Fast Re-Tx: Mobile Latency Optimization with Cross Layer Analysis

Haokun Luo
University of Michigan
{haokun}@umich.edu

ABSTRACT

Slow network experience on smartphone could hurt user experience and downgrade application reputation. When users are urgent about specific information, the carried information size is small but the latency of the mobile network service is very sensitive. The discontinuous burst of short data transmission is the most common traffic pattern in the cellular network. But significant latency appears at the beginning part of the data transmission. In order to identify the root cause, I use diagnostic cross layer monitoring tool, QxDM (Qualcomm eXtensible Diagnostic Monitor), and found that inactive response to packet loss in the RLC (Radio Link Resource) leads to unnecessary timeout in the transport layer. I propose a RLC *Fast Re-Tx (Retransmission)* mechanism to avoid sluggish reaction to packet loss, and further reduce the latency during the initial network connection. Based on our simulation results, the *Fast Re-Tx* mechanism could reduce the latency by up to *35.69%* over initial network connection.

General Terms

Mobile, Measurement

Keywords

RRC state machine, Cross Layer Analysis, RLC Fast Retransmission, Latency Optmization

1. INTRODUCTION

3G cellular data networks have recently witnessed rapid growth, especially due to the emergence of smartphones. In this paper, we focus on the UMTS (the Universal Mobile Telecommunications System) 3G network, which is among the most popular 3G mobile communication technologies. However, the bottleneck of the internet resides in the first hop of the network, and large amount of lower layer retransmission could cause significant latency [10]. 3G network has a bad reputation for slow initial network connection [6].

3G network systems operate under more radio resource constraints. To efficiently utilize the limited radio resources, UMTS introduces for each user equipment (UE, for instance, a smartphone) a radio resource control (RRC) state machine, such as in Figure 1, that determines radio resource usage affecting device energy consumption and user experience [4]. Usually a UE (user equipment) can be in one of three states, each with different amount of allocated radio resources. The transitions between states also have significant impact on the UMTS system.

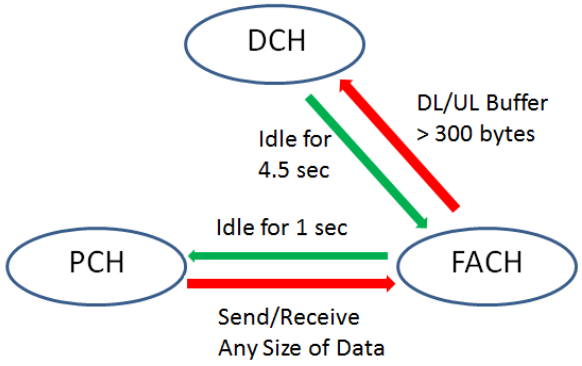

Figure 1: RRC State Machine for the 3G UMTS network for T-Mobile

The network topology provides a nature isolation between different layers. The design of transport layer protocol doesn't require the knowledge of lower layer information. However, the abstraction of the design could lead to sub-optimal scenario, i.e. large significant latency will occur during the RRC state transition. Since the root cause of the abnormal delay behavior resides in the data link layer (i.e. RLC, radio link control, layer), the visibility of lower layer information will help us identify the root cause of the bizarre behavior in upper layer. Fortunately, we have a real-time monitoring tool, called QxDM as in Figure 2, to enable use capture the radio traffic information cross multiple layers. Therefore, cross layer analysis over QxDM logs would help us have a in-depth understanding the correlation between different layers, and identify the root cause of abnormal latency behaviors during the initial state of data transmission.

2. RELATED WORK

One of the previous cross layer study combined inferred RRC state information with collected device power trace to

