as the PTx acts like a network bridge by processing only the data link and physical layers for the appliance, as shown in Figure 4. This architecture enables the appliance to be an IoT device with its own stack. In this architecture, the load on the NFC channel increases due to the communications overhead introduced by the TCP/IP headers, TCP handshake, acknowledgment and retransmission mechanisms.

3.2. Experimental Setup

We perform experiments in a test setup of the cordless kitchen. The experimental setup consists of three Linux based systems that behave as a cordless appliance, PTx and end-user device. The block diagram of the experimental setup is illustrated in Figure 5. The Lightweight IP (LwIP) stack [4] is installed on all devices, where only the required layers of the stack are utilized. An Ethernet connection is used between the PTx and the end-user device. An NFC channel is setup between the PTx and the cordless appliance. As the PTx operates in the NFC Reader/Writer (RW) mode and the appliance operates in the NFC Card Emulator (CE) mode, these are also created in the setup with NFC RW and CE devices. Figure 6 shows the test setup used in the experiments.

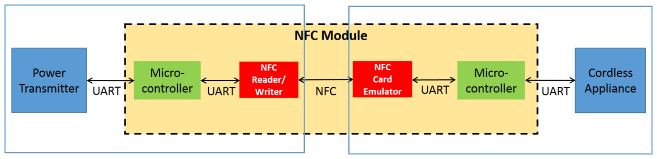


Figure 5: Block diagram of our test setup

Our NFC devices support bit rates of 212 kbps and 424 kbps. They are capable of transferring a chunk of 14 B (at 212 kbps) and 30 B (at 424 kbps) in one communication time slot of 1.5 ms, which occurs every 10 ms. So the effective data rate in the time-slotted mode would be 11.2 kbps (at 212 kbps) and 24 kbps (at 424 kbps). The modules require the data chunk to be available at least 2 ms before the start of a time-slot.

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.

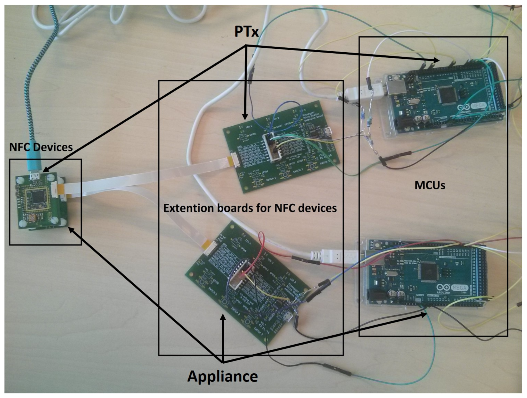


Figure 6: Our test setup of the cordless kitchen

The MCUs used in the module are responsible for fragmentation of the incoming packets from the TCP/IP stacks. The reassembly is done in the respective stacks. The MCUs are also responsible for synchronizing the data transfer with the communication time-slots. They have an incoming packet buffer of 2 kB. Serial communication links (UART) are used between the MCUs and the NFC devices at 115200 baud. Both MCUs have an interrupt driven UART reception, and store and process only one packet at a time.

A proprietary UI protocol is used between the end-user device and the appliance. The TCP server and client applications are run on these to exchange data using the UI protocol. Table 2 summarizes the communication overhead in the system. In the experiments, the appliance is assigned with an IP address of 192.168.1.102, and the end-user device with 192.168.1.202.

TABLE 2: Communication overheads

Overhead type	**Size (Bytes)**
IPv4	20
TCP	20
UI protocol	8
Packet handling	8
NFC protocol	4 per time-slot
Total	**56 + (4 * Num. of time-slots per packet)**

4. Adapting TCP to the NFC Channel

TCP is a heavy protocol for the NFC channels with low data rates, which are further lowered by the time-slots in the cordless kitchen scenario. In this section, we present two major problems that prevent a standard TCP implementation from being used directly in the cordless kitchen scenario, and then we present our solutions.

4.1. Challenges

4.1.1. TCP spurious retransmissions. Spurious retransmissions occur when the sender experiences a timeout before the ACK is received, due to the RTO value being small compared to that of the packet Round Trip Time (RTT). The presence of such retransmissions has a large impact on the TCP session latency. This is because the latency of the system is already high due to the constrained NFC channel, and transmitting these extra packets would increase the latency even further. It is therefore important to eliminate these retransmissions by identifying the cause of their occurrence.

To find this cause, a TCP session is established over the channel at an NFC bit rate of 11.2 kbps, a TCP MSS of 1024 B and an RTO value of 1 s. A payload size equal to the MSS is exchanged in the session, which generates an NFC payload size of 1080 B, including all the overheads mentioned in Table 2. Figure 7(a) shows the output from the Wireshark tool taken over the Ethernet link. Note that the packets over the NFC channel are not visible in the capture. This figure shows two spurious retransmissions (packets 8 and 9) and duplicate acknowledgement (Dup ACK) sent in response to the retransmission from the appliance.

Table 3 shows the average 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. Smaller data sizes will have a smaller RTT, so the chances of the RTO timer of 1 s getting triggered will be lower, which would result in fewer or no spurious retransmissions. At 11.2 kbps, the RTT of a 500 B packet is about 1.1 s, resulting in a total of two retransmissions, and the RTT of a 250 B packet is about 0.6 s, which results in only a single retransmission. Results obtained at 24 kbps show that fewer retransmissions are observed compared to a bit rate of 11.2 kbps. This is because at higher bit rates the RTT of packets over NFC will be even lower.

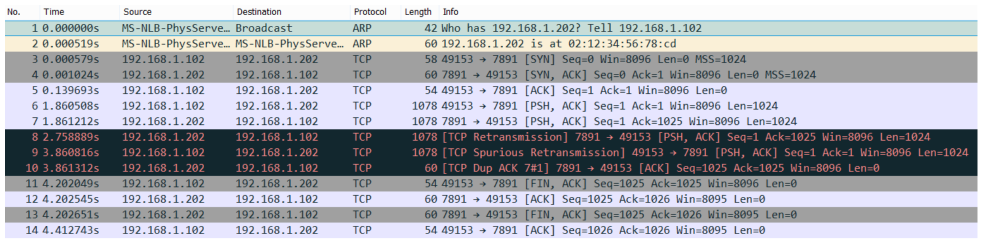
TABLE 3: Number of retransmissions at 11.2 kbps

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

These experiments confirm that the RTO value is too low for the given system, which results in spurious retransmissions. To eliminate these, the RTO values of the packets need to be set appropriately. The TCP/IP stack updates the RTO value for its packets dynamically by constantly measuring the RTT of its data packets. However, the timeout occurs for the very first data packet of the session, for which an RTT measurement has not been made yet. The stack therefore ends up using the initial RTO value set at compile time. The TCP sessions in the cordless kitchen can be short, so there will not be enough time to adapt to TCP's RTO estimation. As this system uses a low data rate and high-delay channel, it is necessary to remove the retransmissions right from the start of the session to ensure a good end-user experience.

