An Evaluation of Mobile End Devices in Multimedia Streaming Scenarios

Michael Ransburg, Mario Jonke, and Hermann Hellwagner

Multimedia Communication (MMC) Research Group, Institute of Information Technology (ITEC), Klagenfurt University, Klagenfurt, Austria
{mransbur,hellwagn}@itec.uni-klu.ac.at,mjonke@edu.uni-klu.ac.at

Abstract. This paper compares handhelds based on the iPhone and Android operating systems in multimedia streaming scenarios. We simulate typical Internet network impairments, i.e. packet delay and packet loss, and evaluate their effects on the end devices. Additional evaluations include bandwidth overhead inflicted by the different streaming approaches and traffic shape and fairness when both handhelds consume media simultaneously. Based on the quantitative evaluation, both approaches show weaknesses and strengths. A final qualitative discussion points out additional advantages for the streaming approach implemented in the iPhone operating system.

Key words: multimedia, streaming, mobile, RTP, HTTP, comparison

1 Introduction

In this work we compare the iPhone 3.0 to the Android 1.6 operating system (OS) in multimedia streaming scenarios. For the iPhone OS a second generation Apple iPod Touch was chosen as a platform and for the Android OS the HTC Magic served as a platform. Both operating systems support the codecs H.264/AVC (AVC) [1] for video and MPEG-4 AAC (AAC) [2] for audio and are able to receive streamed multimedia content but they use different approaches to enable this. While the Android OS relies on the well known RTP protocol for AVC [3] and AAC [4], the iPhone OS facilitates a new approach called HTTP Live streaming. Other than RTP, HTTP Live streaming is a pull-based protocol which relies on breaking the overall stream into a sequence of small HTTP-based file downloads, which are referenced by an extended M3U playlist. Further information on HTTP Live streaming can be found in [5]. Our goal in this work is to evaluate which of the two streaming mechanisms is more suitable for streaming multimedia content to mobile end devices.

The remainder of this paper is organized as follows. In Section 2 we present our evaluation environment, i.e. the test data, test methodology and an overview of our testbed. Subsequently, we first evaluate the influences of packet delay and packet loss on the startup delay and playback in Sections 3 and 4, respectively. The bandwidth overhead which is caused by the two alternative streaming approaches is evaluated in Section 5. Next, we look at traffic shape and fairness in

Section 6, by serving both handhelds simultaneously over the same channel and analyzing the traffic. Finally, Section 7 concludes our evaluation and provides an outlook to future work.

2 Evaluation environment

2.1 Test data and methodology

As source content 100 seconds from a teaser from the movie Ice Age 3 were chosen and encoded in AVC for video and AAC for audio, which are the standard codecs supported by both operating systems. Since both operating systems only support the AVC baseline profile, B-frames, the CABAC filter and the Trellis algorithm were disabled during encoding. Audio was encoded with 96 kbps and two channels. For video we created variations differing in the temporal, spatial and quality domain. Table 1 shows the different encoding characteristics for the test sequences, based on common usage and on the end device capabilities, i.e. the highest resolution which is supported by both handhelds is 480x320 pixels and the highest bit rate is 500 kbps.

Resolution [px]	480x320	320x240	240x160	160x120	
Frame rate [fps]	30	25	24	20	12.5
Bit rate [kbps]	500	400	200	100	

Table 1. Encoding characteristics for test sequences

In the case of the iPhone OS two extra steps need to be performed, in order to prepare the content for HTTP Live streaming. First, the content is multiplexed into a MPEG-2 transport stream and second, it needs to be fragmented into segments. We used a minimum length of 10 s for each segment, as suggested in [5]. However, not all segments are of equal length, since the segmentation can only be performed at Instantaneous Decoder Refresh (IDR) frames within the video sequence. Therefore, the segmenter has to wait for the next IDR-frame to appear, before a new segment can be started. In our case, we used an IDR-frame interval of 250. This is the default for *MEncoder*[1], which we used for encoding the contents. In the worst case this means that an IDR-frame may occur shortly before the specified segment duration in which case the specific segments duration may almost double. Additionally, this encoder may decide to introduce additional IDR-frames in case of scene cuts.

We did not modify the handhelds in any way and used the native media players in order to get representative results. This means that our startup delay measurements were done manually with an external timer which leads to higher errors in measurement. To compensate for this factor and for unexpected behavior of the handhelds 10 repetitions were performed for each measurement.

