Towards QoS Improvements of TCP-based Media Delivery

Martin Prangl, Ingo Kofler, Hermann Hellwagner
Dept. of Information Technology
Klagenfurt University, Austria
Email: {firstname.lastname}@itec.uni-klu.ac.at

***Abstract*—The amount of audiovisual data available on the Internet and thus of multimedia communication over today's networks is increasing at a rapid pace. Despite the availability of specific media transport protocols like RTP, most content providers make use of the well-established and reliable TCP protocol to deliver audiovisual content over the Internet. The reason is that TCP-based data delivery in general is much less complicated for the clients to be served and over today's networks traversed (including proxies and firewalls), than making use of UDP-based RTP connections. However, in case of network bandwidth fluctuations and packet losses, TCP-based media delivery may lead to annoying jerky playback at the client side, due to retransmissions and late arrival of media data. This papers deals with TCP-based perceptual QoS improvement mechanisms for increasing the media experience for the consumer under unstable network conditions. Our approach is based on media content adaptation (transcoding) to fit the actual network bandwidth continuously monitored by the sender. The proposed mechanisms are applied at the application level at the server side, leaving the existing TCP implementation untouched and therefore enabling transparent use of existing media players. An evaluation of a realistic use case is presented which underlines the efficacy of our approach.**

I. Introduction

The transport of multimedia contents over the Internet is getting more and more popular. Video on Demand as well as live television services are distributed over the Web, enabling online consumption of a huge and rapidly growing set of media contents. However, providing an acceptable Quality of Service (QoS) level to the consumer forms a challenging problem for several reasons. In contrast to common data transfer, e.g., the exchange of documents or data files, the real time consumption of audiovisual content is sensitive to bandwidth fluctuations, delay and jitter. For solving this problem, several QoS aware network architectures like Integrated Services [1], Differentiated Services [2] or Multi-Protocol Label Switching (MPLS) [3] are proposed for reserving resources for such critical flows and prioritize them along the network chain, respectively. Nowadays such intelligent network mechanisms are rarely activated in network nodes. Guarantees on link bandwidth, delay and jitter cannot be granted for an individual end-to-end connection through the Internet, that is still a best effort network. Even Voice over IP (VoIP) services, which rely on similar QoS requirements, are becoming rapidly popular. VoIP traffic consumes much less bandwidth and is therefore less critical than traffic caused by audiovisual applications. Furthermore, the bit rates of VoIP flows can be assumed as constant, whereas video flows are characterized by variable bit rate and burstiness.

Special transport protocols were developed and standardized to support the requirements of real-time audiovisual applications. The so called Real-Time Transport Protocol (RTP) was designed on top of the UDP transport protocol that does not guarantee a reliable connection. In contrast to TCP, the design goal was to avoid end-to-end retransmissions that lead to a higher average delay and jitter. Instead, the protocol provides means for timing of the media data and a sequence number mechanism to detect lost packets. In combination with content adaptation techniques [4], RTP is well suited for supporting the vision of Universal Multimedia Access [5].

Nowadays a clear trend can be observed toward the use of wireless technologies. Modern devices like PDAs or mobile phones are connected to the Web by WLAN or UMTS carriers, enabling convenient mobile and location independent consumption of Web contents. However, the use of wireless connections is critical in case of real time applications. Delay and bandwidth are varying depending on the signal quality which is sensitive to the location of the client as well as to natural influences.

As a consequence of the extended connectivity, also the users' demand for security increased during the last years. Today, security mechanisms like firewalls are integrated along the network chain and at the client's network access (end) points. Modern operating systems are equipped with firewalling mechanisms as well. Enterprises usually separate their LANs from the Web by the use of proxy servers. This fact represents a problem for media content providers because incoming RTP traffic is blocked by such security mechanisms in most cases. Home users or small companies also often use a Network Address Translation (NAT) gateway for connecting multiple clients to a single shared Internet connection. Such popular NAT devices represent a barrier for incoming RTP traffic as well. In order to distribute media contents to a large group of users, independence of security mechanisms and network components within the whole network chain is required. For this reason, most public content providers make use of the simple and well-established TCP protocol for transporting media contents to their clients. Of course, TCP delivery is easy and uncomplicated but leads to drawbacks in case of real time media consumption. The reason is that TCP is a reliable protocol, thus lost media frames are retransmitted, and delayed

(potentially useless) frames are delivered. Furthermore, TCP specific features like congestion control and flow control, are harmful in this use case. As a consequence of insufficient bandwidth, TCP-based on-the-fly consumption is annoying for the user because the media session may suffer from missing data periodically.

