Trace-based Simulation of C2X-Communication using Cellular Networks

Norbert Goebel* Markus Koegel‡ Martin Mauve* Kalman Graffi*
*Department of Computer Science, University of Düsseldorf, Germany
{ goebel, mauve, graffi }@cs.uni-duesseldorf.de
‡ESCRYPT GmbH – Embedded Security, Bochum, Germany
markus.koegel@escrypt.com

***Abstract*—Simulation of cellular network communication is complex and typically requires a high degree of knowledge about the underlying network and its parameters. At the same time simulating cellular networks is important for the automotive industry in order to be able to test the feasibility of applications that use car-to-x-communication applications before performing costly field tests. In this paper we propose a trace-based simulation model derived from real-world measurements. It does not require any information about the network besides information that can be readily measured by a regular user, it is much faster than regular simulation, and it has been validated by comparing simulation results to real world measurements.**

I. Introduction

In the context of *Car-to-X (C2X)* communications, the impact of a particular C2X application is a pivotal question that needs to be answered in the development process. To this end, it is vital to get a thorough understanding of the underlying communication channel. Currently, C2X applications utilize either local, ad-hoc short-range communication (e. g., IEEE 802.11p) or conventional cellular mobile communication (e. g., GPRS, UMTS or LTE). While there has been good progress evaluating the performance of IEEE 802.11p based on theoretical models, network simulations or real-world measurements, the knowledge of the performance of C2X communication via cellular networks is much more limited. The main reason for this is the very complex nature of cellular wireless networks which makes them hard to model and simulate accurately.

In this paper, we propose a way to simulate today's cellular network communication in order to determine the availability of information in the networked vehicles. Instead of simulating particular aspects of the radio communication or the network architecture, we propose to use a trace-based simulation approach: first, we collect measurements on the key characteristics of the cellular network in the spatial area that we want to examine in the simulation (like illustrated in figure 1). The collected *traces* are in turn used as basis for our simulation model that we employ to determine what information is available in which vehicle at any specific point in time. We validate the simulation model by comparing simulation results with real world measurements.

In contrast to regular, model-based simulation of network and radio communication parameters, a trace-based simulation provides results that directly follow from real-world measurements. Therefore, compared to regular simulations, they are much less likely to deviate significantly from the measured real-world *ground truth*. Furthermore, a trace-based simulator requires significantly less computational complexity than a model-based simulator, thus achieving a very good scalability in the number of simulated vehicles. In addition, collecting measurement traces with a small probe vehicle fleet and running simulations on this basis is much more cost-efficient in an early development phase than repeatedly conducting large field tests with numerous equipped vehicles.

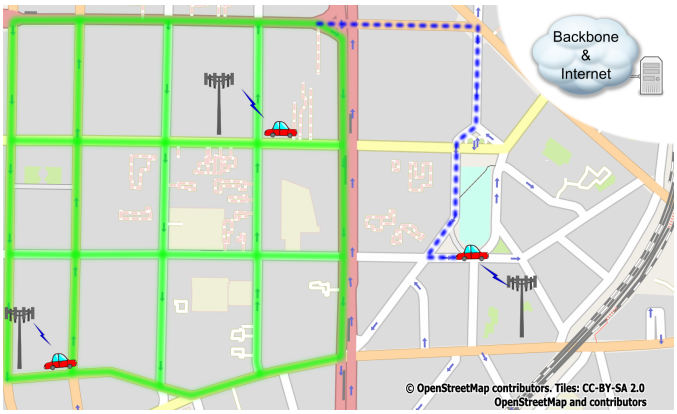


Figure 1: Simulation area.

The key contributions of this paper are twofold:
(1) We introduce a novel simulation model using measurement traces to simulate UDP traffic over mobile cellular networks.
(2) We validate the simulation model by comparing simulation results to independent real-world measurements.

In the remainder of this paper, we give a brief overview of related work in Section II. We explain our tracing methodology and the arising challenges in Section III and describe our basic simulation model and our implementation in Section IV. Section V contains an analysis and a validation of the proposed simulation model. We further on introduce improvements to our approach in Section VI and evaluate them in Section VII. We conclude this paper in Section VIII.

II. Related Work

For our work, we do not require any insights into the cellular mobile network and thus treat it as a black box. We therefore focus on existing end-to-end measurement methods only, where two network hosts actively participate in the

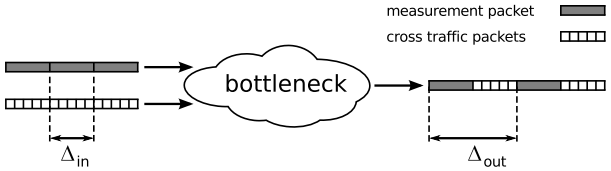


Figure 2: Operating mode of the *Probe Gap Model (PGM)*.

measurement process. All existing black box methods either utilize *packet pairs* — two packets usually sent back-to-back — or *packet trains* — a sequence of measurement packets sent with varying gaps between consecutive packets. Measuring data rates in networks has been studied extensively in the last two decades and the examined methods mainly differ in the ways the packets are sent and analyzed.

Considering multiple slightly different notations in the related work, we use the following terminology:

The *capacity* C_i of a network link i is the maximum *IP-Layer* throughput that can be achieved *without* the presence of competing *cross traffic*. The *available data rate* A_i of a network link i is the maximum *IP-Layer* throughput that can be achieved while competing *cross traffic* exists. The *utilization* U_i of a network link i is the ratio of its *capacity* that is used by *cross traffic*: $U_i = 1 - (A_i/C_i)$.