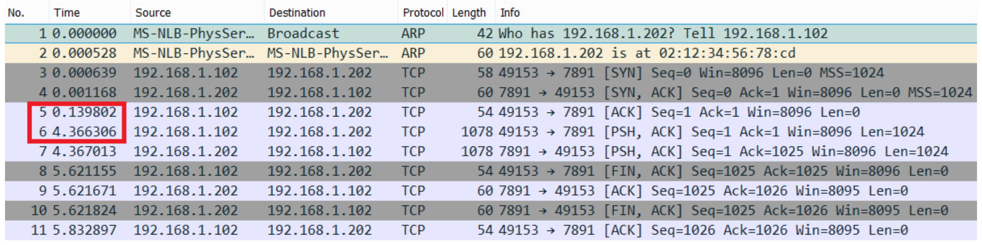
4.1.2. Packet drops due to the small inter-packet delay. In Figure 7(b), spurious retransmissions and duplicate ACKs have been removed by setting an initial RTO value of 3.5 s (see Section 4.2.2). However, there is still one retransmission at the appliance, as indicated by the stack logs. The total time of the TCP connection increased to about 5.9 s compared to the one with an initial RTO value of 1 s, which was about 4.56 s. This unexpected increase takes place between packets 5 and 6 (highlighted in Figure 7(b)). The time difference of about 4 s between these packets suggests that packet 6 is a retransmitted packet from the appliance. The result of the experiment at 24 kbps with an exchange of 1080 B of data and an initial RTO value of 2.5 s (see Section 4.2.2) also shows a time difference of about 2.5 s between the packets 5 and 6. This again suggests that packet 6 has been retransmitted by the appliance stack, just like the previous case. Similar results are observed for other data sizes.

The average delay between packets 5 and 6, representing the ACK of the TCP handshake and the first data packet



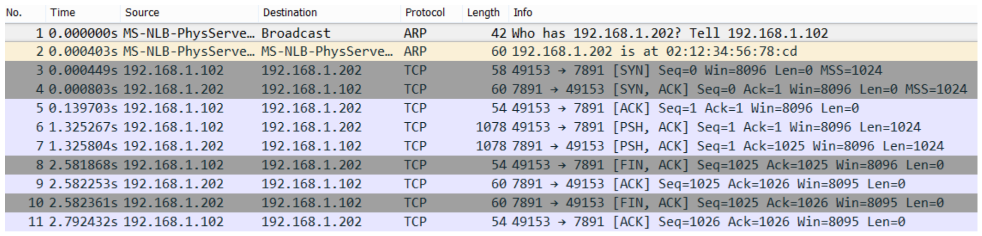
```
No. Time       Source             Destination        Protocol Length Info
  1 0.000000s  MS-NLB-PhysServe…  Broadcast          ARP        42 Who has 192.168.1.202? Tell 192.168.1.102
  2 0.000519s  MS-NLB-PhysServe…  MS-NLB-PhysServe…  ARP        60 192.168.1.202 is at 02:12:34:56:78:cd
  3 0.000579s  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
```

(a)



```
No. Time      Source            Destination       Protocol Length Info
  1 0.000000  MS-NLB-PhysSer…   Broadcast         ARP        42 Who has 192.168.1.202? Tell 192.168.1.102
  2 0.000528  MS-NLB-PhysSer…   MS-NLB-PhysSer…   ARP        60 192.168.1.202 is at 02:12:34:56:78:cd
  3 0.000639  192.168.1.102     192.168.1.202     TCP        58 49153 → 7891 [SYN] Seq=0 Win=8096 Len=0 MSS=1024
  4 0.001168  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 0.139802  192.168.1.102     192.168.1.202     TCP        54 49153 → 7891 [ACK] Seq=1 Ack=1 Win=8096 Len=0
  6 4.366306  192.168.1.102     192.168.1.202     TCP      1078 49153 → 7891 [PSH, ACK] Seq=1 Ack=1 Win=8096 Len=1024
  7 4.367013  192.168.1.202     192.168.1.102     TCP      1078 7891 → 49153 [PSH, ACK] Seq=1 Ack=1025 Win=8096 Len=1024
  8 5.621155  192.168.1.102     192.168.1.202     TCP        54 49153 → 7891 [FIN, ACK] Seq=1025 Ack=1025 Win=8096 Len=0
  9 5.621671  192.168.1.202     192.168.1.102     TCP        60 7891 → 49153 [ACK] Seq=1025 Ack=1026 Win=8095 Len=0
 10 5.621824  192.168.1.202     192.168.1.102     TCP        60 7891 → 49153 [FIN, ACK] Seq=1025 Ack=1026 Win=8095 Len=0
 11 5.832897  192.168.1.102     192.168.1.202     TCP        54 49153 → 7891 [ACK] Seq=1026 Ack=1026 Win=8095 Len=0
```

(b)



```
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
```

(c)

Figure 7: Packet capture (a) indicating spurious retransmissions (packets 8 and 9) and Dup ACK (highlighted in black), (b) showing data exchange of 1080 B with an initial RTO value of 3.5 s at 11.2 kbps wherein a retransmission is observed, and (c) showing data exchange of 1080 B with NFC channel sensing mechanism and initial RTO value of 3.5 s at 11.2 kbps.

respectively, is around 50.6 μs in normal situations, i.e., without the NFC channel. This inter-packet delay between consecutive packets is too small for the NFC channel as it is half-duplex and the NFC module used in this setup can store and process only a single packet at a time. It discards all packets that it receives while 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.

4.2. Proposed Solutions

Both these 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.