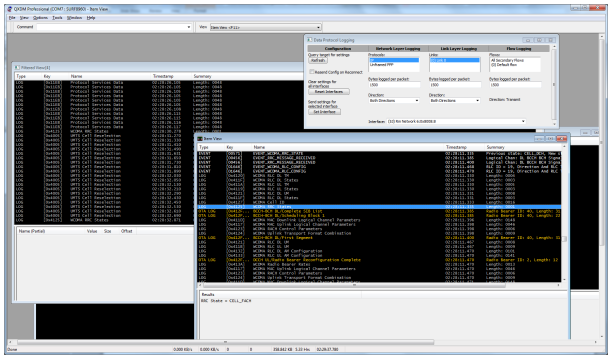

Figure 2: Ongoing monitoring and logging activities on QxDM

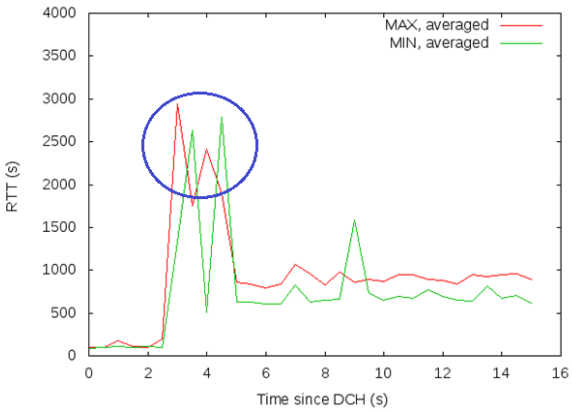

Figure 3: Unexpected large RTT during initial part of FACH state and FACH to DCH promotion transition

imply application behaviors leading to unnecessary energy consumption [9]. Their primarily focus on the application level energy optimization instead of latency. The QxDM in our study provides more fine grained performance, latency and power information. We could have more detailed insights to understand data link layer root cause for abnormal latency and inefficient energy consumption behaviors in upper layer.

The other previous study provides cross compare analysis over the TCP and RLC protocols, and optimized the configurations based on simulation [7]. Their primary goal is to improve the performance behavior by introducing TCP congestion control mechanism into RLC layer. However, their simulation result is purely based on simulated traces, which is less convincing to the real world network traffic scenarios. I collected traces from real-time cellular network traffic, and the identified latency problem is more realistic than their cases.

3. MEASUREMENTS

3.1 RRC State Inference

The previous RRC state inference study provides a decent methodology to infer the RRC state machines [8]. They want to infer the DCH demotion timer and FACH demotion timer in the RRC state machine. The methodology is to send a MAX (1024) bytes size packets to let the handset promote to DCH. Then wait for various amount of times (or called gap periods) to allow the handset stay in DCH, or demote to FACH, or even further demote to PCH. And send another packet with size of Max or Min (30) bytes. They measure the RTT (Round Trip Time) delay for each fixed gap period. They will infer the demotion timer by observing a larger RTT difference between two consecutive pre-defined gap periods. For example, if the RTT at gap period of 4 s is 3 times larger than that of 3.5 s, then they will conclude the FACH demotion timer is around 4 s. Since the handset will demote to FACH and promote to DCH (sending another packet) again at 4 s, the extra state transition delay contributes to the larger RTT. I repeat their experiments using UDP packets, and let the server respond an echo packets once the transmitted packet get received. In Figure 3, I could observe an unexpected delay over FACH initial state and FACH to DCH promotion transitions, which implies the latency that mobile user could experience during the initial stage of data transmission.

3.2 Control Experiments

The purpose of the control experiments is to verify the unexpected delays over noisy FACH state, and identify the root cause of the problem. First, we repeated our active RRC inference test for 160+ hours and dump both the client QxDM and server side tcpdump traces to perform offline analysis [1]. In order to distinguish the identities of each UDP packets, we manually injected a ”sequence number” into the first four bytes of the sender’s randomized payload, and the echo server sent back the received sequence number as acknowledge number in its payload. I refer the trace as *UDP_Trace*. Second, we designed a packet train probing using TCP packets so that we could recreate the ”pseudo-state” issue [11]. Basically, we send a ”train” of TCP packet size of 10 KB bytes with interleaving gap period of 3 seconds to 5 seconds incremented by 0.5 seconds with the total of 500 packets. The gap period is the DCH demotion timer period considering the variance in the lossy channel, and adjacent packet will transmit the packet starting from FACH state. Therefore, we will have more transition over the troublesome FACH state to allow us to take a closer look the root cause. Since TCP has its own ARQ (automatic repeat request) mechanism, it would be interesting to investigate the RLC retransmission’s delay influence over TCP retransmission. I will refer the trace as *TCP_Trace*.

