RapidStream: P2P Streaming on Android

Philipp M. Eittenberger, Matthias Herbst, Udo R. Krieger
Faculty of Information Systems and Applied Computer Science
Otto-Friedrich University
Bamberg, Germany
Email: {philipp.eittenberger, udo.krieger}@uni-bamberg.de

***Abstract*—In this paper we present the architecture of the first mobile P2P streaming prototype for the operating system Android. At first, we discuss the application of P2P streaming in the scenario of mobile networking. Then, the system and software architecture of our prototypical implementation is elaborated. In addition, an initial field test to evaluate the feasibility of the proposed approach is presented. Finally, we report our insights arising from the practical experience with Android.**

I. Introduction

Mobile video traffic is growing rapidly with yearly growth rates of more than 90 % according to [2]. The rapid deployment of new multimedia services including video streaming as well as many new video portals, like PPLive, PPStream or SopCast, indicates the evolutionary path towards the next generation of mobile networks. Hence, one important aspect that needs to be investigated is given by the optimal dissemination of video data in the next generation of wireless networks. One viable possibility to distribute the traffic load more evenly in the network and thereby, provide lower costs and higher scalability is given by the usage of peer-to-peer (P2P) technology. However, current P2P applications are not tailored for these new requirements; quite the opposite, P2P streaming applications tend to use the network resources very aggressively and at least P2P streaming applications have no or little preference to exchange data among nearby peers [3]. This is not surprising, because current P2P video streaming applications have been specifically developed for a "wired" scenario, where users run the application at their PC, which is connected via the customer premises network to the ISP. However, with the advent of ubiquitous computing users want to use their accustomed applications wherever they are. Thereby, new requirements arise that need to be addressed by P2P applications in order to provide a sufficient quality of experience.

Let us first clarify the terminology of the P2P domain: Users, so called *peers*, are connected with each other in overlay networks to share resources, in this case, to disseminate video data. The main difference to the client/server paradigm is given by the fact that each peer can be at the same time client and server. A common approach to disseminate video data via P2P is to split up the data in smaller units, called *chunks*. They are then distributed between the peers of a swarm. A source node, in this domain called *seed*, provides the initial upload of the data; subsequently, the downloaders distribute the data further to other peers. To indicate the chunks a peer currently holds, the peers exchange their *buffer maps*, also called *chunk maps*. Mainly two different service types can be distinguished within P2P streaming: A *video on demand (VoD)* system provides users with VCR functionality, e.g. stop, rewind or fast forward of the video. In contrast, by *live streaming* the users have a more TV-like experience, where all users view the same playback time within a certain range of delay. Regarding the system architecture and, in particular, the implementation of the data dissemination, the systems can be coarsely divided into two main groups: *Mesh-pull* systems build an unstructured overlay, hence "mesh", and each peer requests, i.e. "pulls", the data from other peers. *Tree-push* systems explicitly construct a dissemination overlay and "push" the data along the constructed "trees".

P2P streaming applications have attracted a lot of attention in recent years. Numerous scientific studies investigated their properties, large research projects have been founded to develop prototypes (e.g. NapaWine[12] or PPNext[13]), but more important, real systems have also been deployed successfully. These P2P streaming applications are able to serve simultaneously up to hundreds of thousands of users nowadays. It is therefore just a matter of time when these applications also pervade mobile networks. The goal of our work is to investigate these necessary adaptations of P2P techniques for the mobile, wireless dissemination of video content. For this purpose, we have developed the prototypical P2P streaming application *RapidStream*. In this paper we will present its current architecture and report our insights of using Android as a client platform for P2P applications.