A *network path* is a sequence of network links from a sender to a corresponding receiver. The *capacity* C of a network path is the minimal capacity of all links on the path, the *available data rate* A for a network path is defined analogously. The link with the smallest capacity on the path is called the *narrow link*, while the one with the smallest available data rate is referred to as the *tight link*. Following equations apply to a network path with N links and corresponding link utilizations U_i:

$$\begin{aligned} C &= \min_{i=0..N-1} C_i \\ A &= \min_{i=0..N-1} [C_i(1-U_i)] \end{aligned} \quad (1)$$

With this terminology in mind, we can now look at the existing work in this area. Early approaches for data rate measurements used simple packet pair [1] or packet train techniques [2]. However, these approaches either did not reflect today's Internet practice or were later on proven to measure the available data rates incorrectly, for example by the authors of [3]. We therefore only focus on recent end-to-end measurement methods that either base on the *Probe Gap Model (PGM)* or the *Probe Rate Model (PRM)*.

A. Probe Gap Model (PGM)

Figure 2 illustrates the modus operandi of the PGM: the model uses *Packet Dispersion* — the temporal gap at the receiver between two packets sent back-to-back — to estimate the available data rate of the network path.

While PGM only needs two packets to estimate the available data rate, it has some drawbacks: first, the estimates exhibit a very large variance if the measurement covers only a short time interval. Therefore, multiple estimates with some kind of smoothing are needed to achieve more stable estimates. Second, PGM assumes *FIFO* queues in the routers and depends on one link of the path being the tight link and the narrow link at the same time. Finally, the capacity of the network path needs to be known a priori to estimate the available data rate which is not given in black box scenarios. These drawbacks leave PGM unsuitable for black box measurements in mobile cellular networks.

B. Probe Rate Model (PRM)

PRM uses the concept of *self-induced congestion* to estimate the available data rate of a network path: PRM systems send packet trains starting with large temporal gaps between consecutive packets that decrease towards the end of the packet train. Thereby, the data rate is increased over the transmission time of the packet train.

As long as the send rate is smaller than the available data rate of the network path, the receive rate is equal to the send rate. But, once the send rate exceeds the available data rate, a queue builds up at the *tight link* within the network which causes the delay of the packets to increase so that the receive rate eventually will be smaller than the send rate. Using repeated measurements, the available data rate can be estimated by locating the point at which the receive rate exceeds the send rate. PRM is used, for example, by TOPP, Pathload, Pathchirp, PTR and Assolo [4]–[8].

As long as the cross traffic remains constant, the measurements can be repeated with varying send rates, so that theoretically, arbitrary estimation accuracy can be achieved. High accuracy degrees, however, result in measurements that take several *Round Trip Times (RTTs)* to be completed. Both the required stability of the cross traffic and the long measurement duration render PRM unsuitable for mobile measurement scenarios.

III. Tracing Cellular Networks

We propose a trace-based simulation model of cellular networks, which is based on real-world measurements and not on assumptions or artificial simplifications. Since measurement traces play such a crucial part in our simulations, the way how the respective measurements are being gathered is important. In this section, we discuss the setup that we used as well as how the values were traced during our measurements.

A. Measurement Setup

We performed the network measurements with a mobile node and a stationary server node, both running Linux operating systems. We equipped the mobile node with a 3G cellular network interface (Sierra Wireless MC8775), the stationary node had access to a gigabit Ethernet. To avoid reachability problems due to *Network Address Translation (NAT)* that is usually employed by cellular network providers, the mobile node always initiated the measurements.

Additionally, both the mobile and the server node were connected to GPS receivers (Garmin 18x LVC) that provided *Pulse Per Second (PPS)* signals via serial connections. We therefore used the devices both for self-positioning and high-accuracy time-synchronizations, thus turning both nodes into

Stratum 1 servers with mean synchronization errors below 1 ms. To this end, we ran the *gpsd* software [9] in combination with an *ntp* daemon [10] for the time synchronization. The usual ntp time synchronization over the network is not an option, because ntp requires symmetric communication delays that are not given in cellular networks and we aim at keeping any additional traffic load as low as possible to not distort our own measurements.

For spatial repeatability, we selected two routes of the local public transport: one bus route through a rural/urban area and one tram route through the urban area of a major German city. On the bus route, we performed one round-trip measurement run logging data for 119 minutes. The round-trip measurements on the tram route took 83, 91 and 92 minutes. We logged every packet's send and receive event with the respective timestamps and GPS trajectory data at a frequency of 1 Hz as well as all available modem status data.

B. Measuring Available Data Rates, Delays and Packet Loss

As discussed in Section II, we cannot use PGM and PRM in mobile networks. We therefore resort to simple bulk traffic measurements: we determine the available data rate with sequences of $n \geq 2$ UDP datagrams, each of size s_p including all headers, that we send as back-to-back *packet trains*. In cellular mobile communications, network characteristics are usually not symmetric, so we measure them separately for each communication direction. We perform the measurements once per second for each direction to provide a good spatial measurement coverage. In the following, we only regard *upstream* measurements from the mobile client to the measurement server; the *downstream* measurements work analogously.