4. CROSS LAYER ANALYSIS

This section briefly describes the how I collect ground truth data transmission information in IP and RLC layer using QxDM in subsection 1. Subsection 2 talks about the

QxDM Log ID	Description
0x11EB	IP data packets
0x4132	WCDMA RLC downlink acknowledge mode configuration
0x4133	WCDMA RLC uplink acknowledge mode configuration
0x413B	WCDMA RLC uplink acknowledge mode PDU
0x418B	WCDMA Flexible RLC downlink acknowledge mode PDU
0x4125	WCDMA RRC states

Table 1: QxDM log entries used in cross layer analysis

core algorithm to enable cross layer mapping. Subsection 3 verifies the latency observations using cross layer algorithm. Subsection 4 describes about how I calculate the RLC retransmission ratio in QxDM. Subsection 5 identifies the root cause of latency by results from both emphUDP_Trace and emphTCP_Trace, and propose RLC *Fast Re-Tx* mechanism to reduce the RLC delay.

4.1 QxDM Tool

QxDM is a real-time data collection and diagnostic logging tool for measuring mobile-based RF (Radio Frequency) performance [2]. It is a Windows based monitoring application. When I perform control experiments and real application measurements, I plug in the device to the desktop or laptop with QxDM software installed. Once the experiments finishes, I filtered out the real-time monitoring information related to IP packets, RLC PDUs (protocol data unit, the smallest data transmission unit in RLC layer), and RRC states in Table 1, and dump the results into a log file. The 0x11EB log entry includes IP headers, IP payloads, and its customized header. Since large IP packets will be fragmented into smaller segments, the customer header could indicate the segment index of the whole IP packet. The 0x4132 and 0x4133 unveil the RLC AM (acknowledge mode) configurations, i.e. the polling function timers, the retransmission limit for a single PDU, and etc. The 0x413B, and 0x418B provides RLC PDU header and first byte payload information for both data PDUs and control PDUs (or STATUS PDUs) in both uplink and downlink directions. We wrote a QxDM log parser to aggregate the filtered entries, and apply cross layer analysis to understand the correlation between different layers. I conduct all the experiments on two devices – Galaxy S3 with Android OS 4.1.1 and HTC One S with Android OS 4.0.4, to observe any device dependent behavior.

4.2 Verification Latency in QxDM

QxDM provides the ground truth information about RRC state, so I want to validate the delay behavior based on the inferred RRC model. To verify the packet delay in the transport layer over each RRC state, I need to know the RRC state for transport layer packets and RLC layer PDUs. Since each RRC state log is only a single log entry indicating the change of RRC state, I don't have explicit RRC state information tied to each packet and PDU. Therefore, I determine the RRC states by backtracking the most recent RRC state log, then assign as attributes to the packets and PDUs so that I could acquire RRC state information directly from the packet or PDU itself.

Because the the promotion period for PCH and FACH differs from their initial period due to additional signaling overhead, I want to distinguish between a RRC state with and without involving transitions. Since I focus on the initial period of data transmission, the RRC state promotion is more meaningful than state demotion, which is the end of whole data transmission period. So RRC state promotion is our preliminary concern for this project. We break down the normal FACH into FACH_INIT and FACH_PROMOTE states. Similarly, PCH was divided into PCH_INIT and PCH_PROMOTE. Since the FACH promotion timer is around 2 seconds and PCH promotion state is around 0.5 seconds. I define the FACH_PROMOTE state as the time slot of counting back 2 seconds from the point of promoting to DCH. For the same reason, PCH_PROMOTE state describes the time slot of counting back 0.5 seconds from the point of promoting to FACH.

From the *UDP_Trace*, I calculate the UDP RTT by mapping the client transmitted UDP packets to the served echoed back packets through comparing the manually injected "sequence number". If both packets have the identical "sequence number" in their first four byte payload, then the RTT for the transmitted data is the timestamp difference between the two packets. Figure 4 shows the UDP RTT broken down into different RRC state and state transitions. The average RTT at FACH_INIT state is *2.52 s* for HTC One S and *2.08 s* for Galaxy S3, which around both around *190%* greater than each one's RTT in DCH. The average RTT for FACH_PROMOTE state is *1.14 s* for HTC One S and *1.05 s* for Galaxy S3, which are around *40%* greater than those in DCH. The result could validate the abnormal delay behavior especially during the initial part of the FACH state, namely the FACH_INIT state. We could also observe a tremendous latency over PCH_PROMOTE state. Because packets cannot stay in PCH during the transmission, the promotion to FACH and DCH will double the signaling overhead, which introduces more latency.

