Mitigating Latency Inflation in V2C Transactions Using Periodic Sidelink Communication

Seungmo Kang, Hojeong Lee, and Hyogon Kim

***Abstract*—In the vehicle-to-cloud (V2C) interactions using 3GPP sidelink, vehicles transmit data to a server on the Internet and receive responses in return. This paper shows through real-life measurement that bi-directional end-to-end communication latency in V2C can be dominated by that caused by sidelink communication if the application data arrivals and the sidelink transmit resources are not synchronized. It is counter to our expectation that the delay on the non-sidelink leg that spans the 5G RAN, 5G core network, and the Internet will account for the majority of the latency. For reducing inflated latency on the sidelink, this paper explores several solution approaches: reduced selection window size, decreased packet delay budget, and adaptive selection window postponement. Through real-life measurements, we show that such measures can slash sidelink latency inflation by more than 50% so that 5G V2X sidelink can be more easily applied to real-time applications built on V2C communication.**

***Index Terms*—Vehicle-to-cloud (V2C), end-to-end delay, sidelink, resource allocation timing**

I. Introduction

THE rapid progress of connected vehicle technology and autonomous driving capabilities necessitates vehicle-to-everything (V2X) communication to enhance the safety and efficiency as well as to evolve the intelligence and functionalities of vehicles. These vehicles can locally use vehicle-to-vehicle (V2V) and vehicle-to-pedestrian (V2P) communications for safety and traffic efficiency as well as vehicle-to-cloud (V2C)[1] communications among others for more extended and rich services. In particular, as the vehicles are increasingly connected to the cloud and more tightly coupled with services provided therein, the communication latency in the wireless leg must be reduced to the minimum. For example, real-time V2C communication available for production vehicles will be able to enable such services as automotive digital twins or remote driving.

In cellular V2X communication, the on-board units (OBUs) on vehicles as well as the road-side units (RSUs) are User Equipment (UE). They can connect to a base station (*i.e.*, gNB) and use uplink/downlink or alternatively communicate directly amongst themselves by using sidelink. As the base station may not always be available, sidelink communication is indispensable. Accordingly, vehicles can use wireless resource either centrally allocated by gNB if available (called Sidelink Mode 1) or autonomously selected by themselves in a distributed manner (called Sidelink Mode 2). In terms of scheduling, there are three methods that vehicles can choose to use. Dynamic scheduling is performed by gNB, where each transmission by vehicles needs to be requested, granted, and then executed. This method using three transmissions per message can burden gNB with control traffic. When there are many vehicles, it could increase the latency due to the signaling exchange between the UE and the gNB that can be detrimental to certain safety-critical V2X services [1]. In case gNB is not available or vehicles want to use distributed scheduling, Sensing-based Semi-Persistent Scheduling (SPS) [2] can be used. Finally, random scheduling is the method to use if channel sensing cannot be afforded, as in the case of extremely energy-constrained devices.

In this paper, we explore a scenario where vehicles use V2C communication through RSUs in Sidelink Mode 2. On behalf of vehicles (*i.e.*, OBUs) that access RSUs over sidelink, the RSUs may directly talk to the cloud server itself (see Fig. 1) or to a closer Multi-Access Edge Computing (MEC) node in order to reduce communication delay. We assume that both OBUs and RSUs use SPS for allocating semi-static transmit resources for V2C communications. One might argue that dynamic scheduling instead of SPS can better serve the variable delay traffic from the server. However, in the absence of a central coordinator, dynamic scheduling is not usable. Even if it is, the control overhead can be significant as discussed above. As V2C applications such as remote driving or digital twins will require long-lived conversations, dynamic scheduling will not be a scalable approach especially when there are many vehicles that use long-lived V2C services. Therefore, we consider the case where the RSU also uses semi-static scheduling for client-server interactions achieved through the sidelink.

In SPS [2], each participating node selects and uses a frequency resource (subchannels) at periodically spaced subframes at $t_{\text{TX}}+(j-1)\cdot RRP \ \ (1 \leq j \leq RC)$ where t_{TX} is the time index of the first resource subframe, RRP is the vehicle's Resource Reservation Period (RRP), and RC is the randomly selected resource reservation counter value. When the first resource in the series of j periodic resources is selected, SPS in 3GPP New Radio (NR) Sidelink Mode 2 uses the sensing information from the 1,100 ms window in the recent past to avoid booking the same resource used by other nodes. This process prevents most simultaneous uses of the same resource by multiple vehicles. RRP can be set to 1 to 99 ms or multiples of 100 ms up to a maximum value of 1000 ms. The RC value depends on the RRP. For instance, it is randomly set between 5 and 15 for RRP = 100 ms [3]. Every transmission decrements RC, and when RC reaches zero, the device selects new subchannels and subframes with a probability of $1-P_{keep}$, where P_{keep} is the resource keeping probability.

[1]a.k.a. vehicle-to-network-to-vehicle (V2N2V)