The client fills each packet of the train with the train length n, the packet's position index in the train, its send timestamp, the packet train sequence number, and the last measurement results (delay, available data rate, drop rate) for the other communication direction. Additionally, the packets in the upstream direction have an extra field for the currently used network technology of the modem (GSM, GPRS, EDGE, UMTS or HSDPA). Using this information, the server can approximate the client's available data rate A as follows:

$$A = \begin{cases} \frac{s_p \cdot (l-f)}{t^{\text{rx}}(l) - t^{\text{rx}}(f)} & n' \geq 2 \\ \frac{s_p}{t^{\text{rx}}(f) - t^{\text{tx}}(f)} & n' = 1 \\ \text{NaN} & \text{else } (n' = 0) \end{cases} \tag{2}$$

where f and l are the indexes of the first and last received packets of the packet train, $n' = l - f + 1$ is the number of actually received packets of the train and $t^{\text{tx}}(i)$ and $t^{\text{rx}}(i)$ are the send and receive timestamps of the i-th packet, respectively. Due to the high-accuracy time synchronization the server can safely use the absolute timestamps from the client.

Unfortunately, bulk traffic measurements have a significant drawback, which was one of the reasons PGM and PRM were developed in the first place: they are prone to overloading the measured network path. This *self-induced congestion* would then make it hard to accurately determine the delay that one could expect from an uncongested network. We deal with this drawback by dynamically adapting the length of the packet train so that its total size is approximately half of the currently available data rate of the network path that is under investigation, multiplied by the length of the measurement interval. This allows the measurement data to leave the network before the next packet train is sent and thus enables us to provide accurate delay measurements at least for the first packet of the train.

There are two ways how the currently available data rate can be estimated in order to adjust the train size: first, given that the wireless link is most likely the bottleneck, the mobile device can use the information about the network technology currently in use; this is usually provided by the cellular network interface and implies an upper bound for the available data rate. Depending on the refresh rate of this information, this technique allows for quick adaptations of the packet train size for the upstream direction. In the downstream direction, however, the server cannot adapt its train parameters that quickly, because this can only be done as a reaction to the currently used network technology announced in the upstream packets from the client. Second, the client can employ the data rate measurements that were made based on its last packet train: to this end, the server piggybacks this approximation onto its own packet train. Once the client receives the server's packet train, the approximation can be read out and the client's packet train size can be adjusted, if necessary. Since the client needs to wait for the server's approximation, the reaction time for this method is potentially longer than using information about the currently used network technology. For our implementation, we use the minimum of both values as an estimation of the current available data rate.

Not every sent packet is delivered to its receiver, as some packets get lost during transmission. Given that every packet carries the sequence number and the total number of packets of its packet train, we can determine how many packets of a train reached their destination and thus can calculate the packet loss rate P for each packet train using the formula:

$$P = 1 - \frac{n'}{n} \tag{3}$$

For the approximation of the packet transmission delay, we expect that the client's transmission queue is empty and the delay is measured for the first packet in line. In doing so, we measure the network transmission delay d_{total} as the total packet transmission delay of the first received packet of a packet train:

$$d_{\text{total}} = t^{\text{rx}}(f) - t^{\text{tx}}(f) \ .$$

If we were able to calculate $A \neq \text{NaN}$, we can estimate a packet's transmission delay d_{air} for the wireless link:

$$d_{\text{air}} = \frac{s_p}{A}$$

Finally, this enables us to calculate the packet's backbone delay d_{bb}:

$$d_{\text{bb}} = \begin{cases} d_{\text{total}} - d_{\text{air}} & 0 < d_{\text{air}} < d_{\text{total}} \\ \text{NaN} & \text{else} \end{cases} \tag{4}$$

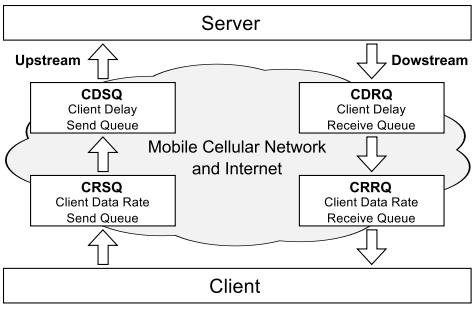


Figure 3: Simulation model for client-server communication.

Now that we have obtained the loss probability P, the available data rate A, and the delays in the wireless link (d_{air}) and the backbone (d_{bb}), we create our simulation model.

IV. Basic Simulation Model

In this section, we show how the measurement traces can be used as a basis for realistic network simulations. We discuss the necessary preprocessing of the measurement data and then introduce our simulation model. Finally, we provide a brief insight into the implementation of our simulator.

A. Spatio-Temporal Preprocessing

Before the collected measurements can be utilized for simulations, they first need to be assigned to geographic positions. During the generation of the traces, position information is recorded once every second. On the basis of this information, the spatio-temporal preprocessing maps each event to the respectively latest measured position.

B. Simulation Model

Based on the measured network characteristics, we simulate the traversal of packets through the network. We use two main assumptions in our model: (1) the wireless link is the *tight link* and (2) the vast majority of packet drops occur on the wireless link.

Given these assumptions, we build our simulation model as shown in Figure 3. We simulate the upstream (left) and the downstream (right) using two queues each, that are traversed sequentially: one queue (CRSQ/CRRQ) simulates the available data rate (and thus indirectly the transmission delay d_{air}) and the packet loss, while the other queue (CDSQ/CDRQ) simulates the backbone delay d_{bb}. In all our trace measurements, we did not observe any out of order packets; therefore, we use FIFO queues, implying that packets cannot overtake each other in the simulation.