4.3 Cross Layer Algorithm

QxDM tool provides fine grained lower layer RLC layer transmission information. If I could correlate the transport layer packets with the RLC layer transmitted PDUs, I will have a transparent view of the link layer behaviors, especially the RLC retransmission. One of the fundamental limitation of QxDM is the partial logging issue. For example, only the header and first byte data payload will be logged for

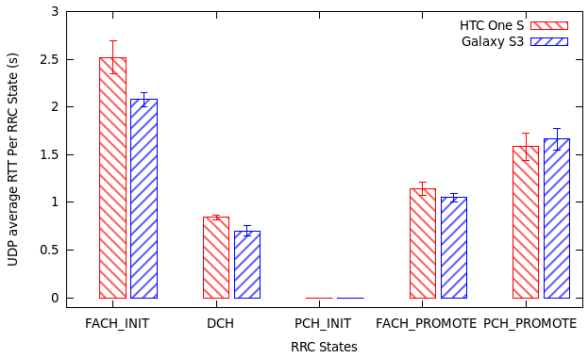

Figure 4: UDP RTT calculated from the QxDM log result

each RLC PDU. It is also possible that a small fraction of RLC PDUs cannot be captured, which lead to a unnecessary sequence number gap. Our mapping algorithm will handle all the limitations I have mentioned.

The cross layer mapping algorithm is essentially a map between the complete IP packets (known as SDU) and corresponding fragmented RLC payload data bytes (known as PDU). Due to the partial logged information in QxDM, only the first data byte is captured in the log. Thus, I have to skip over the rest of the PDU, and try to match for the first data byte in the next PDU. The problem at this point is to determine the end of IP packets while I iterate through the consecutive RLC PDUs. Since each PDU could either contain the payload data dedicated to a single SDU or belongs to two SDUs. If the reminder size of the SDU cannot fulfill the largest size of PDU, then RLC protocol will concatenate the part of the next SDU to fill the rest of space [3]. Ultimately, if the accumulative mapped index equals the size of SDU, I claim to find a mapping successfully; otherwise no mapping discovered. Algorithm 1 states the detailed information of the mapping mechanism.

There is a corner case in the mapping algorithm such that the QxDM cannot capture the some of the SDUs. Similar to TCP protocol, the sequence number in RLC PDUs could uniquely distinguish between every PDUs. If there are some missed PDUs, then I cannot map the first byte data for every PDU size. In that case, I could even skip over the missed PDUs and add up multiple of PDU size to hunt for a match, which is represented as ”factor” variable in the mapping algorithm. However, the aggressive leap mapping mechanism cannot fully recover the corner case, especially when the missed PDUs were either the beginning or the end part of the mapped RLC list. I evaluate the improved mapping algorithm by checking the percentage of mapped IP packets in all the traces of control experiments, and the average mapping ratio is *99.8%*.

4.4 RLC Retransmission Calculation

The RLC layer retransmission is a protection mechanism that maximize the reliability over a loss data transmission channel. However, it also causes the delays due to duplicate

Algorithm 1 Decide whether one TCP/UDP packet maps to all corresponding RLC PDUs

Function *isCrossLayerMapped(SDU, PDU_list):*
PDU_index = 0
SDU_byte_index = 0
while *SDU_byte_index* $<$ size of *SDU* **do**
  if *SDU[SDU_byte_index]* == *cur_PDU* **then**
    cur_PDU = *PDU_list[PDU_index]*
    $factor$ = *cur_PDU*.sn - *last_PDU*.sn // sn is sequence number
    last_PDU = *cur_PDU*
    if *cur_PDU* has LI field in header **then**
      Increase *SDU_byte_index* by the value of LI field
    else
      Increase *SDU_byte_index* by the size of *cur_PDU* multiples $factor$
    end if
    if *cur_PDU*'s HE field is 2 **then**
      Break
    end if
    Increase *PDU_index* by 1
  else
    Break
  end if
end while
if *SDU_byte_index*== size of *SDU* **then**
  return true
else
  return false
end if

transmissions without noticing the upper layers. To assist further identifying the root cause of the latency, we first introduce how we calculate the RLC retransmission from the QxDM logs.