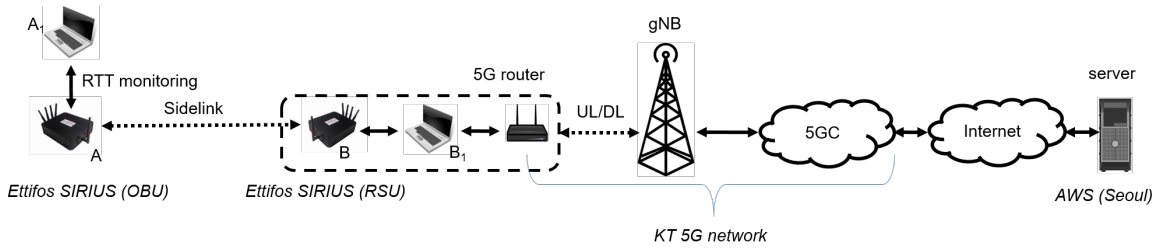


Fig. 1: Real-life V2C round-trip time measurement setup

Note that the periodic resource allocation is performed on the medium access control (MAC) layer, whereas the message arrivals are dictated by applications on end devices (OBUs and cloud servers). Without cross-layer signaling, therefore, these events may not be synchronized. Specifically, the strict periodicity of the resource availability enforced by SPS may well become out of sync with the message arrivals. This problem can be more pronounced with the messages from the cloud where the processing delay for the request from the vehicle may vary. This paper shows that the lack of synchronization significantly amplifies the end-to-end delay in a V2C transaction. It is counter to our expectation that the delay on the non-sidelink leg that spans the 5G RAN, 5G core network and the Internet (*e.g.* RSU $\leftrightarrow$ AWS server in Fig. 1) will account for the majority of the latency. Especially when edge computing is used, the latency to access the service will significantly dwindle to a small number. Then unnecessary delay caused on the sidelink will be even more pronounced and become a major factor to degrade the service experience. In real-time applications such as remote driving and digital twins that demand low latency, this latency inflation can be particularly prohibitive.

In the rest of this paper, we first explore the end-to-end V2C delay performance across 3GPP NR Sidelink. Our real-life measurements show that involves sidelink, operational 5G network and cloud service indeed confirm that the sidelink communication latency can be as large as three times the total delay caused by the non-sidelink leg of the communication path. Then we investigate how we can avoid experiencing unnecessary latency in V2C communication. The proposed modifications can reduce the overhead to less than half of the original delay. We conclude that V2C communication needs to marry more tightly the application layer messaging and MAC layer transmit resource allocation timings for better latency performance.

II. Related work

Research related to latency and throughput measurements to assess 5G V2X communication performance in the V2C contexts exists. While not identical to our study, similar experimental designs are employed to evaluate the performance of 5G V2X links. Ficzere *et al*. [4] estimated the latency of a 5G link in actual V2X use-case conditions by installing a 5G modem connected to an end device on a moving vehicle and measuring end-to-end round-trip time (RTT) to a server. Kutila *et al*. [5] measured latency and throughput for data transfer between vehicles and a server or a mobile edge computing unit via 5G network. Coll-Perales *et al*. [6] provided an extensive list of prior work that measured end-to-end latency, so interested readers are referred to the paper. The work also presented an analytical model that includes radio, transport, core, Internet, peering points and application servers to quantify the latency of 5G V2N and V2C communications. However, these works did not investigate the impact of the resource scheduling and its interactions with message arrival times on the end-to-end RTT performance as our work does.

Novel applications developement by using V2C communication is recently gaining attention. Sisbot *et al*. [7] presented a use case for V2C communication, where traffic data from connected vehicles are periodically collected in the cloud and then be leveraged to provide lane-level congestion information back to the vehicles, which is more fine-grained information than the current state of the art. Our work also assumes similar scenarios where vehicles continuously interact with the cloud to create and implement useful services.

Lucas-Estan *et al*. [1] presented an analytical model that estimates the latency of 5G radio access network (RAN). The model provides a solid analytical foundation as it accounts for various 5G New Radio (NR) parameters such as subcarrier spacings (SCS), slot durations, scheduling, and retransmission mechanisms. Interestingly, it regarded vehicle-to-network-to-vehicles (V2N2V) communications as a replacement for direct V2V communications. Pannu *et al*. [8] also considered V2C communication as an indirect path to implement V2V communication while the V2V OBU market penetration is low, rather than a means to deliver various cloud-based services. It proposed so called virtual edge computing where some vehicles serve as gateway nodes. The main purpose is to enable V2C-only vehicles to upload their data to the cloud through the gateway vehicles so that they can interact with V2V-capable vehicles. However, these works did not account for the variable processing delay at the cloud server that interacts with sidelink resource scheduling or how to deal with it on the sidelink to minimize the total end-to-end delay.

III. Problem description

In order to better understand the sidelink latency issue for future V2C applications, we turn to real-life measurements by using commercial 3GPP Release 16 compliant 5G-V2X sidelink devices that connect to the Amazon Web Services (AWS) through Korea Telecom's 5G network, as illustrated in Fig. 1. Through comparing the sidelink delay and the total

end-to-end delay obtained in the measurement, we find that the sidelink resource scheduling contributes excessive delay to the total end-to-end delay. For a test service that simply echoes the packet, for example, the delay on the RSU–AWS leg accounts only 25% of the total end-to-end delay; the rest is mostly incurred by the sidelink communication.

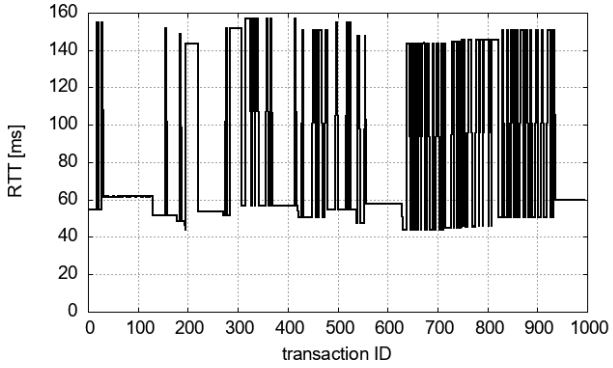
A. Real-life 5G V2C latency

In the experimental configuration as shown in in Fig. 1, the two Ettifos SIRIUS devices [9] talk to each other over the 3GPP Release 16 NR sidelink, where one acts as an OBU and the other as a RSU. The SIRIUS units A and B were respectively connected with Ethernet to laptop computers A_1 and B_1 that are used for monitoring and application execution. We measure two types of RTT. First, the end-to-end RTT is the time it takes for a message from "vehicle" (*i.e.*, A_1) to the destination server (*i.e.*, AWS) and then for a response to return from the server back to the vehicle. The entire end-to-end path traverses through the vehicle's OBU, RSU, 5G RAN, 5G core network, and the server on the Internet, and back to the vehicle in the reverse direction. Second, the sidelink RTT is the time it takes for A to transmit to B plus the time from B to A over the sidelink, subject to the resource schedules on both ends of it. By comparing these two RTT's, we can quantify the contribution of the sidelink latency to the total end-to-end delay. Table I summarizes the parameters used to configure the SIRIUS devices for the sidelink communication.