II. Related Work

Despite the fact that there is a large amount of scientific studies investigating P2P video streaming, there is relatively little work regarding the usage of P2P streaming applications on mobile devices. The first studies introducing P2P video streaming applications that are able to operate on mobile devices are presented by Venot and Yan [14], who introduce a JXTA based P2P video streamer. Yet, their implementation was not able to stream the video content progressively if it was operating on a mobile device, i.e. it could only display the video file when it was completely downloaded. Also Zhang *et al.* [15] presented a Symbian based P2P video streamer, but

The final version of this paper appeared in the 19th International Packet Video Workshop (PV), 2012.
DOI: 10.1109/PV.2012.6229724

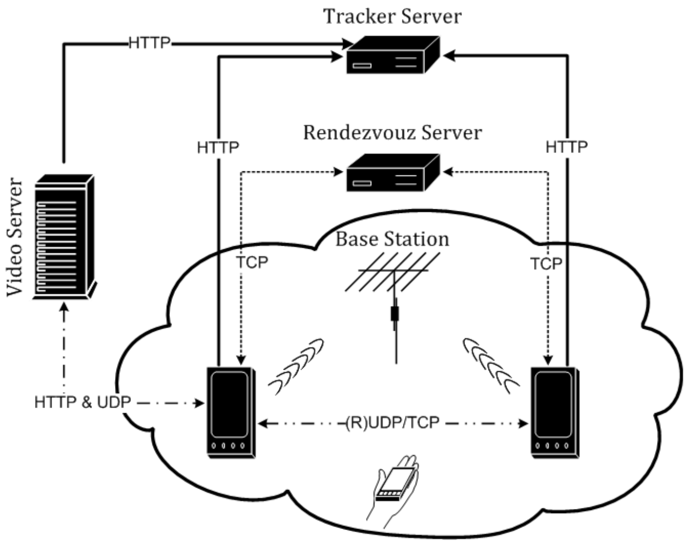


Figure 1. Mobile P2P Streaming Architecture

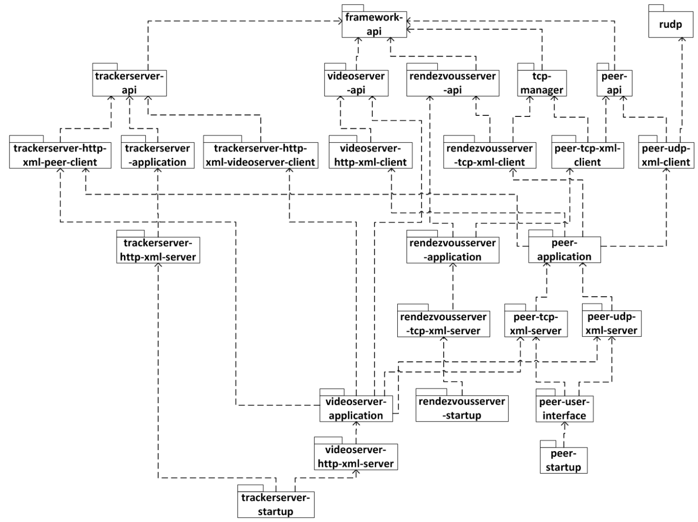


Figure 2. UML Packet Diagram of the Framework

the study did solely evaluate the energy consumption of the proposed system on the mobile devices. Furthermore, Diaz *et al.* [6] conducted a measurement study on a Symbian based P2P video streaming application over cellular networks. All of these first presented P2P video streaming prototypes for mobile devices suffered from the limited resource capacities of mobile devices given at that time (2007). Only recently by the appearance of much more powerful mobile devices this approach has become feasible in practice. In 2010, Peltotalo *et al.* [11] presented a fully working RTSP based P2P video streaming application for Nokia smart phones. Other work that investigates certain aspects of this approach consists of Noh *et al.* [10], who proposed a transcoding scheme to enable video streaming to mobile peers. Cycon *et al.* [5] introduced a H.264 video encoder that operates in real-time on mobile phones for P2P video conferencing. Finally, Leung and Chan [9] proposed a protocol for the P2P dissemination of multimedia content to mobile devices.