The sequence number of the RLC PDU will uniquely identify each RLC packet. Therefore, I determine the RLC retransmission based on the duplicate of sequence numbers. Since the size of RLC PDU is only 42 bytes and is very limited, if the RLC keeps an effective throughput, it has to reduce the number of bits in the header to hold the sequence number. Therefore, the sequence number only consumes 12 bits (range of 0-4095) in the current RLC protocol [3]. As a result, sequence number is circularly reused every 4096 PDUs, and it is always increasing. To avoid over-count the duplicate sequence number in different cycle, I hard set a smaller window size of 512 to count the reappearing sequence number within that range. I determine the RRC state for each RLC PDU by backtracking to most recent RRC state log entry, and fetch the RRC state based on that log content. I break down RLC retransmitted PDUs based on their RRC states, and I calculate the retransmission ratio through dividing total number of retransmitted PDUs per RRC state by the total number of PDUs in that RRC state.

4.5 Root Cause Analysis and Solutions

There are two possible behaviors of RLC retransmission that leads to transport layer delay. The first one is that retransmission is necessary because of noisy channel during FACH state. Based on *UDP_Trace*, I could see a strong correlation between RLC retransmission ratio and FACH state as an evidence. One possible solution is that application could batch the data transmission to reduce the RRC state promotion frequency [8]. The second one is the retransmission get unnecessary delayed caused by the lagging response to the PDU loss signal. I observe it from *TCP_Trace* when I studied the relationship between TCP retransmission and RLC retransmission. I will also provide a improved RLC mechanism to avoid the unnecessary delay later.

Analyzing the *UDP_Trace* based on the RLC retransmission calculation methodology, we calculate the RLC retransmission ratio per RRC state in Figure 5. As we can see, the RLC retransmission among FACH and FACH_PROMOTE in total are *53.46%* for HTC One S device with standard deviation of *4.58%*, and *13.74%* for Galaxy S3 devices with standard deviation of *1.14%*. It strongly suggested the significant delay over the FACH_INIT and FACH_PROMOTE states, which indicates the higher chance of RLC retransmission in a resource shared and noisy channel during FACH state. We could also observe that the HTC device has a higher loss rate than that of Galaxy S3. One possible explanation for device dependent behavior is that the difference in hardware module. The HTC One S was produced over Snapdragon S3 MSM8260, while the Galaxy S3 contains Snapdragon S4 MSM8960, which is the next generation of the one for HTC One S. The wireless radio techniques

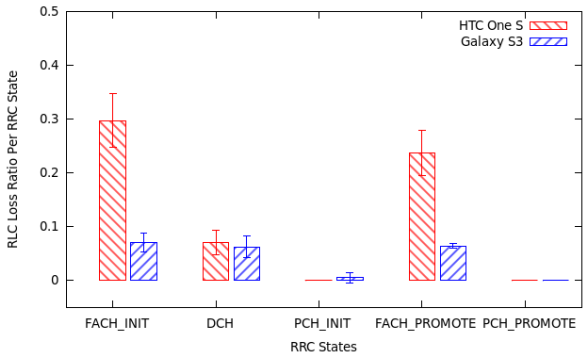

Figure 5: Significant RLC retransmission ratio over stable FACH state and FACH promotion state strongly implies the upper layer latency during the beginning part of the data transmission

for Snapdragon S4 has support for larger range of network types [5]. The detailed discussion about the hardware differences is out of the scope of this paper.

TCP retransmission timeout (RTO) could cause apparent round trip delay due to congestion window drop by half and restarting the data transmission from the slow start phase [13]. From the *TCP_Trace*, I correlate the TCP retransmission behavior with the RLC retransmission through our mapping algorithm. By capturing the root cause of the TCP RTO, I found that current RLC protocol has a sluggish response to the PDU lost signals – the duplicate ACKs, in Figure 6. The delayed RLC retransmission PDUs leads TCP RTO, which further introduces latency in the transport layer. In 3GPP RLC specification, the sender only retransmits the PDU once it receives the STATUS LIST (or non-acknowledged) control PDU from the receiver, but ignores to duplicate ACKs, which is a strong hint for PDU loss [3]. If we could bring the group of retransmitted RLC PDUs from 2.8 s to 0.5 s, then we could avoid TCP RTO, reducing the latency more than 2.3 s. Therefore, I propose a RLC *Fast Re-Tx* mechanism, which will retransmit the unacknowledged PDUs once the sender receives three duplicate RLC PDU ACKs. The faster reaction to loss signals could reduce RLC layer latency and further reduce the latency in transport layer.

