TCP-Based Dynamic Media Stream Adaptation in Ubiquitous Environment

Gergely Hományi, Gábor Paller
Nokia Siemens Networks
gergely.homanyi@nsn.com, gabor.paller@nsn.com

Abstract

More and more media material is consumed in ubiquitous environment where the service can be experienced over different devices, network conditions, network bearers, etc. Unlike in traditional wireless environments where at least changes of the QoS parameters are controlled by the network, ubiquitous networks are characterized by abrupt changes. Transition from complete disconnectivity to low-cost, high-bandwidth connection can happen in a relatively short time span. Media consumption over such networks are influenced by two, contradictory user requirements. The first requirement is that the users would not like to see quality changes in the media stream. The second requirement is that the user would like to consume the media with the best available quality.

This paper presents an adaptation algorithm that aims to reconcile these requirements. Architecture variants of the media adaptation problem and prototyping experiences are also presented.

1. Introduction

Ubiquitous computing is a term that characterizes a computing system where the user interacts not with a single terminal (e.g. PC) but with a number of computing devices and each aspect of the service is accessed using the most appropriate device. Ubiquitous networking [1] covers the networking aspects of such applications. Ubiquitous networks are characterized by the integration of diverse network technologies. For example, a device may be accessing a network service over cellular network when a short-range access point becomes available. The device migrates the service access session to this latter network technology which yields higher bandwidth. Once the user leaves the proximity of the short-range access point, the session is migrated back to the cellular network causing a drop in the bandwidth.

This integration of highly different network technologies causes a significant variation of QoS properties when the service access session is migrated from one network technology to another. Wireless networks have always been challenging from the service developer point of view due to QoS variations. The variations exposed by e.g. one cellular network bearer can be described, however, relatively easily providing a simple QoS model to the service developer. Ubiquitous access introduces additional dimensions into this QoS model making it much more complex than before. Contrary to the traditional approach of guaranteeing certain QoS properties [11], fluctuation in QoS provided by the network is a fact of life and the sole means for applications to tackle it is adaptation [13].

Service developers need to handle this complexity to provide satisfying experience for their users. Handling the complex QoS nature of ubiquitous networks is governed by two, contradictory requirements.

- Users would like to consume the service with the best available quality.
- Users do not like quality changes because they have to re-adapt continuously to the new quality the service can provide. This can be demonstrated more easily with a car radio example. If there are obstacles against radio waves around the car (e.g. hills), the broadcast can be heard say, 95% of time with perfect quality but time by time there can be noisy interruptions. The average QoS is high; still, users tend to look for another radio station.

This paper presents an adaptation algorithm that is able to control media stream bandwidth considering these two contradictory requirements. The paper is organized as follows. Section 2 discusses the high-level media adaptation architecture so that our adaptation algorithm can be put into context. Section 3 presents the media adaptation algorithm itself. Section 4 presents the prototyping results and measurements. Conclusion and discussion of further work closes the paper.

2. Media adaptation architectures

The general media adaptation problem can be formalized as follows: given a set of media options m_i and context variables c_i, we look for an adaptation process $P(m_i, c_i)$ that maximizes some utility function. The utility function incorporates measurable media properties (e.g. bandwidth), non-measurable, subjective media properties (e.g. the user's preference of the adapted media format influenced by e.g. the user's mood) and cost incurred on the network infrastructure performing the adaptation. Considering this information flow, the complete architecture of the media adaptation system is depicted in Figure 1.

The architecture is divided into terminal- and network-side parts. The adaptation happens on the network side. The terminal collects and aggregates client-side context information that only the terminal can measure (e.g. battery life). The terminal is also responsible of interacting with the user if that is necessary (e.g. prompting the user with cost-related questions). The actual media player application may be involved in context collection and end-user interaction. Having that support built into the media player improves the quality of the context (e.g. media player buffer size does not need to be estimated, the player can report it) and the consistency of the user interface (e.g. the end user prompts have integrated look and feel with the media player's user interface) but explicit support of the context and user interaction support is not essential. The aggregated client-side context is transferred to the network side using a QoS signaling protocol where further context inference happens adding the context information gathered on the network side. Using the context set and the policy rules, an adaptation decision is made and the quality of the media stream is changed accordingly.