1) Upstream: If a packet is sent upstream, it first is queued in the *Client Data Rate Send Queue (CRSQ)*. Once a packet is the head of the queue it waits until its transmission delay d_{air} has passed and then leaves the CRSQ. As client movements often imply changing network characteristics and thus changing available data rates, we need to adjust the calculation of d_{air} whenever the available upstream data rate changes in our simulation. Let $k \geq 1$ be the number of prior changes for the simulated available data rate for this packet and let $d_{\text{air}}(i)$, $A(i)$ for $1 \leq i \leq k$ be the already passed fractions of the transmission time and corresponding available upstream data rates for the packet. Then, the remaining transmission time $d_{\text{air}}(k+1)$ can be calculated as:

$$d_{\text{air}}(k+1) = \frac{s_p - \sum_{i=1}^{k} d_{\text{air}}(i) \cdot A(i)}{A(k+1)} \tag{5}$$

Our event-driven simulation implementation ensures that $s_p > \sum_{i=1}^{k} d_{\text{air}}(i) \cdot A(i)$ always holds.

Once d_{air} has expired, it is determined if the packet is dropped. To this end, we calculate the mean drop probability for this packet using $(d_{\text{air}}(i))_{1 \leq i \leq k}$ and the corresponding spatial drop probabilities $P(i)$ for the same locations:

$$P = \sum_{i=1}^{k} \frac{d_{\text{air}}(i)}{d_{\text{air}}} \cdot P(i) \quad \text{, where} \quad d_{\text{air}} = \sum_{i=1}^{k} d_{\text{air}}(i) \tag{6}$$

If the packet is not dropped, it is enqueued to the *Client Delay Send Queue (CDSQ)*, together with the time of its earliest possible delivery to the server, $t =$ current simulation time $+ d_{bb}$. If the packet becomes head of the queue, it is delivered to the server once t is reached in the simulation.

2) Downstream: The downstream direction is slightly different in respect to the ordering of the queues: a packet first passes the *Client Delay Receive Queue (CDRQ)*, which simulates the backbone delay valid at the time the packet becomes head of the queue. Afterwards, the packet is enqueued into the *Client Data Rate Receive Queue (CRRQ)* which simulates the available data rate, the transmission delay and finally the drop probability for the packet.

C. Dealing with Problems in the Measurement Traces

After some preliminary tests we discovered that there are some rare cases where problems with the measurement traces would lead to significant artifacts in the trace-based simulation.

First, in some situations, the initial method for adapting the packet train size is not fast enough to avoid overloading the network. During those periods of self-induced congestion, we cannot use delay measurements to calculate the backbone delay d_{bb} accurately, because in such situations, d_{bb} includes an unknown queuing delay $d_{\text{queue}} > 0$ that cannot be estimated.

As first approach to alleviate this problem we disregard delay measurements that clearly indicate self-induced congestion: we eliminate any delay measurement that yield a delay of more than one second. Instead of calculating a delay for those situations we reuse the estimated delay of the previous measurement.

Second, certain measurement outliers in our available data rate estimations originate from lag while timestamping due to using a non-real-time OS and from packets arriving with a higher data rate than they were sent with. The latter indicates that the network introduced very heterogeneous queuing delays to different packets of the packet train. To tackle this issue, we filter out all available data rate measurements that are clearly not feasible, e.g. that exceed the limit of the transmission

run	trace direction	duration	single packets packets	ploss	P(ploss)	packet trains trains	tloss	P(tloss)	congested trains con	P(con)	phases	bytes sent
Bus	upstream	6957	103417	5,204	5.03	6958	50	0.72	93	1.34	24	$7.5 \cdot 10^7$
Bus	downstream	6953	103261	379	0.37	6954	18	0.26	135	1.94	30	$1.37 \cdot 10^8$
Tram1	upstream	4972	72259	6,552	9.07	4973	75	1.51	108	2.17	44	$5.11 \cdot 10^7$
Tram1	downstream	4972	73427	1,142	1.56	4973	68	1.37	163	3.28	30	$9.63 \cdot 10^7$
Tram2	upstream	5512	78966	4,440	5.62	5513	22	0.4	241	4.37	46	$5.55 \cdot 10^7$
Tram2	downstream	5512	82256	1,210	1.47	5513	72	1.31	98	1.78	23	$1.08 \cdot 10^8$
Tram3	upstream	5579	81456	1,493	1.83	5580	64	1.15	172	3.08	52	$5.76 \cdot 10^7$
Tram3	downstream	5579	83076	1,705	2.05	5561	103	1.85	106	1.91	23	$1.1 \cdot 10^8$

The first group of three columns identify the measurement run, the communication direction and the duration in seconds. The second group shows the number of packets send and lost and the loss rate. This is followed by the number of packet trains sent, completely lost and the complete loss probability during the trace. The next group shows information on self-induced congestion during the traces. *con* is the number of packet trains during congestion followed by the congestion probability for a packet train in this run and the number of identified congestion phases.

Table I: Statistical data of our measurement traces.

technology (GSM, GPRS, EDGE, UMTS, HSDPA) currently used by the modem. Again, we reuse the results of prior measurements to "estimate" the correct value at the point where we discarded the measurements.

D. Simulation Implementation

Simulating networks can be accomplished using a multitude of simulation paradigms. We opted for a discrete event simulation, as it promises a timely and accurate simulation of every event and because of its linear complexity in the number of simulated events. An event-driven simulator advances the simulation time only between discrete events, as it jumps from one event to the next event in time; this has to be taken into account when simulating applications. One main aim is the easy portability of current *Car-to-X (C2X)* applications to our simulation framework. As many C2X applications—like those in the sim$^{\text{TD}}$ project [11]—are implemented in Java, we chose this programming language for our simulator.