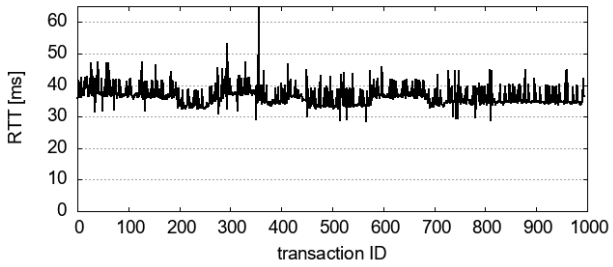
TABLE I: 5G-V2X sidelink configuration at SIRIUS devices

Parameter	Value
Transmit power	20 dBm
Subcarrier spacing	15 kHz
Channel bandwidth	20 MHz
Carrier frequency	5865 MHz
Subchannel size	15
NumSubchannel	7
Bandwidth	20 MHz
PSCCH time resource	2
PSCCH frequency resource	10
MCS index	8
ReselectAfter	4
Packet size	300 B
Resource reservation period (RRP)	100 ms
P_{keep}	0.8

For end-to-end RTT data collection, A periodically generated packets with an Inter-Transmission Time (ITT) of 100 ms and a packet size of 300 bytes, while time-stamping on each packet. The generated packet was then transmitted from A to B via NR sidelink. Without a particular application in mind, we set the packet size to that of a typical periodic vehicle status message such as Basic Safety Message (BSM) [10] also used by 3GPP [11] in evaluation studies. B, in turn, transmitted the packet to B_1. We connected B_1 to a 5G router via Ethernet so that packets from B_1 use the 5G router to enter KT's 5G network. The packets traveled through the gNB and KT's 5G core network and eventually accessed the AWS server on the Internet. To avoid unnecessary latency due to intercontinental travel, we implemented the server by using a *t2.large instance* of AWS using *ap-northeast-2 region* located in Seoul, South Korea. Upon receiving a packet from the vehicle, the server simply echoed the received packet in reverse, towards A. A recorded the time of packet reception and computed the time difference through the timestamp on the packet, and transmitted this information to A_1 that collected the end-to-end RTT values. A total of 1,000 packets were exchanged along the same route for measuring RTT. Fig. 2(a) shows the end-to-end RTT recorded at A_1. Here, we call each round trip by "transaction." One can immediately notice that the delay has two distinct components. The first is the base delay at approximately 60 ms. Then there are frequently occurring delay spikes whose magnitude is approximately 100 ms above the base delay. Note that Fig. 2(a) is only a single instance that exhibits the problem addressed in this paper. Depending on how the message arrival times and the sidelink resources relatively align on time axis, the end-to-end RTT can be much worse, sometimes exceeding 200 ms.



(a) End-to-end RTT



(b) Non-sidelink (RSU↔AWS) leg RTT

Fig. 2: Measured RTTs comparison

In order to figure out how much sidelink latency contributes to the end-to-end delay, we measured the RTT values of only the RSU↔AWS leg. Fig. 2(b) plots the non-sidelink leg RTT measurements. We notice that the RTTs are approximately 40 ms, with occasional spikes reaching up to 60 ms. Prior real-life measurement works [4], [5] report similar numbers for 5G networks. Considering the difference between the RSU↔AWS leg delay from the end-to-end delay, we can see that the sidelink delay is contributing the majority of the end-to-end delay. This is an unexpected and important aspect that needs to be addressed when we use semi-static scheduling on both sides of sidelink. Moreover, we will show later that the variable non-sidelink leg RTT values dictates the magnitude of the delay inflations (*i.e.*, the delay spikes in Fig. 2(a)). The non-sidelink

leg delay has complex interactions with the application-level and SPS parameters in incurring the delay surges.

Although the AWS server in this paper was programmed to simply echo the request packet, in reality the server will perform application-specific processing and will generally have larger RTT values on the non-sidelink leg. Nevertheless, the out-of-sync arrivals of application messages can cause the same problem. Moreover, the sidelink delay can still incur significantly delay costs because the processing and the communication delay on the non-sidelink leg can be small owing to hyperscaler cloud services and MEC.

B. Causes of inflated sidelink latency

There are two main causes of the large or abrupt end-to-end RTT inflations. First, the selection window sizes used on either end of the sidelink can be large. For instance, with typical SPS parameter values for resource selection $T_1 = 1$ ms and $T_2 = 100$ ms and packet delay budget (PDB) of 100 ms, just the sidelink delay can amount to nearly 200 ms in the worst case where the OBU and the RSU happen to pick a resource near the end of the selection window. In Fig. 3, for example, the application message G_1 at the vehicle side should wait until a resource is available to carry it to the RSU. If the resource, V_1, has been selected close to the right extreme of the selection window, nearly 100 ms latency will be incurred. Likewise, similar unfortunate timing relation can occur at the RSU side for the response packet S_1, adding another 100 ms to the end-to-end RTT. If tighter resource allocation is forced, the inflated RTT can be mostly avoided.

