Computer Communications 97 (2017) 81–95

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Moving measurements: Measuring network characteristics of mobile cellular networks on the move

Norbert Goebel*, Tobias Krauthoff, Kalman Graffi, Martin Mauve

Department of Computer Science, University of Düsseldorf, Germany

ARTICLE INFO

Article history:
Received 1 December 2015
Revised 9 June 2016
Accepted 11 June 2016
Available online 14 June 2016

Keywords:
Available data rate
Available bandwidth
Network measurement
Mobile cellular network

ABSTRACT

Trace-based simulation of mobile cellular communication requires valid measurements of real-world delay, available data rate and packet loss. These network characteristics need to be measured simultaneously and with high frequency. Previously introduced measurement methods either do not combine the measurement of all three key network characteristics, suffer from low measurement frequencies or are not able to adapt to rapidly changing network conditions. In this paper we propose a novel measurement method that allows the simultaneous measurement of those three key characteristics for both communication directions while moving at high speed. Furthermore it is able to adapt to rapidly changing network conditions.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Understanding the behavior and impact of a given *Car-to-X (C2X)* application is a challenging problem. One promising way of addressing this problem is trace-based simulation, where real world network measurements form the basis for a simulation environment that is used to execute real C2X applications.

A key challenge in trace-based simulation is how to measure the network characteristics, in particular in infrastructure-based networks. This is a hard problem since it has to observe three main constraints:

- Frequent measurements
- Simultaneous measurements of delay and available data rate
- Rapidly changing network conditions

On the one hand, frequent measurements are needed since mobile networks can rapidly change their characteristics. On the other hand, mobile networks have very limited capacity. Thus frequently measuring their characteristics can easily lead to massive distortions in what is measured. In the worst case the measurement traffic itself causes significant congestion in the network — a situation which we call self-induced congestion. Furthermore valid delay measurements require that no previous measurement packets are in the network any more when the delay probe is sent. But arbitrarily long gaps in the measurement traffic conflict with the need of frequent measurements.

Especially the current measurement methods of available data rates, which often are named “available bandwidth measurements”, cannot handle rapidly changing network conditions or render simultaneous delay measurements impossible.

The key contributions of the paper at hand are twofold:

(1) The proposal of a novel measurement method that allows frequent and simultaneous measurement of delay, available data rate and packet loss for both communication directions while moving. Furthermore, our implementation is able to adapt to rapidly changing network conditions and thus allows generating accurate position-based network traces of conventional cellular networks while moving. To reach this goal we had to eliminate — or at the very least minimize — the impact of self-induced congestion.
(2) The evaluation of our measurement implementation, showing that all of the identified challenges induced by the high measurement frequency, simultaneous available data rate and delay measurements and rapidly changing network conditions are indeed solved.

In the remainder of this paper, we give a brief overview of related work in Section 2. We outline our basic measurement methodology in Section 3 and point out arising challenges and our solutions in Section 4. In Section 5 we explain implementation details of our measurement framework. In Section 6 we further present a short overview of the used measurement equipment which we used for the evaluation presented in Section 7. We conclude this paper in Section 8.

* Corresponding author.
E-mail addresses: goebel@cs.uni-duesseldorf.de (N. Goebel), krauthoff@cs.uni-duesseldorf.de (T. Krauthoff), graffi@cs.uni-duesseldorf.de (K. Graffi), mauve@cs.uni-duesseldorf.de (M. Mauve).

http://dx.doi.org/10.1016/j.comcom.2016.06.002
0140-3664/© 2016 Elsevier B.V. All rights reserved.

2. Related work

We aim at providing a solution for the simultaneous measurement of application level available data rates, delays and loss rates for cellular networks with mobile devices. We therefore focus on end-to-end measurement methods that require no insights into the mobile cellular network but treat it as a black box. Solely measuring the one-way delay is straight forward. And, as long as the *Transmission Control Protocol (TCP)* is not used to transfer the measurement traffic, the loss rate can be measured without significant overhead, too. The measurement of end-to-end available data rate on the other hand is an interesting research challenge, which has been the focus of significant prior research. All relevant existing methods either utilize *packet pairs* — two packets sent back-to-back — or *packet trains* — a sequence of measurement packets sent with varying gaps between consecutive packets. They can be divided into three groups using either the *Probe Gap Model (PGM)*, the *Probe Rate Model (PRM)* or *Bulk Traffic*.

2.1. Probe Gap Model (PGM)

Tools like Spruce [1], IGI [2] and Delphi [3] that use the Probe Gap Model use the temporal gaps (*packet dispersion*) of two consecutive packets sent from a sender to a receiver to estimate the available data rate of the network path. While PGM is able to measure the available data rate faster than any other method, it suffers from a high variance in the measurements. Thus multiple estimates with some kind of smoothing are needed to achieve more stable estimates. Furthermore, the capacity of the network path needs to be known a priori to estimate the available data rate, which is not possible in black box scenarios. These drawbacks leave PGM unsuitable for black box measurements in mobile cellular networks.

2.2. Probe Rate Model (PRM)

PRM systems like TOPP [4], Pathload [5], Pathchirp [6] and Assolo [7] use packet trains to estimate the available data rate of a network path. Each packet train consists of packets starting with large temporal gaps between consecutive packets that decrease towards the end of the packet train. Thereby the send rate is increased during the transmission 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 bottleneck link of the network path and *self-induced congestion (SIC)* occurs. This leads to increasing delays of the packets and the receive rate eventually will be smaller than the send rate. Assuming constant cross traffic, the available data rate can be estimated by locating the point at which the receive rate exceeds the send rate. As the send rate is raised step-by-step the process either needs to be repeated to enhance the estimation accuracy or the send rate has to be raised in smaller steps which leads to a longer measurement duration. During the complete measurement cycle the cross traffic needs to be constant and the network characteristics must not change. This makes PRM unsuitable for mobile measurement scenarios where network conditions change frequently and rapidly.

2.3. Bulk traffic measurements

The most promising way to measure available data rates in cellular networks is using bulk traffic measurements. Using packet trains of packets sent back-to-back, the measured available data rate of the network path equals the receiving rate. In contrast to PRM and PGM methods, bulk traffic measurements do not rely on constant network characteristics during each measurement period. Instead, the measurement represents the average available data rate during the transfer time of the complete packet train. The downsides of this approach are the large amount of data sent per measurement and the possibility to overload network queues in the network path, which might lead to dropped packets.

The easiest, but error prone, way to measure the available data rate using bulk traffic is a simple file upload or download. Here the available data rate equals the size of the file divided by the transfer time. Typically such measurements are conducted using the *hypertext transport protocol (HTTP)* or the *file transfer protocol (FTP)*, which use TCP as the transport protocol. TCP employs congestion avoidance, which minimizes packet loss induced by the measurement. On the other hand, congestion avoidance on a loss prone mobile cellular network connection can lead to dramatic underestimation of the available data rate. Furthermore, TCP guarantees packet delivery by means of retransmissions. Therefore these methods can not be used to determine the loss rate. Additionally, the retransmissions also influence the data rate measurements on the application level as TCP delivers the received segments in order. While waiting for a retransmission, the application level receiving rate drops to zero until the missing segment arrives. When it arrives, TCP hands this segment and all further received consecutive segments over to the application. This raises the application level receive rate far above the receiving rate of the underlying network device.

To lower the impact of these TCP bulk traffic problems, more sophisticated tools like the widely used ookla speedtest.net [8,9] were developed. Speedtest.net first uses ping probes to choose the nearest measurement server. For the speed test it uses up to 8 simultaneous HTTP streams. Thus one or multiple stalling TCP connections have minimal influence on the measurement. The received throughput samples are divided into 20 slices. The highest 10% and the lowest 30% of the measured throughput slices are discarded. The presented result is the average of the remaining 60% of the measurement slices. As stated by ookla their aim is not the measurement of the available data rate but the measurement of the maximum sustainable HTTP throughput between the client and the measurement server. Each measurement needs multiple seconds to finish. In addition to the throughput measurement the *round trip time (RTT)* between the client and server is measured while estimating the best measurement server — based on the lowest ping. Packet loss rates can not be measured by speedtest.net.

Another commonly used bulk measurement tool is *iperf*. It is available for multiple platforms and included in the packet repositories of all major Linux distributions. Currently two major versions of iperf exist: iperf2 [10] and iperf3 [11]. Both offer similar measurement methods and can either measure TCP or UDP throughput. As TCP throughput is limited by many things like loss, congestion, out of order delivery and buffer starvation, and as C2X messages utilize UDP, we focus on iperf's UDP throughput measurement algorithm. Iperf is client/server based. On one end a server needs to listen for measurement requests. On the other end the client is started with parameters defining the desired sending rate, the overall measurement time and the sampling rate of the measured throughput results. The UDP throughput is estimated as received bytes per time-period. It is obvious that the sending rate, configured at measurement start, restricts the maximum receiving rate that can be measured, as the gaps between consecutive packets typically do not shorten during network traversal. Thus, to be able to measure the maximum achievable throughput, the connection's capacity has to be used as the sending rate. Alternatively multiple consecutive measurements with rising sending rates have to be conducted — which would mimic the probe rate model. If the sending rate exceeds the current available data rate of the

Table 1
Notation table.