[1] http://www.mplayerhq.hu, SVN-r29411

Considering all the entries in Table 1, a total of 80 different variations of the original sequence were created by incorporating every possible permutation. Because of space limitations we will focus on a relevant subset of our results in the following.

2.2 Testbed

Figure 1 shows our testbed, which consists of a combination of several different Linux computers with kernel version 2.6.27, running the Ubuntu distribution. The computers `pl05` and `pl06` act as content providers and are the locations where the HTTP Web server (in our case Apache HTTP Web server[2]) and the RTP streaming server (in our case Live 555 Media Server[3]) are located. The

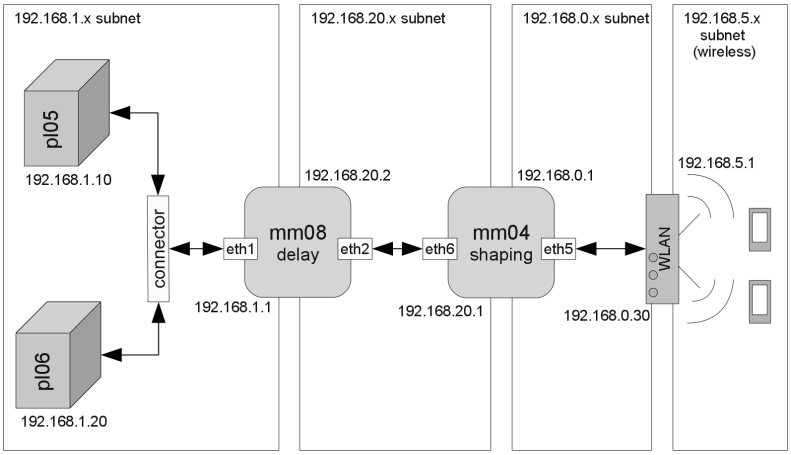
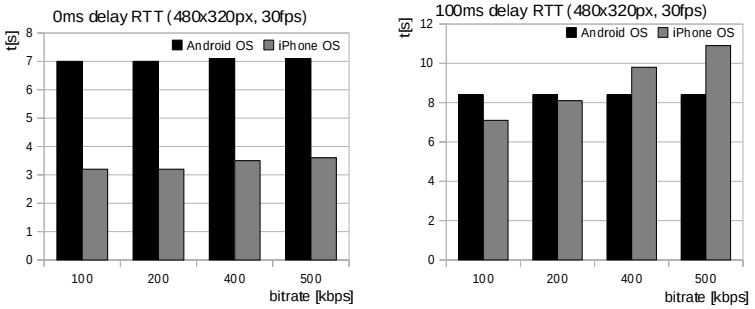


Fig. 1. Testbed

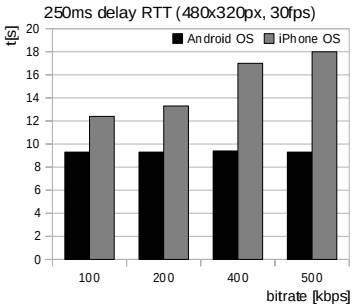
machine `mm08` is used to introduce additional network delay by increasing the round trip time (RTT) of packets, whereas `mm04` is used to apply traffic shaping and packet loss. For this, the Linux traffic control mechanism and its extension *Netem*[4] was used. *Tcpdump*[5] and *wireshark*[6] were used to record the traffic at the client side as well as the server side.

With the help of the test environment several measurements were performed, using the presented test set. These measurements included the startup delay, the impacts on playback, as well as the observation of traffic fairness, shape and data overhead. In the following sections the results are presented.

[2] http://www.apache.org, v2.28
[3] http://www.live555.com, v0.3
[4] http://www.linuxfoundation.org/en/Net:Netem, Kernel v2.6.27
[5] http://www.tcpdump.org, v3.9.8 (libcap 0.9.8)
[6] http://www.wireshark.org, v1.2.7



(a) no network delay (b) Round trip time 100ms



(c) Round trip time 250ms

Fig. 2. Comparison of different round trip times

3 Evaluation of startup delay

The term startup delay describes the time between the request of a sequence and the actual start of the playback.

