RaptorStream: Boosting Mobile Peer-to-Peer Streaming with Raptor Codes

Philipp M. Eittenberger
University of Bamberg, Germany
philipp.eittenberger@ieee.org

ABSTRACT

As mobile devices and cellular networks become ubiquitous, first apps for popular P2P video streaming networks emerge. We have observed that when these applications operate in cellular networks, they don't upload video traffic back to other peers. This paper presents a reason for this behavior and proposes a viable solution to exploit the uplink capacity of mobile devices more efficiently. To the best of our knowledge, this paper is the first to propose the usage of Raptor codes to increase the upload throughput of mobile P2P applications.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Measurement, Performance

Keywords

Android, Mobile P2P Streaming, Raptor Codes

1. INTRODUCTION

Recently, client applications of popular P2P video streaming services have been published for mobile devices (e.g. TVU-Player and SopCast). To assess the performance of this new breed of application, we conducted a measurement campaign with the SopCast client that is operating in a High Speed Packet Access (HSPA) network. One of our key findings is that the client doesn't upload any video data back to other peers. To investigate the reasons of this behavior, we have implemented a measurement framework for Android and conducted several measurement runs in the HSPA network of T-Mobile in Bamberg (Germany). Our measurement framework comprises one Android device that either sends packets to one server or receives packets from it. Packet loss is one metric that has been gathered, in particular, we wanted to investigate the influence of the packet inter-arrival time and packet size on the resulting packet loss rate. The packet inter-arrival time is generated by the sender by delaying each packet transmission for a specified time span. Figure 1 illustrates an excerpt of the outcome of one measurement run, for each combination of packet size and inter-arrival time 1,000 UDP packets have been send from a stationary mobile device to the server. Despite HSPA's inherent hybrid ARQ (HARQ) mechanism, the packet loss rate is excessively high (around 60 - 70 % without a clear effect pattern). More importantly, for smaller batch sizes (about 100 packets) the packet loss drops towards zero, which means that for this number of sent packets the underlying HARQ is working. Yet, when batch sizes increase, the Android device is not able to process the retransmission request messages in a timely manner and thereby, a overflow of the UDP buffer happens, which is responsible for the high packet loss. As shown in Figure 1, even relatively large packet inter-arrival times of 20 ms can't redeem this failure. Not surprisingly, the high packet loss negatively affects the achievable uplink throughput. For instance, Figure 2 shows the outcome of one measurement run conducted with our framework to estimate the available instantaneous uplink throughput in relation to different packet sizes. It presents the 25 % quantile of instantaneous TCP uplink throughput measurements for different packet sizes. The instantaneous throughput T is computed by

$$T = \frac{l\,[Byte]}{\Delta t\,[s]}$$

with frame size l and the inter-arrival time Δt between two consecutive packets, such that $\Delta t = t_{i+1} - t_i$. On the receiving side lost packets are ignored when calculating T_r such that $\Delta t^r = t^r_{i+1} - t^r_i$ is the difference of two successively received packets. Due to buffering effects on the path between the stationary measurement server and the Android client, it can be observed that sometimes the 25 % quantile of the received instantaneous throughput values is higher compared to the sending throughput.

Due to retransmissions and large latencies commonly encountered in cellular networks, the uplink TCP throughput suffers and doesn't even reach 128 Kbit/s in this measurement run. We were able to reach under identical conditions a maximum uplink throughput of 480 Kbit/s in our experiments by using UDP as transport protocol. However, it is hardly possible to transmit useful video data in time by UDP due to the high packet loss, because UDP doesn't ensure in-order delivery or retransmissions. It is thus no surprise that the investigated mobile P2P streaming clients didn't upload any video data.

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
ACM 978-1-4503-1419-0/12/08.

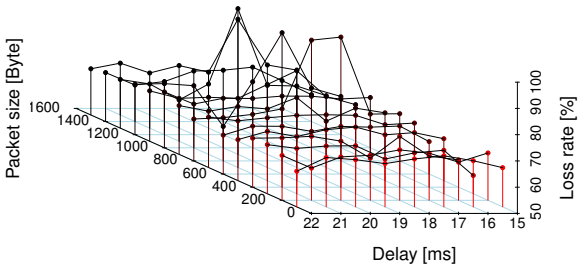

Figure 1: Packet loss in a HSUPA network for different packet sizes and inter-arrival times.