This complete architecture is able to handle all media adaptation use cases in terminal-network setup (media adaptation in P2P environment is not discussed in this paper). In practical cases, however, subsets of the complete architecture are used.

In the server-side adaptation decision case, the server obtains context information from the client-side (e.g. user preferences, client characteristics like screen size), collects additional

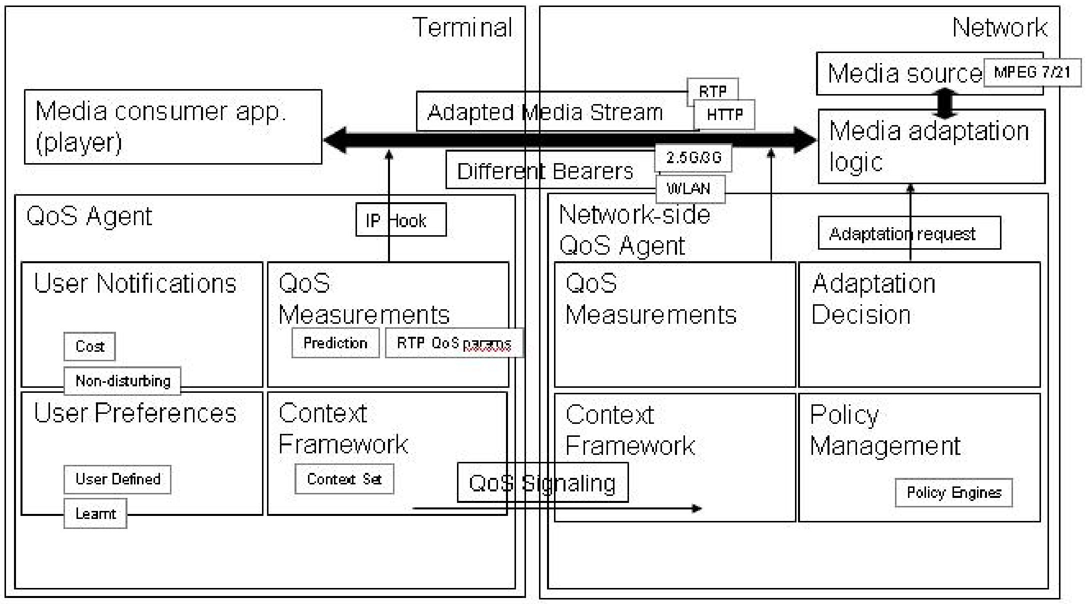

Figure 1 Complete architecture of the media adaptation system

context information on the server-side [12] then does the adaptation decision and the actual adaptation itself. In simplest form of this solution, no changes are needed on the client side. In more complicated setup, however, solving user interaction may be very problematic. Additional disadvantages are that content servers need to be upgraded and the client may not offer all the context information the server may use (e.g. environmental light or noise conditions). Obtaining the missing context information from the client complicates the solution.

In the client-side adaptation decision case, the entire adaptation framework is run by the client. It is the client that collects environmental information (e.g. network, user conditions) and makes the decision, which variant of the media material offered by the server to consume. The server's role is to attach meta-information to the media material so that the client can retrieve this meta-information and examine the options.

Advantages of this solution are the following: works almost seamlessly with legacy servers and content although the meta-information framework (e.g. [2] or [4]) needed for the client-side decision is not widely deployed. There are number of disadvantages, however: hostile client can attack the server by requesting adaptation steps that consume large amount of resources, client base need to be entirely updated with advanced player and media option advertisement need to be standardized else there will not be large enough server base so that the solution becomes economically viable.

Proxy-based adaptation is a variant of server-based adaptation. The solution does not require the content servers to host the entire adaptation logic as the adaptation decision engine and the adaptation logic is hosted by the proxy. The server does need, however, to serve meta-information about the media material although limited amount of meta-information may be extracted from the media material by the proxy. (Contrary to the client-based solution, the server may partially download multiple variants of the media material from the server and try to extract meta-information from the media files). Advantages and disadvantages of the solution are similar to those of the server-based solution except that the media servers do not need to be upgraded.

The solution presented in this paper is based on the assumption that client-side modifications are not realistic because they affect large installed terminal base. Our algorithm tries to estimate the client-side context information most relevant for adaptation by inspecting properties of the media stream. This makes the algorithm ideal for server-side and proxy-based solutions while keeping a number of advantages of the complete architecture.