5. EVALUATION

5.1 RTT Estimation

We first define the RTT in RLC layer. In the RLC protocol, the STATUS PDUs (ACK or NACK) don't generate by every received RLC PDU, but triggered either by receiving a polling request from the sender or by detecting one or more missed PDUs from its receiver buffer [3]. Since the QxDM traces are collected at the client side, we don't have the information of when the server side receive the PDU. Thus, I estimate the RTT of a RLC PDU based on the timestamp difference between the most recent sender's polling

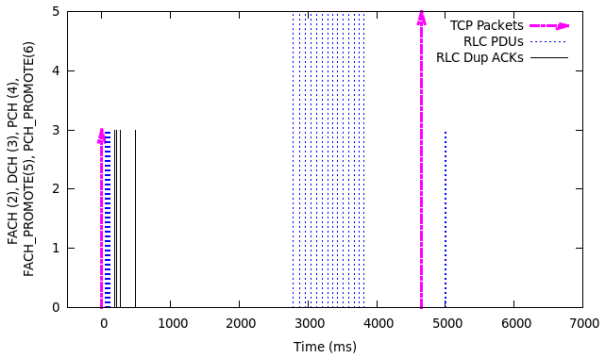

Figure 6: Two Latency Causes: short DCH demotion timer and lagging response to the PDU lost signal

request and received ACK. Based on RLC configuration, the maximum polling request frequency is 500 ms. One of the previous mobile RTT estimation study shows the autocorrelation coefficient of two RTT measurements within 500 ms is more than 0.6 [12]. Therefore, my estimation is still reasonable to be considered as the real RLC RTT value.

5.2 Cost-Benefit Analysis

In order to know whether the new proposed RLC mechanism works, I apply a cost-benefit analysis over the existing *TCP_Trace*. The definition of benefit and cost is straight forward. Basically, if the fast retransmitted PDUs will be transmitted in the future, then we compare the RTT if it transmitted right after the duplicate ACKs with the real RTT value in the trace. If the difference is less than 0, we call that is a benefit. In the same way, if it is greater than 0, we call it a cost. However, we want to know if the PDU is really lost over the channel, or it just gets delayed due to channel contention. That depends on whether the sender receives a ACK or NACK (a list of unreceived PDU sequence numbers). We categorize the situations into four cases – Win, Draw_Plus, Draw_Minus, and Loss. If the sender will receive a NACK, and more than 50% of the fast retransmitted PDUs will retransmit in the real trace, then we call the case "Draw_Plus" as in Figure 8. Since it would brings us benefit if the fast retransmit RTT is less than the RTT in the real trace. The "Win" case is defined that if we could avoid a TCP RTO based on the "Draw_Plus" in Figure 7. In that case, RLC layer benefit is the same as "Draw_Plus", but I want to highlight that it could bring further latency benefit over the transport layer. "Draw_Minus" occurs when less than 50% of the fast retransmitted PDUs get really retransmitted in the real trace as in Figure 9. If the sender gets a ACK with a larger sequence number, then all the PDUs were successfully delivered to the receiver. In that case, we called the case "Loss", since all the retransmitted packets are redundant as in Figure 10.

As we can see from Table 2, around 75% of the time we will have benefit on RLC latency reduction if we count the "Win" and "Draw_Plus". The overall RLC delay could be

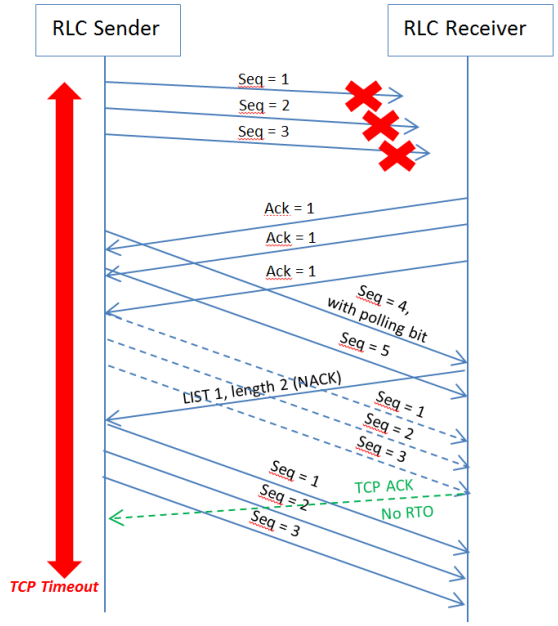

Figure 7: Win: RLC Fast Re-Tx avoid a TCP RTO

Case Name	Percentage of Occurrence (%)
Win	10.32±1.89
Draw_Plus*	64.68±8.32
Draw_Minus	20.63±3.45
Loss	4.25±0.06