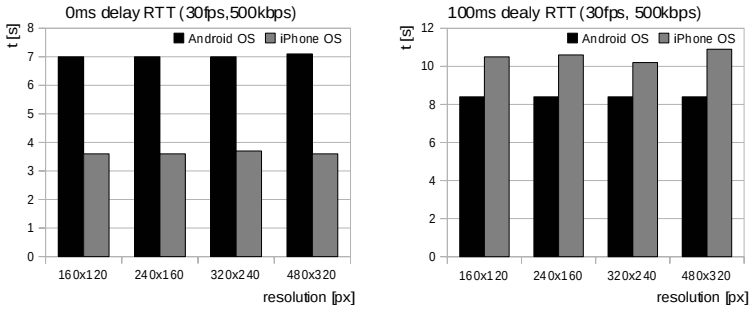
3.1 Effect of packet delay on startup delay

In the Internet, the RTT depends on the infrastructure, its utilization and the geographical localization of hosts in the network. The study about round trip time conducted by Acharya et al. [6] shows that the wide majority of investigated hosts have shown an inherent RTT lower than 250 milliseconds. Therefore, RTT values of 0, 100 and 250 milliseconds where chosen for the evaluation. The delay distribution of forward and backward channel was set symmetrically.

The bar charts in Figure 2 show the startup delay for the test sequences with a resolution of 480x320 pixels and a frame rate of 30 fps. As depicted in Figure 2(a), the startup delay is about seven seconds for Android OS and three seconds for the iPhone OS. Without introducing packet delay, the startup times are relatively constant. However, once packet delay is introduced, the characteristics of the operating systems start to differ as can be seen in figures 2(b) and 2(c).

The startup delay on the Android OS increases linearly when increasing the RTT, on the iPhone OS however, we can observe an exponential increase of startup delay depending on both the RTT and the bit rate.

The explanation for this difference is twofold. First, with RTP only a few frames of a sequence need to be received (i.e. until the playback buffer is full) before playback can be started, which is contrary to HTTP Live streaming where the playback does not start until a whole segment is received. Second, since the HTTP Live streaming approach is based on TCP, the amount of data it can send out at once is constrained by the contention window, limiting the number of packets that can be sent before receiving an ACK. Thus, the server is only allowed to send out a small amount of a packets at once. Due to this condition the overall delay multiplies with the amount of data and the increasing delay in the network. This restriction does not apply to the Android OS, since UDP, which RTP relies on, does not involve acknowledgement packets.



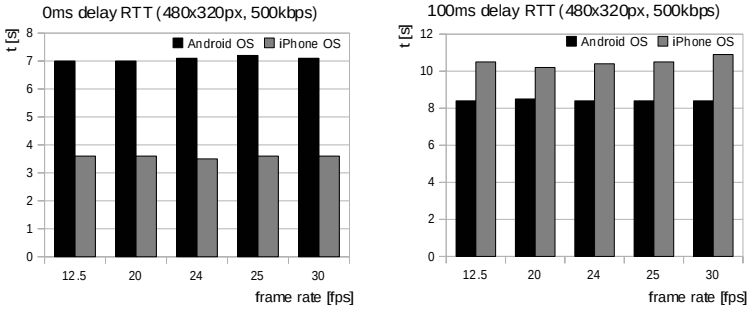
(a) no network delay (b) Round trip time 100ms

Fig. 3. Impacts on startup delay considering different video resolutions

So far, the evaluation focused on variations in bit rate, leaving the parameters for frame rate and resolution selected in the test set unchanged. Figures 3 and 4 show the impact of various resolutions and frame rates on the startup delay. As can be seen, in both cases and for both operating systems the startup delay increases linearly when increasing the resolution or frame rate (while keeping the bit rate constant at 500 kbps). This can be explained by the fact that for the transmission of a segment in HTTP Live streaming only the bit rate (and thus segment size) is a relevant factor. In case of RTP, we assume that the startup delay mostly depends on the size of the playback buffer. We cannot verify this, since the playback buffer is not user-configurable on the Android OS.

3.2 Effect of packet loss on startup delay

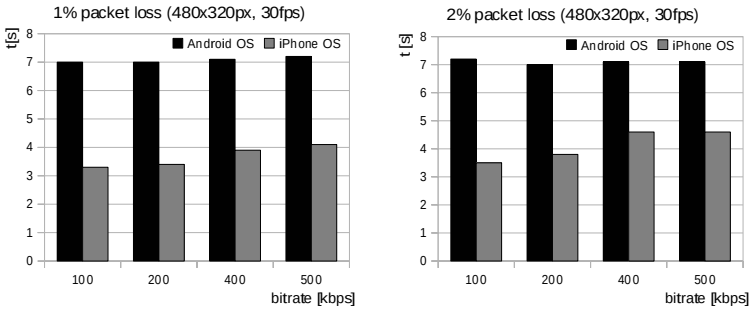
Packet loss is typically caused by congested networks, since routers can only buffer a finite amount of packets at a time, thus certain selected packets have to



(a) no network delay (b) Round trip time 100ms

Fig. 4. Impacts on startup delay considering different frame rates

be dropped. Investigations performed by Wang et al. [7] have shown that about 95 % of tested Internet links have a packet loss rate lower than 2 %. Therefore, a maximum packet loss rate of 2 % was chosen to be considered for the evaluation.