3. Client buffer estimation and media adaptation algorithm

Streaming media delivery traditionally assumes non-reliable protocols, e.g. RTP. The media delivery landscape has changed dramatically, however, with the popularity of Flash-based media players (e.g. YouTube). These players provide streaming experience but actually use HTTP/TCP for media delivery. It has been shown that HTTP-based streaming media delivery already comprises 41% of all HTTP traffic (36% for streaming video, 5% for streaming audio) [10]. The assumption of TCP-based streaming media delivery is quite reasonable therefore.

The adaptation point should be aware of the current end-to-end capabilities of the whole system. If the adaptation takes place at the server side, the client must report back its measurements to the server using a signaling channel. TCP provides built-in signaling as part of the transmission control logic. Our system tries to exploit TCP acknowledgements to be used as QoS signaling facility. The requirement of our system was that the QoS measurement should be separated from the media player. Since the player was treated as black box, the monitoring component has no direct access to the playout buffer. To provide good adaptation the media source and the adaptation logic must be aware of, however, the client buffer status. Significant effort has been spent on evaluating the actual transcoding methods [3], our paper therefore concentrates only on the adaptation decision making.

3.1. Client buffer estimation

TCP ACK mechanism provides good measurement of the other end's status and the network path between the communicating parties [5]. The receiver sends back window advertisements in the ACK packets to report back the buffer status. Therefore both ends of the TCP socket have approximately same information about each other. The time offset between one end's knowledge about the other's status is equal to the network round trip time.

The synchronicity of the parties' status can be exploited so that it is enough to measure TCP parameters only on the server side. With this approach there is no need to additional signaling overhead by using the ACK mechanism already present.

The QoS monitoring component captures TCP packets at the IP level providing low-level access to the raw packet data. The monitoring component will 'see' the transmitted, retransmitted data packets and the acknowledgement packets as well. The ACK packets holds pointers to the acknowledged data (*P(t)*). As *P(t)* changes in discrete steps (when new ACK is received) and there is a network delay before ACK packets arrive, *P(t)* can be considered as a lower limit of the received bytes on the remote side.

The server provides media stream with a rate changing in time due to adaptation (*R(t)*). The buffered time can be calculated using the simple

$$B_t(t) = T_{sent} - T_{played}$$

formula, where T_{sent} and T_{played} denote the amount of sent and already played media in time units. The T_{played} value can be estimated using the time elapsed since the start of the streaming minus the initial buffering period, since the player is expected to play continuously in time.

The calculation of length of the already sent media in time units is a bit more complicated. Instead of the elapsed time, the ACK pointer at the current time ($p = P(t)$) is used.

$$T_{sent} = \int \frac{1}{R(p)} dp$$

where $R(p)$ is the bitrate of the emitted stream at the byte position p.

Since the solution is intended to be media player independent, the initial buffering period should be left out. This does not harm the estimation because we made effort to provide a lower limit of the playout buffer level. Additionally the received time amount on the player side can be greater then T_{sent} because of the ACKs on their way on the network back to the client.

It is important to distinguish our approach from the widely used playout buffer control algorithms [8]. We focused on the server side, trying to estimate the buffer levels on the other end without application-level feedback. The playout control mechanisms operate on the player node, trying to manipulate the audio/video output to react changes in the incoming flow. Both aspects use QoS measurements and make assumptions about the network state, however the point of view and the available means are different. Only the server side is able to change the bitrate of the stream serving continuous media in the presence of long-term network fluctuations.

3.2. Media adaptation using the estimated client buffer level

Given the QoS measurements and the client buffer level estimation the server has to make adaptation decisions. The requirements of the adaptation: 1, To select the possible best quality media stream in the dynamically changing environment. This best quality is always changing as the network conditions vary. 2, In contrast the system must avoid player buffer exhaustion to provide continuous playing. 3, Additionally users do not like changes in the perceived quality, so we have to keep quality switches minimal.