4.2.1. Avoiding packet drops. To avoid packet drops, we implement an NFC channel sensing mechanism. The link layers on the appliance and PTx keep track of the channel busy status and sends packets only when the channel is free. By implementing this mechanism at both the ends of the NFC channel, i.e., in the appliance and in the PTx interfaces, packet drops can be avoided.

Figure 7(c) shows the result after implementing the mechanism. The TCP session is free from retransmissions and Dup ACK packets. This results in a reduction of the overall latency. With a payload size of 1080 B, 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 found to be 1.33 s, which was initially 2.45 s.

4.2.2. Avoiding spurious retransmissions - Generalized approach. To avoid spurious retransmissions, the initial TCP RTO value must be greater than the RTT of the maximum packet size traveling through the NFC channel. This guarantees that there are no spurious retransmissions and also ensures quick retransmission in case of packet loss. The RTO value is automatically updated after TCP starts making RTT measurements. The RTT estimation of the data packets must be done by considering the NFC bit rate being used, the bandwidth of the WiFi/Ethernet channels and the size of the packets that will be transmitted. A TCP/IP packet from the appliance travels through the NFC and Ethernet/WiFi channels before reaching the end-user device. So the packet RTT can be broadly defined as:

$$RTT = RTT_{\text{NFC}} + RTT_{\text{NW}}, \tag{1}$$

where, RTT is the total packet round trip time, RTT_{NFC} is the RTT over the NFC channel and RTT_{NW} is the RTT over the WiFi/Ethernet channel. The initial RTO value recommended for standard wireless (or Ethernet) channels should be used as RTT_{NW}. Authors of [5] recommend a minimum value of 1 s as the TCP RTO value 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.

RTT_{NFC} is the parameter that is significantly higher of the two RTTs. When the appliance stack transmits a packet, it first travels over the UART channel to reach the NFC module, as shown in Figure 5. The NFC module then fragments the packet into chunks and transmits them over the NFC channel to the PTx stack. RTT_{NFC} is given by the following equation,

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

t_{UART} is the packet transmission time over the UART. It is calculated as $size_{\text{pckt}}/baud_{\text{UART}}$, where $size_{\text{pckt}}$ is the total packet size sent to the NFC module and $baud_{\text{UART}}$ is the UART baud. As per Section 3.2, the chunks must be present in the NFC module at least 2 ms before the start of a time-slot. When the stack sends a packet, it can arrive at the NFC module at any point between two time-slots. So the maximum amount of time a packet needs to wait for a time-slot would be $t_{\text{maxslotwait}}$ = 12 ms.

t_{NFC} is the theoretical transmission time over the slotted NFC channel. It is given by $slots_{\text{pckt}} * 10$ ms (assuming a 50 Hz mains power region), where $slots_{\text{pckt}}$ is the number of time-slots needed to transmit the packet. The latter is calculated as $size_{\text{pckt}}/size_{\text{chunk}}$, where $size_{\text{chunk}}$ is the size of the payload section of the NFC protocol. It varies with the bit rate of the NFC being used. $t_{\text{UARTchunk}}$ is the time to transmit the last chunk to the PTx stack over the UART. It is represented as $size_{\text{chunk}}/baud_{\text{UART}}$.

The initial TCP RTO value must accommodate the maximum packet size that is transmitted through the NFC channel. In this experiment, the maximum packet size is 1080 B as the TCP MSS is set to 1024 B. The total RTT for this packet is estimated using Equation 1, and it is found to be 3373.24 ms. As the timer period of the LwIP stack is 500 ms, this value is rounded up to the nearest multiple of 500 ms. This results in optimum initial RTO values of 3.5 s for an NFC bit rate of 11.2 kbps and 2.5 s for 24 kbps. By using 3.5 s as the initial RTO value at 11.2 kbps, no spurious retransmissions are observed in the TCP session, and both server and client stacks wait sufficiently long to receive an acknowledgment (see Figure 7(b)). Similar results are observed at 24 kbps.

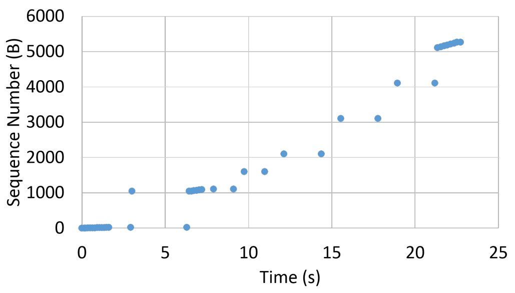
4.2.3. Avoiding spurious retransmissions - New algorithm for dynamic TCP RTO estimation.

TCP in the LwIP stack calculates the RTO values by measuring the RTT of the data packets using Van Jacobson's (VJ) RTT estimation algorithm [6]. VJ's algorithm uses the Smoothed RTT (SRTT) calculation method to predict RTO values. It measures the RTT of the data packets to estimate the RTO value of the next packet to be sent. Therefore, the RTO value which is assigned to a packet is based on the RTT of the previous packet, which is done irrespective of the packet size. So let us consider situations where the TCP sessions are long and have a high initial RTO value (e.g., 3.5 s). If an application sends small data packets of less than 10 B for a long time, VJ's algorithm would adjust the RTO value to a smaller value of about 1 s. Now, if the application suddenly sends large packets, like recipes greater than 1 kB, an RTO value of 1 s would be too small. This would result in spurious retransmissions until TCP adjusts the RTO value according to the new packet size. On the other hand, if the application sends very small packets right after sending large packets, the RTO values of the small packets would be large initially until it is gradually adjusted to an appropriate value. In the meantime, if one of these packets gets lost, the system would take longer to timeout resulting in delayed retransmission. These cases would increase the overall system latency.

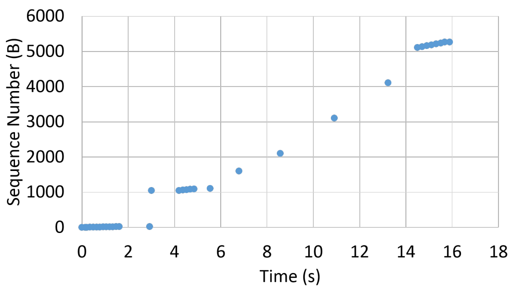
Figure 8(a) shows a TCP stream diagram of the client stack in a long session with 68 data packets of varying sizes. Points with the same sequence number denote retransmissions. It can be seen that every time a large packet (denoted by large jump in sequence number and/or time) is sent after a series of small packets, spurious retransmissions occur. This is because VJ's algorithm would have adjusted the RTO value for small packets. But when large packets are suddenly sent, this RTO value becomes too small considering the RTT of large packets. In Figure 8(a), 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 TCP session latency in the order of seconds, due to the constrained nature of the NFC channel. Here too a new algorithm is introduced that sets the RTO value 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 as follows.