(a) 1% packet loss (b) 2% packet loss

Fig. 5. Impact of packet loss on startup delay

The Figures 5(a) and 5(b) show that packet loss does not influence startup delay for the Android OS. For the iPhone OS however, packet loss results in an increased startup delay, since TCP is a reliable protocol which retransmits lost data packets. Although packet loss increases the startup delay, the impacts are not as high as they are in the case of introduced network delay.

4 Impacts on playback

After evaluating the startup delay in the previous section, we now focus on the playback, i.e., how will factors such as packet delay and packet loss impact the

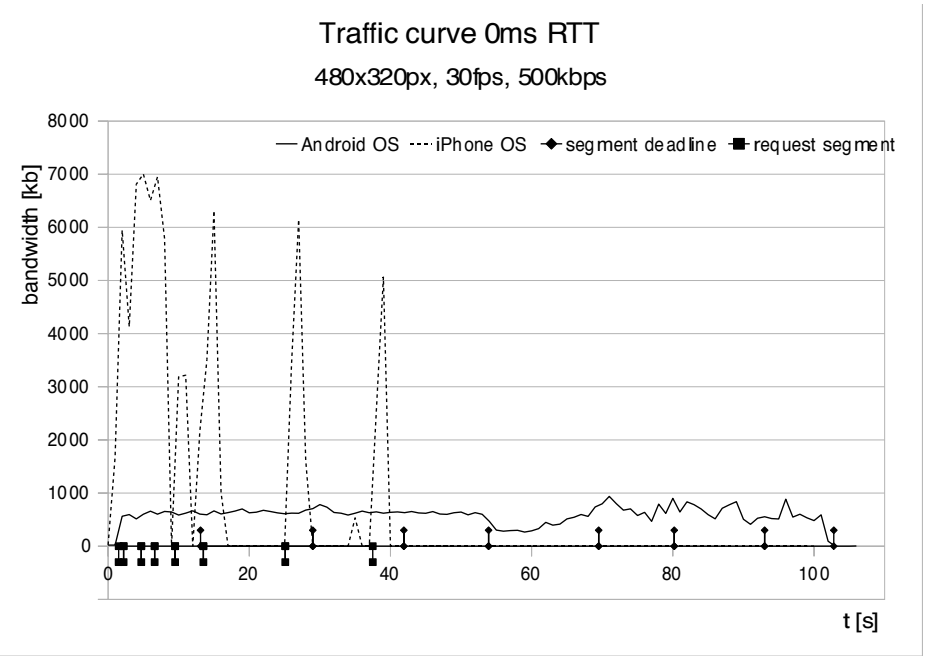


Fig. 6. Traffic under ideal network conditions

continuity of the playback. Figure 6 shows the traffic distribution for both operating systems under ideal network conditions. The solid line shows the traffic occurred during streaming to the Android OS handheld, whereas the dashed line shows the traffic for the iPhone OS handheld. For HTTP Live streaming, the markers facing downwards from the x-axis display the times when a segment is requested. The markers facing upwards from the x-axis on the other hand show the deadline for each segment until its download has to be completed in order to guarantee continuous playback. The moment when the first segment is completely received, i.e. the start of playback, is also the start of the duration until the first segment deadline expires. The times for the deadlines correspond to the length of the segments. These segment deadlines are not applicable for RTP streaming, since the content is not split into segments. Additionally it has to be noted that the measurements of both handhelds were not performed simultaneously, rather the results where merged into one diagram in post-processing. As can be seen from the figure, HTTP Live streaming uses the maximum amount of bandwidth to gather the first few segments, in this case the initial four. After downloading enough segments the request of further segments is paused until the playback advances in time. This behavior can for example be observed at the 20th second. On the other hand, looking at the bandwidth of the Android OS it can be seen that the traffic is continuous.