The continuous playing requirement is the most important one and using the estimated client buffer level it is quite easy to implement. When the estimated playout buffer level decreases towards zero, the adaptation logic must change to a lower quality. The selected new bitrate should guarantee that the playing remains continuous, however should try to use the best of the available streams. Our system uses a short-term TCP throughput estimation to select the new stream. The best quality media stream variant whose bandwidth is lower than the measured throughput is chosen. Since we want to provide the best quality whenever possible this is the only rule to decrease quality.

In case of upgrading stream quality the other two requirements are considered. In order to avoid frequent changes in the stream quality we introduced a system parameter. This parameter (t_{sw}) specifies the timeout. Stream quality changes happening more frequently than t_{sw} are undesirable because they can be disturbing to the user. Therefore, the system records the time of each quality change and does not consider stream upgrades until the timeout elapses.

During the prototype development we found that an extension to the third requirement is needed (illustrated on Figure 2). If the adaptation logic switched to better quality based on local decision, but the long-term TCP throughput (solid thin line) remained under the new bandwidth needs, the client buffer level started to fall slowly. If the initial level was high, this behavior did not harm the 3 requirements listed above, but resulted in a cyclical bitrate increase-decrease even under approximately constant network circumstances.

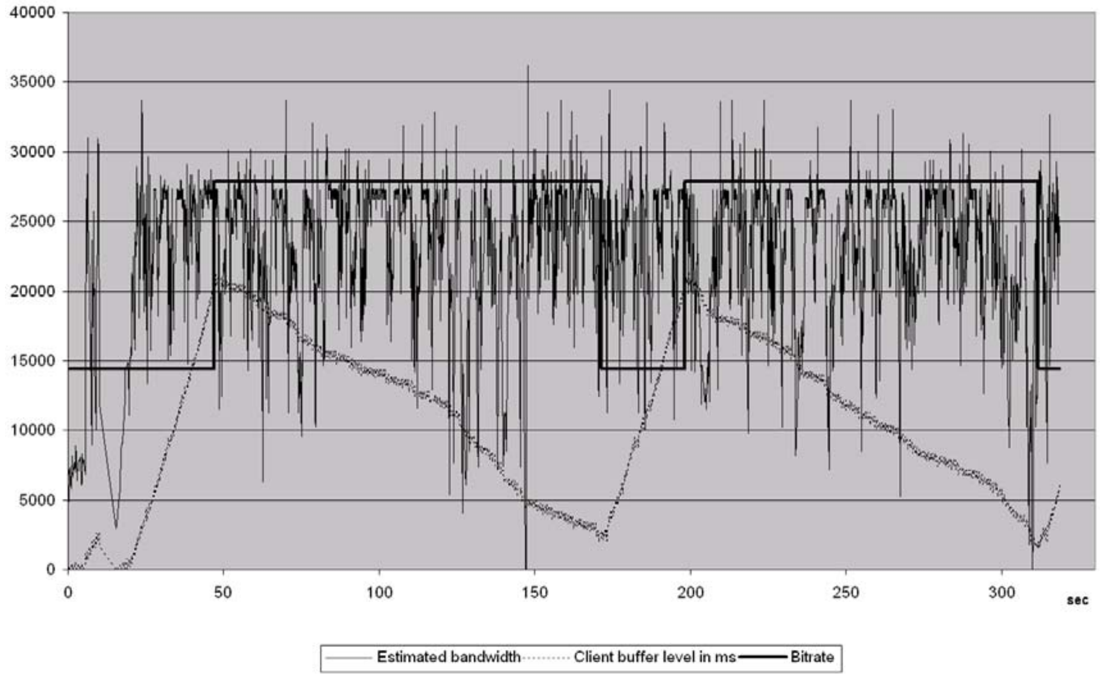

Figure 2 Cyclic adaptation in constant conditions

This was because the decision were based only on the client buffer level (dotted line), which increased nicely during the low media bitrate (thick line) periods, but decreased constantly during the high bitrate periods. Therefore we made effort to avoid such cyclic behavior.

To meet the described requirements we employed regression method to predict the expected future parameters of the end-to-end system. Due to the highly dynamic behavior of the common networks such predictions cannot be very accurate in practice, however detecting trends of the bandwidth fluctuations was useful. The more sensitive task of degrading quality (avoid buffer exhaustion) is based on concrete measurements, meanwhile the media bitrate increase decisions are made using the predicted data.