V. Validating the Basic Simulation Model

We validated our trace-based simulation model using the real-world data from our measurement traces.

A. Simulation Setup

To validate our simulation model we take advantage of having logged every sent and received packet with timestamps and packet train parameters during trace generation. Our basic validation idea is to simulate every single measurement using the GPS positions measured as *movement simulation* for a mobile node representing our measurement equipment. While on the other communication side, an Internet node represents our measurement server. Throughout the time the node traverses the GPS trace, we simulate the measurement traffic by injecting the same packets at exactly the same points in simulation time as they were recorded during the measurements. This includes both upstream and downstream measurement packets. Using the preprocessed network characteristics of the same trace as simulation basis, the packets traverse our simulated network. During the simulation, we append timestamped traversal information of the queues to every packet. This, combined with the measurement data of the received packets from the measurement run, enables us to compare the simulated packet flow with the real world measurements. As simulation base we took our four previously mentioned detailed traces summarized in table I.

B. Analysis

In our evaluation, we compare the packet delays and the packet drops of each measured packet with its counterpart in the simulation. As we inject the packets at exactly the same send time into our simulation as they occurred in our measurements, $treal_i^{\text{tx}}(j) = tsim_i^{\text{tx}}(j)$ holds for all packets j of each packet train i. The only difference the simulation can generate is dropping different packets than the simulation and causing different end-to-end delays to the delivery of the packets. Since the determination whether a packet is lost in our simulation is probabilistic, it is improbable that the exact same packets that were lost in the measurement run are lost in the simulation, too. We therefore analyze the following two parameters in our evaluation: first,

$$d_i^{\text{diff}}(j) := tsim_i^{\text{rx}}(j) - treal_i^{\text{rx}}(j) \tag{7}$$

for all packets that are neither lost in reality or simulation and second,

$$drop_{\Delta}(t) := drops^{\text{sim}}(t) - drops^{\text{real}}(t) \tag{8}$$

where $drops^{\text{real}}(t)$ is defined by

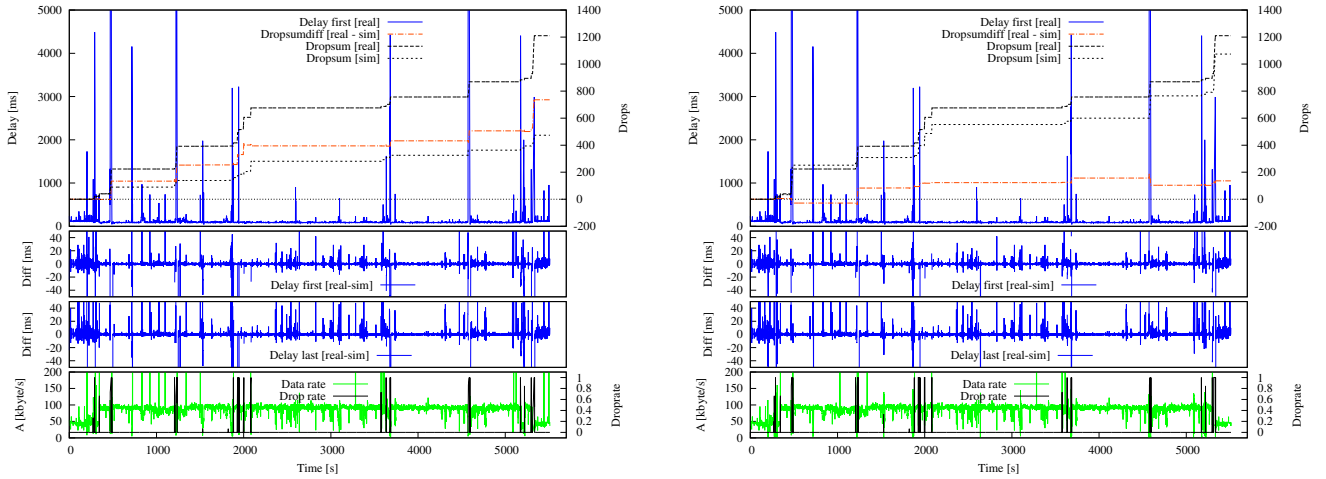
$$drops^{\text{real}}(t) := \sum_{\forall i,j\,:\,treal_i^{\text{tx}}(j) \le t} drop_i^{\text{real}}(j) \tag{9}$$

and $drop_i^{\text{real}}(j) := \begin{cases} 0 & \text{packet } i.j \text{ was received in reality} \\ 1 & \text{packet } i.j \text{ was lost in reality} \end{cases}$

and analogous definitions for the simulation counterparts.

While (7) defines the delay error packet $i.j$ encounters in the simulation, (8) defines the difference in the number of dropped packets in the simulation versus reality up to the time t. As our simulation model evenly spaces consecutive packets as long as no data rate change event occurs and as long as there is no packet loss, we expect that $d_i^{\text{diff}}(j) > 0$ for almost all packets of a train that are not the first or last received packet of a train. If simulation mirrors reality, $d_i^{\text{diff}}(f)$ and $d_i^{\text{diff}}(l)$ should be small or zero.