4.1 Effects of packet delay on playback

As shown by the results in Section 3.1, the network latency has a much higher impact on HTTP Live streaming than it has on RTP streaming. In order to evaluate how network latency influences the playback the same network delays were chosen. Figure 7 shows the results of these measurements. The plot includes

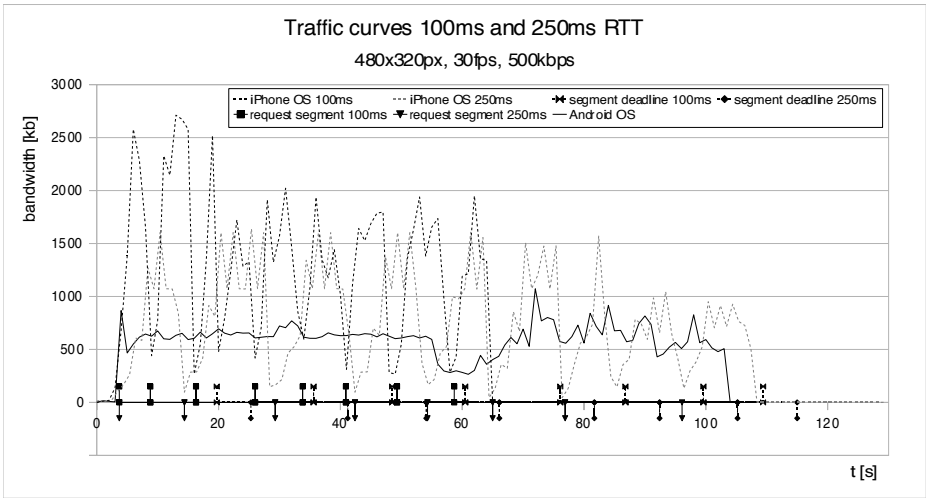


Fig. 7. Traffic considering different packet delays

three traffic lines. Two of them describe HTTP Live streaming at latencies of 100 ms and 250 ms respectively and one shows RTP streaming at a delay of 250 ms. Furthermore, the figure also shows the segment request segment deadline markers, which hold the same semantics as the ones in Figure 6. As can be seen from the markers, all the segment deadlines can be kept in the case of 100 ms latency. For a latency of 250 ms, however, the download of the second segment (third marker) does not finish until the first segment deadline, thus resulting in non-continuous playback. Considering the other segment deadlines, only the 4th and the 5th can be kept. The cause of this problem is the same as in the case of startup delay. The high network latency forces TCP to wait much longer for acknowledgment packets, thus the utilized bandwidth becomes much lower and it takes much longer to fully retrieve a segment. For comparison to the Android OS, the solid line shows the RTP stream at 250 ms network delay. In contrast to the optimal case, duration of the playback is only slightly longer, caused by the initial RTSP negotiation, which builds on TCP. However, the traffic characteristic do not show any significant changes compared to Figure 6. The conclusion that can be drawn from this observation is that high network delay causes HTTP Live streaming to result in non-continuous playback, whereas RTP streaming is nearly unaffected.

4.2 Effects of packet loss on playback

Packet loss is a common problem concerning RTP-based multimedia streaming, since lost packets directly affect the quality of the played content, because the decoder is missing data to restore the uncompressed state. Thus, different concealing techniques exist to deal with the problem of lost packets [8]. On the other hand, streaming content with the help of HTTP does not degrade the quality of the content in case of packet loss, because of the reliability of TCP. However, this does not come for free, since it involves retransmissions. In this section we therefore investigate the impacts of several packet loss rates on the playback. The lines in Figure 8 show the traffic characteristics for the Android OS under