III. RapidStream - P2P Streaming on Mobile Devices

The migration of P2P video streaming applications to a mobile environment requires certain adoptions, since mobile devices are battery powered and have in general less computing power compared to standard PCs. In addition, P2P applications running in a mobile environment encounter different network dynamics compared to a "wired" scenario. Although next generation mobile networks, like LTE, strive to provide broadband-like downstream capacity, hand-overs and the fluctuating link quality negatively affect the transmission performance of P2P applications [7]. Moreover, the protocols of such P2P applications even increase the rate of control messages due to connection disruptions, and thereby, produce more signaling overhead. Apart from these network induced conditions, there are also new requirements caused by the used hardware platforms. As already mentioned, mobile devices provide less computing resources, i.e. primarily less memory and CPU speed. For instance, on current Android versions the heap space of a Java application is limited per default to a maximum of 32 MB. Only with root access on the device we could increase the heap space to the "hard wired" maximum of 48 MB. As a consequence, this implies that the video buffer must be kept quite small and that it is crucial to hold not too many open connections in parallel. In addition, to enable the P2P video data dissemination on Android, the protocol of the P2P application should be as lean as possible, to avoid too much signaling traffic and in general, to keep the communication overhead at a minimum. Therefore, an important development objective of mobile P2P video streaming applications must be the sustainable usage of the network resources, i.e. to be as energy and resource efficient as possible.

As a platform for our prototype we have chosen Android, which is a partly open source, Linux based OS mainly tailored for mobile devices. User programs written for Android are executed in the *Dalvik Virtual Machine (DVM)*, which is based upon the *Java Virtual Machine (JVM)*. We found that there is generally a good fit between both, but in some cases the Android's Java implementation provides only stubs and no implementations for some classes of the Java API.

A. System Architecture

For the prototypical implementation we have chosen to build a mesh-pull based live streaming network. Regarding the system design we went for a hybrid P2P overlay maintenance design, i.e. the system is not fully decentralized, since there is a dedicated, centralized infrastructure to retain control of the P2P network and to relieve the load of the peers. The central infrastructure consists of a tracker server, a video server and a rendezvous server. In the simplest case, only a single instance of each component exists in the system. Of course, to increase the reliability and the performance of the system, one could always use more, redundant instances of each component. The video server needs to register his content at the tracker server to be able to stream video content. In the proposed system, each video results in an own dissemination swarm, i.e. only the peers watching the same video exchange video data. In future extensions more advanced techniques could also be considered to increase the dissemination performance. The process of joining of a swarm, i.e. watching and re-distributing a chosen video, can be described as follows: Upon registration at the tracker server, each peer receives a unique identifier (PeerID). In addition, the tracker server returns a list of the currently broad-casted channels in combination with the connection information of the video servers. If the peer has chosen a video channel, it informs the tracker server of joining the particular swarm. The tracker server supplies the peer with an initial peer list in the bootstrap process. The implementation of the tracker protocol follows closely the standard proposal given in the PPSP Tracker Protocol [4] and it is probably one of the first working implementations. With the help of the rendezvous server, the peer is able to communicate with other peers, even if they or itself are behind a NAT. Right now, only UDP hole punching is implemented, as it yields the highest success rate. However, in future releases we will include TCP hole punching as well to increase the chance of a successful NAT traversal. Upon successful connection setup, the peers exchange their buffer maps and they request missing chunks from each other. The exchange of the buffer maps and the chunk transfer are conducted iteratively as long as the peer is watching the video. Due to the structure of the proposed system, every communication relationship needs a particular type of connection. Figure 1 sketches the system architecture of RapidStream and illustrates the usage of the transport protocols for the different scenarios. For the communication with or between the servers, HTTP is a well suited protocol (as described in [4]), as it is the protocol that is the more likely to work in any case. The communication with the rendezvous server is performed by TCP. The video data dissemination requires a more efficient transport protocol, thus, the connectionless UDP is used. However, to ensure the successful transmission of signaling and control messages between the peers, the framework can use TCP respectively RUDP [1] for this kind of communication too. To ease the performance analysis and the gathering of measurement data, we have also included a statistics server, which receives periodically measurement data from all network participants. Since this server is not a vital part of the P2P network, it is not depicted in the system architecture (cf. Figure 1).