Figure 4(a) shows the downstream simulations and measurement data of one of our data sets (Tram2). In the topmost chart, $treal_i^{\text{rx}}(f)$ is plotted against the left y axis (Delay [ms]), and $drops^{\text{real}}(t)$, $drops^{\text{sim}}(t)$ and $drop_{\Delta}(t)$ are plotted against the



(a) Basic simulation model.

(b) Enhanced simulation model.

Figure 4: Downstream simulation and measurement data for Tram2.

right y axis (Drops). We capped the delay graph at 5000 ms—the three spikes top out at about 16, 14 and 17 seconds. The two middle charts show $d_i^{\text{diff}}(f)$ and $d_i^{\text{diff}}(l)$ while the bottom chart shows the measured data rate and drop rate. Visual inspection clearly shows that our basic simulation model works well in general but has problems to mirror the real packet drop rates in cases of high self-induced congestion. Closer examinations showed that the number of dropped packets is underestimated significantly in those situations, in particular if complete packet trains are lost. In all other situations the simulated packet losses and delays match those of our real-world measurements quite closely. The top chart of Figure 6 shows that the problem of underestimated packet-drop rates is less severe in the upstream direction.

VI. Improvements

The basic simulation model faces two limitations: (1) it underestimates the pack loss rates in the simulation for situations where the real-world data indicates massive packet loss and (2) it uses a rather unflexible ad-hoc approach to deal with self-induced congestion for the delay measurements. For both aspects we now present improvements.

A. Interpolating Timestamps for Lost Packet Trains in the Downstream

In the downstream direction, we use the receive timestamp of the first received packet of each packet train to estimate the location at which the measured data rate and drop rate was observed. For packet trains that were lost entirely, no such timestamp exists and we must find an approximation for the correct timestamp for the beginning of the 100% packet loss. We chose linear interpolation to estimate the missing timestamps using the timestamps of the neighboring packet trains that were not lost as references. In the upstream direction, we do not face this problem as we use the send timestamps in that direction.

B. Available Data Rate for Lost Packet Trains

The measured data rate for lost packet trains surely is $A = 0$. However, the simulator needs to know how many packets (or bytes/s) it has to drop during a complete loss phase, otherwise the send queue will be emptied too fast or too slow. While we have to investigate in future work if it is possible to derive a better approximation for the data rate using the information gained in Section VI-C we opted for $A = 2 \cdot n \cdot s_p$ as a first attempt to improve the drop rate estimation.

C. Detecting Self-Induced Congestion

In order to deal with phases of self-induced congestion in a more fundamental way than before, we first need a way to identify those phases.

Let $t_i^{\text{tx}}(l)$ be the send timestamp of the last packet of sequence i that started without congestion and $t_{i+1}^{\text{tx}}(f)$ be the send timestamp of the first packet of the following packet train. Ideally, this would allow the use of the formula

$$con(i+1) = \begin{cases} \text{true} & t_{i+1}^{\text{tx}}(f) < t_i^{\text{tx}}(l) + d_{\text{air}}(i.l) \\ \text{false} & t_{i+1}^{\text{tx}}(f) \geq t_i^{\text{tx}}(l) + d_{\text{air}}(i.l) \end{cases} \tag{10}$$

to detect a congestion start ($con(i+1)$ is true and $con(i)$ is false) or end ($con(i+1)$ is false). Unfortunately, we neither know the air delay $d_{\text{air}}(i.l)$ of the last packet of i nor do the send timestamps reflect the exact time when the packet left the device. Instead, they indicate when our application handed it over to the network stack. As an approximation for $d_{\text{air}}(i.l)$ we can use $d_{\text{air}}(i)$. But still, all but the send timestamps of first packets of packet trains that started in a congestion-free phase are incorrect and we need to apply a correction to those false timestamps. Let $t_i^{\text{tx}}(f)$ be correct, i.e. the send queue was empty when $i.f$ was enqueued. With our 1 Hz measurements, this also implies that $t_i^{\text{rx}}(l) - t_i^{\text{tx}}(f) > 1s - d_{\text{air}}(i)$ holds. Using the receive timestamps, the corrected $t_i^{\text{tx}}(l)$ is:

$$tcorr_i^{\text{tx}}(l) = t_i^{\text{tx}}(f) + t_i^{\text{rx}}(l) - t_i^{\text{rx}}(f) \tag{11}$$

This allows us to replace $t_i^{\text{tx}}(l)$ with $tcorr_i^{\text{tx}}(l)$ in equation (10) to decide if the network queue was empty when sending packet $i+1.f$:

$$con(i+1) = \begin{cases} \text{true} & t_{i+1}^{\text{tx}}(f) < tcorr_i^{\text{tx}}(l) + d_{\text{air}}(i.l) \\ \text{false} & t_{i+1}^{\text{tx}}(f) \geq tcorr_i^{\text{tx}}(l) + d_{\text{air}}(i.l) \end{cases} \quad (12)$$

If $con(i+1) = true$ we are in a congestion phase and now have to correct $t_{i+1}^{\text{tx}}(f)$ to

$$tcorr_{i+1}^{\text{tx}}(f) = tcorr_i^{\text{tx}}(l) + d_{\text{air}}(i) \quad (13)$$

This correction is repeated for each consecutive packet train until $con(i+1) = false$ again — which marks the end of the congestion phase and the need to correct $t^{\text{tx}}(f)$ ends.