Notation	Section	Description
RMF	5	Rate Measurement Framework — the measurement software presented in this paper
pre RMF	7	The predecessor of the RMF which showed the need for a more advanced measurement algorithm with faster feedback loops
s_i^k	3.1	kth packet of chirp i
p_0	3.1	Constant size of the first packet of each chirp
p_i	3.1	Size for packets $s_i^2..s_i^{n_i}$ of chirp i
n_i	3.1	The number of packets of chirp i
d_i	3.2	Delay of chirp i (precisely of s_i^1)
d_{last}	4.3	The currently last valid measured delay
A_i	3.3	The available data rate measured by chirp i
lr_i	3.3	Loss rate if chirp i
ICSIC	4	*Inter-chirp self-induced congestion* occurs if sending queues do not empty completely between chirps
Tail drop	4	Dropping of large amounts of packets at the tail of the chirps
t^{guard}	4.1	Guard time (silence) introduced between chirps
MI	4.1	The *measurement interval* is the period between the transmit times of the first packets of consecutive chirps
UT	4.1	The *utilization* is the fraction of the *MI* used by measurement traffic
$chirpsize_i$	4.1	Length of a chirp in bytes
t^{tx}	4.2	Transmit time of a packet
t^{rx}	4.2	Receive time of a packet
t^{etx}	4.2	Estimated transmit time of a packet avoiding ICSIC
t^{erx}	4.2	Estimated receive time of a packet
CC	5	*Control channel* — TCP, 1st physical link - used for parameter exchange at the beginning of a measurement
MC	5	*Measurement channel* — UDP, 1st physical link — transports measurement packets
BC	5	*Back channel* — virtual channel inside the payload of the measurement packets used for feedback
SC	5	*Safeguard channel* — UDP, 2nd physical link — transports dynamic payload only

connection, the buffers at the network device before the link with the lowest available data rate will be filled — self-induced congestion occurs. Eventually, the buffer's capacity is reached and packets are dropped. Iperf has no way of telling if the dropped packets — which it recognizes — were dropped due to transmission errors or due to overflowing network queues. As iperf does not offer any option to insert gaps in the measurement traffic to assure empty network queues, it is not suitable to measure neither loss rates, nor delays in highly volatile networks like cellular networks.

But while bulk traffic measurements need some modifications to be usable for combined data rate, delay and loss rate measurements, they are still much better suited for measurements under fast changing network conditions than PRM and PGM methods.

In an earlier paper [12], which introduced a novel simulation model for trace-based simulation of C2X-communication using cellular networks, we presented and evaluated a basic bulk traffic measurement algorithm. While this algorithm worked well in many situations, it had significant problems with self-induced congestion due to growing and overflowing network queues inside the network. In the paper at hand, we solved this major challenge by introducing multiple feedback loops, an inter-chirp self-induced congestion detector and a second physical network connection.

3. Basic measurement methodology

Our measurement setup consists of a mobile client using cellular access and an Internet homed server. In the following subsections, we explain how the traffic used for the measurement is composed and how this measurement traffic is used to determine the main network characteristics. Table 1 highlights the notations introduced and used for this purpose.

3.1. Construction of a measurement chirp

As a foundation for our measurements we use bulk traffic: we adapt the idea of a *chirp*, a short packet-train, for our purpose. Fig. 1 shows a single packet train *(chirp)* of packets send back-to-back. Each chirp i consists of n_i packets $s_i = (s_i^1, .., s_i^{n_i})$. The first packet of each chirp (s_i^1) has a size of $p_0 = 126$ Bytes in our implementation. All $n_i - 1$ other packets of the chirp (s_i^j with $2 \leq j \leq n_i$) have the same size p_i which can be larger than p_0 but do not exceed the *Maximum Transmission Unit (MTU)*. This allows us to alter the number of measurement packets per chirp (n_i) as well as the size of the packets (p_i) of a chirp for every single chirp in order to adapt to changing network conditions. Each chirp is used to measure the available data rate, the loss rate and the delay of the network path.

3.2. Measuring delays

Measuring the *round trip time (RTT)* of a packet does not suffice for delay measurements as the up- and downstream in mobile cellular networks often have very different characteristics. We therefore rely on well synchronized clocks in both measurement devices. As the up- and downstream delay in mobile cellular networks seldom are symmetric, using the *Network Time Protocol (NTP)* [13] to synchronize the mobile node against an Internet homed timeserver via the cellular network is no suitable option (compare [14]), as half of the difference in up- and downstream delay would be added to the NTP time estimation as an error. Equipping the mobile node with a *pulse per second (PPS)* capable GPS receiver solves this by making it a stratum-1 timeserver itself. GPS receivers with PPS support issue a pulse precisely at the start of each second. This pulse is used to trigger a hardware interrupt at the connected computer. Gpsd [15] and ntpd (the GPS and NTP daemons) use this precise timing in conjunction with the jittery reception of the timestamps in the GPS data to achieve a time synchronization accuracy of a few microseconds. No external communication except the reception of the GPS signal is needed to achieve this accuracy. The Internet homed server instead synchronizes against stratum-1 timeservers using its gigabit network connection.

Stabilized time synchronization of both nodes is crucial for the delay measurements. It takes some time to achieve sub-millisecond synchronization accuracy after system power-on. Especially the mobile nodes need 15 to 60 minutes after their initial GPS fix to reach sub-millisecond accuracy. To assure good time synchronization of the mobile node and the server we constantly monitor the synchronization offset reported by NTP using *ntpq*. No measurement is started if the offset is not less than 0.5 ms for at least

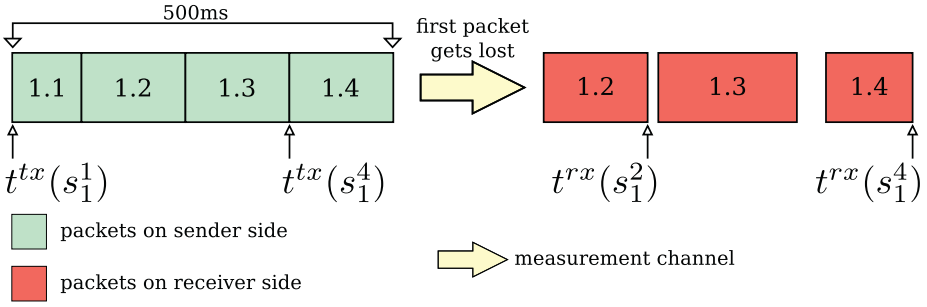


Fig. 1. Structure of a measurement chirp.

one minute. Measurements would be aborted if the reported offset rises above 5 ms — the latter never happened during our measurements. As the measurement server typically runs without interruption, its sync-offset typically ranged between 0.0 and 0.01 ms. The sync-offset of the mobile node typically ranged between 0.0 and 0.5 ms with a clear tendency towards the lower end the longer the mobile node was switched on.

Assuming these stabilized time synchronization and a stable GPS fix, a clock difference of a few microseconds can be achieved with this setup. Even short losses of the GPS fix do not significantly harm the time synchronization. During phases of good GPS reception the NTP daemon calculated an accurate drift of the local clock which leads to a quite accurate computer clock. We thus have a highly accurate clock and gain position information as a side effect.

Using synchronous clocks in the mobile node and the server allows us to use the transmit and receive timestamps to calculate the one-way application-to-application delay. Therefore the sender embeds the *transmit timestamp* t^{tx} in each packet and the receiver records a *receive timestamp* t^{rx} for each received packet. Each packet further carries the chirp-id i identifying the chirp and a packet-id $\in [1..n_i]$ identifying the packet in the chirp.

Assuming that the packet s_i^1 — the first packet of chirp i — is sent when all send network queues at the sender are empty, the transmit and receive timestamps of this packet can be used to calculate the one-way delay as:

$$d_i = t^{rx}(s_i^1) - t^{tx}(s_i^1) \tag{1}$$

Without empty sending queues, d_i includes an unknown queuing delay d_{queue}. In Section 4.2 we describe in detail how we ensure empty sending queues at the start of a chirp.

3.3. Measuring data rates and loss rates

Let $s_i = (s_i^1, .., s_i^{n_i})$ be the n_i packets send back-to-back of chirp i and let $r_i = (r_i^1, .., r_i^{m_i})$ be the m_i received packets of this chirp with $m_i \leq n_i$. As the packets are send back-to-back, all consecutive Bytes of a chirp are assumed to leave the sending interface without temporal gaps between each other. Thus the receive rate is the available data rate for this network path (in the sending direction). As long as at least two packets of a chirp arrive, the receive timestamps of the packets of chirp i plus the knowledge of the packet sizes allow us to estimate the available data rate A_i as the fraction of received Bytes per time period:

$$A_i = \frac{p_i \cdot (m_i - 1)}{t^{rx}(r_i^{m_i}) - t^{rx}(r_i^1)} \tag{2}$$

As the receive timestamp always marks the reception of a complete packet, the size of the first received packet is irrelevant in the aforementioned received Bytes as they do not belong to the measurement period.

If none or just one packet of a chirp arrives, no feasible estimation on the available data rate is possible.

Calculating the loss rate is straight forward as we just need to calculate the fraction of packets per chirp that were lost:

$$lr_i = 1 - \frac{m_i}{n_i} \tag{3}$$

4. Challenges in cellular networks with mobile devices

While the basic measuring methods for delay, available data rate and loss rate, work well in wired networks, they suffer from fast changing network characteristics and high loss rates in cellular networks with mobile devices. In order to enable trace-based simulation and to account for the fast changes of network characteristics in cellular networks with mobile devices, the characteristics of the network need to be measured frequently. Unfortunately the frequent simultaneous measurement of delay and available data rate poses a significant challenge. On the one hand, the sending queue needs to be empty before the delay measurement packet is sent (compare Section 3.2). On the other hand, we have to send sufficiently large chunks of data for the data rate measurements. Therefore the amount of data sent per chirp needs to be chosen very carefully to allow fast measurements and, simultaneously, still induce enough traffic into the mobile cellular network to gain high data rates from the Radio Network Controller (in 3G) or the Base Station Controller (in 2G).

The major challenge is to choose the right length of each chirp and the length of the gaps between the measurement chirps to avoid *inter-chirp self-induced congestion (ICSIC)* — the sending queues are not empty at the start of the next chirp — but still allow the insertion of enough measurement data into the network while keeping a high measurement frequency.

The second challenge — which partly is introduced by using bulk traffic measurements — are increased drop rates for the tails of our chirps caused by overflowing network queues. As we can not distinguish between dropped packets originating from a bad cellular network connection and those caused by overflowing network queues we have to eliminate the latter to gain correct loss rate estimates.

In the remainder of this section we present our solutions to address these challenges.