The RTT estimation needs to consider the current channel delays. In this system, the WiFi/Ethernet and the NFC channels could have variable delays. The NFC channel in the cordless kitchen would be used to send non-TCP/IP messages related to power control, state transition, etc. every now and then. This would affect the RTT of the TCP/IP messages. If the delay of the combined channels increases over time, it is difficult to identify if the increase is on the NFC channel or on the WiFi channel. If the delay decreases below the initial value it will be due to a reduced delay only on the WiFi channel because the RTT on NFC channel will not go below the theoretical value (maximum reduction can be 10 ms when the packet gets a time slot as soon as it arrives).

Considering this, the theoretical RTO is calculated for



(a)



(b)

Figure 8: Long TCP session using (a) VJ's algorithm, and (b) new RTO estimation algorithm

each packet before its transmission, using Equation 1. RTT_{NW} is set according to the initial RTO value recommended for WiFi/Ethernet channels. RTT_{NFC} is calculated using Equation 2. (Note: The LwIP stack uses an initial RTO value of 3 s. As we use an Ethernet channel with < 1 ms delay, an initial RTO value of 1 s is used in the experiments). The RTT of each data packet is dynamically measured to estimate the current channel delay. 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 deviates from the theoretical value is calculated. It is given by $RTT_{\text{measuredPrev}}/RTT_{\text{Prev}}$.

When the factor $r \geq 1$, the theoretical values of both RTT_{NFC} and RTT_{NW} are scaled up by this value. If $r < 1$, then only RTT_{NW} is scaled down. This is because 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_{NW} as recommended by [6]. The new RTT is calculated with these scaled values using Equation 1. To better estimate 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 estimate. The window size should be chosen depending on the type of the applications being supported and the rate of packet transmission. The LwIP stack has a timer period of 500 ms. So the estimated RTO value is calculated as a multiple of 500 ms. In case of packet loss, the exponential backoff algorithm is used with the estimated RTO value of the lost packet.

$$RTT_{\text{NFC}} := r * RTT_{\text{NFC}} \text{ if } r > 1 \tag{3}$$

$$RTT_{\text{NW}} := max(1000, r * RTT_{\text{NW}}) \quad \forall r \tag{4}$$

The RTT is estimated considering the time it takes to transmit the packet in both the directions. The receiver may not always send back a packet of the same size. If only an ACK is received, the estimate will be larger than anticipated. However, if the receiver replies with a bigger packet, for example, using delayed ACK or Nagle's algorithms, the estimated RTO value will be smaller than the actual value. This will lead to spurious retransmissions.

To solve this problem, 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. As the RTO values are rounded up, 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 above case.

Algorithm 1 New RTO estimation algorithm

```
1: expBackoff(): computes binary exponential backoff based on retransmit count
2:
3: procedure
4:     $r \leftarrow 1$ // Initialize r to 1
5:     while Packet queue is not empty do
6:         $RTT_{\text{NFC}} \leftarrow$ Calculate theoretical $RTT_{\text{NFC}}$ using Eq. 2
7:         $RTT_{\text{NW}} \leftarrow$ Use recommended initial RTO value
8:         $RTT \leftarrow RTT_{\text{NFC}} + RTT_{\text{NW}}$ // Store theoretical RTT to calculate $r$
9:         if $r \geq 1$ then
10:            $RTT_{\text{NFC}} \leftarrow r * RTT_{\text{NFC}}$
11:            $RTT_{\text{NW}} \leftarrow r * RTT_{\text{NW}}$
12:        else
13:            $RTT_{\text{NW}} \leftarrow max(1000, r * RTT_{\text{NW}})$
14:        end if
15:        $RTO \leftarrow \lceil (RTT_{\text{NFC}} + RTT_{\text{NW}})/500 \rceil * 500$ //Round-up to the next 500ms
16:        if Retransmission = true then
17:            $RTO \leftarrow RTO *$expBackoff() // Backoff procedure
18:        end if
19:        $RTT_{meas} \leftarrow$ Measure and update RTT of the packet transmitted
20:        $r \leftarrow RTT_{meas}/RTT$ // Compute r
21:    end while
22: end procedure
```

The new RTO estimation algorithm is presented in Algorithm 1. It is tested on the previous TCP session shown in Figure 8(a). The same experimental setup with the Ethernet channel is used for testing. Without the modification in the delayed ACK algorithm, a latency of around 15.96 s is achieved as shown in Figure 8(b); 6.62 s lower than 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, which are 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 in the previous case. Note that the percentage improvement of the new RTO estimation algorithm depends on the data set in consideration. It varies with different data sets.

5. Results

This section presents the results of the proposed solutions for different NFC bit rates and data sizes. The system performance is analyzed by measuring latency, throughput, number of retransmissions in the TCP sessions, NFC channel bandwidth utilization, etc. The results are averaged over 20 TCP sessions.

5.1. Packet retransmissions

Tables 4 and 5 show the number of retransmissions, DUP ACKs and keep-alive messages in the TCP session after using the mitigation techniques in Sections 4.2.1 and 4.2.2, at 11.2 kbps and 24 kbps respectively. The retransmitted packets are depicted by the symbol 'R', DUP ACKs by 'DA' and keep-alive packets by 'KA'. The experiments are carried out with TCP sessions exchanging single packet with NFC payload sizes of 250, 500, 1000 and 1080 B at 11.2 kbps and 24 kbps.