If $d_{\text{air}}(i)$ cannot be measured, because less than two packets of train i arrived, we approximate it using the last valid available data rate measurement $A(i-1)$ and the packet size $s_p(i)$ of the train i. As $d_{\text{air}}(i)$ is derived from $A(i)$, which is measured over the time span of the complete packet train i, it is only an approximation. Its repeated use might lead to added errors in long congestion phases and thus might lead to $tcorr_{i+n}^{\text{tx}}(f)$ being larger than $t_{i+n}^{\text{rx}}(f)$. To solve this, we altered the correction of $t_{i+1}^{\text{tx}}(f)$ to

$$tcorr_{i+1}^{\text{tx}}(f) = \min \begin{pmatrix} tcorr_i^{\text{tx}}(l) + d_{\text{air}}(i), \\ t_{i+1}^{\text{rx}}(f) - \dfrac{s_p(i+1)}{A(i)} - d_{bb}(i) \end{pmatrix} \quad (14)$$

D. Improving the Delay Estimates

The ability to determine phases of self-induced congestion without using an arbitrary constant enables us to implement a much more flexible correction for delay measurements than the one introduced in Section IV-C. We can now fix the delay measurements during all phases of self-induced congestion and not just those indicated by a rather arbitrary constant. As in the basic simulation model, we replace the delay values measured during self-induced congestion with the last valid value preceding it.

E. Splitting of Packet Trains

Self-induced congestion periods always start with a few packet trains that were sent with a higher data rate than the medium offers at that moment. This leads to packet trains where the first packet of the train arrives far more than one second before the last. This imposes three problems for our simulation: First, the frequency of the measurements decreases (recall that we try to generate 1 Hz measurements with moving nodes). Second, the send timestamps of the first packet of each train during a congestion phase cannot be used for delay measurements. Third, as we also use the send timestamp to derive the location of the packet for the upstream direction, we map the measurements to the wrong location (as we assume they were sent before they actually left our network device). To reduce these effects, we altered the examination of our measurement traffic and split packet trains with more than one second *train arrival time* $\Delta_{\text{train}} := t_i^{\text{rx}}(l) - t_i^{\text{rx}}(f) > 1\text{s}$ into sections of approximately one second length. Figure 5 shows an example of a packet train with sequence number one consisting of eight packets (1.1 to 1.8), that was split into three sections.

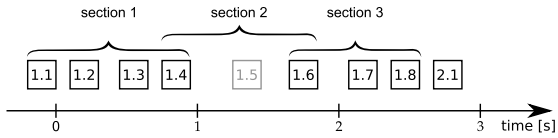


Figure 5: Splitting of the packet train with sequence 1.

We split packet trains using the following five rules: (1) If less than two packets of the packet train arrived, then the packet train is not split up. (2) A section contains at least two packets that arrived at the destination. (3) A lost packet (like 1.5 in Figure 5) cannot be the last packet of a section except for the last section of a packet train. (4) If rules 1-3 are fulfilled and $\Delta_{\text{section+next not lost packet of the train}} > 1\text{s}$ create a new section following rule 5, otherwise add the packet to this section and repeat rule 4 until all packets are in sections. (5) The last packet of each section serves as the first packet of the following section.

By treating each section as a separate packet train we gain more measurement points for available data rate and packet loss probability. This does not alter the number of usable delay measurements, though. Without dividing packet trains in sections, we use the send timestamp of the first packet of a train in the upstream to match it to a location. As the first packets of each section (except for the first section if $con(train) =$ false) were sent during congestion the improved version uses (11) and (13) with $f :=$ first packet of a section and $l :=$ last packet of a section to correct those timestamps.

VII. Evaluation of the Enhancements

We validated the improved simulation model with the evaluation setup as in Section V. Comparing the results of our simulation model with those with improved preprocessing in figure 4(b) indicates that the simulation of packet losses during self-induced improved significantly. The remaining differences originate in the uncertainty of the available send data rate during 100% packet loss phases. Overestimating the send data rate results in too many packets being dropped and a send queue that decreases too fast, resulting in lower simulated packet delays. Underestimating the send data rate on the other hand leads to too few packets being dropped in the simulation and a queue building up larger than during our measurements.

A comparison of the two graphs of Figure 6 reveals, that the small differences of the drop sums $drops^{\text{sim}}(t)$ and $drops^{\text{real}}(t)$ shown with our basic simulation model are almost completely resolved using the enhancements described the last section. The statistical data of all four measurements in table II shows, that out of 78966 packets in the upstream, the improved simulation model dropped 4440 packets in the simulation out of 4442 in the real measurements, only a ratio of 0.0000025 of all packets was not dropped correctly. Out of the 82256 packets in the downstream, the improved simulation model dropped 1062 out of 1210, thus a ratio of 0.0018 of all packets were not dropped correctly.

trace		send vs. dropped			real delay				simulated delay				sim-real delay			
run	dir	packets	rdrop	sdrop	rmin	rmax	ravg	r95	smin	smax	savg	s95	dmin	dmax	davg	d95
Bus	up	103417	5204	5125	51	18516	216	304	51	45729	299	305	−2609	26222	31	70
Bus	down	103261	379	401	37	11227	274	308	19	11769	254	299	−5598	2012	−19	35
Tram1	up	72259	6552	6302	49	22549	379	563	49	14228	292	586	−19815	8933	−93	98
Tram1	down	73427	1142	876	36	34676	433	525	16	19856	427	586	−19758	9175	−10	44
Tram2	up	78966	4440	4442	49	38371	412	562	50	31100	336	566	−21908	2231	−79	95
Tram2	down	82256	1210	1062	46	18307	282	465	38	12862	277	452	−17125	8329	−6	42
Tram3	up	81456	1493	1465	48	50742	484	607	48	14016	296	650	−47703	2983	−188	100
Tram3	down	83076	1705	1074	37	22919	311	398	38	22927	388	434	−3109	9219	20	41

Columns one and two identify the trace and the communication direction. The following three column groups show statistical information of the real (measured) packet delays, the simulated delays and the differences of both. The *95 columns show the 95percentile. While dmin, dmax, davg were gained using real-sim delays, d95 was gained using |sim-real|.