4.1. Measurement frequency and guard times

A first approach to prevent inter-chirp self-induced congestion is the division of each measurement period into a fixed amount of time where the chirp is sent, followed by a fixed guard time t^{guard} (silence). One reasonable but arbitrary choice could be 500 ms for both intervals in order to reach a measurement frequency of 1 Hz. In most cases an *utilization (UT)* of 50% would allow good estimations on available data rates and loss rates. The additional guard

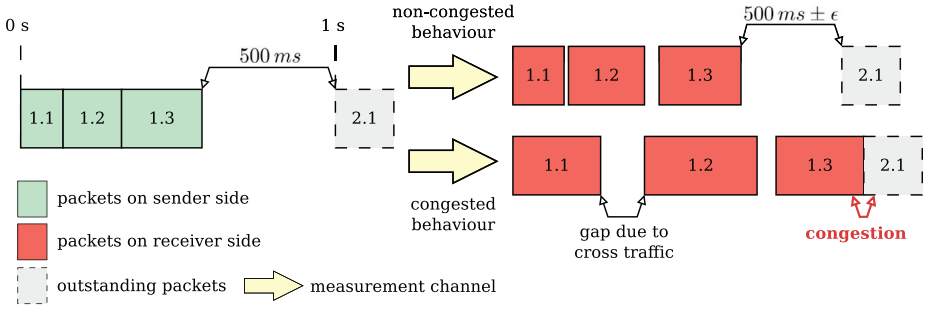


Fig. 2. Example for congested and uncongested chirps.

time would — usually — allow any queues built up in the network path to be empty again when a new chirp starts. An example with *measurement interval* $MI = 1000ms$ and an *utilization* $UT = 0.5$ is depicted in the upper part of Fig. 2.

Unfortunately the volatile network conditions make it hard to achieve this ideal situation. We simply do not know the available data rate a priori and thus can not assure perfect intervals. Instead we use the payload of our measurement packets of the opposite direction to piggyback the previous available data rate measurement as a feedback to adjust the size of the next measurement chirp i:

$$chirpsize_i = p_0 + (n_i - 1) \cdot p_i \approx A_{i-1} \cdot MI \cdot UT \tag{4}$$

We start sending chirp $i+1$ exactly after the measurement interval of chirp i ended. In our experience with real-world deployments this works reasonably well for most situations.

However, the lower part of Fig. 2 shows a situation where this approach fails. In this example, the current available data rate is lower than half of the estimated available data rate that we received via piggybacking. This severe data rate reduction leads to *inter-chirp self-induced congestion ICSIC* — the network queues are still filled with measurement packets of the previous chirp when the first packet of the current chirp is sent. This renders delay measurements during ICSIC invalid and changes the measurement frequency, which is reduced below 1 Hz during the build-up of the queues and raised above 1 Hz until the queues are empty again.

In [12] we identified ICSIC periods of up to 50 seconds in our measurements. We further on identified that 1% to 5% of all chirps in our traces suffered from ICSIC. In the following three sections we present our solution to prevent — or in some cases at least shorten — ICSIC. This is followed by three sections explaining how we solved the tail drop problems.

4.2. Inter-Chirp Self-Induced Congestion detector

To eliminate or at least reduce ICSIC in phases of severe data rate reduction we developed an ICSIC-detection algorithm. The basic idea of this algorithm is to estimate the sending time of the first packet of each chirp in a way that significantly raises the chance of empty network queues at that time while still maintaining a high measurement frequency.

Without loss of generality, we describe our algorithm for the downstream measurements. Thus the server sends the measurement packets and the mobile node receives them.

Let us assume the mobile node just received the kth packet of chirp i (s_i^k) with $1 \le k \le n_i$ as shown in Fig. 3. This triggers the calculation of a new congestion free transmit time of the first packet of chirp $i+1$ ($t^{etx}(s_{i+1}^1)$). If $t^{etx}(s_{i+1}^1)$ is later than the normally scheduled transmit time $t^{tx}(s_{i+1}^1) := t^{tx}(s_i^1) + MI$, it is transmitted piggybacked in the upstream measurement payload to the server. The server then can adjust the sending time accordingly and, if our estimation is correct, send the first packet of chirp $i+1$ after the ICSIC phase is over.

As the starting point for the estimation of $t^{etx}(s_{i+1}^1)$ we use the delay calculation formula 1, which is equivalent to:

$$t^{tx}(s_{i+1}^1) = t^{rx}(s_{i+1}^1) - d_{i+1} \tag{5}$$

Here, assuming no congestion occurred, d_{i+1} is the downstream delay for the packets of chirp $i+1$. As we do not know the future one-way delay d_{i+1} we need to approximate it with the last measured downstream one-way delay in a non-congested situation: d_{last}. Section 4.3 explains how we achieve d_{last}. And as packet s_{i+1}^1 is not received yet, we do not know $t^{rx}(s_{i+1}^1)$, either. Thus we approximate it using the receive time of the last packet of the current chirp $t^{rx}(s_i^{n_i})$, adding the transmission delay of packet s_{i+1}^1 and a guard time t^{guard}. We use the same $t^{guard} = MI \cdot (1 - UT)$ we introduced in the previous section.

$$t^{erx}(s_{i+1}^1) \approx t^{rx}(s_i^{n_i}) + \frac{p_0}{A_i} + t^{guard} \tag{6}$$

As long as $k \neq n_i$, the packet $s_i^{n_i}$ has not been received yet and thus $t^{rx}(s_i^{n_i})$ needs to be approximated. Assuming that we know A_i, we can use the known receive time $t^{rx}(s_i^k)$ and the uniform size p_i of the measurement packets to estimate $t^{erx}(s_i^{n_i})$ with the following equation:

$$t^{erx}(s_i^{n_i}) = t^{rx}(s_i^k) + (n_i - k) \cdot \frac{p_i}{A_i}. \tag{7}$$

Using Eq. 7 to estimate $t^{erx}(s_i^{n_i})$ we finally get an approximation for an uncongested transmit time for the first packet of chirp $i+1$ as:

$$t^{etx}(s_{i+1}^1) \approx t^{rx}(s_i^k) + \frac{(n_i - k) \cdot p_i + p_0}{A_i} + t^{guard} - d_{\text{last}} \tag{8}$$

Except in the rare case where we just received the last packet of chirp i we still remain with no valid available data rate measurement A_i as at least some of the packets of the chirp i still have to arrive. If at least two packets of chirp i were received we use those packets to estimate A_i. If less than two packets of chirp i have arrived we use A_{i-1} as an approximation for A_i. Surely, both of these approximations introduce an error, but this will likely be corrected when further packets of chirp i are received and new approximations for $t^{etx}(s_{i+1}^1)$ are calculated and transmitted piggyback to the server.

4.3. Filtering delay measurements during congestion

As explained earlier, measurements of the network delay using the first packet of each chirp can only work if no self-induced congestion occurred. While the ICSIC-Detector we introduced in the previous Section 4.2 eliminates most ICSIC phases, it can not eliminate it in situations where all measurement packets are lost or an

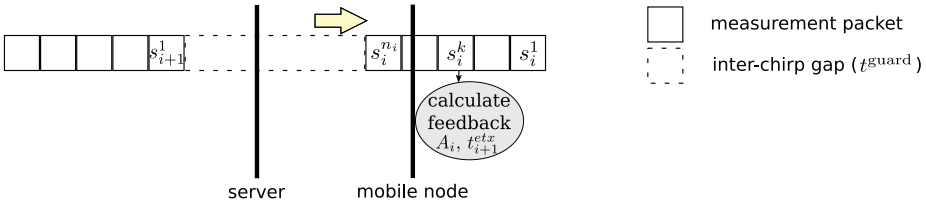


Fig. 3. $t^{etx}(s^1_{i+1})$ calculation example.

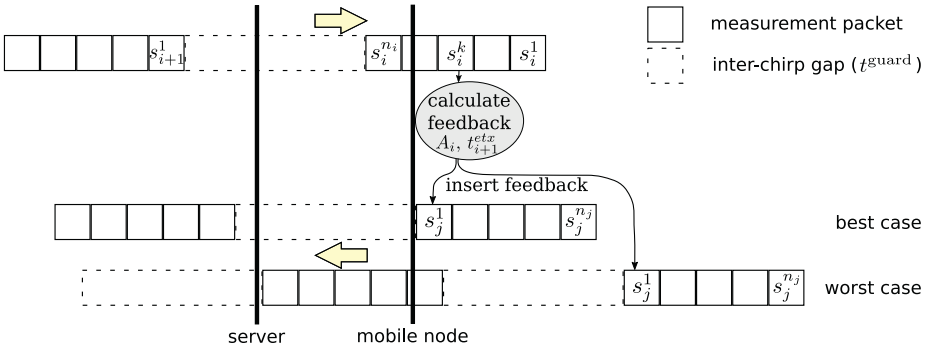


Fig. 4. Feedback delay sketch using measurement channel piggybacking only.

arbitrarily large delay is introduced between two consecutive packets of a chirp by the "black box". This might, for example, happen on context switches from 3G to 2G.

Therefore we discard any delay measurement where the following equation does not hold:

$$t^{rx}(s^1_{i+1}) \geq t^{rx}(s^{n_i}_i) + t^{guard} \tag{9}$$

Instead, we keep using the last valid measured delay d_{last}.

4.4. Reducing feedback delays using a second physical link

Using the payload of the measurement packets sent in the other communication direction to transport measurement data back to the other node can introduce large feedback delays which we sketched in Fig. 4.

Even without the transmission delay of the packet, the feedback can only be inserted in future measurement packets. Even in uncongested situations, the feedback might have to wait until the inter-chirp gap $t^{guard} = MI \cdot (1 - UT)$ has passed by before getting piggybacked. If the feedback direction currently suffers from ICSIC, the delay can reach 50 seconds as we showed in [12]. Even if only one communication direction is in an ICSIC-phase the other direction is influenced, too, as it has to rely on old feedback and thus old measurements to adjust the chirp size. Therefore we introduced an optional second physical link using a second cellular modem in a separate cellular network which is different from the one that is being measured. As it is only used to transport feedback packets it usually does not suffer from self-induced congestion. We keep piggybacking the measurement results in the measurement packets even when using the optional second, feedback-only physical link. Thus the feedback can reach the other node at least as fast as before. Especially in ICSIC situations it has a high chance of reaching the other node much faster using the safeguard channel.

4.5. Solving tail drops

While the ICSIC detection and delay filtering worked nicely in our implementation, some measurement phases suffered from large amounts of packets at the tail of chirps being dropped – especially in the downstream direction. In some situations we saw the first roughly 5% of the packets of a chirp arriving without dropped packets, followed by the first 5% of the packets of the next chirp — leaving 95% of the chirp as dropped packets.