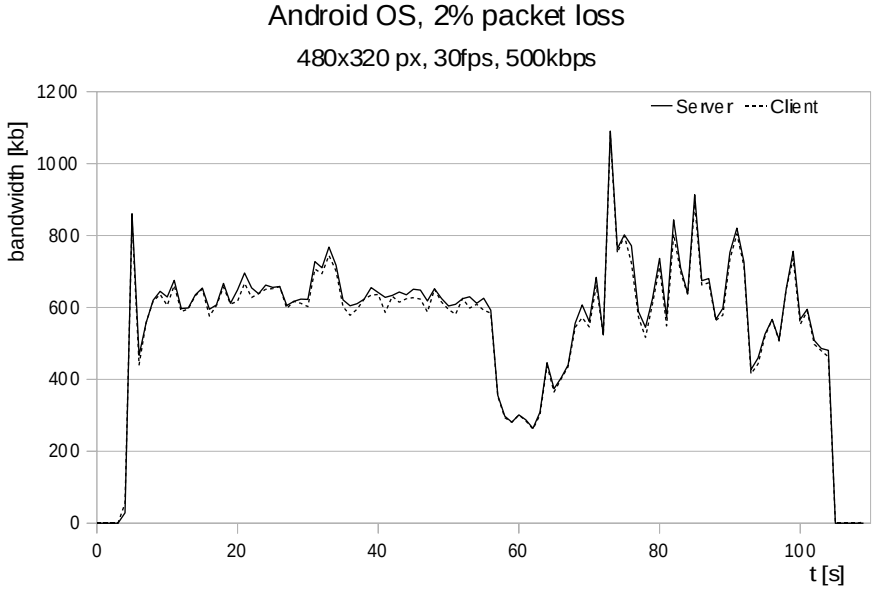


Fig. 8. Playback impacts during 2% packet loss for Android OS

the influence of 2 % of packet loss. Measurements were performed at the server and the client simultaneously. Thus, the solid line shows the traffic at the server, whereas the dashed line describes the client side, which is measured at the outgoing interface of machine `mm04` as described in Section 2.2. For the Android OS packet loss results in the creation of artifacts during the playback. In addition to packet loss rates of 1 % and 2 %, we also tested 5 % packet loss, in which case the Android OS was not able to finish playback.

The measurements for the iPhone OS can be seen in Figure 9. Only one line is shown, since the server and the client side did not show any noticeable difference. When comparing the traffic characteristics to those in the optimal case in Figure 6, one can see that in case of packet loss the utilized bandwidth

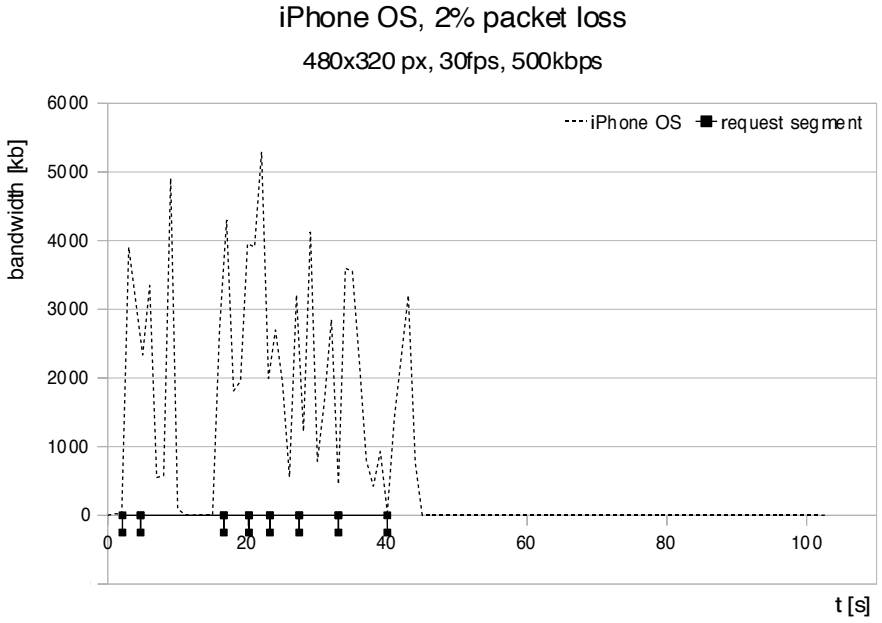


Fig. 9. Playback impacts during 2 % packet loss for iPhone OS

is much lower and consequently the segment request times are more distributed. The explanation for this is the behavior of TCP in the case of packet loss. As TCP detects a lost packet, it halves the maximum allowed bandwidth according to the Additive Increase Multiple Decrease (AIMD) algorithm [9], by reducing the size of the contention window. After receiving further packets without loss, the contention window is increased again, but only in a linear way. Nonetheless, the tested packet loss rates did not influence the quality of the playback in terms of continuity. Furthermore, the content is displayed at full quality without the creation of artifacts, due to the reliability of TCP.

5 Bandwidth overhead

When transferring content across a network additional information is needed. The amount of additional bytes spent depends on the protocols involved during the transmission. Therefore, the purpose of this section is to compare the amount of overhead caused by RTP streaming and HTTP Live streaming, respectively. As test sequence for the measurements the highest available quality was chosen, i.e. 480x320 pixels at 30 fps and 500 kbps. Actually, before considering the overhead introduced through streaming, the additional amount of data caused by container format multiplexing needs to be considered. Since RTP enables to stream AVC and AAC in their raw bit stream format, there is no additional

overhead for the Android OS. On the other hand, the raw bit streams need to be multiplexed into the MPEG-2 transport stream format to enable HTTP Live Streaming for the iPhone OS. The selected test sequence requires 596 kbps including audio, increasing to 706 kbps after multiplexing and segmentation, i.e. the final bit stream includes an overhead of about 18.5 %. The reason lies in the stuffing of transport stream packets [10]. Overhead introduced during streaming, due to network protocols, was measured under optimal network conditions. In the case of HTTP Live streaming about 9000 TCP packets were observed. The additional overhead introduced by the TCP header is 32 byte, resulting in a total amount of about 23 kbps including the additional data required for requesting the playlist file and the single HTTP requests for each segment.