Since our QoS measurement component primarily provides ACK pointer series, the regression model has been built on these measurements. The third requirement forced us not to switch too frequently, therefore during the t_{sw} timeout ACK pointer data can be collected. The collected time series are used to calculate the linear regression parameters.

Then the regression model is used to predict future ACK pointers. Since the system had to make effort not to switch within the next t_{sw} timeout period, the system predicts ACK pointer in the $t_{current} + t_{sw}$ time span. Using predicted $\hat{P}(t + t_{sw})$ ACK pointer the system searches for appropriate bitrates that fulfill the requirements. Using the selected bitrate the $\hat{P}$ prediction is converted into predicted client buffer level estimation.

To select the best quality based on the predicted buffer level, first we investigated the following strategy. If the predicted client buffer level is above a minimal low buffer limit, the next quality switch is not likely to happen within the t_{sw} timeout. Since deviation from the prediction is expected, the predictions should be compensated using the expected error of the prediction. The error values are measured during the streaming and are used to improve next prediction. The decision rule can be formalized:

$$B_t(t) + \frac{\hat{P}(t + t_{sw}) - P(t) - \sqrt{E\{\varepsilon^2\}}}{R} - \Delta t_{played} > L$$

where B_t is the current buffer level, $E\{\varepsilon^2\}$ is the expected value of the collected squared prediction errors, R is the checked new bitrate, Δt_{played} shows how much *time* will be consumed from the buffer until the prediction point (in fact it is equal to t_{sw}) and L is the low buffer limit. Since the system has no initial error estimation, the first switches may break the switching timeout rule to avoid buffer exhaustion in case of bad conditions. Additionally this strategy may lead to cyclic up and down switches even if the available bandwidth is nearly constant at certain levels. Therefore we modified the strategy.

In the new approach the requirement for the predicted buffer level $\hat{B}_t$ was that it should not decrease within the next t_{sw} timeout. Let us consider a possible quality increase situation at time t. Since the last switch at least t_{sw} time has been elapsed. During switch timeout ($[t - t_{sw}, t]$) the available bandwidth was higher than the current playing rate, because the selected bitrate was chosen to be so (see Figure 3). Therefore the client buffer level has also been increased. If we use the regression model to select the new bitrate, the new strategy ensures that the expected client buffer level at the prediction point (at $t + t_{sw}$) will be greater or equal to the current level (denoted by the horizontal line). So if it turns out that the prediction was too optimistic, the client buffer level will fall under the estimated level. However, this is not likely to cause problems, since the high buffer level at the decision point enables a relatively wide range (the block arrow on the right of Figure 3) of error without breaking the requirements. Therefore in this new strategy there is no need for very accurate error estimation.