Table II: Statistical data of the simulation showing packet delays and drops.

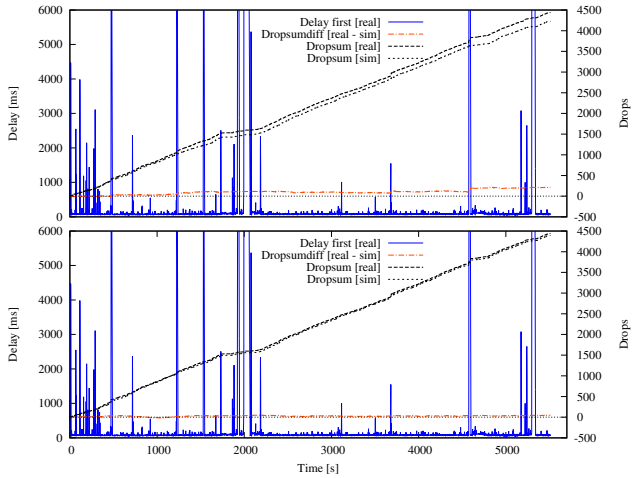


Figure 6: Upstream simulation and measurement data for "Tram2". The top shows the basic simulation model, the bottom the simulation model with improved preprocessing.

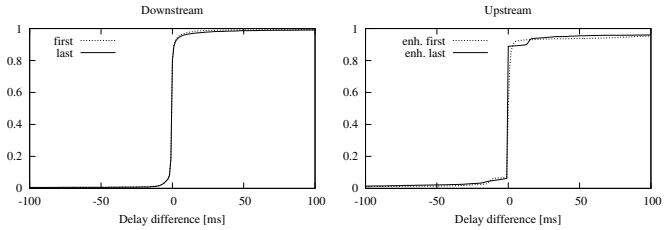


Figure 7: CDF plots of the differences in packet delay measurement versus simulation.

The cumulative distribution functions of the packet delay differences of our measurements and simulation results displayed in Figure 7 show that 87% of the downstream packets and 82% of the upstream packets are off by at most 3 ms.

The higher number of outliers in the upstream are the result of much longer self-induced congestion phases in the upstream — we measured up to 50 seconds delay for some packets in "Tram3" upstream. These rare cases of extreme self-induce congestion (between 1.34% and 4.37% of the packet trains are send in congestion phases, see table I) lead to errors in the estimation of packet loss and delays. In future work, we will focus on improving the *measurement* of the traces to detect self-induced congestion earlier and thereby reduce the negative impact of it on the simulation model.

VIII. Conclusions and Future Work

In this paper, we introduced a novel trace-based simulation model for C2X-communication using mobile cellular networks. We validated the simulation results by simulating our measurement traffic and comparing the results to the real world packet flow. The evaluation revealed two challenging situations caused by inaccurate measurements: self-induced congestion with high delays and phases with 100% packet loss.

We then introduced several improvements to correct false timestamps and filter incorrect delay measurements. Furthermore, we split packet trains with large *train arrival times* to gain more measurement points. Finally, we showed that all these improvements enhanced the simulation results significantly.

For the future, we plan to improve our measurement framework to further reduce the number and length of self-induced congestion phases, to extend our simulation model to properly estimate available data rates for multiple users of a network cell, and to release an OMNet++ implementation of our simulation model.

References

[1] S. Keshav, "A Control-Theoretic Approach to Flow Control," in *Proceedings of ACM SIGCOMM*, 1991, pp. 3–15.

[2] R. Carter and M. Crovella, "Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks," Boston, MA, USA, Tech. Rep., 1996.

[3] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?" in *Proceedings of IEEE INFOCOM*, 2001, pp. 905–914.

[4] B. Melander, M. Bjorkman, and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proceedins of GLOBECOM*, 2000.

[5] M. Jain and C. Dovrolis, "Pathload: A measurement tool for end-to-end available bandwidth," in *Proceedings of Passive and Active Measurements Workshop*, 2002, pp. 14–25.

[6] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "Pathchirp: Efficient available bandwidth estimation for network paths," in *Proceedings of Passive and Active Measurement Workshop*, 2003.

[7] N. Hu, S. Member, P. Steenkiste, and S. Member, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 879–894, 2003.

[8] E. Goldoni, G. Rossi, and A. Torelli, "Assolo: an Efficient Tool for Active End-to-End Available Bandwidth Estimation," *International Journal on Advances in Systems and Measurements*, vol. 2 no 4, pp. 283–292, 2009.

[9] "gpsd: a GPS service daemon.https://savannah.nongnu.org/projects/gpsd."

[10] "NTP: The Network Time Protocol. http://www.ntp.org."

[11] C. Weiß, "V2x communication in europe – from research projects towards standardization and field testing of vehicle communication technology," *Computer Networks*, vol. 55, no. 14, pp. 3103 – 3119, 2011.