The following paragraphs describe the three independent methods that we combined to eliminate these tail drops:

1. We throttle the sending rate using a self-written kernel module. This helps to reduce tail drops of chirps sent by the mobile node.
2. We throttle the sending rate to the technology limit of the modem. This is especially useful for packets sent by the server because the first method does not help due to the gigabit link of the server, which is much faster than the bottleneck of the network path.
3. We change the measurement interval $MI = 1000$ ms to $MI = 200$ ms resulting in 100 ms chirps followed by 100 ms silence (t^{guard}).

To eliminate any chance that our own measurement devices drop these packets we implemented a kernel module that monitors the network queues of the measurement device in the Linux kernel. Our measurement implementation was altered to postpone the sending of each packet until at most one packet is left in the kernel sending queue of our measurement device. While still maintaining the application-to-application measurement approach, as a side effect, this implementation significantly raised the transmit timestamps' accuracy in our measurement packets, which are inserted just before they are handed to the network layer.

Further measurements showed that this proved to eliminate the tail drops in the upstream direction, but it had no significant effect on the tail drops in the downstream direction. These tail drops are likely to be the result of an overflowing network queue at the bottleneck of the network path – probably just before the air interface is reached. Given the characteristic of our measurement traffic this is no surprise as we are sending a large number of packets into the network at the fastest possible rate that we can generate using a gigabit Ethernet interface.

Further reduction of the tail drops on the downstream were possible by throttling the sending rate of the server. Still assuming that the bottleneck of the downstream network path is the air interface to the mobile node, we throttled the sending rate of the server to the maximum receiving rate the modem can handle. While this reduced the severeness of the tail drops in the

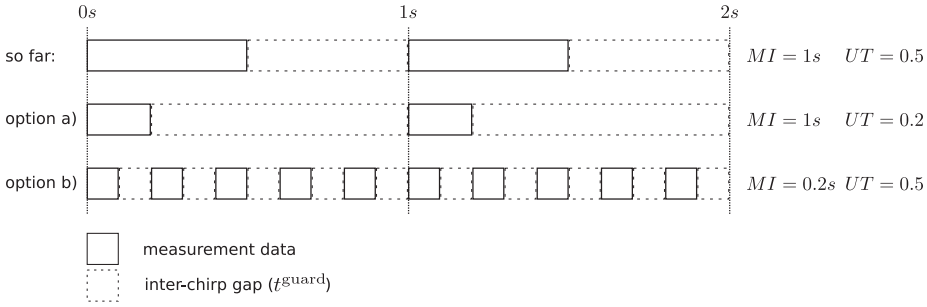


Fig. 5. Examples for changes of the measurement interval *MI* and the utilization *UT*.

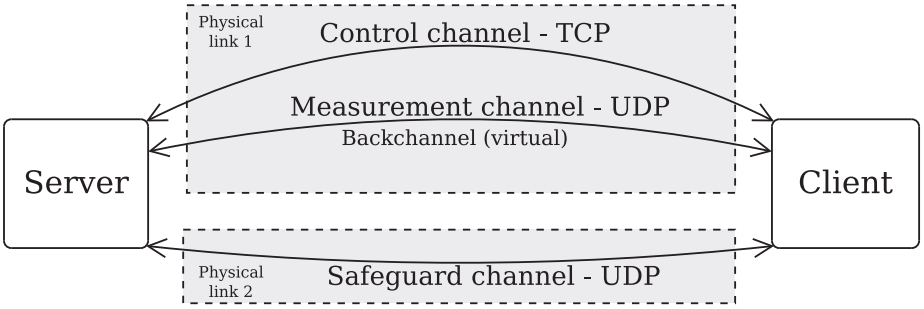


Fig. 6. Structure of the rate measurement framework.

downstream direction, there were still time spans in which a fast declining receive rate of the mobile node allowed the network queues — most likely near to the mobile node — to tail drop large amounts of packets. Further throttling the sending rate of the server below the maximum possible receive rate of the mobile node is no valid option as it would disable the measurement of data rates within the technological limits of the modem.

The only other way to reduce the *chirp delay product* — the amount of bytes of one chirp currently traveling in the network — is to shorten the chirps themselves.

This leaves two possibilities, which we depicted in Fig. 5:

(a) Lower the utilization *UT* of each measurement interval
(b) Shorten the measurement intervals without changing the utilization *UT*

While option (a) would surely work and even reduce the risk of self-induced congestion, it would also lead to less accurate measurements as less time per second would be used for measurement traffic and more time spent in silent phases. We therefore opted for solution (b) and conducted measurements with different measurement intervals *MI* of 1000 ms, 500 ms, 300 ms and finally 200 ms while keeping the utilization at 50%.

While the reduction to $MI = 500$ ms already heavily reduced the tail drops in the downstream, it still did not eliminate them completely. They vanished when switching from 300 ms to 200 ms. Lowering the *MI* significantly below 200 ms does not make sense with typical 3G one-way delays of 20 to 70 ms, as the feedback needs at least another one-way delay to travel back through the network and thus would always be “late”. Therefore we opted for 200 ms measurement intervals which, combined with the kernel module and the throttling of the sending rate, proved to work well (compare Section 7.3).

5. Implementation details

To overcome the aforementioned challenges we implemented our ideas in the *rate measurement framework (RMF)*. The RMF is developed and implemented by us using *C++* to minimize effects like garbage collection and is intended to run on Linux devices. The source code is available at github[1].

Fig. 6 shows the basic network setup the RMF uses. It employs up to two physical links (using different network devices) between the two measurement nodes. The first physical link always hosts two discrete network channels, the *control channel (CC)* and the *measurement channel (MC)*. The *CC* is used to exchange parameters before the measurement starts and uses TCP for reliable data transfers. All measurement data – the chirps – are sent via the *measurement channel (MC)* using UDP. Inside the MC another virtual channel, the *back channel (BC)*, is established. It contains information of the last measurements conducted in the other direction piggybacked in the payload of the measurement packets.

The optional second physical link uses a second cellular modem in a separate cellular network. This link is solely utilized to transfer measurement feedback that is also transferred in the virtual *back channel*, but avoiding the delays introduced by inter-chirp self-induced congestion and inter-chirp gaps.

Service providers of mobile cellular networks often employ *Carrier Grade Network Address Translation (CGN)* inside their networks. These CGNs typically do not allow user-defined port forwardings. To simplify NAT-holepunching we assume that the measurement server is reachable via static ports on at least one public IP-Address. Consequently, all connections are initiated by the mobile node to open up NAT ports.

The RMF can be configured by various command line parameters which for example, allow the selection of network devices, port and IP usage, IPv4 and IPv6 selection, thread priorization, initial sending rates, maximum sending rates for up- and downstream, the default length of a measurement interval *MI*, the utilization ratio *UT* specifying the amount of time spend sending data during each measurement interval, the maximum number of packets per chirp and the usage of the safeguard channel. The workload is handled by various threads: separate send and receive threads for each UDP channel, dedicated low priority logging threads and

[1] https://github.com/hhucn/rmf

Table 2
Fixed and dynamic payload contents.

Type	Data type	Description
Fixed payload	uint64_t	Send time [ns] of this packet
	uint64_t	Send time [ns] of the first packet of this chirp
	uint32_t	Chirp number
	uint32_t	Packet number
	uint32_t	Packet size including headers [Byte] of this packet
	uint32_t	Number of packets used in this chirp
	uint32_t	Packet size including headers [Byte] of packets 2..n
Dynamic payload	uint64_t	Send time [ns] of this dynamic payload
	uint32_t	Chirp network characteristics are based on
	uint32_t	Packet network characteristics are based on
	uint64_t	Last data rate measurement [Byte/s]
	uint32_t	Last loss rate measurement times 10^9
	uint32_t	Chirp number t^{etx} calculation is based on
	uint32_t	Packet number t^{etx} calculation is based on
	uint64_t	Calculated t^{etx} [ns]
	uint32_t	Last safeguard channel one-way delay [ns]

a separate thread that generates the measurement chirps based on the latest available measurement results.

All timestamps used in the RMF are epoch times in nanoseconds stored in int64_t data types.

The packet payload of RMF measurement packets is divided into fixed payload, dynamic payload and (if needed) padding. While the fixed payload stores static data identifying the current packet and chirp, some chirp characteristics and the send timestamp, the dynamic payload carries information on the last network measurements and, if applicable, information on a changed t^{etx} for the opposite sending direction. Table 2 explains the various payload parts.

The payload of safeguard packets does not need to carry fixed payload nor does it need padding and thus only carries the dynamic payload. The smallest possible measurement packet carries $36 + 48 = 84$ Bytes of measurement payload and results in an Ethernet frame of 126 Bytes including Ethernet (14), IPv4 (20) and UDP headers (8). All safeguard packets carry 48 payload Bytes and thus generate 90 Bytes Ethernet frames.

To limit the traffic and thus the probability of self-induced congestion on the safeguard channel a two-fold filter was implemented. On a normal basis safeguard messages are sent periodically every 50 ms. But whenever the self-induced congestion detector calculates a t^{etx} that is larger than the normal t^{tx} of that chirp there might not be enough time for transmission left if we wait for the next periodic safeguard message sending slot. Hence we send a safeguard packet immediately if the following inequation holds:

$$\text{now} + 3 \cdot d_{\text{safeguard}} > t^{etx} > \text{now} + d_{\text{safeguard}} \tag{10}$$

On reception of any dynamic payload the receiver checks if the contained t^{etx} is based on a more recent measurement packet than the last previously used t^{etx}. Older t^{etx} are trashed immediately. Further on, network delays in real world might lead to the reception of a new $t^{etx}(s_i^1)$ based on chirp $i-1$, but the next chirp to be sent is chirp $j > i$. Then we estimate $t^{etx}(s_j^1)$ as:

$$t^{etx}(s_j^1) = t^{etx}(s_i^1) + (j - i - 1) \cdot MI \tag{11}$$

If the optional safeguard channel is used in combination with a server with only one public IP address, source-based routing on the client is used. This allows the usage of different routes for measurement packets and safeguard packets to the same destination.