TABLE 4: Number of retransmissions at 11.2 kbps

NFC payload size (Bytes)	Retransmissions in TCP session			
	Original TCP/IP configuration	NFC channel sense (Sec. 4.2.1)	Optimum TCP RTO (Sec. 4.2.2)	NFC channel sense + Optimum TCP RTO
250	1R	1R + 1DA	1R	0R
500	2R	2R + 1DA + 2KA	1R	0R
1000	3R + 1DA	3R + 2DA + 2KA	1R	0R
1080	3R + 1DA	3R + 2DA + 2KA	1R	0R

TABLE 5: Number of retransmissions at 24 kbps

NFC payload size (Bytes)	Retransmissions in TCP session			
	Original TCP/IP configuration	NFC channel sense (Sec. 4.2.1)	Optimum TCP RTO (Sec. 4.2.2)	NFC Channel sense + Optimum TCP RTO
250	1R	0R	1R	0R
500	1R + 1DA	0R	1R	0R
1000	1R	2R + 1DA + 2KA	1R	0R
1080	3R + 1DA	2R + 1DA + 2KA	1R	0R

Tables 4 and 5 show that by using only technique of Section 4.2.2 the total number of retransmissions can be brought down to one. By using only technique of Section 4.2.1, the total number of packets increases compared to the respective original TCP sessions in most of the cases. However, when both these techniques are used together, all types of retransmissions, DUP ACKs and keep-alive packets are removed. Before concluding on the performance based on the number of packets in the TCP session, it is important to study the latency of the session (see Section 5.2).

Figure 9 depicts the RTO values estimated by the new algorithm (Section 4.2.3) in a long TCP session with randomly varying packet sizes. These values are compared with the ones estimated by VJ's algorithm and the packet RTT values obtained over an Ethernet channel with < 1 ms delay. The estimations are however, still carried out considering the WiFi channel with a minimum RTO of 1 s, which results in an offset of about 1 s between the measured RTT and estimated RTO values as seen in Figure 9. The new algorithm clearly gives a more accurate estimate of the RTO values compared to VJ's algorithm as it takes the packet sizes and bit rates of the channels into account, therefore avoiding all the spurious and delayed retransmission scenarios.

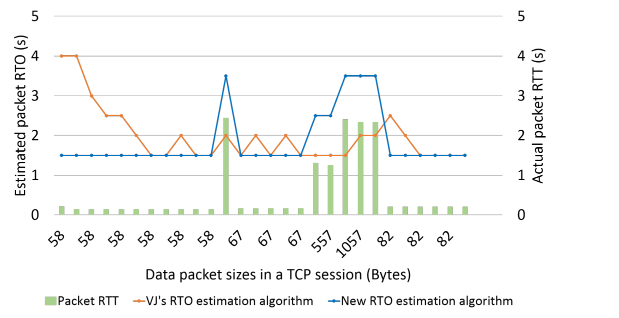
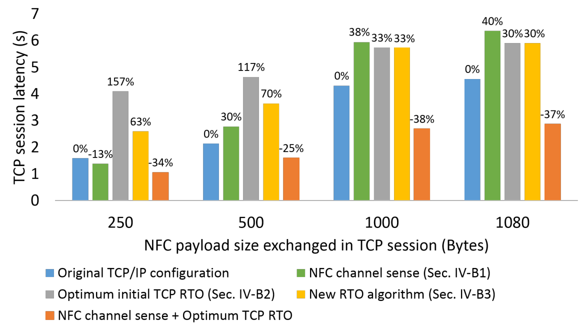
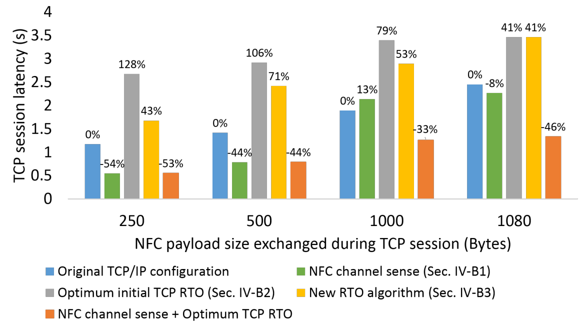


Figure 9: Comparison of packet RTO values with the new and VJ's RTO estimation algorithms



(a) Data rate 11.2 kbps



(b) Data rate 24 kbps

Figure 10: Latencies of TCP sessions

5.2. Latency

Reduction in the number of packets in a TCP session need not necessarily improve the latency of the session. This is because the time-delay between packet generation, especially in case of retransmitted packets, is also an important factor that affects the overall latency. Figures 10(a) and 10(b) show the graphs of latencies of TCP sessions with and without the retransmission mitigation techniques of Section 4.2 at 11.2 kbps and 24 kbps respectively.

The percentage by which the latencies increase or decrease using the mitigation techniques compared to the original latency is indicated in the graphs. At lower NFC bit rates, for example 11.2 kbps, the TCP session latency with only technique of Section 4.2.1 becomes higher than that with only of Section 4.2.2 when there are more extra packets resulting from spurious retransmissions. This is because even though the packet drops are prevented, there will be too many extra packets transmitted over a low bandwidth channel. On the contrary, at higher bit rates like 24 kbps, the latency with only technique of Section 4.2.1 will be lower than that with only technique of Section 4.2.2. This is because when only technique of Section 4.2.2 is used, the time-delay created by retransmission resulting from packet drop, will be more significant compared to the packet transmission time on a channel with relatively higher data rate. The TCP/IP stack has to wait for the timeout to realize that the packet has been dropped and resend it. This waiting time will be long compared to the time taken to transmit the extra packets. To achieve best results, it is recommended to use both the mitigation techniques together. Using both, up to 38% reduction in latency can be achieved at 11.2 kbps and up to 53% at 24 kbps.