In the case of RTP streaming 10300 packets were encountered. With a packet header of 8 byte for UDP and 16 byte for RTP, this sums up to about 19.3 kbps overhead. Note that the 16 byte for RTP is an average number, since the RTP payload format headers for AAC and AVC use a different number of bytes. Streaming with RTP involves additional traffic caused by RTCP. Actually, about 230 RTCP packets were measured, resulting in about 13 kB of data, considering the 8 byte of UDP header and 48 bytes of data of an RTCP packet. Furthermore, also the traffic caused by RTSP needs to be considered, which was about 4 kB. That is, both RTCP and RTSP cause only insignificant overhead. At this point it has to be mentioned that the streaming server only used fragmentation units A and B (FU-A and FU-B)for packetizing AVC packets [3]. An single time packet aggregation packet (STAP) implementation would have resulted in a minor decrease of the overhead.

To conclude, without considering multiplexing, the overhead of both approaches is similar. Thus, the main amount of additional overhead on the iPhone OS is the result of the necessary multiplexing.

6 Traffic shape and fairness

In this section we evaluate the behavior of both devices in case of scarcity and rivaling traffic by limiting the available bandwidth to 1 Mbps and serving both end devices simultaneously. The traffic characteristics under these circumstances are depicted in Figure 10. The solid line describes RTP streaming, whereas the dashed line shows the characteristics of HTTP Live streaming. Additionally the total amount of traffic is depicted by the bold solid line. Furthermore, also the request segment and segment deadline markers for HTTP Live streaming are displayed. The streaming session for RTP is started first. After about 23 seconds HTTP Live streaming is also started. Comparing both lines to the optimal case in Figure 6, it can be seen that the traffic characteristics for HTTP Live streaming are completely different, while RTP almost equals the optimal case. After the end of the RTP session, HTTP Live streaming utilizes the full bandwidth to gather the next segments which already fell behind the deadline. From the observation of these traffic lines it can be seen, that the RTP traffic treats the HTTP Live streaming in an unfair manner. Although the playback by the Android OS

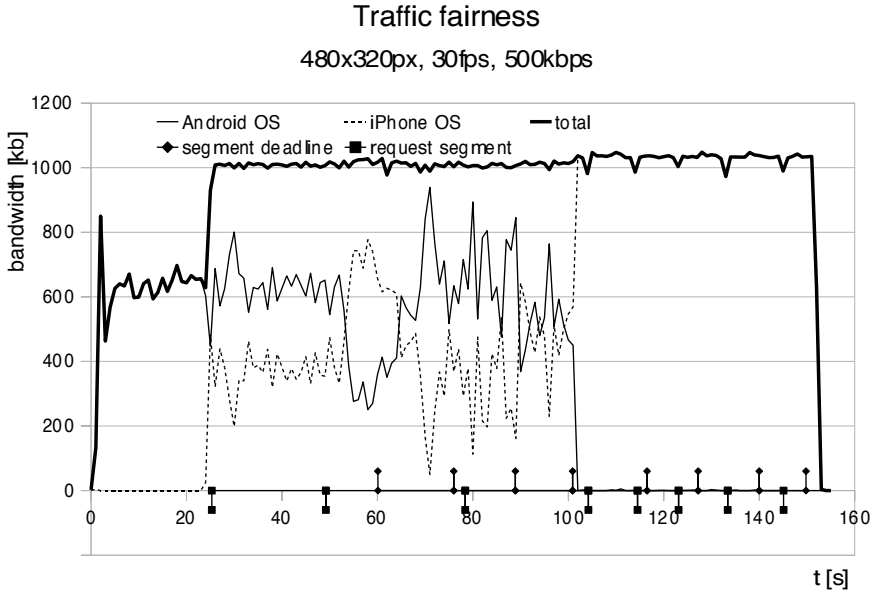


Fig. 10. Traffic fairness under bandwidth limitation of 1Mb/s

has shown artifacts, caused by delayed packets due to network congestion, the traffic shape did not change in a noticeable extent. In contrast, the available bit rate for the iPhone OS adjusts to the bit rate required by the Android OS. This is due to the congestion control mechanisms implemented in TCP but not in UDP. A remedy to this problem could be the usage of the Datagram Congestion Control Protocol (DCCP) [11] instead of UDP, which provides congestion control mechanisms for unreliable traffic. In fact, measurements in wireless network environments performed by de Sales et al. [12] have shown that DCCP and TCP behave fair to each other, as long as no UDP flow is involved.