A high-level overview of our implementation showing the essential threads running on each node is shown in Fig. 7. The flowchart outlines where the methods introduced in Sections 3 and 4 are working and how they influence other threads. The top two solid boxes mark receiving threads, which call the two functions surrounded by dashed boxes. The two solid boxes at the bottom of the flowchart embed the sending threads. To enhance the readability, the high-level overview neither shows the measurement initialization phase, nor does it show the logging of packets. Wherever feasible, we added references to the corresponding paper sections describing parts of the implementation or explain the used algorithms.

Both receiving threads (solid boxes at the top) constantly wait for incoming UDP Datagrams and upon reception note the arrival time (t^{rx}) and delegate further handling of the dynamic and fixed payload to two specialized functions.

The function “handle fixed payload” uses the received fixed payload embedded in the measurement packets to calculate the available data rate, loss rate and the delay of the incoming connection. Further on it estimates the next ICSIC safe transmit time t^{etx} for the opposing sender. As all this information is needed by the measurement sender thread of the other node, it forms a new dynamic payload which might be embedded in the next measurement packet(s) or safe guard packet(s). If an emergency transmission via safeguard channel is needed, the safeguard thread is triggered.

The function “handle dynamic payload” is called by both receiver threads, as both channels carry dynamic payload. It first checks if the received dynamic payloads information is based on a newer packet (chirp number, packet number) than the currently stored newest incoming dynamic payload. This is necessary, as both communication channels probably have different delays, and out of order packet arrivals are possible. When the received dynamic payload is “new”, it further checks, if the embedded t^{etx} is earlier than the currently scheduled t^{tx} of the next chirp. If so, the measurement sending thread is woken up to schedule an earlier transmit time. If not, it suffices to change the t^{tx} as the sending thread will check if the t^{tx} changed when waking up to send the next chirp.

The measurement sender thread (bottom left solid box) is responsible for the timing and creation of the measurement chirps. It uses the data rate measurement embedded in the incoming dynamic payload to alter the size of the measurement chirp. The normal sending frequency is one chirp per MI. Whenever the dynamic payloads t^{etx} is later than the periodically assigned start of a chirp, the measurement sender adjusts the sending time to the received t^{etx} in the effort to avoid ICSIC.

The safeguard sender thread (bottom right solid box) is optional. When used, it periodically, with a frequency of 20 Hz, sends the current outgoing dynamic payload via the 2nd physical channel to the other node. Whenever in-Eq. 10 holds, an emergency

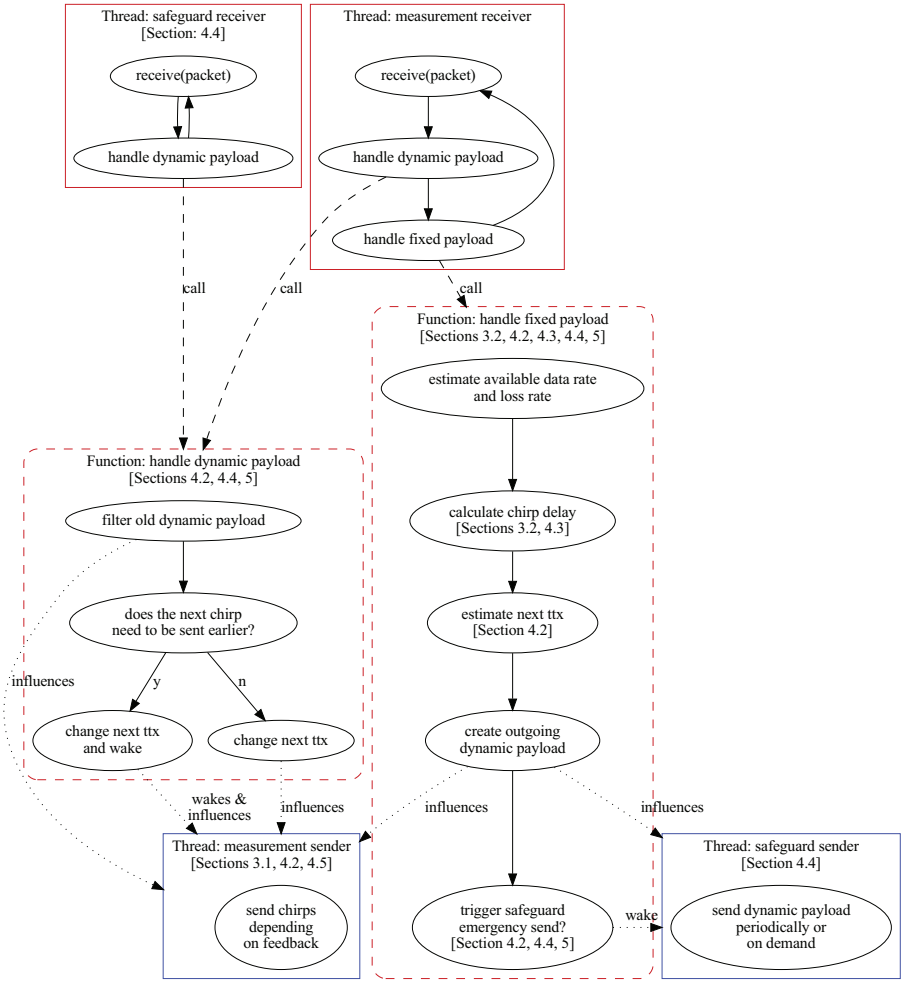


Fig. 7. High-level overview of the RMF except measurement setup and logging.

transmission is triggered, issuing the immediate sending of a safeguard packet. Afterwards, the periodic sending takes over again.

6. Measurement setup

We extensively tested the RMF-implementation using various computers and cellular modems during our measurements in Greater Düsseldorf. We used the following mobile devices:

- Alix 6F2 Board, single core 500MHz AMD Geode LX800 CPU, 256MB RAM, MC8790 cellular modem (max. 7.2MBit/s down, 5.76MBit/s up)
- Thinkpad T530, i7-3520M, 16GB RAM, H5321gw modem (21MBit/s down, 5.76MBit/s up)
- Thinkpad X200t, SL9400 CPU, 8GB RAM, either a F3507g (7.2MBit/s down, 2MBit/s up) or H5321gw (21MBit/s down, 5.76MBit/s up) modem

The mobile devices were connected to a garmin GPS 18 LVC GPS PPS receiver for time synchronization, and the cellular modems were connected to up to two external antennas. During moving measurements the equipment was installed inside a car and the antennas were roof mounted. For the optional safeguard channel we used a USB modem placed behind the windshield. As servers we used two different machines in separate locations:

- A dedicated server housed in the university data center which offered two gigabit network connections, 2 Intel Xeon 5160 Dual Core CPUs and 8GB RAM.
- A virtual server in the Kamp.de dynamic hardware pool with 6 vCores, 8GB RAM and a gigabit Ethernet connection.

The measurements prior to the work presented here, which identified the ICSIC problem, (referenced in Table 3) were conducted using the predecessor of the RMF written in JAVA; using fixed 1Hz measurement intervals with chirp size adjustments. At that time we used a Thinkpad T61 with a dual core CPU, 4 GB RAM and a sierra wireless MC8775 modem (7.2MBit/s down, 5.76MBit/s up) and no safeguard channel. Those former measurements were conducted against a dual core server hosted at our facility.

Fig. 8 shows the main measurement area and the routes of the RMF traces used in this paper. To make the traces distinguishable, the traces are plotted with varying offsets to the measured coordinates to minimize perfectly overlapping traces on the same roads.

7. Evaluation

Existing available data rate measurement methods either can not cope with the fast change of network conditions which occur in mobile cellular networks, or they make simultaneous delay measurements impossible (see Section 2). It is therefore not an option to provide a meaningful comparison with those approaches. What we can do, however, is to show that the ideas presented in this paper, in fact, provide good overall absolute results and a significant improvement in respect to the problem of self-induced congestion as identified in [12].

As a baseline for this we use the measurements we conducted for [12], which we refer to as “pre RMF”. Those were

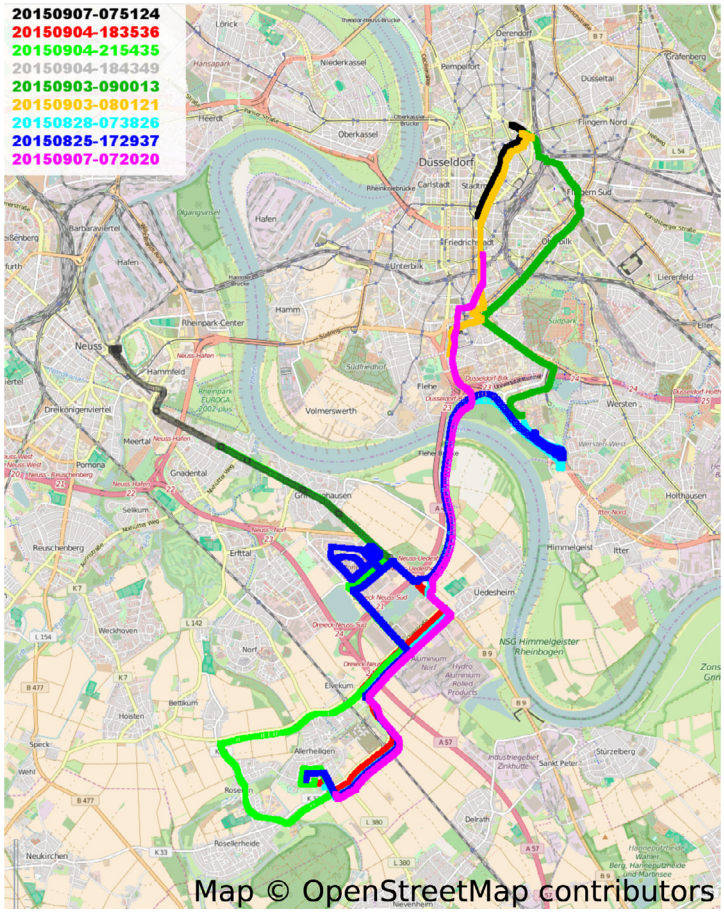


Fig. 8. Overview of the Neuss/Düsseldorf area used in the RMF example measurements of this paper. (Map ©OpenStreetMap [16]).

captured using the basic measurement methods as described in Sections 3 to 4.1. Measurements conducted with the current RMF framework including all enhancements described in this paper are referred to as "RMF".

