State-Recovery Protocol for URLLC Applications in 5G Systems

Anas Alsoliman
University of California, Irvine
United States of America
aalsolim@uci.edu

Forough Shirin Abkenar
University of California, Irvine
United States of America
fshirina@uci.edu

Marco Levorato
University of California, Irvine
United States of America
levorato@uci.edu

2023 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNeT) | 978-1-6654-9321-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/WISNET56959.2023.10046219

***Abstract*—In 5G systems, ultra-reliable low latency communication (URLLC) services are expected to support an end-to-end latency of up to 5ms, while guaranteeing a minimum of 99.999% packet delivery reliability. Providing such strict service requirements is potentially difficult. As a solution, the 5G standard suggests taking advantage of the packet duplication strategy whereby two data sessions are established between one or two base stations and the node running the URLLC service. However, the technical specifications of the duplication strategy are not addressed by the standard, and thus, its implementation is left up to the mobile network operators. In this paper, we propose a state-recovery protocol for URLLC applications that assists packet-duplication frameworks by recovering the state of transmitted packets in the case of acknowledgment loss. Based on such state information, packet-duplication frameworks can intelligently control the duplication rate to provide URLLC-level reliability while using as little resource as possible based on the current network conditions. We validate our protocol through multiple experiments on srsRAN nodes implemented on Colosseum, the world's largest radio frequency emulator.**

I. Introduction

5G Systems (5GS) are expected to bring substantial improvements over its predecessor, 4G LTE. These improvements are designed to be utilized by three main classes of applications, namely Enhanced Mobile Broadband (eMBB), Massive Machine-Type Communications (mMTC), and Ultra-Reliable Low-Latency Communications (URLLC). Among all three classes, URLLC supports mission-critical applications such as connected and autonomous vehicle applications. In URLLC, the end-to-end latency requirement is assumed to be no more than 5ms while the reliability of packet delivery is set to be no less than 99.999%. These two requirements are conflicting in nature and are difficult to be jointly achieved by the current protocols. For example, the packet retransmission scheme used to support the reliability guarantee service of the Transmission Control Protocol (TCP) increases the latency of delivered packets. On the other hand, while User Datagram Protocol (UDP) does not induce any additional latency on transmitted packets, it does not guarantee transmission reliability. Other transport protocols attempt to combine the best features of both TCP and UDP. However, often these protocols are designed with a specific application in mind and fail to meet the reliability guarantee of URLLC applications.

This work was partially supported by NSF under *Grant CNS 2134973.*

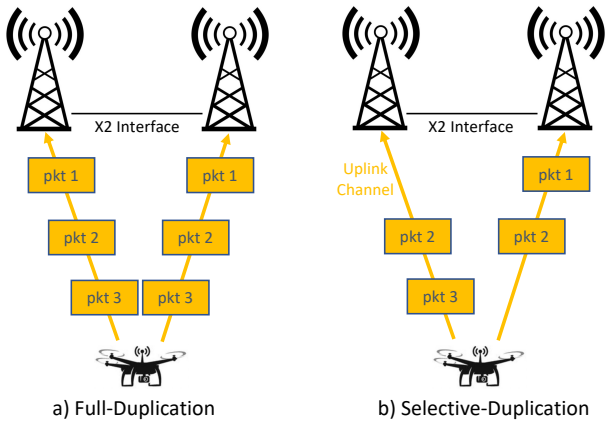

Fig. 1. Packet duplication schemes

In an attempt to combine these two strict and conflicting requirements into the same URLLC service class, the system architecture draft for the 5G System (version 16.6.0 release 16) released by the 3GPP group [1] suggests that URLLC applications could establish two redundant data sessions over the 5G network. This redundant connectivity scheme is referred to as Dual-Connectivity (DC) in the literature, which is part of the general Multi-Connectivity (MC) scheme. These two sessions can be established over two different channels toward either the same base station (BS) or two different BSs. Importantly, URLLC data is not split over the two channels to increase data rate, but for the purpose of packet duplication. That is, each packet generated by a URLLC application is sent twice, once over each channel. This scheme is known as Packet-Duplication (PD). Using the PD/MC scheme ensures that a packet lost over one channel can be recovered by the other channel without any packet retransmission. This results in an improved packet latency and reliability, at the price of a doubled bandwidth usage.