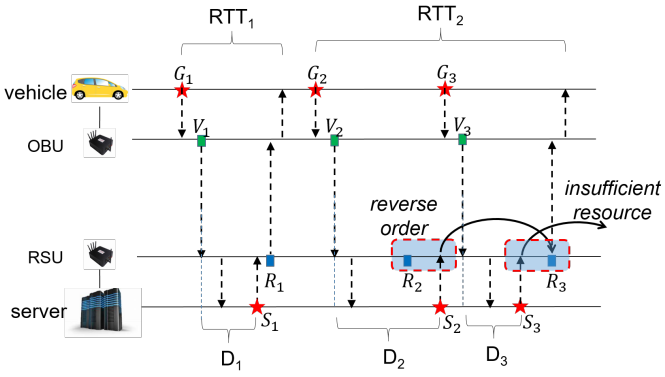


Fig. 3: Problem description: reverse ordering of resource and message causing inflated latency (□=transmit resource, ★=application message)

The second cause of the large RTT is more involved. It is the lack of synchronization between the messages and resources. Even with tight resource selection window it can still cause delay inflation, or can further add to the already inflated delay in case large selection window sizes are used. In particular, the 100 ms (= RRP) delay superpositioned above the base delay in Fig. 2(a) is caused by this cross-layer synchronization issue. In Fig. 3 for example, the server processing plus the communication delay for the second request, D_2, is large so that the response packet S_2 from the server arrives later than resource R_2 ("reverse order"). In this case, the response has to wait until the next resource R_3. The consequence is that the RTT will look inflated by a whole RRP (*i.e.*, time between R_2 and R_3) to the client, compared to the previous RTT. Even worse, the inflated delay may persist over the next few transactions under certain circumstances. Suppose the resource size for the response packets at the RSU is tailored to the response packets, which is the typical practice in wireless resource allocation, in order to minimize waste resource. Then, since S_2 uses most of R_3, S_3 will not be able to use R_3 even if it arrives before R_3 ("insufficient resource"). It will have to use R_4, which will add 100 ms to RTT_3 as well. Such delay inflation will persist unless some counter-measure is taken. First, a size-triggered reselection can be performed. Second, Radio Link Control (RLC)-layer segmentation can be performed [3] across a series of RSU transmissions to amortize the resource shortage. For instance, by including segments from S_3 in what small remaining space the subsequent resources have available, the shortage can be eventually resolved. In Fig. 2(a), the SIRIUS devices recover to the base delay after a few transactions by implementing the RLC segmentation.

C. Replicating results through simulation

In order to replicate the results in Fig. 2(a) and freely modify the sidelink scheduling behavior to explore the impact on the delay performance, we turn to simulation. We developed an in-house NR sidelink simulator that uses the same sidelink configuration settings as in Table I. Using the real-life RSU↔AWS leg delay measurements in Fig. 2(b) as input to the simulator, we obtain Fig. 4 for the predicted end-to-end delay in the baseline scheme. The differences between our simulation model and the SIRIUS device are, first, that we opt to not implement RLC-layer segmentation because it is an implementation option [3] and it does not help solve the delay inflation problem. Second, we assume in this paper that the server responses are small in size so that they can be aggregated in the logical channel. This can be true if mainly control information is sent in response to vehicle's sensory or status reports in such applications as remote driving and automotive digital twins. For example, S_2 and S_3 in Fig. 3 are aggregated and carried by the same resource R_3. Consequently, the detailed end-to-end RTT dynamics changed, but with the same pathological phenomenon: the RTT inflated by 100 ms frequently appears. For instance in Fig. 4, the transaction IDs around 200, 350, and 750 among many others show such surges over the base delay.

To put the excessive contribution of the sidelink delay to the total RTT in perspective, we also juxtapose the measured delay on the RSU↔AWS leg in the figures including Fig. 4 to be compared with the RTT simulation result. For the baseline scheme, both PDB and T_2 are set to 100 ms on both ends of the sidelink. We can observe that the RTT is extremely high, comparing with the low RTT between RSU and the AWS server on the Internet shown at around 40 ms. In the baseline scheme, both the mean and the deviation of the end-to-end RTT are large at $\mu \approx 148.1$ ms and $\sigma \approx 36.3$ ms, which is dominated by the sidelink delay. Such inflated communication delay will make time-critical applications unreliable [5], [12].

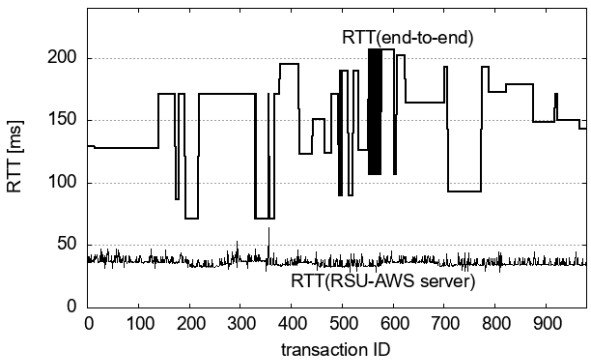


Fig. 4: Baseline scheme: end-to-end RTT disproportionately inflated by sidelink

Besides the large delay itself, the delay variability exhibited in the end-to-end RTT is also harmful because they may lead to a suboptimal user experience in real-time applications. In particular, applications such as remote driving may suffer a more detrimental impact from the variability than from a higher average latency [12]. In the next section, we explore several counter-measures to the delay inflation and variability problems on the sidelink, and evaluate each of them through simulation experiments.