We first take a look at some statistics on *pre RMF* and *RMF* measurements in Section 7.1. Afterwards we analyze the characteristics of the remaining self-induced congestion phases in Section 7.2 and show that they cannot be eliminated or shortened any further. We present an example of the solved tail drop problem in Section 7.3. Finally, we give detailed insight into a 30 minute measurement trace in Section 7.4.

7.1. Statistics

Table 3 presents statistical data of two former (*pre RMF*) and nine chronologically sorted recent measurement traces (*RMF*). All traces were conducted in Greater Düsseldorf. As representatives for our self-induced congestion prone pre RMF implementation we chose two traces. The first trace *best pre* performed extraordinarily well in regard to the maximum delays — caused by ICSIC phases — measured (18 s and 11 s). The second trace *worst pre* on the other hand presents the *pre RMF* trace that suffered the most by ICSIC and showed packet delays of up to 22 s and 50 s.

We split the nine traces conducted with our improved measurement method, using the RMF introduced in this paper, into two groups. The upper traces, *20150825-172937* and *20150828-073826*, were conducted before we introduced the periodic sending interval with emergency sending for the safeguard channel. The remaining seven RMF traces used the current RMF implementation with all the improved measurement methods introduced in the paper at hand.

We included the two traces without the safeguard channel sending optimizations on purpose. They demonstrate that these modifications are indeed necessary. While the statistical data of *20150828-073826* looks quite similar to that of *20150825-172937* — especially regarding the maximum delays measured — the latter trace features a maximum delay of more than 21 seconds in the downstream direction. The analysis of the logfiles of *20150825-172937*, which contain the payload (except padding) and timestamp of every sent and received packet, showed, that we unwillingly created congestion on the safeguard channel while the downstream measurement channel was congested, too. Thus our feedback was "late" even though we used the safeguard channel. As a consequence, we altered the RMF to send periodic updates every 50 ms and issue an emergency transmission if needed. Even though the fix was not implemented at that time, trace *20150828-073826* shows, that this filter is not needed at all times, as we do not see abnormal high delay spikes in this trace.

For the remainder of our evaluation, we will focus on the comparison between the two *pre RMF* traces and the seven *RMF* traces with proper safeguard channel filters (*20150903-080121* - *20150907-075124*). All these RMF traces show significantly lower maximum packet delays for both measurement directions than the pre RMF traces. This underlines the major improvements in ICSIC handling presented in this paper. The significantly reduced packet loss ratio and even more so the extremely low chirp loss ratio show how well our improvements work. While the majority of complete chirp losses formerly happened during the extreme congestion phases, probably caused by tail drops due to overflowing network queues, the newly introduced ICSIC detector proves to shorten ICSIC phases and significantly reduces the number of chirps and packets sent during such phases. This effectively

Table 3
Statistical data of some of our measurement traces.

Trace			Packets sent		Chirps sent		Congested chirps				Delays [ms]		
run	dir	dur	#	%loss	#	%loss	#	%con	ph.	avg	min	max	avg
best pre RMF	up	6957	103417	5.03	6958	0.72	93	1.34	24	3875	51	18516	216
	dn	6953	103261	0.37	6954	0.26	135	1.94	30	4500	37	11227	274
worst pre RMF	up	5579	81456	1.83	5580	1.15	172	3.08	52	3307	48	50742	484
	dn	5579	83076	2.05	5561	1.85	106	1.91	23	4608	37	22919	311
20150825-172937	up	1859	169067	0.99	8460	0.97	1241	14.67	779	644	19	6948	131
	dn	1860	382261	0.37	8568	1	586	6.84	402	718	28	21878	129
20150828-073826	up	883	78548	0.31	4007	0.3	611	15.25	353	686	19	6167	148
	dn	884	193330	0.04	4122	0.07	332	8.05	206	660	20	5107	106
20150903-080121	up	1294	113663	0.21	5863	0.2	822	14.02	491	658	22	6654	128
	dn	1296	285896	0	6125	0	278	4.54	209	572	30	6129	100
20150903-090013	up	1264	115933	0.41	5847	0.39	608	10.4	382	651	12	4334	105
	dn	1265	277564	0.01	6004	0.02	294	4.9	230	547	16	5007	87
20150904-183536	up	446	39811	0.25	2023	0.4	296	14.63	172	692	16	3988	132
	dn	446	89449	0.31	1781	0.79	175	9.83	102	699	21	10246	134
20150904-215435	up	1264	117161	0.15	5827	0.15	519	8.91	308	662	13	6408	106
	dn	1265	288972	0.01	5969	0.02	473	7.92	315	593	17	6529	100
20150904-184349	up	605	55741	0	2826	0	259	9.16	166	592	12	1820	98
	dn	606	150758	0	2899	0	113	3.9	99	492	17	1357	76
20150907-072020	up	1812	167630	0.05	8359	0.05	909	10.87	606	606	12	4604	103
	dn	1813	415052	0	8596	0	397	4.62	330	515	17	4712	80
20150907-075124	up	372	30893	0.06	1668	0.06	316	18.94	184	658	16	1487	136
	dn	373	84059	0	1778	0	119	6.69	77	564	21	1752	89

The first group of three columns identify the measurement run, the communication direction and the duration of the run in seconds. The second group shows the number of packets sent and the loss rate. The third group shows the number of chirps sent and percentage of completely lost chirps. The fourth group shows information on chirps suffering from self-induced congestion. It presents the number of congested chirps, the congestion probability for a chirp, the number of identified congestion phases and the average length of a congestion phase in milliseconds. The final column group presents statistical information of the packet delays of all packets of the chirp.

reduces packet losses caused by too much measurement traffic and leads to significantly better loss rate measurements for the network.

A direct implication of the raised measurement frequency — the pre RMF implementation used measurement intervals of $MI = 1000ms$ while the RMF traces used $MI = 200$ ms — is the larger rate of congested chirps. On the one hand the 100 ms gap might not suffice for feedback delivery and thus might be late for the next chirp. On the other hand, formerly unmeasurable fast fluctuations of the available data rate have an impact on single chirps while they formerly (pre RMF) influenced chirps five times larger. On the other hand, as shown in column *congested chirps / avg*, the average length of a congestion phase is reduced significantly. While it ranged from 3300 ms to 4600 ms for our pre RMF implementation, the RMF was able to reduce the average congestion phase length to less than 700 ms for all recent traces.

The highest measured packet delay of 10 s, measured with the RMF in trace *20150904-183536*, is the direct consequence of massive packet duplication and an abrupt in-chirp delay. An in-depth analysis of the corresponding trace showed, that we received many duplicate packets on four consecutive chirps in that measurement, which additionally arrived interleaved. Of the 81 packets sent in those four chirps, 147 packets reached our receiver. This mass of packets clearly overloaded the network and lead to ICSIC. While our ICSIC detector cannot anticipate duplicate packets we seldom witnessed more than a few consecutive duplicate packets during our traces.

7.2. Analysis of the remaining ICSIC periods

Even though the statistics show that we reduced ICSIC to only a few chirps, it still remains to be shown that those ICSIC phases can not be shortened or eliminated completely. While we can not present a mathematical proof we can offer an in-depth analysis of an exemplary ICSIC phase. Fig. 9 shows an excerpt of the downstream of a measurement trace.

In the lower chart the delay $(t^{rx} - t^{tx})$ of each received packet is plotted against its chirp number, hence the excerpt shows data of chirps 4613 to 4623 of this particular measurement. Usually the measurement packets in the downstream direction are sent faster than the utilized modem can receive them. Provided that the network conditions do not change during the measurements, the graph would show the first packet of each chirp arriving after the current one-way delay of the measurement, followed by 100 ms of evenly distributed packets of the same chirp. While chirp 4613 shows this ideal behavior, most other plotted chirps demonstrate how jittery mobile communication under movement behaves. The upper chart shows the measured gap between the first packet of the current chirp and the last packet of the previous chirp, again plotted against the corresponding chirp's number. This graph ideally would show a horizontal line at 100 ms as we aim for 100 ms measurement traffic followed by 100 ms silence per 200 ms measurement interval. Values near or at 0 ms indicate that, even though we used our ICSIC detector, two consecutive chirps suffered from ICSIC. In the shown excerpt chirps 4615 to 4620 arrived without inter-chirp gaps and thus suffered from ICSIC. Therefore let us have a closer look at chirp 4614 which started the congestion phase. The first packets of chirp 4614 have arrived after the expected delay, followed by a packet with a slightly larger delay triggering a slightly raised $t^{etx}(s^1_{4615})$. But the remaining packets of chirp 4614 that reached the mobile node arrived with a gap of 1.2 seconds. Until the arrival of the first packet of this “slow” group, no possibility to calculate a suitable t^{etx} existed — there is plainly no way to distinguish a “slow” packet from a “dropped” packet until it or a subsequent packet arrives. And even the arrival of a subsequent packet could be the result of out-of-order packets.

Hence our ICSIC detector could not issue the significantly changed $t^{etx}(s^1_{4615})$ before the arrival of this “slow” packet, roughly 1.35 seconds after $t^{tx}(s^1_{4614})$. Even without delay on the safeguard channel the server could not know about this significant $t^{etx}(s^1_{4615})$ change until $t^{tx}(s^1_{4614}) + 1.35s$. With our standard measurement interval $MI = 0.2s$ the server thus already sent chirps 4615 to 4620 before he received the changed t^{etx}. Hence the first opportunity to

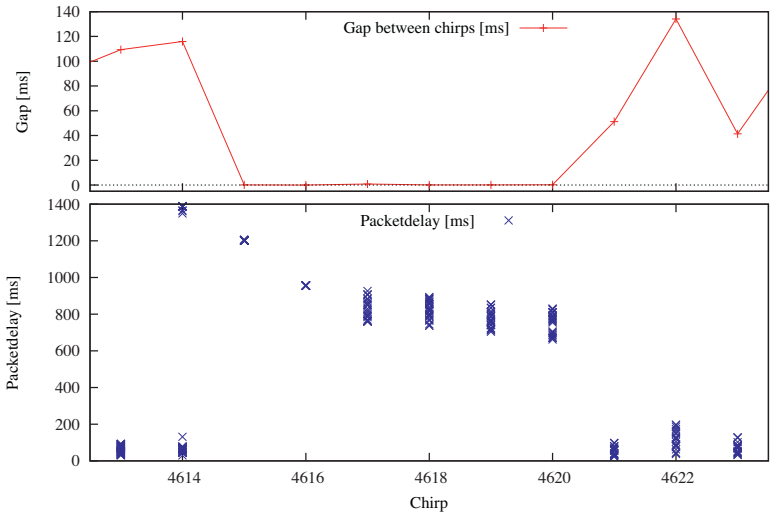


Fig. 9. Example of an ICSIC period caused by sudden large delays.