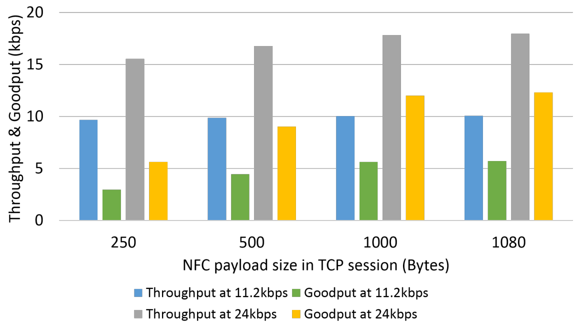


Figure 11: System throughput at 11.2 kbps and 24 kbps

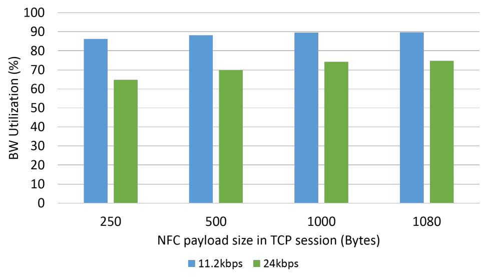


Figure 12: Bandwidth utilization at 11.2 kbps and 24 kbps

5.3. Throughput and goodput

The throughput of the system remains the same with or without the retransmission mitigation techniques of Section 4.2. It is known that the techniques are used to reduce the latency, however, the reduction in latency is achieved by reducing the number of packets or B traveling through the channel. Therefore the throughput, which is the number of B transferred per unit time, will be unchanged. Figure 11 depicts the throughput vs. goodput of the system for different NFC payload sizes exchanged in the TCP session at 11.2 kbps and 24 kbps. On an average the throughput is 9.9kbps at an NFC bit rate of 11.2 kbps and 17.01 kbps at 24 kbps. The goodput of the system improves with increase in the payload size because the overheads from the TCP/IP header and UI protocol become less significant with increase in payload size. Choosing a bigger TCP MSS size will help in increasing the goodput of the system.

The throughput is lower for TCP sessions with small payload sizes, and it gradually increases with increase in payload size. This is because with small payload sizes, the time spent in waiting for a time-slot will be significant compared to the packet transmission time. At higher bit rates this becomes more noticeable because the transmission time will be even smaller. The NFC bandwidth is fixed in the system, so other ways of increasing the throughput would include TCP/IP header compression techniques [7], [8], and employing 6LoWPAN technology [9], [10].

5.4. Bandwidth utilization

The bandwidth utilization of NFC channel for the experiments performed is illustrated in Figure 12. The average bandwidth utilization is found to be 88.4% at 11.2 kbps and 70.89% at 24 kbps. The utilization is limited by the processing overhead which includes packet transmission time over UART and WiFi/Ethernet channels, packet processing time by the stack, etc. Lower bandwidth utilization is observed at higher bit rate because the processing overhead remains constant irrespective of the NFC bit rate. This overhead will be more significant at higher bit rates because it has smaller NFC transmission time. The bandwidth utilization in the system can be improved by parallelizing the packet processing and transmission operations, increasing bit rate of serial communication (UART) or by eliminating the MCUs and directly interfacing the appliance and PTx stacks to their respective NFC devices. This will reduce the processing delay caused by the serial communication. These techniques could not be tested due to limitations in the available hardware.

6. Parametric Analysis

This section focuses on analyzing parameters that could possibly affect the latency of the TCP connection in cordless kitchen. Simulations and theoretical calculations have been made to analyze the effects of parameters such as TCP MSS, CWND, NFC BER and non-TCP/IP messages.

6.1. Effect of TCP MSS value

Large MSS may cause IP fragmentation and reduce the efficiency of transmission. If the MSS size is too small, the TCP/IP header overhead would become very prominent resulting in inefficient use of the channel bandwidth, thus increasing the latency. Therefore, choosing the right maximum segment size is very important to achieve minimum latency.

Table 6 summarizes the average results of transferring 5 kB of data from the end-user device to the appliance using different TCP MSS sizes at 11.2 kbps. The results show that unless a very small value (< 512 B) is chosen, the latency will not increase by a large number. A small MSS of 256 B increases the latency by 26.28%, however, choosing a size greater than or equal to 512 B increases the latency only by $< 10\%$. So an MSS value of 1024 B or greater would give a very high performance with minimal latency. It is important to note that in case of an erroneous NFC/WiFi channel with a high bit error rate (BER), large packets would be more susceptible to errors compared to small packets. So the TCP MSS should be chosen depending on the conditions of the channel in order to avoid too many retransmissions caused by packet errors.

TABLE 6: TCP session latency for different TCP MSS values

TCP MSS (Bytes)	1460	1024	512	256
TCP session latency (s)	6.24	6.29	6.78	7.88

6.2. Effect of initial TCP CWND size

When TCP encounters a packet loss, the slow start process starts with an initial CWND size. It can be hypothesized that if the initial CWND is very small then the latency of the TCP session would increase as the slow start process would take longer to reach the maximum threshold, making the channel idle for a significant amount of time. This hypothesis is tested by considering an MSS of 1024 B and a maximum CWND size of 8192 B at 11.2 kbps. The experiment is performed to find out the optimum initial CWND size for the system such that there is minimum latency considering the congestion on the Ethernet/WiFi channels.

Figure 13 shows TCP session latencies with 100 kB data transfer for different initial CWND sizes over a lossy channel, where one out of twenty five packets are lost on an average. Only up to 7.5% reduction can be achieved when a bigger CWND size is chosen. This is because the NFC channel has a very small bandwidth and has almost maximum utilization even for small window sizes. Larger initial CWND size does not increase the channel utilization further. Therefore it can be concluded that the size of the initial CWND does not have a significant effect on the latency of this system.

6.3. Effect of NFC BER

Bit errors in the NFC channel introduce errors in TCP/IP packets being tunneled through the channel, causing checksum failures. In the given setup, there is no error correction

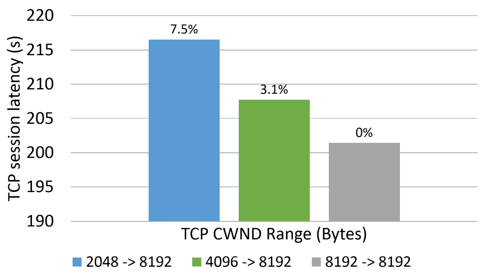


Figure 13: TCP session latency for different initial CWND sizes over a lossy channel

mechanism in the NFC layer. So even a single bit error in the packet would lead to retransmission as the packet will be dropped by the stack. Therefore, the presence of bit errors in the NFC channel will have a huge impact on the latency of the system.