This paper focuses on how to improve the perceived QoS of TCP-based media content delivery. The aim of our work is to improve the possibly unsatisfactory behavior of the TCP-based media consumption in case of bandwidth fluctuations. In order to be independent from media player architectures and to avoid annoying player software updates at the client side, our approach is applied at the server side. In order to achieve transparency w.r.t. the player's point of view, our improvements are applied at the application level, keeping the existing TCP implementation untouched. With the help of periodical link throughput measurements, the original media content is shaped in such a way that it fits the actual network conditions.

The remainder of this work is structured as follows. Section II introduces the problems of efficient RTP-based delivery in today's network environment. Section III gives a brief introduction into TCP-based media delivery. Section IV is dedicated to our proposed adaptive server architecture for TCP-based media delivery. Subsequently two algorithms for QoS improvements are introduced. In Section V an evaluation of the proposed approach based on media delivery over a shared bottleneck link is given. Section VI finally concludes the main findings of this work.

II. Problems of RTP-based Media Delivery

Typically, multimedia content is delivered using the Real-time Transport Protocol (RTP) [6], a connectionless, unreliable communication protocol based on UDP. The main reason is that a reliable communication protocol (such as TCP) produces too much overhead because it makes no sense to retransmit lost packets. Such retransmitted, and therefore potentially late packets (containing stream information of the past) may be useless at the rendering client side and may have to be discarded. Usually, the content delivery over RTP is controlled by the server, which pushes the media content to a certain UDP port given by the client. In order to give the client the option to choose a certain content, to inform the server of the listening port(s) and to enable interactive control to the user (e.g., start, stop, pause) of the current session, another communication protocol is needed. This protocol is called Real-Time Streaming Protocol (RTSP, RFC2326) [6] and is based on TCP. The RTSP requests are always initiated by the client. Another optional protocol named Real-Time Control Protocol (RTCP) can be used to provide statistical feedback, e.g., lost packets, for QoS reactions (management) on the server side.

However, delivering multimedia data over the Internet using RTP/RTSP has drawbacks as well. First, the elementary streams have to be packetized according to specific RFCs, depending on the used media formats (e.g., one packet for one encoded frame). This is required because in case of packet loss, the lost information should be confined to have as little impact on the media presentation as possible. Furthermore, this additional task implies that the delivery module has to be extended if a new encoder is included in the system. The second, most critical drawback is the traversal of restrictive network nodes like firewalls, NAT routers or proxies in the network chain.

A. Firewalls

Assume that a client is located behind a firewall and requests content from the media server by sending a TCP-based RTSP Message. This (outgoing) message passes the firewall and consequently the media server receives the content request including the port and IP address of the client for establishing the RTP/UDP connection for content delivery. After the server receives the *RTSP PLAY* message, the server starts the RTP packetization and transmits the data via the connectionless UDP protocol to the user's terminal. The problem in this case is that the firewall blocks incoming connections from the Web (from the media server) because it does not know anything about the session (the request is coming from the user). It would be possible to configure the firewall to pass the UDP packets of the media server but the challenge is that the media servers (IP addresses) as well as the available ports on the clients are dynamically changing. Clients are usually using a set of media servers, not only one particular. Furthermore the user is in general not familiar and/or not allowed to configure the firewall according to his/her individual preferences. A solution would be to implement additional logic within the firewall, that keeps track of the exchanged RTSP messages and allows incoming UDP packets on the ports and IP addresses negotiated within the RTSP session. A novel draft of such a session awareness of firewalls for avoiding the problem of blocking incoming RTP packets is discussed at the end of this section. However, to the best of our knowledge such a session tracking logic is not implemented in current firewall products.