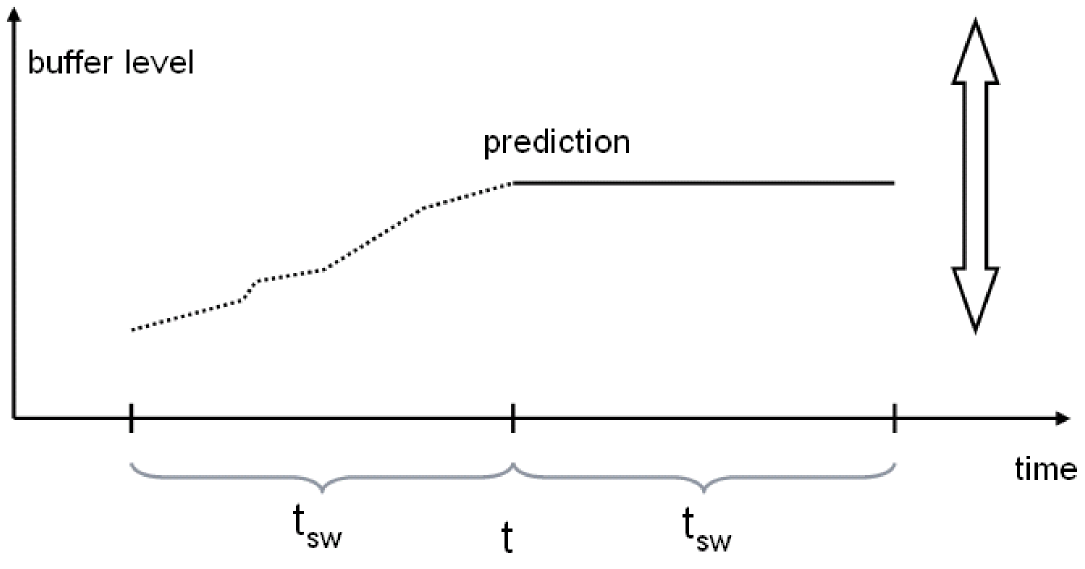

Figure 3 Prediction error insensitivity band

If the quality upgrade is not possible due to the network conditions, the regression model is updated continuously. However the regression line should be sensitive to positive bandwidth changes, therefore the model is updated using only the measurements of the last t_{sw} period. Additionally, when a quality switch occurred, the regression model is dropped and started again from the point of switch.

4. Prototyping and measurement results

We used WinPcap [6] packet capture library in our prototype. The media server, the QoS measurement module and the adaptation logic is implemented in a Java servlet. The servlet and the packet capture library communicate via Java Native Interface (JNI).

An emulated network environment was set up for prototype development purposes. A computer running network emulator software has been inserted between the server (running the adaptation servlet) and the client (running the media player). The emulator machine had two network cards and the NISTnet network emulator [7] software filtered and delayed the packets passing through the network interfaces. Using NISTnet we were able to control the available bandwidth, delay and packet loss ratios. Since TCP was used, the measured end-to-end throughput was heavily affected by not only the maximum bandwidth setting but also the delay.

The emulator enabled us to analyze the prototype in different scenarios, eliminating the real-world randomness we managed to simulate several network circumstances and the adaptation response of our algorithms. However later we verified the measurements in real 3G networks.

The prototype provides video-on-demand services. The video streams have been transcoded offline and stored in separate files on the server. Therefore there are a limited set of files for each media in different quality and bitrate. In case of adaptation the server simply switches between these file sets, and forwards the content of the new files to the stream. The switch can be carried out only at frame boundaries, so the media files are tagged with the positions of possible switch points.

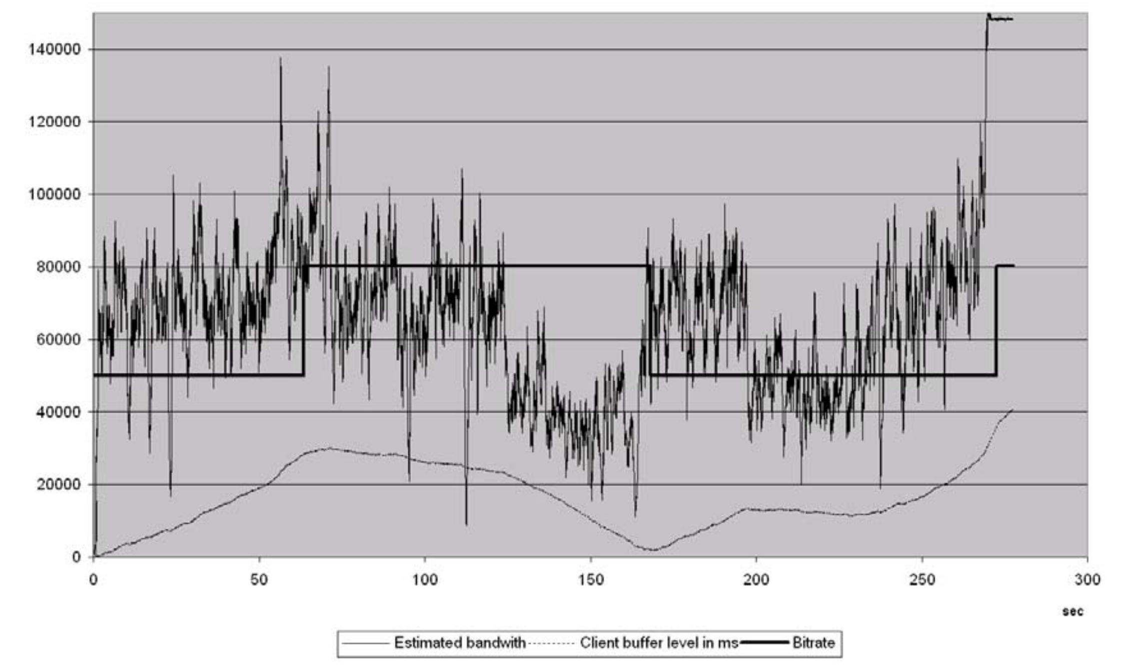

Figure 4 Adaptation in emulated, changing environment

The servlet collects the measurement and decision data in order to enable offline analysis. Figure [4] shows a playing session in the emulated environment described above.