A. Related Work

Different research efforts have addressed various issues related to URLLC applications. Research in [2] and [3] investigate the effect of correlated channels on URLLC applications, as well as approaches to find the least correlated channels. As correlation in interference and path-loss decreases diversity,

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on September 29,2023 at 17:46:21 UTC from IEEE Xplore. Restrictions apply.

a packet duplicated on two distinct but correlated channels is likely to have a similar outcome on both channels. The authors in [4] extend the channel correlation problem and investigate the effect of correlated queues, i.e., the queues of different BSs with similar buffering loads. Other contributions study the minimization of PD/MC overhead on 5GS caused by URLLC applications. For example, in [5] a technique is provided to optimally decide which URLLC users are qualified to obtain an MC privilege – and thus increase reliability – and which users can use PD on top of MC. Other contributions, such as [6] and [7], study a PD scheme, called selective-duplication, where only packets that meet a set of criteria, such as poor channel quality at the time of transmission, are tagged for duplication. The main purpose of this scheme is to lower the URLLC emitted traffic, which is inversely proportional to the overall number of admissible URLLC users.

B. Problem Formulation & Contribution

As discussed earlier, a selective-duplication scheme can reduce the bandwidth usage of URLLC applications. Figure 1 compares a full-duplication scheme (1a) with its selective-duplication counterpart (1b). In Figure 1b, it is assumed that the delivery probability of packet 2 is below the URLLC threshold of 99.999%, while packets 1 and 3 meet the requirement. Figure 1a shows that the full-duplication scheme duplicates all packets regardless of the specific network conditions. On the other hand, the selective-duplication scheme only duplicates packet 2. The latter approach reduces bandwidth usage for the incoming packets. However, the duplication decision (whether it is optimization-based [5] or machine learning-based [7]) relies on channels' feedback and overall state (both current and previous states) of the network such as current channel conditions and delivery statistics of previously transmitted packets.

Different network statistics, such as signal-to-noise ratio, can be acquired directly by the transmitter, but other important statistics can only be reported back by the receiver in the form of direct acknowledgments (ACKs). However, the ACK message itself could be lost and thus, the state of the transmitted packet would be lost as well. This in turn would likely degrade the performance of selective duplication decisions for the next packets. In other words, it is difficult to precisely determine whether the packet or its ACK is lost - i.e., the so-called lost acknowledgments problem (LAP). Furthermore, the retransmission of packets (or their ACKs) is not feasible for URLLC applications due to their strict delay requirement.

There is a plethora of research efforts [8]–[10] addressing the LAP with the main focus on enhancing the reliability of packet delivery or the throughput using packet retransmission. Nonetheless, the objectives of acknowledgments in URLLC applications are different (deriving the state of the transmitted packets and its underlining network) than the mainstream networked applications, where delivery guarantee comes at the expense of a larger delay.

In this paper, we propose a state-recovery protocol for URLLC applications. The main motivation behind the proposed protocol is to aid and enhance the selective-duplication performance by identifying whether the loss is related to the transmitted packet or its ACK. We implemented our protocol on the Colosseum lab [11]. The results reveal that the proposed protocol enables a perfect selective-duplication scheme to outperform the full-duplication scheme by enhancing channel efficiency up to 95%. The protocol proposed in this paper is designed to support selective duplication in the uplink (UL). However, the protocol can be repurposed to support downlink (DL) as well. Importantly, our protocol improves the efficiency of the existing Selective-Acknowledgment (SACK) schemes [12] used by modern TCP implementations. SACK requires 8-bytes to represent a single contiguous gap (maximum of four gaps) of an unacknowledged stream of bytes, while our protocol represents an entire unacknowledged packet with a single bit. Furthermore, our protocol offers additional advantages such as indicating the delivery deadline state for previous packets and combining multiple acknowledgments from multiple BSs.

II. State-Recovery Protocol

The proposed state-recovery protocol enables the differentiation of the loss of a packet and its acknowledgment. To this end, the protocol recovers the state of a transmitted packet in the case that its ACK is not received. The state of the packets can then be used to aid the duplication decision process of the next packet. The design and implementation of such a process can take the form of either an optimization-based or machine learning-based algorithm. However, the details of this process are out of the scope of the paper.