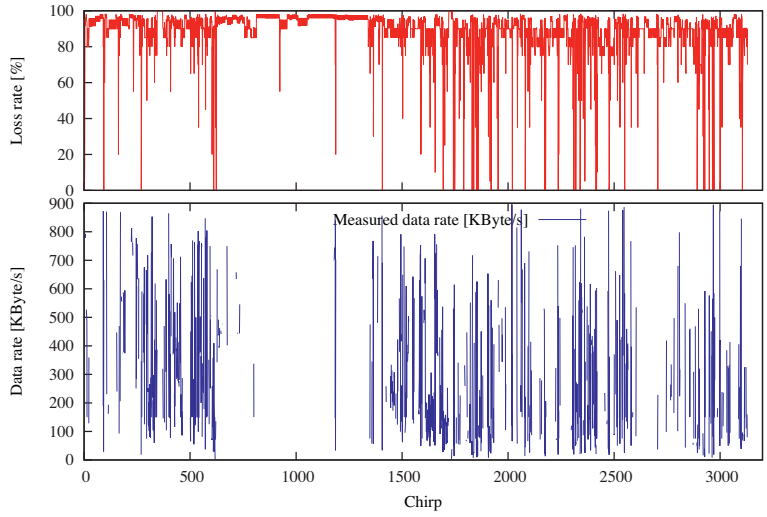


Fig. 10. Example of a downstream measurement suffering from massive tail drops (using an older version of the RMF).

stop this unpredictable ICSIC phase was the changing of $t^{tx}(s^1_{4621})$ — which evidently happened as the gap between chirp 4621 and 4620 was slightly above 50 ms.

7.3. Eliminating tail drops

In Section 4 we stated, that we encountered a tail drop problem likely caused by overflowing network queues due to too much measurement traffic. We also outlined that the problem is more severe in the downstream direction than in the upstream direction. Fig. 10 shows downstream measurement data conducted with an old version of the RMF without our tail drop fixes. The upper graph shows the percentage of dropped packets per chirp while the lower graph shows the measured data rates for the same chirps. Obviously, this measurement shows an unusually high loss rate which almost never reaches 100%, but is well above 80% for the majority of the trace.

In contrast, Fig. 11 shows the downstream measurement data from a trace using a version of the RMF that incorporated our tail drop fixes. The trace starts at the same place and time in Greater Düsseldorf as the trace shown in Fig. 10, but lasts 20 minutes longer as it was not aborted due to unusable measurements. Apparently the second trace shows zero packet loss for the vast majority of the 8550 chirps. While there are still some loss spikes, those top out at 100% each and back down quickly which is a typical sign for areas of bad reception.

7.4. Overview on measurement results

In this section, we give a brief overview of some of the measurement results from a sample trace.

Figs. 12 and 13 show the measurement results for the downstream and upstream direction, conducted on September 7th 2015. The measurements were conducted during the morning rush hour in the area shown in Fig. 8. The route taken is marked in magenta, named *20150907-072020* and starts in Allerheiligen, a suburb of Neuss. The trace ends after 30 minutes in the city center of Düsseldorf. For this measurement, the Thinkpad x200T with a H5321gw modem (21MBit/s down, 5.76MBit/s up) with two roof mounted antennas was used. The measurement channel connected using a *t-mobile* contract, the safeguard channel used the network of *E-Plus*. The measurement interval *MI* was set to 200 ms and the utilization *UT* to 50%. Both multiplots show four subplots which all share the same x-axis (the chirp number).

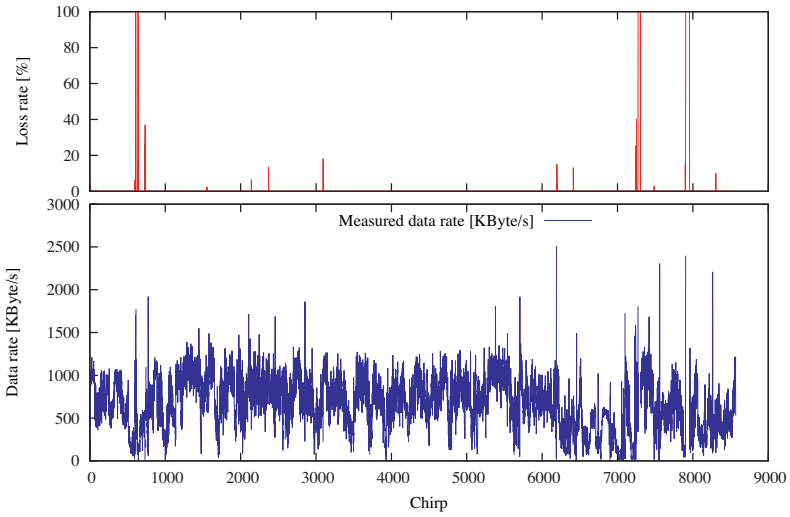


Fig. 11. Example of a downstream measurement using a current version of the RMF with tail drop fixes.

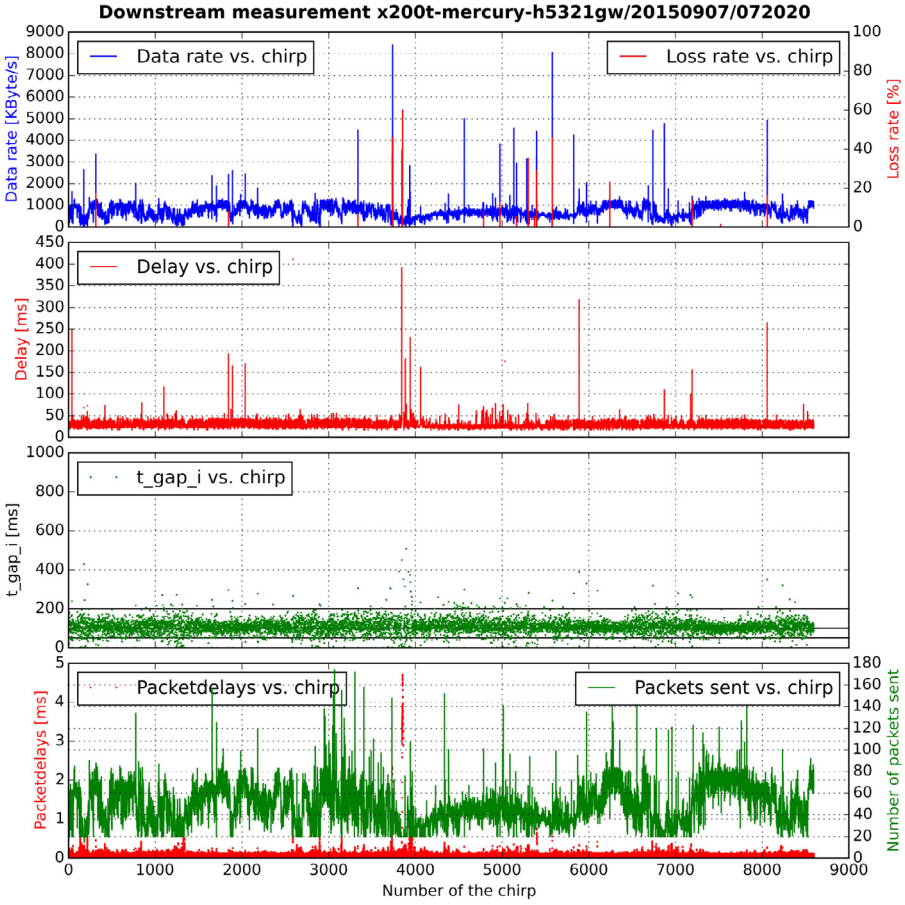


Fig. 12. Detailed downstream measurement results using a current version of the RMF.

The upper most plot shows the measured available data rate and the loss rate for each measurement chirp.

The second plot shows the delay measurements without the measurements we discarded due to identified ICSIC.

The third plot shows the gap between the last received packet of chirp i and the first received packet of chirp $i+1$ (t^{gap}). In a perfect, unchanging network, the third graph would show a horizontal line at 100 ms due to the used *MI* and *UT*. t^{gap}s between 0 ms and 50 ms indicate ICSIC and we thus discard delay measurements there. Whenever t^{gap} is far above 100 ms, our algorithm significantly overestimated t^{etx}. This mostly happens when significant changes in available data rate and high loss rate occur simultaneously. We judge every t^{gap} value between 50 ms and 200 ms as good. The graph thus features black horizontal lines at 50 ms, 100 ms and 200 ms as a guidance.