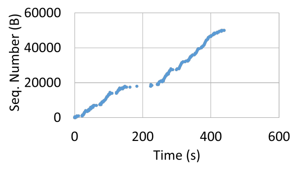
Bit errors in the NFC channel can be random or bursty. Random errors would lead to more number of retransmissions as the errors are randomly distributed and can affect any packet. On the contrary, the burst errors come as a block, so the errors would be confined to fewer packets. The burst error would have less impact on the TCP session latency compared to random error. An experiment is designed to verify this hypothesis where the appliance and end-user device exchange 100 packets of 500 B each. Random and burst errors of 10^{-4} and 10^{-5} are introduced in the NFC channel. The burst errors are introduced based on the Gilbert−Elliott model by taking the average burst length as 4 bits.

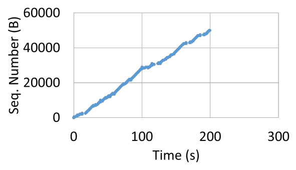
Figures 14(a) and 14(c) show the output of the TCP session with random NFC BERs of 10^{-4} and 10^{-5} respectively. It can be seen that as the BER reduces, the number of retransmissions decreases and hence the TCP session latencies. The same behavior is observed with burst error as shown in Figures 14(b) and 14(d). As per the hypothesis, at a given NFC BER, fewer retransmissions are observed with burst errors compared to that with random errors. This proves that burst errors have less impact on the system latency as the errors come in bursts which affect fewer TCP/IP packets.

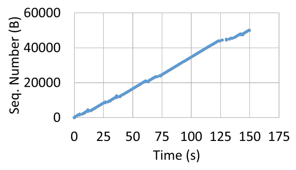
Results show that at a BER of 10^{-4}, the latency with the burst error is around 54.56% less than that with random error. However, as the BER reduces, the difference in latency between the two types of errors reduces. At a BER of 10^{-6}, there is only about 1.8% difference in the session latencies. Therefore, it can be concluded that at lower BERs the type of error will not matter much but at higher BERs burst errors will have lesser impact on the overall latency.

6.4. Considering non-TCP/IP messages over the NFC channel

For the ease of analysis, in all of the experiments the NFC channel was assumed to comprise of only TCP/IP messages. However, in real case scenario, the NFC channel would also carry other types of messages such as power control, measurement, state transition, negotiation, etc. The


(a) Random BER of 10^{-4}


(b) Burst BER of 10^{-4}


(c) Random BER of 10^{-5}

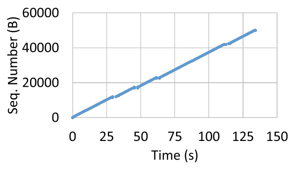

(d) Burst BER of 10^{-5}

Figure 14: TCP session with random and burst BER

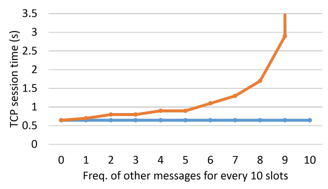


Figure 15: Latencies for different freq. of non-TCP msgs.

frequency of these messages would depend on the type of application being used. An experiment is performed to analyze the performance of TCP in the presence of such messages at different frequencies of their occurrence. The frequency of non-TCP/IP messages is taken as a fraction, for example, 2 slots every 10 slots used for other messages. This will impact the RTT of a TCP/IP packet over the NFC channel, i.e. t_{NFC} will increase by the number of slots used by other messages while the packet is being transferred over the NFC channel. t_{NFC} now becomes,

$$t_{\text{NFC}} = \frac{slots_{\text{pckt}}}{1 - freq_{\text{ctrlmsgs}}} * 10,$$

where, $freq_{\text{ctrlmsgs}} = a/b$. a/b signifies 'a' out of every 'b' slots format. b is taken as section size, so $b-a$ will be usable slots per section. t_{NFC} has to be rounded up to the nearest section size b because the usable slots can occur anywhere in the section.

As the frequency of other messages increases, the TCP session latency also increases. This increase will be non-linear because the number of usable slots varies inversely with latency. If $freq_{\text{ctrlmsgs}} = 0$, all slots will be available for TCP/IP packets. If $freq_{\text{ctrlmsgs}} = 1$, all slots will be used for other messages, so t_{NFC} becomes ∞. This means that the TCP/IP messages cannot be transferred. A TCP session with 1 kB data exchange and a section size b of 10 slots is considered for the experiment. Theoretical calculations are made for TCP session latencies using the Equation 1 (refer Section 4.2.2) and for an NFC bit rate of 848 kbps.

The results of the experiment is depicted in Figure 15. The graph shows an inverse variation function, so the rate of increase in latency will be steep as the frequency of non-TCP/IP messages increases. Results show that an efficiency of around 72% can be achieved in the transmission of TCP/IP messages at a frequency of 5/10. Appropriate frequencies can be chosen depending on the criticality of the Internet application.

7. Related Work

Over the last few years there has been research on providing Internet connectivity to NFC enabled IoT devices. For the ease of comparison and analysis, the methods of enabling connectivity in literature can be broadly classified as follows.

Tunneling standard TCP/IP protocol over NFC: Juan et al. introduce a concept called WebTag in [11], which enables direct IP based access to a sensor tag using NFC technology. It supports secure applications by tunneling the TCP/IP traffic over the NFC carrier. This work gives a brief overview of performance issues and their mitigation techniques. However, it lacks a detailed analysis of TCP characteristics that affect the system performance. Stefan et al. propose a concept for a test system in [12], that can establish a TCP/IP connection over an NFC channel and tunnel all the TCP/IP data through it. They provide some test results in terms of TCP retransmission rate and measured data rate using three different IP Maximum Transmission Unit (MTU) sizes. However, they do not take other aspects of the TCP/IP protocol into account. They only do a preliminary analysis to show advantages and disadvantages of tunneling TCP/IP over NFC devices.

6LoWPAN adaptation for TCP/IP protocol over NFC: In [9], YongGeun et al. have proposed a method of transferring IPV6 packets over an NFC channel using 6LowPAN functionalities. In this, they remodel the IPV6 stack to include the data link and the physical layers of the NFC stack. Junhwan et al. propose a 6LoWPAN adaptation protocol in [10] for transmitting IPV6 packets over NFC devices. It involves modifying the standard TCP/IP stack to enable IPV6 communication over an NFC channel using 6LoWPAN techniques described in [9]. The authors provide some simulation results in terms of latency and total packet count. They also compare the number of IPV6 packet transmissions with and without the IP header compression. However, other aspects of the performance have not been discussed in detail.