B. Network Address Translation (NAT)

Clients behind a NAT router are using a local IP address which is not valid in the Internet. The NAT router is translating the local requests to the Web by replacing the client's local IP address to the router's global (valid) IP address and manages/tracks the ports for forwarding the answer from the Web back to the requesting local client. From the media server's point of view, the NAT router is requesting a content for consumption. The forwarded *RTSP OK* server answer reaches the client as in the previous use case because the NAT router knows to which local client the answer belongs. The problem occurs if the media server starts sending the RTP/UDP packets to the NAT router. The NAT router does not know what to do with these packets because it does not consider the semantics of the previous RTSP/TCP communication. As a consequence, the RTP packets are rejected at the NAT router, the client does not receive the media stream.

C. Proxy Servers

Assume that a client is connected to the Internet by a proxy server. This approach for a connection to the Internet is widely used in companies and large work groups. The proxy server receives the RTSP request and forwards it (by replacing the client IP address with the proxy IP address) to the media server. Note that this proxy server has to support an RTSP-specific TCP port. If the proxy only supports the HTTP protocol, it does not understand the RTSP message and as a consequence rejects the client request. The RTSP forwarding step is similar to the NAT router's task in the previous use case. The proxy is not able to forward the media stream reply from the server because it is not accepting and distributing UDP packets. This feature would require a special multimedia proxy [7], which is not widespread. Only a connection to the server without intermediate network nodes, that perform either NAT or act as proxy, enables the user to receive the media stream from an RTP-based media server. Nowadays, this "optimal" unsecure use case rarely exists in real life because the security awareness of the community is growing rapidly. Furthermore, modern network devices like routers as well as operating systems are equipped with firewalling mechanisms by default.

Recapitulating the previous use cases, the problem receiving the RTP stream results from two UDP protocol specific features. First, the RTP connection is initiated by the server. Such uncontrolled connection from "outside" alerts firewall mechanisms. Second, the underlying UDP protocol is connectionless, which means that NAT routers or proxies do not know to which connection the RTP packets belong.

These drawbacks of the RTP media delivery are well known in the multimedia research community. Currently, there is a draft of the so called NSIS Signaling Layer Protocol (NSLP) [8] available, which is designed to request the dynamic configuration of firewalls and NAT routers along the data path. This proposed functionality includes the dynamic configuration of firewall rules as well. Another approach is followed in the scope of the Universal Plug-and-play (UPnP) framework [9]. There, clients like a media player can make use of services offered by routers or NAT devices and can configure them to pass through the RTP stream to the player. This technology is already in use in network devices like DSL modems. However, the telecommunication industry is demanded to enforce this or similar mechanisms, otherwise the existing implementation of RTP becomes more and more unattractive.

III. TCP-based Media Delivery

Content providers like YouTube[1] usually want to broadcast their content to a huge set of clients. In order achieve this goal, TCP-based media delivery is currently seen as the best choice for the reasons already mentioned above. Furthermore, the TCP-based communication between the requesting client and the server is easy. Many multimedia players (e.g., VLC[2]) as well as Web-based plug-ins (e.g., Adobe Flash[3]) support the Hypertext Transfer Protocol (HTTP) for receiving the media stream, which is also known as *progressive download*. In principle, the media content is published by simply moving it into the documents directory of an HTTP Web server. The player at the client side downloads the requested content piecewise into its buffer and decodes and renders the media data on-the-fly.

If the client does not have enough bandwidth available or significant bandwidth fluctuations occur during the media consumption, the client's buffer is drained periodically. In such a case, the player is unable to decode the media content in time which leads to a jerky playback. A solution for keeping the session running smoothly, which consequently increases the perceptual QoS for the consuming user, is given as follows.

IV. Increasing QoS of TCP-based Media Sessions

Content adaptation is seen as a key concept to deliver media content independently of the terminal capabilities, network conditions, and user preferences. For example, content adaptation enables a client, equipped with an H.264-AVC/MP3 compatible media player, to consume an audiovisual content, originally encoded in MPEG-2/MP2 at a high bit rate. Also if the client's network link suffers from insufficient bandwidth to consume the original content (at high bit rate), the content can be adapted so that its resulting degraded version meets the given resource limitations. Audiovisual content can be adapted in many dimensions. The video part can be adapted in the spatial domain, which means to change the spatial resolution of each frame; in the temporal domain, by reducing the number of frames per second; and in the signal-to-noise ratio (SNR) domain, by modifying the encoder specific quantization value for the frame compression. A degradation of the audio part can be achieved by reducing the sample rate, the amount of bits per sample, and/or the number of audio channels.