* The Draw_Plus case excludes the percentage of Win

Table 2: The RLC Fast Re-Tx Cost-Benefit Table

reduced by *2.66%* over all the RRC state based on trace simulation analysis. If we break down the cost-benefit over different RRC states, the RLC RTT latency could reduce by up to *35.69%* over initial FACH state or FACH promotion transitions. Therefore, we could have a large latency benefit over the initial period of data transmission.

6. DISCUSSION

6.1 Future Work

The current study is primarily focus on the latency improve over the RLC layer. It is also possible to measure the delay improvement over application layer directly, especially for the initial period of the data transmission. Another possible direction is to evaluation the energy efficiency for the improved RLC protocols with fast retransmission mechanism. Since QxDM provides device data transmission power information, we could also evaluate the energy saving over each RRC state and state transitions.

6.2 Limitation

Since we could not modify the handset's NIC (network interface card) driver, we only evaluate the modified trace in-

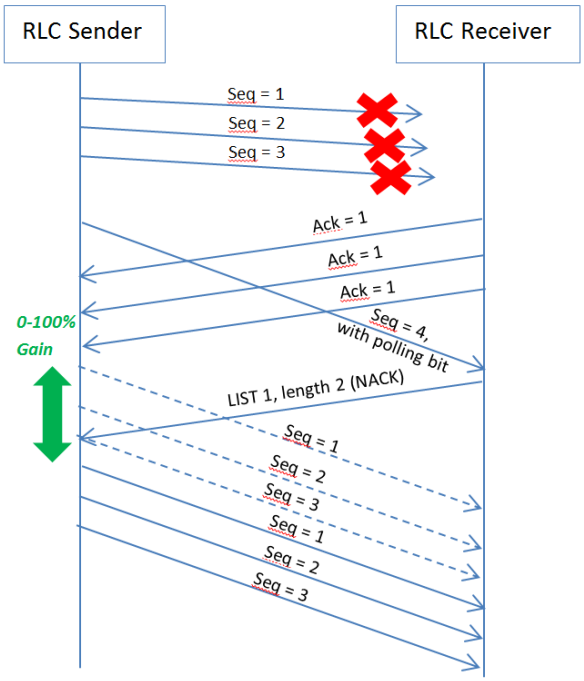

Figure 8: Draw Plus: the predication accuracy is more than 50%

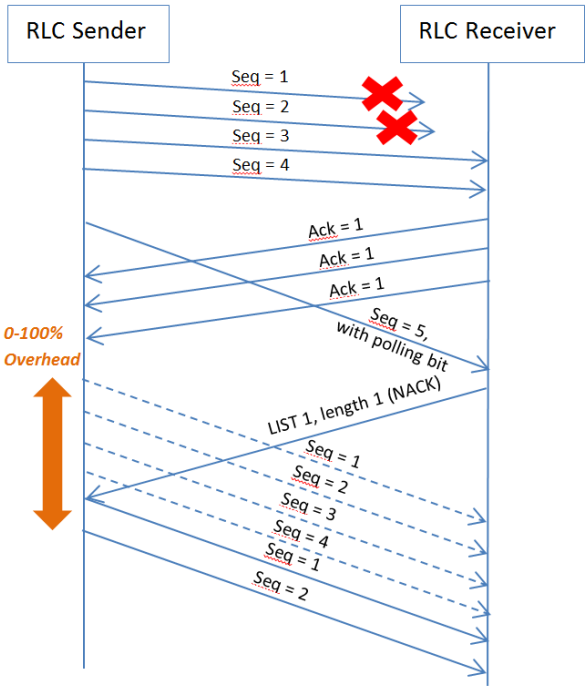

Figure 9: Draw Minus: the predication accuracy is less than 50%

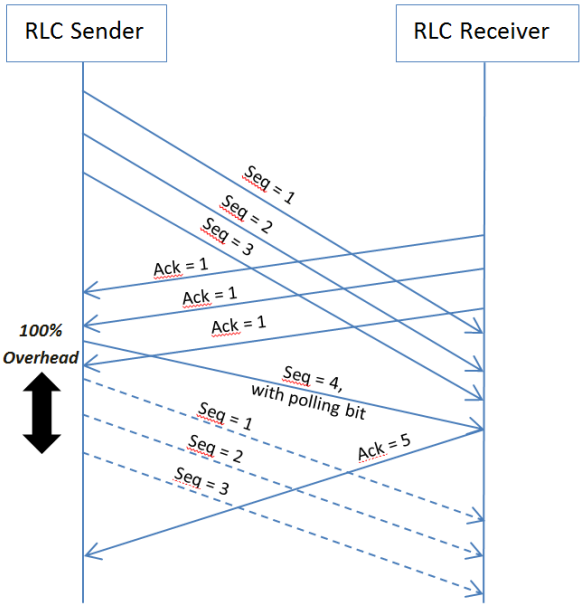

Figure 10: Loss: all predications fail

formation based on the QxDM traces. Currently we assume the fast retransmitted RLC PDUs guarantee to be received on the base station, but the actual channel is lossy and the actual benefit might be lower than the simulated results. In addition, since the modification of RLC protocol requires hardware modification, the deployment of the modified protocol could be slower than a software solution.

7. CONCLUSIONS

From the RRC state inference model, I evaluated the latency across each RRC state and state transitions. I observed the extra latency in the FACH state and FACH promotion transitions. Utilizing the QxDM monitoring tool, we have a better visibility over RLC layer traffic information. I wrote a QxDM log parser and analyzer to cross mapping the transport layer and data link layer information, and identified the root cause of