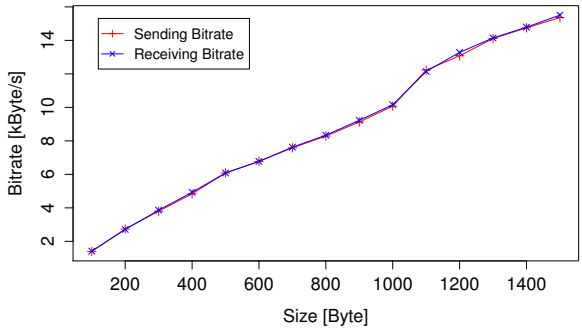

Figure 2: Uplink TCP throughput in a HSUPA network for different packet sizes.

2. RAPTORSTREAM

One possible solution to exploit the uplink capacity of mobile devices in cellular networks is given by the usage of one particular class of forward error correction (FEC) codes. Due to their desirable property of ratelessness, Fountain codes have the ability to generate theoretically an unlimited amount of uniquely encoded data on-the-fly. This property eliminates completely the need for content reconciliation, as no redundant content exists in the network. Fine grained chunk scheduling would not be needed any more, since a receiving peer could restore the original data needing just a slightly larger amount of encoded data from any set of peers.

Raptor codes have been developed by Amin Shokrollahi [4] as an advancement of Tornado codes and represent the first practical class of a Fountain code with near optimal error correction functionality. It is possible to reduce the scheduling complexity of the P2P dissemination and to cope with the packet loss in cellular networks by using Raptor codes as application layer forward error correction (AL-FEC). For further practical details regarding the usage of Raptor codes in the context of P2P streaming, we would like to refer the reader to [2].

As every data unit is uniquely encoded by a Raptor code, any arbitrary subset of encoded data is sufficient to restore the original data, given that a small price, i.e. overhead, has to be paid. Therefore, the usage of Raptor codes in combination with UDP as transport protocol would be a viable option to increase the uplink throughput. We implemented a first base line version of the R10 Raptor code [3] in plain Java. The implementation was not specifically optimized for encoding/decoding speed, e.g. by using a native language like C or GPU instructions. However, it is able to run on every Android certified device without further adjustments due to the pure Java implementation. First performance measurements conducted on a commodity Android device (Galaxy Tab 7 - GT-P1000 with a 1.0 GHz ARM processor) yielded average encoding throughputs of approx. 700 KBit/s and mean decoding throughputs of approx. 800 KBit/s (by using the fastest parameter combinations for encoding and decoding). This performance is already enough to fill the measured available uplink capacity of recent 3.5G networks without further optimizations.

We have already presented RapidStream [1], which is a first prototypical implementation of a mobile P2P video streaming application for Android devices. Right now, we are in the process to incorporate our Raptor code implementation into the streaming engine of RapidStream, which will be then called RaptorStream.

3. CONCLUSION

Our measurements have raised some questions that need further investigations with regard to the future deployment of P2P applications in cellular networks: For instance, at which number of sent packets does the high packet loss emerge? How can that behavior be avoided? In addition, we have proposed the usage of Raptor codes as AL-FEC to exploit the upload capacity of mobile devices and to reduce the complexity of P2P chunk scheduling. In general, Raptor codes require only a small overhead rate and are able to support the necessary streaming rates. For future work, we plan to conduct extensive field trials to evaluate the performance of our proposal after the completion of RaptorStream.

Acknowledgments

The author would like to thank Marcel Großmann for his assistance with the measurements and Todor Mladenov for his guidance through Raptor coding. Furthermore, the author is grateful to Dr. Ingo Dahm of the Deutsche Telekom AG for the provided access to the T-Mobile 3G network.

4. REFERENCES

[1] P. M. Eittenberger, M. Herbst, and U. R. Krieger. RapidStream: P2P Streaming on Android. In *19th International Packet Video Workshop.*, 2012.

[2] P. M. Eittenberger, T. Mladenov, and U. R. Krieger. Raptor Codes for P2P Streaming. In *20th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.*, 2012.

[3] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. RFC 5053, 2007.

[4] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52:2551–2567, June 2006.