Typically, the decision in which way the content should be adapted in order to maximize the utility for the user is complex and depends on the given resources, the individual user preferences, and the terminal capabilities. This adaptation decision taking process is usually performed initially right after the content request. The resulting decision is expressed by unique target media stream parameters like frame rate, spatial resolution, etc., which are used to configure the content adaptation engine. In our implementation, the adaptation decision is encoded into an HTTP URL which is dynamically created according to the terminal capabilities and user preferences. This URL includes the content identification as well and is used by the media player or the Web-based player plug-in for requesting the adapted media content. This work addresses the dynamic case of adaptation, where the link conditions are assumed to be changing dynamically. The initial adaptation decision for the specific request is assumed to be given (by the URL). For more information about the adaptation decision taking process, the reader is referred to [10].

[1] http://www.youtube.com
[2] http://www.videolan.org
[3] http://www.adobe.com

The main part of our system consists of an adaptation engine and a delivery module, each running in a separate process. The delivery module implements a very slim HTTP server, which basically relies on a simple TCP socket communication. Only the *HTTP GET* command (RFC1945) handling is needed for receiving, parsing, and serving the player's media request. The simple GET request enables the player to request a specific media file by submitting a corresponding URL to the server, e.g., *http://mediaserver.com:8080/starwars.avi*. This simple request invokes (like in case of a usual HTTP server) the delivery module to open the file *starwars.avi* and to send the byte stream of the file subsequently to the client. This use case does not invoke any adaptation on the content and is used to evaluate our improvements in Section V later. In order to enable our system to perform on-the-fly content adaptation, the URL can contain adaptation parameters as discussed before. For example, a transcoding request for downscaling the video to 320x200 pixels, reducing the frame rate to 10 fps and encoding it as an H.264 video can be signaled in the URL as follows: *http://mediaserver.com:8080/starwars.avi?vc=h264&s=320x200&fr=10*. After receiving the URL from the client, the delivery module decodes the URL and parses the parameters for further use.

The adaptation engine's task is to adapt or transcode the requested content. In our framework, the well known and very powerful multimedia library *ffmpeg*[4] is used, which enables the system to support the most common video and audio codecs. As already mentioned, each client request is served by the delivery module in a separate process. If there is no need for adaptation, the delivery process transmits the media data to the client and terminates if the end of the media stream is reached or the client is terminating the session. Otherwise, if there is a need for content adaptation, the corresponding delivery process spawns an adaptation process, which is responsible for the adaptation task. The information in which way the content has to be adapted is known by the delivery process that takes the values from the parsed URL parameters. Therefore, it is able to spawn and initialize the adapting child accurately.

During the content delivery, there is a need for permanent information exchange between the parent (delivery) and the child processes (adaptation engine), as shown in Figure 1. One communication channel is needed to forward the adapted content from the child to the parent. There is a need for synchronization between these two processes because the adaptation process must not produce the adapted content faster than the delivery process can send it to the consuming client. This issue is solved by using a special communication channel, a *pipe* which is known as one of the core concepts of interprocess communication. Note, that the media flow to the client is slowed down according to the available bandwidth since the client-server communication is based on TCP. If the player pauses the session (by simply stopping reading from the socket), the delivery module pauses the delivery as well because it is writing to a blocking TCP socket. Since in this case the delivery process is blocked and does not read from the pipe anymore, the pipe will get full which causes the adaptation process to block as well. If the available link bandwidth becomes lower than the encoding media bit rate, the media flow is not continuously consumable by the client.

However, there is one fact which can be used to enable "smooth" media rendering at the client side. Because the delivery module writes the media data into a blocking TCP socket, it can estimate the real actual delivery bit rate with ease (by simply counting bytes within a time slot of one second). Based on this information the adaptation engine is able to react to the dynamic bit rate fluctuations. As shown in Figure 1 the delivery process writes the delivery bit rate statistics periodically into a shared memory segment, which is accessible by the adaptation child process as well. An efficient mechanism for avoiding media rendering problems caused by bandwidth fluctuations is given as follows.