A. Protocol Design Overview

We consider a system including one URLLC device, simply called device, and M base stations, where $1 \leq m \leq M$ indexes the m-th BS. There are a total number of M channels in the system each assigned to one BS. $s_i = \{r_i, d_i\}$ is the state of packet i (pkt$_i$), where both r_i and d_i are binary variables. r_i indicates the packet delivery status, where r_i is equal to "1" if pkt$_i$ is successfully received by the receiver, and "0" otherwise. On the other hand, d_i represents the packet deadline status, based on the relationship between the packet's delivery delay (sum of the transmission and queuing delays) dl_i and a predefined deadline parameter d_p. Specifically, d_i is equal to "1" if $dl_i \leq d_p$ and "0" otherwise. These two binary variables are attached to each acknowledgment ACK_i. Moreover, the receiver tracks the states of the last n packets. Then, with each sent out acknowledgment ACK_i, the receiver attaches the state of the last n packets to the acknowledgment, *i.e.,* $ACK_i = s_i + \{s_{i-1}, \ldots, s_{i-n}\}$. Lastly, for each BS$_m$, a binary status $b_{m,i}$ is defined, where $b_{m,i}$ is mapped to "1" if BS$_m$ has received a copy of pkt$_i$ and "0" otherwise. Then, $b_{m,i}$ is attached to ACK_i such that $ACK_i = s_i + \{s_{i-1}, \ldots, s_{i-n}\} + \{b_{1,i}, \ldots, b_{M,i}\}$. Notably, it is assumed that all BSs can communicate via a reliable channel such as the X2 interface in LTE networks.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on September 29,2023 at 17:46:21 UTC from IEEE Xplore. Restrictions apply.

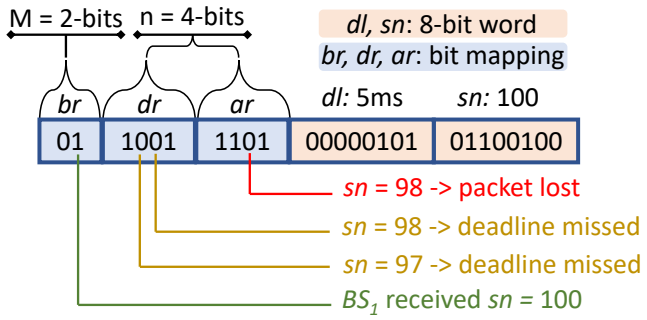

Fig. 2. An example of the proposed ACK header

B. ACK Header Structure

Figure 2 depicts the proposed ACK header, which consists of five fields: 1-byte sequence number (sn), 1-byte packet delivery delay (dl), n-bits ACK array (ar), n-bits deadline array (dr), and m-bits BS array (br). For each pkt$_i$, $sn = i$ is simply the sequence number of the packet successfully received at the BS, and dl_i shows the total delivery delay in milliseconds. Both fields can take any number ranging between [0, 255]. ar_i and dr_i on the other hand are the delivery state and the deadline state (since the reception of pkt$_i$) of the last n packets, respectively. For instance, for pkt$_{i-2}$, we have $r_{i-2} = ar_{i-2}[-2]$[1] and $d_{i-2} = dr_{i-2}[-2]$, where the position of each bit in the array represents the position of pkt$_{i-2}$ with respect to $sn = i$ within a series of n packets. Each bit in br_i is mapped to "1" if a BS has successfully received pkt$_i$ and "0" otherwise.

C. Acknowledgments Duplication

The selective-duplication process requires an immediate ACK response to determine the duplication decision for the next packet. If an ACK is lost, the channel recovery must be postponed until the next ACK is successfully received. This waiting time would force the selective-duplication process to either rely on outdated information from the last received ACK or simply consider the previously sent packet as lost. To mitigate this issue, every ACK sent on one channel is duplicated over all the other channels, regardless of whether the packet was duplicated or not. When a BS receives a packet from the UL channel, it first duplicates the ACK to all other BSs over a reliable backbone network, *e.g.,* X2 interface in LTE networks. Upon receipt of an ACK from the backbone network, the BS relays the ACK toward its own DL channel. To prevent sending the same ACK multiple times, the BS backs off for d_p seconds when it receives a UL packet in order to collect ACKs from all other BSs and fills up br before forwarding the ACK to the DL channel.

D. Protocol Scenarios via Sequence Diagram

The proposed protocol can recover the state of packets under different scenarios. Figure 3 shows that the recovery

[1]A negative index indicates indexing from the array tail where [-1] is the last element in the array while [-2] is the second to last element, and so on.