The thick line shows the output of the adaptation decision: the media bitrate of the transferred stream. System parameters used for this diagram were $t_{sw} = 20\ sec$, $L = 2\ sec$ and bandwidth measurement window was 2 seconds. The system conformed to the requirements; the playout was continuous tried to use the best quality whenever possible and avoided too frequent changes, thus 20 seconds in this example. The continuous thin line represents the short-term (in this case 2sec) bandwidth estimation, while the dotted line is the estimated client buffer level during playout.

In spite of the existing complex TCP bandwidth measurement solutions [9], there is no need for very accurate bandwidth estimation in this system. The only purpose is to help stream degrading switches in low buffer situations. The calculated value therefore must reflect the most recent TCP throughput.

The prototype has also been tested in real mobile environment. The client media player was running on a notebook computer with 3G modem card. The server was connected to the public Internet so that it can be accessed from the 3G network. Therefore the network path contained a wireless part, the mobile operator's internal network, the Internet and the ADSL line which was connected to the server machine.

Figure [5] presents the collected measurements in case of UMTS connection ($t_{sw} = 20sec$).

The mechanisms of the adaptation strategy can be seen in action. When the network provided high bitrate connection, the average bandwidth varied around 24kB per second. Obeying the requirements the new media bitrate was chosen so that the predicted buffer level should not change in the next t_{sw}. The higher bitrate section started with high client buffer level, therefore there was no problem until the network dramatically changed. When the lower bitrate provided by the network caused the client buffer to become empty, the adaptation logic switched to lower bitrate. Then as the throughput stabilized again around 24k, the system switched to a higher bitrate, but not higher than the long-term bandwidth.