IV. Solution approaches

To cope with the two root causes of the inflated delay problem, we explore a few solution approaches in this section. As to the large selection window size issue, recollect that the right extreme of the SPS selection window is set to $W_r = \min(T_2, PDB)$ in effect [2]. Therefore, to tighten the selection window, we can set T_2 to a small value on the MAC layer or set PDB to a small value through the application requirement. Either will force a small selection window size. As to the second problem of the lack of synchronization, we can attempt to select a resource that comes after the server's message arrives at the RSU, by setting T_1 to a larger value. So, the three solution approaches we will explore are:

1) Reducing selection window size
2) Reducing PDB
3) Postponing selection window

We will investigate each case and evaluate it in terms of the delay reduction effect and any side effects as the cost aspect. The baseline configuration to be compared with them is given as follows: $[T_1,\ T_2,\ PDB] = [1,\ 100,\ 100]$ ms (which implies the selection window postponement = off). Below, we will change one or more of them for each explored approach and investigate its impacts.

A. Tightening selection window - decreasing T_2

Here, we change to $T_2 = 10$ ms from the baseline, so that $[T_1,\ T_2,\ PDB] = [1,\ 10,\ 100]$ ms on both sides of the sidelink. We obtain Fig. 5 for the new configuration. Comparing with Fig. 4, we notice that the base latency is formed around 60 ms here. The gap between the base and the RSU↔AWS latency is indeed reduced by the much smaller selection window size on either side of the sidelink. However, frequent delay spikes that add 100 ms (*i.e.*, RRP) to the base latency can still be seen. These delay spikes happen when the response message S_i from the server does not arrive in time for the next transmit resource R_i so that S_i has to be carried by the next resource R_{i+1}, as we saw the case of S_2 in Fig. 3. Note that since this resource R_{i+1} is less than 100 ms away from the late-arriving S_i, PDB-triggered reselection [3] does not happen. Although this approach reduces the mean RTT and the deviation to $\mu \approx 65.9$ ms and $\sigma \approx 31.3$ ms, the consequent delay spikes still need to be removed by other means.

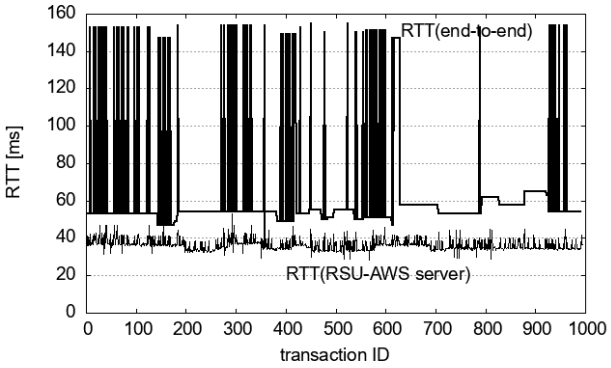


Fig. 5: End-to-end RTT; $T_2 = 10$ ms

B. Tightening PDB

An alternative approach is reducing PDB. In terms of the resulting selection window size, PDB reduction has the same effect as the reduction of T_2 due to the effective relation $W_r = \min(T_2, PDB)$. However, there is one additional consequence to the PDB reduction: PDB-triggered reselection. When the time to the next scheduled resource is longer than the PDB of the arriving packet, the RSU can perform a reselection so that a new resource can be found within the PDB [3]. If the resource picked in the narrow selection window happens to precede the message arrival due to large server delay (*e.g.* like the impact of D_2 in Fig. 3), the small PDB can act as a trigger to invoke a reselection. If T_2 was used to reduce the selection window under large PDB, on the other hand, the reselection would not occur.

1) PDB = 10 ms: Here, we change PDB from 100 ms to 10 ms so that $[T_1,\ T_2,\ PDB] = [1,\ 10,\ 10]$ ms on both sides of the sidelink. Note that we must also set $T_2 \leq PDB$ ([2], Section 8.1.4). Fig. 6 shows the result of decreasing PDB. It shows that the end-to-end RTT is highly dynamic. This is because the tight PDB indeed triggers a reselection whenever the time that response message from the server has to wait until the next resource exceeds the reduced PDB value. In that case, the reselection cancels the existing resource allocations that would violate the PDB [3]. An immediate consequence is that the large delay spikes that we saw in Fig. 5 are prevented because a different resource within $W_r = 10$ ms of the late response message will be newly allocated instead. Consequently, the mean RTT has been reduced by more than 100 ms compared to Fig. 4, and the deviation reduce by a factor of 8. We obtain $\mu \approx 54.7$ ms and $\sigma \approx 4.2$ ms.

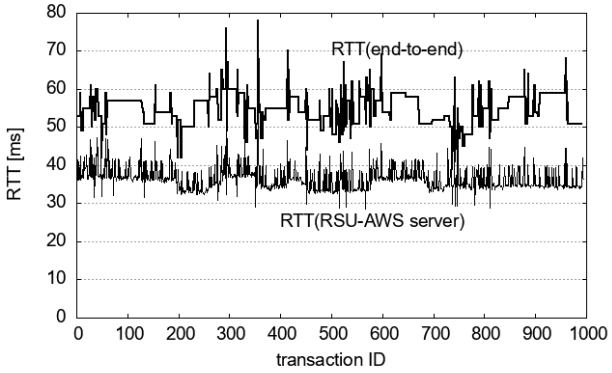


Fig. 6: End-to-end RTT; PDB = 10 ms, $T_2 = 10$ ms

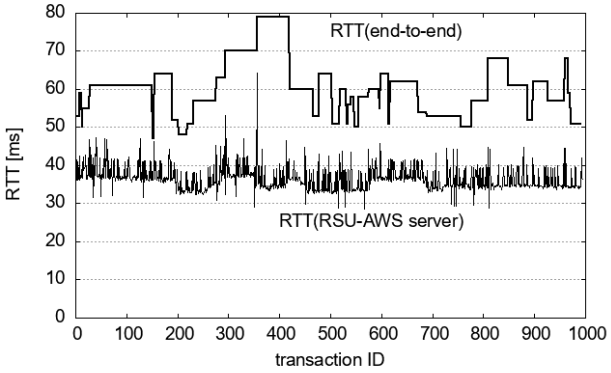


Fig. 8: End-to-end RTT; PDB=80 ms

Unfortunately, the fact that this approach is subject to rampant reselections is concerning (see Table II). Moreover, this is for only a single pair. If there are many vehicles that use the cloud service that go through the RSU, the unpredictability of sidelink resource use caused by the aggregate number of reselections will lead to excessive packet collisions and eventually reduced packet reception ratio among the vehicles using SPS [13]. In order to investigate this problem, we attempt to incrementally increase PDB in Fig. 7. It shows that the PDB-triggered reselections become excessive when the PDB is less than 30 ms. Such PDB "threshold," if any, will depend on the distribution of the server response times D_i, *i.e.*, on the application. Finally, setting the PDB at 100 ms may seem to remove the PDB-triggered reselections. But as Fig. 5 showed, this configuration is subject to excessive delay spikes.