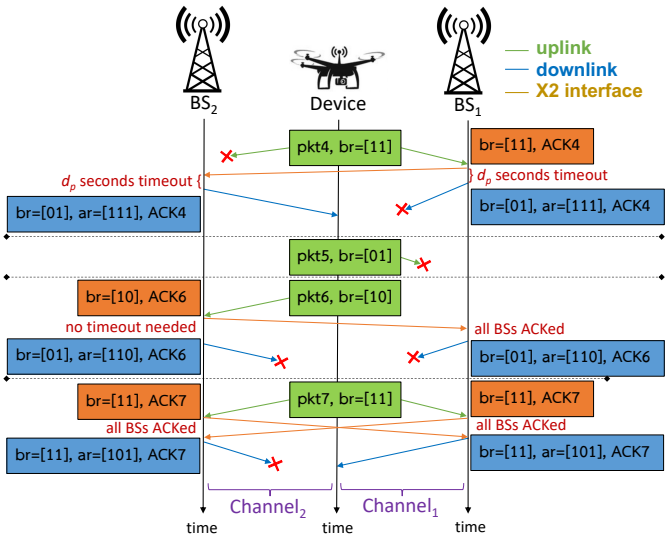

Fig. 3. Sequence diagram of the proposed protocol

starts right after successfully transmitting four packets. Note that, dl and dr were omitted from the ACK header in Fig. 3 due to a space limitation. The first transmitted packet in Fig. 3 starts with pkt$_4$ ($sn = 4$) and is duplicated on both channels. The packet and its ACK are lost in $channel_2$ and $channel_1$ respectively. However, ACK_4 is recovered over $channel_2$ by BS_2. From $br = [01]$ of the received ACK, the device infers that the packet sent to BS_2, and the ACK sent by BS_1, are lost. Note how BS_1 backed off for d_p seconds waiting for a duplicated ACK from BS_2 to fill its br before sending its own ACK over $channel_1$. Similarly, BS_2 backed off for d_p seconds waiting for the UL packet to arrive. Thereafter, the device attaches its own br before sending the packet. This br is for optimization purposes to inform the BS about what other BSs should have received a packet duplicate. Hence, an ACK duplicate from other participating BSs is also expected over the X2 interface. Next, pkt$_5$ is sent over a single channel, but is lost. Then, pkt$_6$ is sent over a single channel as well, but all its ACKs (original and duplicates) are lost. The loss of all packet duplicates and/or its ACKs is the worst-case scenario for the state-recovery protocol as there is no feedback from the network. However, after sending pkt$_7$, the device is able to recover the state of the previous two packets. Also, neither of the BSs backed off for d_p seconds since all duplicate ACKs were received. Interestingly, the device can infer from ACK_7 that pkt$_7$ was received by both BSs (by looking at 1s in $br = [11]$) and also, ACK_7 from BS_2 was lost. However, if ACK_7 from both BSs are lost, the next successfully received ACK (*e.g.,* ACK_8) can only predict that pkt$_7$ was successfully received by *some* BSs yet it cannot identify which one actually received the packet. This limitation is determined by the need to compressed information in the ACK header. The knowledge of previously used channels, *i.e.,* which BSs exactly received the previous packet, increases the header size w.r.t br and ar from $m + n$ to $m \times n$ bits.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on September 29,2023 at 17:46:21 UTC from IEEE Xplore. Restrictions apply.

TABLE I
EXPERIMENT RESULTS

	Experiment 1: -50dBFS	**Experiment 2: -70dBFS**
Total # of packets	37451	37459
# of packets received on both channels	35350	29695
# of packets received on one channel	2073	7620
# of packets lost in UL	28	144
# of ACKs lost in DL	0	12
Max # of continuous lost packets	2	3

III. IMPLEMENTATION & EVALUATION:

A prototype of the protocol is implemented in the Colosseum [11], which emulates RF signals using a network of high-performance Software-Defined Radios (SDR), specifically, X310 USRP radios. The protocol is built on top of srsRAN, formerly known as srsLTE – an open-source implementation of the LTE stack, where User Equipment (UE) devices connect to and exchange traffic with BSs. We design an experiment where a single device is connected to two BSs via LTE. The device is configured to transmit a series of 1000-byte packets on UL for five minutes. The inter-departure of the packets is set to 8ms, which corresponds to a network traffic of $\approx$ 1 Mbps. During the experiment, the UE duplicates each packet toward each BS. Then, both BSs are connected via the Colosseum's internal cabled network to emulate a reliable LTE's X2 interface for the BSs. The implementation evaluates the effectiveness of the proposed protocol under different channel conditions. Specifically, we test the protocol under two experiments that use different signal decibel Full Scale (dBFS) receive powers: -50 dBFS and -70 dBFS. For each experiment, we compare the efficiency of the proposed protocol against the efficiency of the full-duplication scheme. The former is defined as the percentage of packets successfully received on both channels, while the latter is expressed as the percentage of packets successfully received on only one channel. In this regard, we collect the number of packets lost in the UL, the number of ACKs lost in the DL, and the number of packets delivered via each channel.