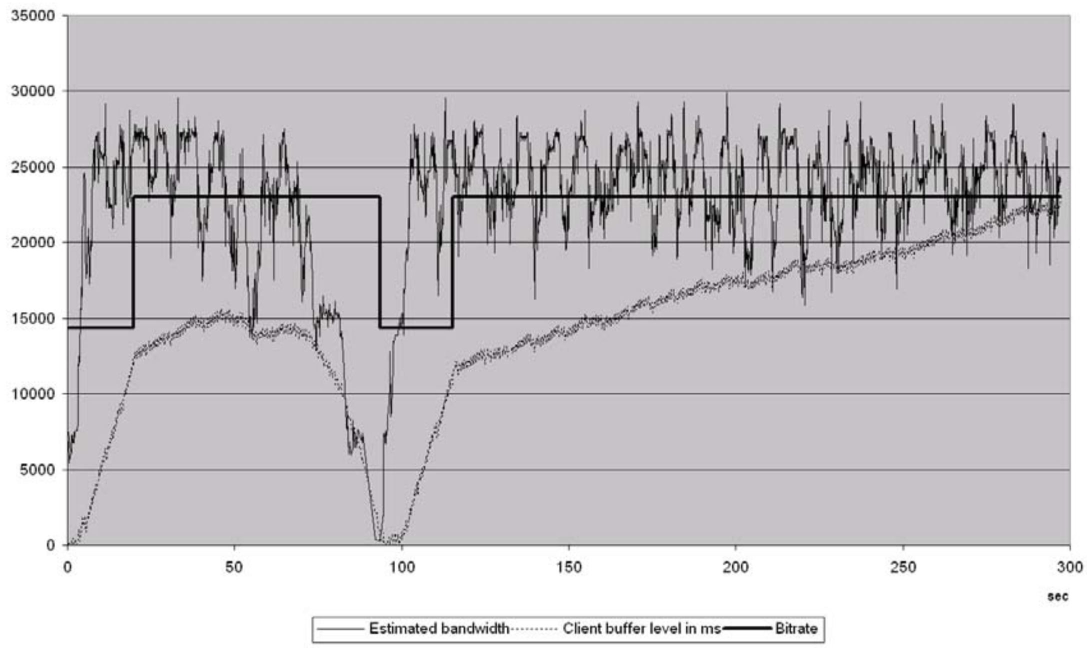

Figure 5 Adaptation in real 3G environment

5. Conclusions

Consuming media in ubiquitous environment frequently requires media adaptation. Media adaptation is influenced by two, contradictory requirements: media should be consumed by the highest available bandwidth provided by the channel but the end users dislike frequent media quality changes. The media adaptation logic may be deployed on both the client and the server side but as it was demonstrated in section 2, server-side deployment of the adaptation logic has significant advantages. It was also demonstrated that context information from the client side must also be collected and the simplest solution is to use a dedicated QoS channel for this purpose.

We made the interesting observation that it is possible to extract important QoS properties of the client (namely, the observed bandwidth on the client side) by observing the behavior of the TCP behavior on the server side. TCP sounds like an inappropriate solution for delivering streaming media content but in fact, it is widely used for this purpose. The adaptation algorithm described in this paper is able to use bandwidth estimation extracted from the TCP channel and is able to consolidate the two conflicting requirements of the media adaptation. Prototype implementation was created and the proposed algorithm showed promising results during these tests. Further research direction is to test the behavior of the algorithm in more advanced testbed environment.

6. References

[1] J. Weatherall and A. Jones, Ubiquitous networks and their applications, IEEE Wireless Communications, Volume: 9, Issue: 1, pages 18-29, Feb. 2002.

[2] A. Vetro and C. Timmerer, Digital Item Adaptation: Overview of Standardization and Research Activities, IEEE Transactions On Multimedia, Vol. 7, No. 3, June 2005.

[3] A. Vetro, C. Christopoulos, and H. Sun, Video transcoding architectures and techniques: an overview, IEEE Signal Processing Magazine,vol. 20, pp. 18u 201329, March 2003.

[4] 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs (Release 7) [26.234]

[5] Douglas E. Comer, Internetworking with TCP/IP. Principles, protocols and architectures, fourth edition. Prentice Hall

[6] WinPcap: The Windows Packet Capture Library, www.winpcap.org

[7] Mark Carson and Darrin Santay, NIST Net, A Linux-based Network Emulation Tool

[8] Kohei Fujimoto, Shingo Ata and Masayuki Murata, Adaptive Playout Buffer Algorithm for Enhancing Perceived Quality of Streaming Applications, Telecommunication Systems, 2004

[9] T. Tateoka, Y. Kurita and K. Abe, A Bandwidth Measurement Technique for Mobile Computers

[10] Ellacoya Networks, Inc., Ellacoya Data Shows Web Traffic Overtakes Peer-to-Peer (P2P) as Largest Percentage of Bandwidth on the Network, http://www.ellacoya.com/news/pdf/2007/NXTcommEllacoyaMediaAlert.pdf

[11] C. Aurrecoechea, A. T. Campbell and L. Hauw, A Survey of QoS Architectures, Multimedia Systems, Volume 6, Issue 3 (May 1998), p. 138-151

[12] Q. Han, N. Venkatasubramanian, Information Collection Services for QoS-Aware Mobile Applications, IEEE Transactions on Mobile Computing, Volume 5, Issue 5 (May 2006) p. 518-535

[13] K. Curran, G. Parr, A middleware architecture for streaming media over IP networks to mobile devices, Wireless Communications and Networking, 16-20 March 2003, Vol. 3, p. 2090- 2095

Authors

Gergely Hományi works as a research engineer at Nokia Siemens Networks in Hungary. He received his M.Sc. degree from the Budapest University of Technology and Economics. Previously, he worked at Nokia Research Center on various research projects. The spectrum of his research included planning and management of wireless mesh routers, location-based services, and applications of proximity networking (Bluetooth, ad-hoc WLAN). Recently, he has been involved in mobile media research, especially quality-of-service issues in a ubiquitous environment.

Gábor Paller reveived his MSc. and PhD. degrees from the Technical University of Budapest in 1992 and 1996, respectively. Dr. Paller joined Nokia in 1998 and held positions in Nokia R&D and Nokia Research Center.

His interest included wireless protocol development, mobile device management, mobile Java and middleware. He was also involved in standardization and joint research program activities. Since 2008, he works at OnRelay Ltd. on fixed-mobile convergence technologies.