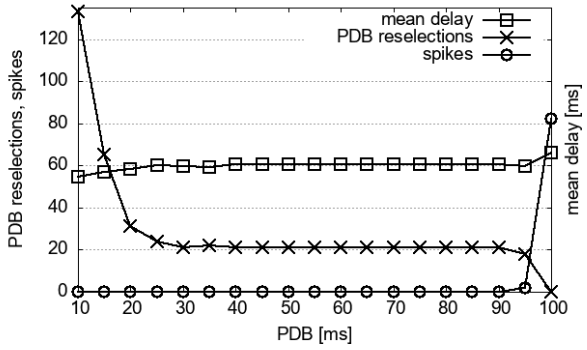


Fig. 7: Mean RTT and reselections as functions of PDB

2) PDB = 80 ms: As to the mean delay, Fig. 7 shows that it is not reduced significantly even if we use very small PDB values, risking rampant reselections. Therefore, it would be wise and possible to set PDB to a higher value without excessively sacrificing the mean delay. Fig. 8 shows the result of setting PDB = 80 ms, only slightly lower than the baseline configuration. Indeed, the PDB-triggered reselection becomes less frequent than with PDB = 10 ms (see Table II). In return, the delay and its variability slightly increase: $\mu \approx 60.5$ ms and $\sigma \approx 7.2$ ms.

Note that reducing PDB at the MAC layer has issues. First of all, barring cross-layer signaling, MAC is a wrong layer to tamper with PDB. Although reducing the delay is generally good for the application, PDB should better be not arbitrarily adjusted by the MAC layer because it is an application layer parameter. In the current standard specification, the PDB value is handed down through the communication stack, and it is only used as a constraint at the MAC layer when the right extreme of the selection window W_r is set. The selection window size reflects the application's PDB requirement in such an indirect manner. Second, in the ignorance of the application characteristics, the PDB "threshold" in Fig. 7 cannot be easily figured out. If PDB is adjusted too small, it will cause rampant reselections. For example, the server delay D_i could frequently be over 80 ms. It is known that excessive reselections undermine the predictability of resource use that is exploited in the SPS algorithm. The consequence is that packet collisions increase and the packet reception ratio (PRR) is adversely affected [13], which in turn may affect application performance.

C. Postponing selection window - increasing T_1

Apart from reducing the selection window or PDB, we could consider postponing the selection window by increasing T_1 so that it comes after the arrival of the response packet from the server. It can help avoiding the "reverse order" in Fig. 3, thereby preventing the 100 ms delay surge. One remaining question is how long the postponement should be, as the server delay D_i depends on the application behavior.

1) Fixed $T_1 = 90$ ms: Without knowing the server delay distribution, we first attempt a simple scheme to set a fixed T_1. For comparison with the same selection window size configuration above, we set $T_1 = 90$ ms and $T_2 = 100$ ms on the RSU side so that $[T_1,\ T_2,\ PDB]_{rsu} = [90,\ 100,\ 100]$ ms. We leave $T_1 = 10$ on the OBU side so that $[T_1,\ T_2,\ PDB]_{obu} = [1,\ 100,\ 100]$ ms because the OBU does not have the same variable processing time issue in our system model. The resulting end-to-end RTT dynamics is shown in Fig. 9. We immediately notice that this scheme produces extremely inflated delay. This is because the selection window has been postponed by 90 ms, so the minimum is raised by so much. Furthermore, there are high-frequency fluctuations as well. These characteristics are reflected in the statistics: $\mu \approx 121.0$ ms and $\sigma \approx 38.8$ ms. This scheme hardly improves the delay

performance over the baseline scheme, especially compared with the previous two approaches.

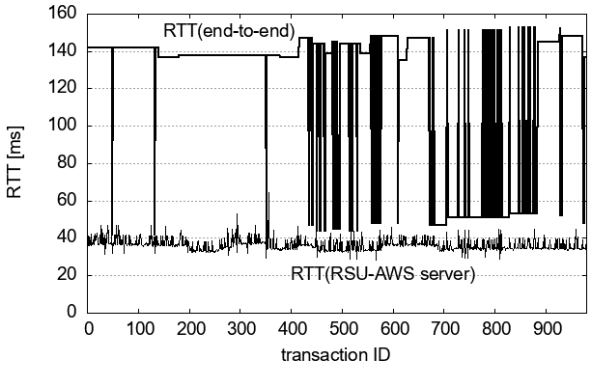


Fig. 9: End-to-end RTT; $T_1 = 90$ ms

2) Adaptively determining T_1: An alternative and better approach to setting T_1 in face of unknown server delay so as to avoid the reverse resource order is to model D_i as a random variable and exploit its statistical property. Instead of attempting to completely prevent the reverse order, this scheme sets T_1 to be larger than *most* server delay values D_i. In order to deal with unknown distribution (*e.g.* due to different applications or due to different network conditions) of the r.v., we choose to turn to Chebyshev's inequality that provides an upper bound on the probability of deviation σ of a r.v. D (with finite variance) from its mean μ, regardless of the type of distribution:

$$Prob[|D - \mu| \geq k\sigma] \leq \frac{1}{k^2}. \quad (1)$$

In this paper, we adaptively set k so that p-percentile of D_i's are covered where $p = 90$ or 99. It requires continuously updating μ and σ for every obtained server delay sample. For this purpose, RSU uses a moving window of the last 100 samples of D. The updated statistics are used to set the starting position (T_1) of the selection window whenever a reselection is performed according to the reselection counter (RC) and the resource keep probability (P_{keep}). Fig. 10 illustrates the idea. Here, μ and σ are the mean and the deviation of D, the processing delay at the server plus the communication delay between the RSU and the server. Fig. 11 shows the RTT measurements when applying Inequality (1).