Table I lists the results of both experiments. In the first experiment, the total number of packets successfully received is 37423, where 35,350 packets are received on both channels and 2073 ones are received on only one channel. This results in an efficiency of 94.5% and 5.5% for the protocol-assisted selective-duplication and the full-duplication strategy, respectively. In the second experiment, the efficiency of the full-duplication strategy increases to 20.4%, and our protocol improves efficiency up to 79.6%. This observation further motivates the usefulness of the selective-duplication scheme, and thus, the need for a state-recovery protocol to support an efficient duplication decision strategy.

In the first experiment, 28 packets are lost on both channels. The missing ACKs of these 28 packets provide an indication that the packets are lost. However, the UE device cannot determine with certainty whether the packets or its ACKs were lost in the absence of additional feedback enabling the recovery of the actual states of these missing packets. In the second experiment, 156 packets have missing ACKs, even though only 12 packets were actually lost. To recover the past packet and network states, n should be selected carefully as it indicates the maximum number of packets that are expected to be lost in a sequence. Moreover, the maximum number of contiguous packets lost in experiments 1 and 2 are 2 and 3 packets, respectively.

IV. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a state-recovery protocol that tracks the state of previously lost packets to address the Lost Acknowledgement Problem (LAP), that is, the problem of differentiating between a lost packet and a lost ACK. This protocol is intended to be utilized by frameworks that require precise network feedback such as the selective-duplication frameworks for 5G's URLLC applications. Notably, the protocol has the potential for additional improvements in the future, such as 1) embedding the reading of the current buffer size of network slices, 2) dedicating a field in the header for reporting individual channel states, 3) introducing an adaptive header size for adding extra network statistics in the case of poor channel conditions, 4) using ACK timeouts, and 5) involving the sender in the message exchange. The latter is beneficial to report the scheduled transmission time of the next packet, whereby the receiver can reply with a triggered ACK if no packet is received within the scheduled time limit.

REFERENCES

[1] 3GPP, "System architecture for the 5g system (5gs)," *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501*, 2020.

[2] Y. Chen *et al.*, "Impact of correlated fading on multi-connectivity," *IEEE Transactions on Wireless Communications*, vol. 20, no. 2, pp. 1011–1022, 2020.

[3] J. Rao and S. Vrzic, "Packet duplication for urllc in 5g: Architectural enhancements and performance analysis," *IEEE Network*, vol. 32, no. 2, pp. 32–40, 2018.

[4] C.-Y. Chen and H.-Y. Hsieh, "Does queue correlation matter in 5g multi-connectivity with packet duplication?," *IEEE Wireless Communications Letters*, 2022.

[5] J. Elias *et al.*, "Multi-connectivity in 5g new radio: Optimal resource allocation for split bearer and data duplication," *Available at SSRN 4102694*.

[6] M. Centenaro *et al.*, "System-level study of data duplication enhancements for 5g downlink urllc," *IEEE Access*, vol. 8, pp. 565–578, 2019.

[7] D. Segura *et al.*, "Dynamic packet duplication for industrial urllc," *Sensors*, vol. 22, no. 2, p. 587, 2022.

[8] B. Kim and J. Lee, "Retransmission loss recovery by duplicate acknowledgment counting," *IEEE Communications Letters*, vol. 8, no. 1, pp. 69–71, 2004.

[9] B. Sinopoli *et al.*, "Optimal linear lqg control over lossy networks without packet acknowledgment," *Asian Journal of Control*, vol. 10, no. 1, pp. 3–13, 2008.

[10] S. Chandra *et al.*, "Hybrid buffer-based optical packet switch with negative acknowledgment for multilevel data centers," *Journal of Optical Communications*, 2020.

[11] L. Bonati et al., "Colosseum: Large-Scale Wireless Experimentation Through Hardware-in-the-Loop Network Emulation," in *Proc. of IEEE Intl. Symp. on Dynamic Spectrum Access Networks (DySPAN)*, December 2021.

[12] E. Blanton *et al.*, "A conservative loss recovery algorithm based on selective acknowledgment (sack) for tcp," tech. rep., 2012.

Authorized licensed use limited to: Access paid by The UC Irvine Libraries. Downloaded on September 29,2023 at 17:46:21 UTC from IEEE Xplore. Restrictions apply.