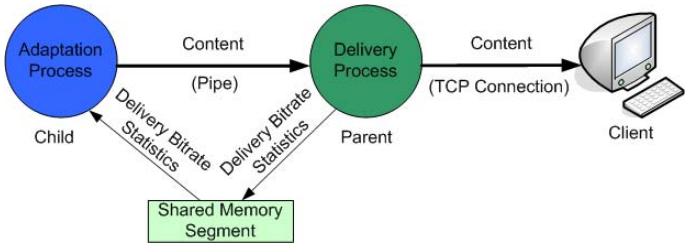

Fig. 1. Interprocess communication between delivery module and adaptation engine.

A. Handling Bandwidth Fluctuations

Figure 2 shows a bandwidth fluctuation diagram containing the available bandwidth (bw), the actual delivery bit rate (dbr) enforced by the player, and the actual media stream bit rate (br). If br, which is enforced by the encoder, is lower than bw, there is no problem. Let this case be the initial point (t=0). The delivery module captures the delivery bit rate (dbr) every second, which is indicated by points in the graph. After t=1 bw begins to decrease, and at t=2.5 the critical level, where $br = bw$, is reached. At the next capture point of dbr, the delivery module and consequently the adaptation process notice that $bw < dbr$. At this time, the adaptation engine adjusts $br = dbr$. Note that, because of the discrete measurement points, this algorithm causes an excess of the media stream bit rate ($br > dbr$) between the measurement points in case of a falling slope of dbr (red line). This exceeding bit rate consumption can usually be compensated by the input buffer of the media player. A continuous falling slope of dbr would empty the client buffer at some point in time, depending on the buffer's size. At t=5 where bw reaches a stable level, br is adjusted to dbr as well. During this stable period the player tries to refill its buffer by trying to read faster from the socket (by increasing dbr). This is not possible because it gets blocked as a consequence of bw not being greater than dbr. At t=6, bw starts increasing monotonically. Now the player is able to increase dbr for buffer refilling.

[4]http://ffmpeg.mplayerhq.hu/

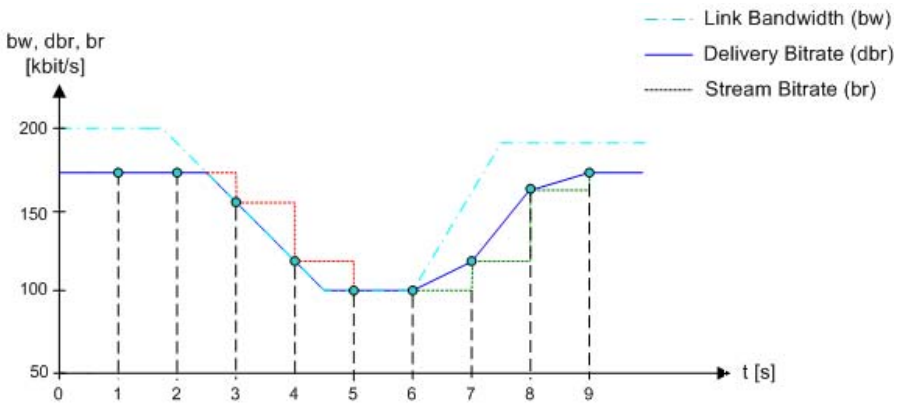

Fig. 2. Handling Bandwidth Fluctuations

As a consequence, at t=7 the delivery module subsequently increases br according to the measurement of dbr (green lines). This action forces the player to amplify the increase of dbr. At t=9, br reached its initial value, the buffer is refilled. As a consequence $dbr = br$ at $t > 9$.
Remark 1: The mechanism works as long as the client buffer does not underrun.
Remark 2: The adaptation engine was implemented by adjusting br at an abstract (encoder independent) level. All encoders under consideration maintain the bit rate adjustment in the SNR domain. As a consequence, this mechanism works only in a certain dynamic range of bandwidth fluctuations. If bw gets too low, the media playback will become jerky again.

B. Determining Available Bit Rate

In general, a higher bit rate leads to a higher perceived quality for the client. Therefore, a QoS aware multimedia framework should efficiently use all available resources. Apart from handling bandwidth fluctuations as shown before, it is possible that the available bandwidth is rising or the user does not know his/her correct bandwidth limit. In this case another mechanism is implemented as shown in Figure 3.