abnormal delays cause by channel contention and imperfection in the RLC protocol. I proposed a RLC *Fast Re-Tx* mechanism to actively response to the PDU loss, and evaluated the delay cost-benefit over real-time traces. The latency reduction over FACH and FACH promotion state is up to *35.69%*, and it could also reduce the overall latency by *2.66%*. Therefore, the RLC *Fast Re-Tx* mechanism could enhance the user mobile experience by introducing less delays, especially during the initial states of data transmission.

8. ACKNOWLEDGMENTS

We would like to thank Yihua Guo, Sanae Rosen, and Z. Morley Mao for supporting and commenting on this paper.

9. REFERENCES

[1] tcpdump. http://www.tcpdump.org/.
[2] QxDM Professional Proven Diagnostic Tool. http://www.qualcomm.com/solutions/testing/diagnostics-software/, 2012.
[3] 3GPP TS 25.322: Radio Link Control (RLC) - UMTS, 2013.
[4] 3GPP TS 35.331: Radio Resource Control (RRC) - UMTS, 2013.
[5] Snapdragon (System on Chip). http://en.wikipedia.org/wiki/Snapdragon_(system_on_chip), 2013.
[6] Why are 3G Networks so Slow? http://blog.ioshints.info/2013/04/why-are-3g-networks-so-slow.html, 2013.
[7] Alcaraz, Juan J., Fernando Cerdn, and Joan Garca-Haro. Optimizing TCP and RLC interaction in the UMTS Radio Access Network. *IEEE Network Magazine*, 2006.
[8] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Z. Morley Mao, and Subhabrata Sen and Oliver Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *Proc. ACM IMC*, 2010.
[9] Feng Qian, Zhaoguang Wang, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. Profiling Resource Usage for Mobile Applications: A Cross-layer Approach. In *Proc. ACM MobiSys*, 2011.
[10] Haiqing Jiang, Zeyu Liu, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Understanding Bufferbloat in Cellular Networks. In *Proc. of the 2012 ACM SIGCOMM workshop on Cellular networks*, 2012.
[11] Hu, Ningning, and Peter Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE J. Selected Areas in Communications*, 2003.
[12] Qiang Xu, Sanjeev Mehrotra, Z. Morley Mao, and Jin Li. PROTEUS: Network Performance Forecast for Real-Time, Interactive Mobile Applications. In *Proc. ACM MobiSys*, 2013.
[13] Vern Paxson and Mark Allman . Computing TCPs retransmission timer, 2000.