7 Conclusions and future work

Our evaluation shows the differences of the two streaming approaches used in iPhone OS and Android OS. We evaluated the startup delay in case of increasing packet delay and packet loss. This showed that the startup delay increased linearly for Android OS and exponentially for iPhone OS in case of increasing packet delay. By injecting packet loss we noticed a minor increase in startup delay for the iPhone OS, while the Android OS was not affected at all. Next, we evaluated the playback characteristics under the same network impairments. This showed that for high packet delay of 250ms the playback on the iPhone OS is non-continuous, while the Android OS was nearly unaffected and showed no

playback disruptions. Packet loss, on the other hand, caused disruptions in the video on the Android OS, while having no impact on the video quality on the iPhone OS. We then analyzed bandwidth overhead, where the MPEG-2 transport stream format required by HTTP Live streaming mechanism of the iPhone OS caused a substantial overhead. Finally, we evaluated traffic shape and fairness, which showed the typical greedy behavior of RTP vs. HTTP.

Both approaches have their strengths and weaknesses. However, the HTTP Live streaming mechanism comes with the advantage that there is no dedicated streaming server needed. Additionally, the problems related to NAT traversal [13] are avoided. Finally, the HTTP-based approach comes at the advantage that existing Content Delivery Networks (CDNs) can be used to distribute the content in a very scalable way. It therefore promises to be a more lightweight and scalable solution, which will certainly help its adoption by industry.

In our future work we plan to evaluate the special characteristics of the wireless network more closely. Additionally, we would like to extend our evaluations based on the just mentioned advantages of the HTTP Live streaming approach, i.e. NAT traversal and scalability.

Acknowledgments. This work is supported by the Österreichische Forschungsförderungsgesellschaft mbH (FFG) in the context of the Celtic SCALNET (CP5-022) project.

References

1. T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7), July 2003.
2. H. Purnhagen. An Overview of MPEG-4 Audio Version 2. In *Proc. 17th International Conference: High-Quality Audio Coding*, August 1999.
3. S. Wenger, M.M. Hannuksela, T. Stockhammer, M. Westerlund, and D. Singer. RTP Payload Format for H.264 Video. RFC 3984, February 2005.
4. J. van der Meer, D. Mackie, V. Swaminathan, D. Singer, and P. Gentric. RTP Payload Format for Transport of MPEG-4 Elementary Streams. Technical report, Internet Engineering Task Force, November 2003. Proposed Standard, RFC 3640.
5. R. Pantos. HTTP Live Streaming. Internet-Draft (work in progress), draft-pantos-http-live-streaming-03, 2010. Expires October 4, 2010.
6. Anurag Acharya and Joel Saltz. A study of internet round-trip delay. Technical report, University of Maryland, December 1996. CS-TR-3736.
7. Y. Angela Wang, Cheng Huang, Jin Li, and Keith W. Ross. Queen: Estimating packet loss rate between arbitrary internet hosts. In *PAM '09: Proceedings of the 10th International Conference on Passive and Active Network Measurement*, pages 57–66. Springer-Verlag, 2009.
8. Yao Wang and Qin-Fan Zhu. Error Control and Concealment for Video Communication: A Review. In *Proceedings of the IEEE*, volume 86, pages 974 – 997, May 1998.
9. M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

10. ISO/IEC 13818-1:2000 Information Technology-Generic Coding of Moving Pictures and Associated Audio, Part 1: Systems. Recommendation ITU H.222.0, December 2000.
11. E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol (DCCP). RFC 4340, March 2006.
12. Leandro Melo de Sales, Hyggo O. Almeida, and Angelo Perkusich. On the performance of TCP, UDP and DCCP over 802.11 g networks. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2074–2078. ACM, 2008.
13. V. Paulsamy and S. Chatterjee. Network Convergence and the NAT/Firewall Problems. In *Proc. 36th Hawaii International Conference on System Sciences*, January 2003.