TCP/IP adaptation mechanisms for high delay networks: Dina et al. proposed a new protocol called the Explicit Control Protocol (XCP) in [13], which gives an improved congestion control mechanism in very high bandwidth-delay product networks. The work mainly concentrates on improving the bandwidth utilization and fairness in bandwidth allocation. As the NFC protocol only supports one-to-one communication and already has a good bandwidth utilization in the cordless kitchen system, these techniques are not applicable. In [14], Spencer et al. give an overview of the performance implications of slow links on the TCP/IP protocol. They recommend header and payload compression techniques to reduce the latency, TCP buffer auto-tuning to avoid packet drops due to buffer overflow conditions and limited transmit algorithm to trigger fast retransmit and fast recovery in case of packet loss. The authors of [15], [16] and [17] provide a couple of techniques to improve the throughput of TCP in wireless networks with high delay variability. They mainly discuss about spurious retransmissions that occur in such networks and provide solutions.

All these works provide some aspects of TCP/IP behavior in high delay, low bandwidth links but none of them studies the performance of a system containing different types of wireless channels with dissimilar characteristics. Furthermore, the works do not study the time-slotted NFC channel characteristics and quantify the TCP performance based on latency, throughput, retransmissions and bandwidth utilization for different bit rates and BERs.

8. Conclusions

This work focused on enabling Internet connectivity to cordless kitchen appliances. In order to provide an efficient and seamless communication with the appliance, we proposed the bridge architecture to enable connectivity via the time-slotted NFC channel of the cordless kitchen.

As most of the IoT applications rely on the TCP protocol, this work mainly focused on adapting TCP to the cordless kitchen system. We identified two major problems, namely spurious retransmissions and packet drops at the NFC interface, that arise while adapting TCP to the time-slotted NFC channel. To eliminate spurious retransmissions, we provided a generalized solution to compute optimum RTO values for TCP/IP packets tunneled over the NFC channel. As TCP does not consider the payload sizes for RTO estimation, spurious retransmissions were observed when there was high variability in packet sizes. To mitigate this, we proposed a new algorithm that dynamically estimates and updates the RTO values of the packets considering their payload sizes and changing channel delays. The algorithm is designed to adapt and perform well even when the WiFi/Ethernet channel has large delay compared to the NFC channel. For the data set considered, a reduction of 29.32% in latency is observed using this algorithm compared to VJ's algorithm used in the LwIP stack. To avoid packet drops in the NFC module resulting from small inter-packet delays, we introduced an NFC channel sensing mechanism in the cordless kitchen system. Using all these techniques, up to 38% reduction in the system latency is achieved at an NFC bit rate of 11.2 kbps and up to 53% at 24 kbps.

We also did a parametric analysis considering parameters that affect the system performance such as the TCP MSS, CWND size, NFC BER and non-TCP/IP control messages. Take-aways of this study include: (a) recommended MSS value is $\geq 1024\,\mathrm{B}$ to get a good performance; (b) the initial CWND size does not have a significant impact on the system latency, and (c) the system has relatively better performance with bursty errors in the NFC channel than with random bit errors.

References

[1] Connected Smart Kitchen, Smart Kitchen Gadgets — 2017 Guide to the Best Cooking Devices. [Online]. Available: https://www.postscapes.com/connected-kitchen-products/.

[2] Cordless kitchen appliances, Cordless kitchen appliances. [Online]. Available: https://www.wirelesspowerconsortium.com/developers/cordless-kitchen-appliances.html.

[3] Wireless Power Consortium, Wikipedia, 19-Jun-2017. [Online]. Available: https://en.wikipedia.org/wiki/Wireless_Power_Consortium.

[4] lwIP - A Lightweight TCP/IP stack - Summary [Savannah], lwIP - A Lightweight TCP/IP stack - Summary [Savannah]. [Online]. Available: https://savannah.nongnu.org/projects/lwip/.

[5] V. Paxon, M. Allman, J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," RFC 6298, Jun. 2011.

[6] V. Jacobson, Congestion avoidance and control, Symposium proceedings on Communications architectures and protocols - SIGCOMM 88, 1988.

[7] V. Jacobson, Compressing TCP/IP Headers for Low-Speed Serial Links, 1990.

[8] M. Degermark, M. Engan, B. Nordgren, and S. Pink, Low-loss TCP/IP header compression for wireless networks, Proceedings of the 2nd annual international conference on Mobile computing and networking - MobiCom 96, 1996.

[9] Y. Hong, Y. Choi, J. Youn, D. Kim, J. Choi, "Transmission of ipv6 packets over near field communication", draft-hong-6lo-ipv6-over-nfc-03, June 2017 (work in progress).

[10] J. Park, S. Lee, S. H. Bouk, D. Kim, and Y. Hong, 6LoWPAN adaptation protocol for IPv6 packet transmission over NFC device, 2015 Seventh International Conference on Ubiquitous and Future Networks, 2015.

[11] J. J. Echevarria, J. Ruiz-De-Garibay, J. Legarda, M. lvarez, A. Ayerbe, and J. I. Vazquez, WebTag: Web Browsing into Sensor Tags over NFC, Sensors, vol. 12, no. 12, pp. 86758690, 2012.

[12] S. Grunberger and J. Langer, "Analysis and Test Results of Tunneling IP over NFCIP-1," 2009 First International Workshop on Near Field Communication, Hagenberg, 2009, pp. 93-97. doi: 10.1109/NFC.2009.21

[13] D. Katabi, M. Handley, and C. Rohrs, Congestion control for high bandwidth-delay product networks, ACM SIGCOMM Computer Communication Review, vol. 32, no. 4, p. 89, Jan. 2002.

[14] S. Dawkins, G. Montenegro, M. Kojo, and V. Magret, End-to-end Performance Implications of Slow Links, 2001.

[15] K. K. Leung, T. Klein, C. Mooney, and M. Haner, Methods to improve TCP throughput in wireless networks with high delay variability, IEEE VTC2004-Fall. 2004.

[16] T. Klein, K. Leung, R. Parkinson, and L. Samuel, Avoiding spurious TCP timeouts in wireless networks by delay injection, IEEE GLOBECOM 04.

[17] G. Fotiadis and V. Siris, Improving TCP Throughput in 802.11 WLANs with High Delay Variability, 2005 2nd International Symposium on Wireless Communication Systems.