The available bandwidth is initially higher than the stream bit rate (t=0). As a consequence, the delivery bit rate is equal to the stream bit rate. The measurement of the delivery bit rate is done in intervals, marked by points as mentioned in the previous section. The automatic bandwidth detection algorithm works as follows. If the delivery bit rate is equal to the stream bit rate at two sequential measurements (e.g., $dbr_{t=1} = dbr_{t=2}$), the stream bit rate is increased by a constant $\Delta\ br = 5\%$ of $br_{t=0}$. ($br_{t=2} = br_{t=1} + \Delta\ br$). In other words, the delivery module tries to push more data to the client. As a consequence, the player at the client side has to consume this amount of data in the same time as before (real time consumption). Consequently, the client pulls the data faster which results in an increased delivery bit rate as well ($dbr_{t=3} = br_{t=3}$). This successive increase of br is repeated until the delivery bit rate is smaller than the stream bit rate (determined at t=11). At this point br is reduced: $br_{t=11} = br_{t=10} - \Delta\ br$. If br is smaller than dbr at the next measurement, br gets increased again (t=12). The temporarily higher delivery bit rate is compensated by the buffer at the client side. Note, that the average delivery bit rate is not changing at t $\geq$ 10. If dbr is reduced as a consequence of a decrease of the available link bandwidth (bw), the dynamic bandwidth adjustment algorithm as discussed before is applied. Please note that the delivery module tries to push more data ($br > dbr$) to the client only if a constant average delivery bit rate (dbr) is detected. This can be compared to the additive increase behavior of the TCP congestion control mechanism. For this reason, both presented algorithms can used in parallel without influencing each other.

V. Evaluation

In order to demonstrate the perceived QoS improvements of our approach, we performed some real-world experiments. For that purpose we used an MPEG-2 video stream that was captured by a DVB-S card with a bit rate of approximately 3.8 Mbps and a duration of 80 seconds. The video stream was made available on our adaptive server implementation. As a client application, we used two instances of the popular multimedia player VLC which were running on the same client machine. The player was configured to use a buffer of approximately 5 seconds which seems to be a suitable value for the playback of content streamed via HTTP. On the link between the server and the client machine a traffic shaper was deployed that limited the throughput to 6 Mbps. The traffic shaper should emulate the behavior of a DSL link that is commonly used to connect a home network to the Internet. In the case of two simultaneous streams with a bit rate of 3.8 Mbps each, the bandwidth required by the streams will exceed the maximum throughput and will cause a choppy playback at the client. To quantify the smoothness of the playback we used the statistics provided by the VLC player. We decided to use the number of events in which pictures arrived too late and were skipped during the playback as a measure of smoothness. In order to show the advantage of our algorithm, we first measured the smoothness of the video in the case where no adaption was performed. For that purpose we requested the video stream with player 1 (t=0). Since the bit

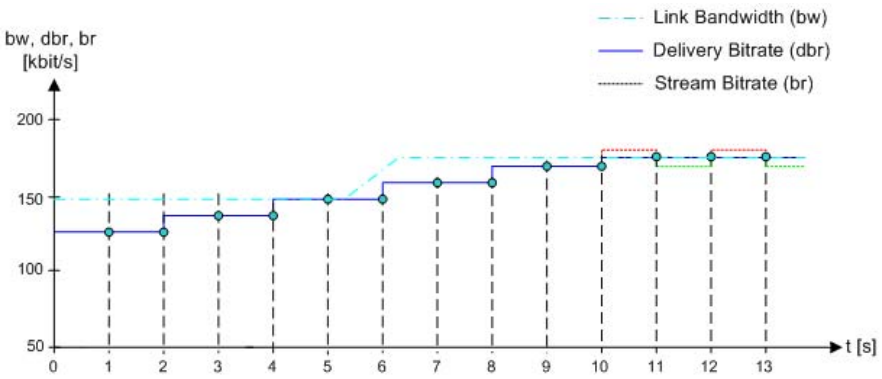

Fig. 3. Available Bandwidth Detection

TABLE I
NUMBER OF FRAME-SKIP EVENTS

	Normal		Adaptive	
Run	Player 1	Player 2	Player 1	Player 2
1	23	30	2	0
2	31	23	1	3
3	20	28	1	10
4	13	31	1	6
5	16	28	4	0
Average	21	28	2	4