The red dots in the bottom plot mark the delay ($t^{rx} - t^{tx}$) for every received measurement packet of each chirp. If our measurement traffic, which is created on the basis of prior measurements,

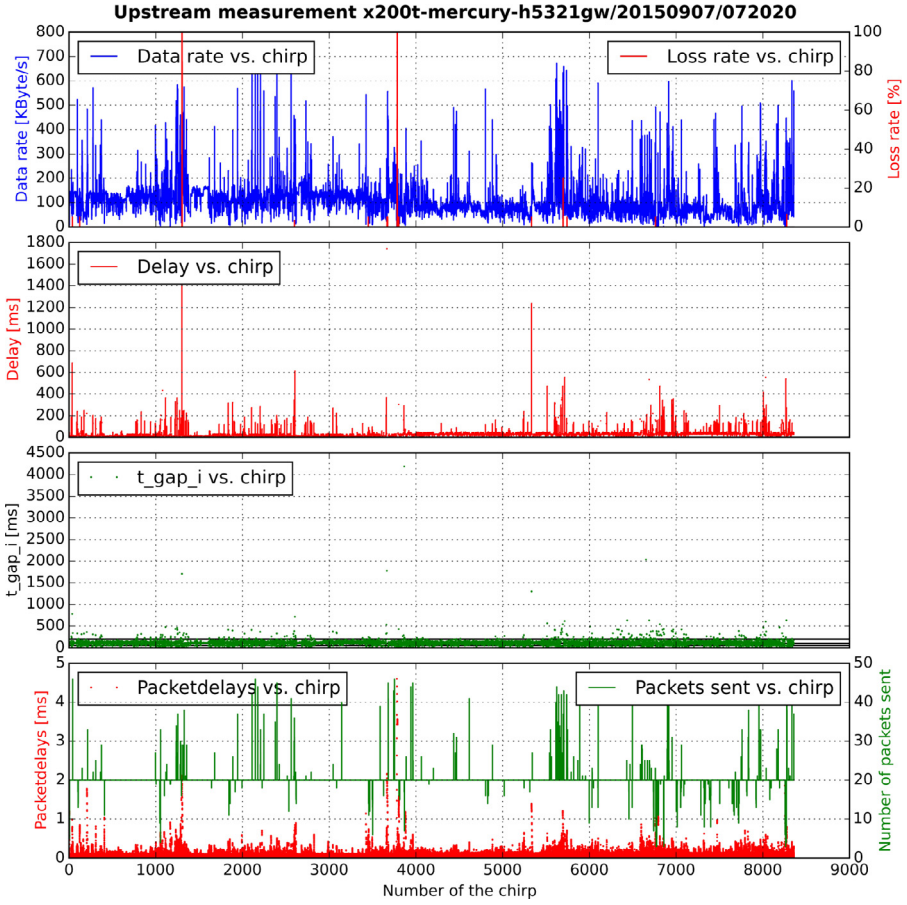


Fig. 13. Detailed upstream measurement results using a current version of the RMF.

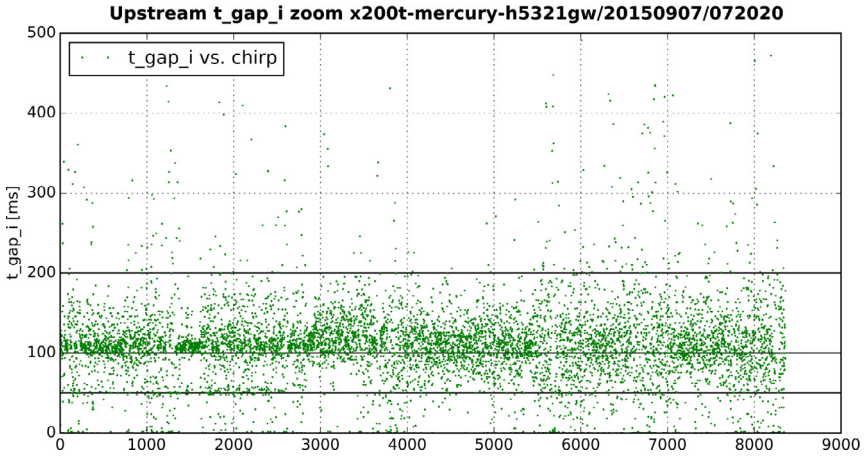


Fig. 14. Zoomed in view of t_gap_i of the upstream measurement results.

perfectly matched the currently available data rate, an equally distributed vertical line of dots with a span (lowest delay to highest delay) of up to 100 ms would be visible. Whenever this line rises above 100 ms, our sending rate was higher than the available data rate. Whenever the line exceeded 200 ms, an ICSIC phase would start if the ICSIC handling did not work. The green lines mark the number of packets sent for each chirp.

The downstream measurements shown in Fig. 12 feature only a few loss spikes. Thus obviously no systematic tail drops occurred. A closer look at the measured available data rate reveals some visible spikes. As the mobile cellular network is a black box for us, we can not explain the cause of these spikes, but often they coexist with ICSIC phases. So probably packets were queued in the network and transferred using diversity or by granting a temporary high downstream data rate by the cellular network. We cannot rule out that the laptop and its OS might have influenced the timestamping of packets at that time, but this is rather unlikely, as the CPU usage of the RMF typically ranges between 3% and 6% during the measurements, with the highest priority given to the receiving threads. The recorded delays are quite stable with a few spikes - probably due to bad reception - of up to 410 ms. The monitored t^{gap}s are mostly evenly distributed around the “perfect” 100 ms. Only a small amount of t^{gap}s is below 50 ms with an even smaller part being near or at 0 ms. Even fewer gaps are longer than 200 ms.

This clearly shows that our algorithm is able to eliminate ICSIC most of the time. The bottom plot also shows that the delay of the individual packets reside in the expected range 'most of the time'. An exception occurs around chirp 3900, where a delay spike of almost 5 s indicates a significant ICSIC phase. Still this is significantly lower than the up to 11 s delay measured in our best *pre RMF* measurement.

When comparing the upstream measurements shown in Fig. 13 to the downstream measurements in Fig. 12, the larger amount of spikes in the available data rate chart is visible. Please note that the scaling of the charts presented in both figures differ to show as much detail as possible in each chart.

The higher number of spikes in the upstream available data rate measurements is probably caused by the much more complex task of controlling multiple competing senders with varying - mostly burstier - demands and sending powers. While in the downstream direction only one station (the cell tower) is sending data, in the upstream direction multiple moving devices have to be coordinated and interferences between the stations have to be compensated. This can result in changes in the chosen forward error correction or a higher spreading factor (in 3G) when transmission gets worse.

The severe available data rate reductions following most spikes are the major reason why the measurement feedback mechanisms introduced in this paper are necessary, as these situations would lead to ICSIC without the usage of the ICSIC detector.

Never the less, even in the upstream direction the packet losses are minimal, except at two spikes with 100% packet loss. The disadvantageous transmission characteristics of the upstream also result in a much higher fluctuation of the measured delays and in five t^{gap}s well above 1 s with a quite severe one at 4 s, the latter one coinciding with a packet delay spike of 4.5 s shown in the bottom subplot.

Fig. 14 is a zoomed view of the 3*rd* subplot of Fig. 13, illustrating that the majority of the inter-chirp gaps in the upstream also range between 50 ms and 200 ms.

8. Conclusion

In this paper, we introduced a novel measurement method and implementation that facilitates simultaneous high frequency measurements of key network characteristics of conventional mobile cellular networks while moving. We identified major challenges induced by the combination of high measurement frequency, rapidly changing network conditions and bulk traffic measurements. Furthermore, we explained why prior measurement methods do not cope well with highly volatile network conditions. We presented our solutions to those problems and demonstrated why the remaining self-induced congestion cannot be shortened or reduced any further.

For the future, we plan to combine our previous work on trace-based simulation and the considerably improved measurements of mobile cellular networks and release our complete tool chain, allowing the simulation of C2X-applications over mobile cellular networks using the OMNet++ implementation of our simulation model.

Acknowledgments

The authors would like to thank Sebastian Wilken, Christian Lange and Franz Kary who helped develop the RMF. Additionally the authors would like to thanks kamps.de for hosting one of the servers used in our measurements. Further on, the authors would like to thank the reviewers for their comments and suggestions which helped to improve the final version of this paper.

References

[1] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available bandwidth estimation tools, in: Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, in: IMC '03, ACM, New York, NY, USA, 2003, pp. 39–44, doi:10.1145/948205.948211.

[2] N. Hu, P. Steenkiste, Evaluation and characterization of available bandwidth probing techniques., IEEE Journal on Selected Areas Commun. 21 (6) (2003) 879–894, doi:10.1109/JSAC.2003.814505.

[3] V. Ribeiro, M. Coates, R. Riedi, S. Sarvotham, B. Hendricks, R. Baraniuk, Multifractal cross-traffic estimation, in: Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management, 2000. 15-1

[4] B. Melander, M. Bjorkman, P. Gunningberg, A new end-to-end probing and analysis method for estimating bandwidth bottlenecks, in: Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE, vol. 1, 2000, pp. 415–420, doi:10.1109/GLOCOM.2000.892039.

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

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

[7] E. Goldoni, G. Rossi, A. Torelli, Assolo, a new method for available bandwidth estimation, International Conference on Internet Monitoring and Protection (2009) 130–136, doi:10.1109/ICIMP.2009.28.

[8] speedtest.net / ookla, Accessed 11/26/2015 http://speedtest.net.

[9] speedtest.net measurement explained, Accessed 11/26/2015 https://support.speedtest.net/hc/en-us/articles/203845400-How-does-the-test-itself-work-How-is-the-result-calculated-.

[10] iperf2 - incremental enhancements to iperf, Accessed 11/26/2015 http://sourceforge.net/projects/iperf2/

[11] iperf3: A tcp, udp, and sctp network bandwidth measurement tool, Accessed 11/26/2015 https://github.com/esnet/iperf.

[12] N. Goebel, M. Koegel, M. Mauve, K. Graffi, Trace-based simulation of c2x-communication using cellular networks, in: 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS 2014) - Special Session on VANETs and ITS3, 2014, pp. 108–115, doi:10.1109/WONS.2014.6814730.

[13] NTP: The Network Time Protocol., Accessed 03/24/2016 http://www.ntp.org.

[14] P. Kovar, K. Molnar, Precise time synchronization over gprs and edge, in: R. Bestak, B. Simak, E. Kozlowska (Eds.), Personal Wireless Communications, IFIP – The International Federation for Information Processing, vol. 245, Springer US, 2007, pp. 277–284, doi:10.1007/978-0-387-74159-8_26.

[15] gpsd: a GPS service daemon., Accessed 03/24/2016 https://savannah.nongnu.org/projects/gpsd.

[16] OpenStreetMap, OpenStreetMap, [Accessed Feb. 14th 2016], Copyright and License: www.openstreetmap.org/copyright. URL http://www.openstreetmap.org/.