We notice that as p increases, so does RTT. For the conservative $p = 99$, RTT is mostly larger than 60 ms, whereas for $p = 90$ it is mostly 80 ms with occasional increases over it. In return, $p = 99$ quickly suppresses the delay spikes even after small number of RTT samples are collected. For $p = 90$, the spikes still appear even after 300 transactions, but eventually they all disappear. The mean and the deviation for $p = 90$ and $p = 99$ cases are $\mu \approx 63.6$ ms and $\sigma \approx 8.3$ ms, and $\mu \approx 81.8$ ms and $\sigma \approx 7.6$ ms, respectively. In future, for known cloud servers, RTT samples will accumulate much faster at the RSU as the RSU handles numerous vehicles using the same application on the same cloud server. It will enable much faster adaptation than shown in Fig. 11. Fig. 11(c) shows the T_1 and $\mu + k\sigma$ adaptation over time. Note that $T_1 = \mu + k\sigma - D_i$ is

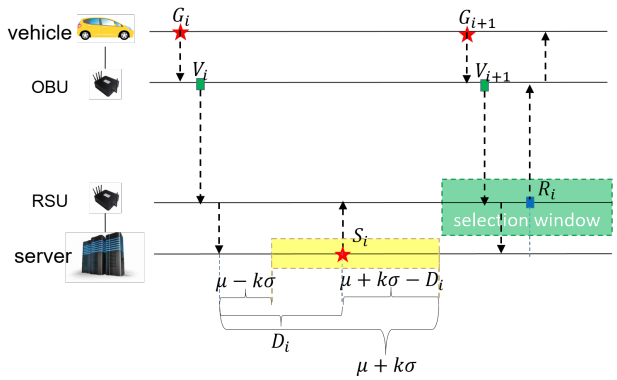
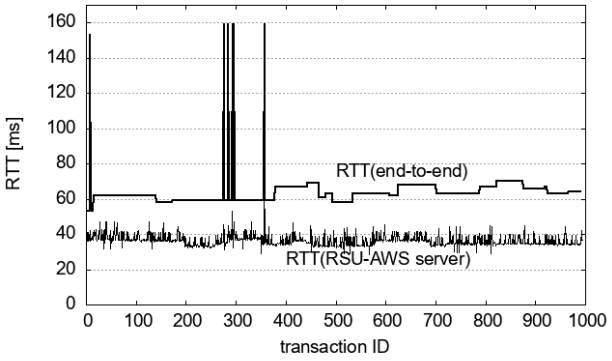
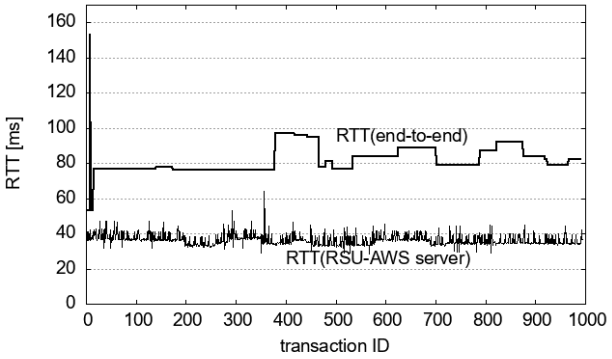


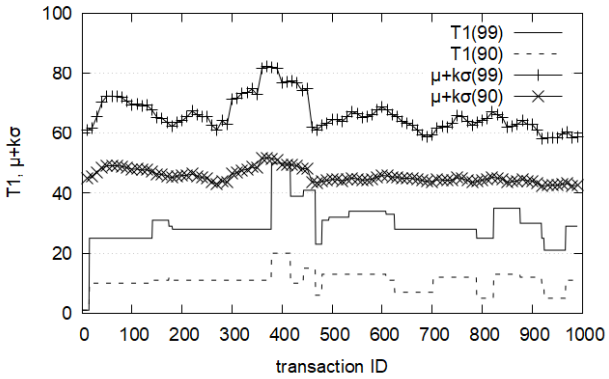
Fig. 10: Adaptively postponing selection window by Ineq. (1)



(a) Adaptive selection window postponement, 90%-ile



(b) Adaptive selection window postponement, 99%-ile



(c) T_1 and $\mu + k\sigma$ adaptation as per Eq. (1)

Fig. 11: Applying Chebyshev on processing delay to position the selection window

measured from the arrival time of the server's response packet S_i, whereas $\mu + k\sigma$ is measured starting from the arrival of the request packet G_i from the vehicle (see Fig. 10). Also note that T_1 is only updated upon a reselection event whereas $\mu + k\sigma$ is updated in every transaction.