rate of the stream was significantly lower than the capacity of the link, the playback was smooth. After 40 seconds (t=40) we requested the same stream with player 2 and tried to consume both video streams simultaneously for additional 40 seconds. As a result of the insufficient capacity, both players were not able to download the video stream as fast as they consumed them. This forced both player instances to skip frames that arrived too late which resulted in a choppy playback. The number of such frame-skip events of both players 1 and 2 in the case of a normal, non-adaptive delivery can be found in Table I. In order to get significant numbers, the experiments were performed five times and finally the mean value was calculated. During the 80 seconds of the experiments, on average each of the players had to skip frames 25 times. Then we performed the same experiment again, but with the dynamic bandwidth adjustment algorithm enabled. The results show that the adaptation of the video led to an improved smoothness at both players. On average only 3 late frames had to be skipped at both players within the 80 seconds of the experiment's duration.

VI. CONCLUSION

In this paper, we presented an approach to increase the perceived QoS for TCP-based video streaming over best effort networks like the global Internet. This improvement is accomplished by the continuous adaptation (transcoding) of the video stream and taking into account TCP throughput measurements at the server side. Based on our experimental setup, we demonstrated that our proposed algorithm leads to a smoother playback at the client in cases where the video stream has to share the bandwidth of a bottleneck link with other TCP flows. Another important advantage of our approach is that it works independently of the player used by the client. No explicit signalling information or feedback to the server is necessary. A drawback of the video encoders considered so far is that a continuous video adaptation can only performed in the SNR domain since other parameters like the frame rate cannot be modified on-the-fly. This limits the bit rate ranges in which the adaptation can be performed. Additionally, this kind of adaptation is very CPU intensive and therefore a tradeoff between perceived QoS for the consumer and operational costs has to be made. In order to tackle this problem, our future work will utilize the emerging scalable extension of the H.264/AVC codec that is designed to support computationally cheap adaptation by simple truncations or extractions of parts of the video bitstream.

REFERENCES

[1] D. Estrin, S. Berson, S. Herzog, D. Zappala, *The Design of the RSVP Protocol*, ISI Final Technical Report, University of Southern California, Information Sciences Institute, July 1996.

[2] V. Fineberg, "A Practical Architecture for Implementing End-to-End QoS in an IP Network", *IEEE Communications Magazine*, vol. 40, iss. 1, pp. 122-130, January 2002.

[3] X. Xiao, L.M. Ni, "Internet QoS: A Big Picture", *IEEE Network*, vol. 13, no. 2, pp. 8-18, March/April 1999.

[4] M. Prangl, H. Hellwagner, T. Szkaliczki, "A Semantic-based Multimodal Utility Approach for Multimedia Adaptation", *Proceedings of the 7th International Workshop on Image Analysis for Multimedia Services (WIAMIS)*, pp. 67–70, April 2006.

[5] M. Prangl, T. Szkaliczki, H. Hellwagner, "A Framework for Utility-based Multimedia Adaptation", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 6, pp. 719–728, June 2007.

[6] J. F. Kurose, K. W. Ross, *Computer Networking - A Top-down Approach Featuring the Internet*, 3rd ed., Addison-Wesley, Amsterdam, 2004.

[7] L. Böszörmenyi, H. Hellwagner, P. Schojer, "Metadata-driven Optimal Transcoding in a Multimedia Proxy", *Multimedia Systems*, ACM/Springer, vol. 13, no. 1, pp. 51–68, Sept. 2007.

[8] NSIS Working Group Internet Draft, *NAT/Firewall NSIS Signaling Layer Protocol (NSLP)*, version 14, July 2007.

[9] UPnP Forum, *Internet Gateway Device (IGD) Standardized Device Control Protocol*, version 1.0, November 2001.

[10] M. Prangl, R. Bachlechner, H. Hellwagner, "A hybrid recommender strategy for personalised utility-based cross-modal multimedia adaptation", *IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1707–1710, Beijing, July 2007.