D. Summary

Table II summarizes the end-to-end RTT statistics, the number of PDB-triggered reselections at the RSU, and the number of delay spikes for the three considered approaches. We can observe that the delay and delay variance are most reduced by using smaller PDB values. However, it can explode the number of reselections significantly higher than the other two approaches. Therefore, this approach needs to be carefully evaluated of its impact on other vehicles' PRR due to increased packet collisions due to frequent reselections. Moreover, setting an appropriate PDB value remains an issue. Next comes small T_2 in terms of delay performance, but it has large variability. In particular, a whole RRP worth of delay (*e.g.* 100 ms) frequently added to the base delay will be difficult to tolerate for certain applications. Reducing the selection window by increasing T_1 by a fixed amount yields the worse delay performance than the other approaches, only comparable to the baseline. Finally, T_1 adaptation at the RSU trails T_2 reduction and PDB reduction, but it has less variability issues and PDB-triggered reselection issues. A few delay spikes can occur at the beginning of the adaptation, but they tend to disappear after it. All in all, this final adaptive approach seems to be the best to slash the inflated sidelink delay with few side effects for V2C communication using cellular V2X technology.

TABLE II: Mean, deviation, and #reselections at RSU

Scheme	μ [ms]	σ [ms]	PDB Resels.	Spikes
Baseline	**148.1**	**36.3**	**0**	16
$T_2 = 10$ ms	65.9	31.3	0	82
PDB = 10 ms	54.7	4.2	133	0
PDB = 80 ms	60.5	7.2	21	0
$T_1 = 90$ ms	121.0	38.8	0	43
Adapt T_1 (90%)	63.6	8.3	0	6
Adapt T_1 (99%)	81.8	7.6	0	1

V. Conclusion

This paper investigated V2C communication scenario where semi-static scheduling is used in both on-board units and road-side units when the NR sidelink is used. Through real-life measurement on commercial 5G user equipments connected through an operational mobile network to a cloud server, it showed that the dominant latency can be caused by the sidelink scheduling. Potential solution approaches were considered including narrow selection window, small packet delay budget and adaptive selection window positioning. Narrow selection window can effectively reduce the mean delay, but when the server processing delay exceeds the selection window size within which the periodic resource was chosen, the delay spike that adds a full RRP to the end-to-end RTT can result. Reducing PDB can also suppress the end-to-end delay, but the PDB-triggered reselections may become rampant if the server delay cannot be correctly estimated. Moreover, it has an undesirable aspect of arbitrarily altering the application layer parameter. Third, adaptively postponing the selection window can be effectively exploited at RSUs especially when the RTT values to cloud servers can be monitored. Compared with the other schemes, both the RTT mean and the deviation can be significantly suppressed without significant side-effects. As a future work, combinations of the three approaches will be explored to improve the latency performance even more. Also, mathematical analysis of the approaches will be conducted. Finally, the impacts of mobility, message size, and the impacts on other vehicles' PRR will be investigated.

Acknowledgements

This work was supported by the Technology Innovation Program (No.20026341, Development of ultra-thin SiP under 3.5mm and roof-integrated 5G TCU) funded By the Ministry of Trade, Industry & Energy (MOTIE, Korea).

References

[1] M. Carmen Lucas-Estañ, Baldomero Coll-Perales, Takayuki Shimizu, Javier Gozalvez, Takamasa Higuchi, Sergei Avedisov, Onur Altintas, and Miguel Sepulcre. An analytical latency model and evaluation of the capacity of 5g nr to support v2x services using v2n2v communications. *IEEE Transactions on Vehicular Technology*, 72(2):2293–2306, 2023.

[2] 3GPP. 3rd generation partnership project; technical specification group radio access network; nr; physical layer procedures for data (release 18). Technical Specification, Dec. 2023. TS 38.214.

[3] 3GPP. 3rd generation partnership project; technical specification group radio access network; nr; medium access control (mac) protocol specification (release 15). Technical Specification, Jan. 2021. TS 38.321.

[4] Dániel Ficzere, Gábor Soós, Pál Varga, and Zsolt Szalay. Real-life v2x measurement results for 5g nsa performance on a high-speed motorway. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 836–841, 2021.

[5] Matti Kutila, Pasi Pyykonen, Qing Huang, Wei Deng, Wenhui Lei, and Emmanuel Pollakis. C-v2x supported automated driving. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–5, 2019.

[6] Baldomero Coll-Perales, M. Carmen Lucas-Estañ, Takayuki Shimizu, Javier Gozalvez, Takamasa Higuchi, Sergei Avedisov, Onur Altintas, and Miguel Sepulcre. End-to-end v2x latency modeling and analysis in 5g networks. *IEEE Transactions on Vehicular Technology*, 72(4):5094–5109, 2023.

[7] E. Akin Sisbot and Yashar Zeiynali Farid. Demo: Lane-level traffic estimation using v2c communication. In *2023 IEEE Vehicular Networking Conference (VNC)*, pages 157–158, 2023.

[8] Gurjashan Singh Pannu, Seyhan Ucar, Takamasa Higuchi, Onur Altintas, and Falko Dressler. Vehicular virtual edge computing using heterogeneous v2v and v2c communication. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2, 2022.

[9] Ettifos. World's first 3gpp release 16 compliant 5g-v2x sidelink platform sirius. https://www.ettifos.com/product-sirius. Accessed: 2024-03-01.

[10] SAE. V2x communications message set dictionary. Standard, Sept. 2023. J2735_202309.

[11] 3GPP. 3rd generation partnership project; technical specification group radio access network; study on evaluation methodology of new vehicle-to-everything (v2x) use cases for lte and nr; (release 15). Technical Specification, June 2019. TS 37.885.

[12] Ruilin Liu, Daehan Kwak, Srinivas Devarakonda, Kostas Bekris, and Liviu Iftode. Investigating remote driving over the lte network. In *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '17, page 264–269, New York, NY, USA, 2017. Association for Computing Machinery.

[13] Jicheng Yin and Seung-Hoon Hwang. Adaptive sensing-based semipersistent scheduling with channel-state-information-aided reselection probability for lte-v2v. *ICT Express*, 8(2):296–301, 2022.