B. Software Architecture

We have developed a general P2P streaming framework, which is based upon a modular software architecture. The framework is purely Java based and can be executed on every Java-capable device. Figure 2 depicts the UML package diagram of the framework. The hierarchical structure illustrates the dependency between the packages. Every package can be easily replaced as long as its dependency is considered.

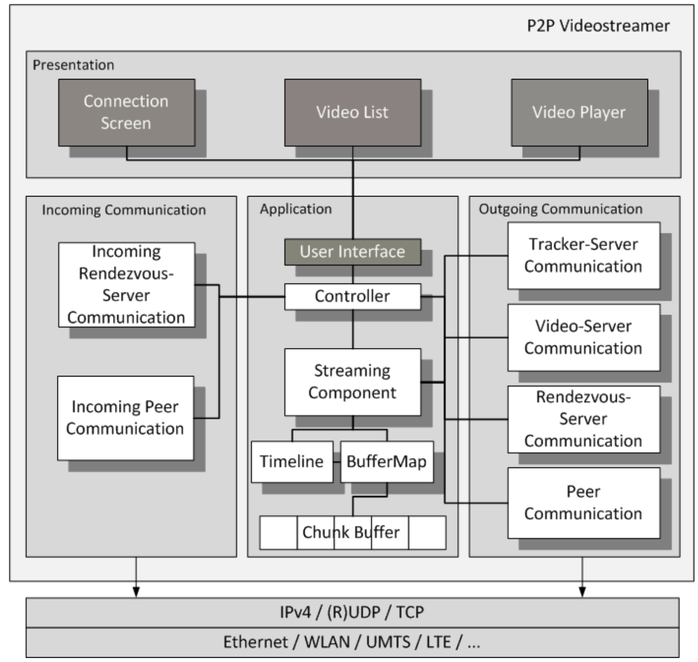


Figure 3. Architecture of RapidStream's Android P2P VideoStreamer

This modular design enables the combination of existing components, it allows for instance the bundling of the tracker server, the video server and the rendezvous server into one executable program. The central *Framework-API* provides the most common data objects, which are shared between all the modules and which are used for the inter-module communication. After the completion of the framework, the next step included porting the peer module to Android. The architecture of the Android P2P streaming application is illustrated in Figure 3. Porting the application was relatively straightforward; the main difference is that there is no Swing on Android. Therefore, a few GUI classes, in the Android terminology called *activities*, had to be added to the presentation layer of the P2P videostreamer: The *Connection Screen* is used to enter the contact information of the different servers, the *Video List* activity displays the obtained list of available videos and upon reception of enough video data, the *Video Player* shows the particular video. The brain of the application is the *Controller*, which manages in coordination with the *Streaming Component* all the connections through the incoming and outgoing communication modules. The *Streaming Component* is also responsible for the internal video buffer and for advancing the time line of the chunk buffer.

C. Interaction Pattern of a Peer

Figure 4 depicts an UML sequence diagram that illustrates the initial operations of a peer without explicit error handling considerations. At first, the device running RapidStream connects to the tracker server receiving its PeerID. With the PeerID and its connection information it registers itself at the rendezvous server and requests the list of video servers from the tracker server. Then, the peer may contact a particular video server and request the list of videos/channels. The connection establishment might be enabled by the rendezvous server, if the video server is behind a NAT. Upon choosing a particular video or channel, the video server provides the necessary meta data of the video. This procedure is further explained in Section III-D. Subsequently, the peer requests the buffer map of the video server and sets its initial buffer map accordingly. In addition, the peer informs the tracker server of joining the dissemination swarm of the video and requests a bootstrap list of other peers participating in the same swarm. The following interactions may be conducted iteratively as long as the peer is a member of the video swarm: The peer contacts the video server to update its peer list, then, it choses a particular chunk to request and a peer and subsequently, tries to establish a connection to the chosen peer. Again, the connection establishment might be enabled by the rendezvous server. After a successful connection, the peer is requesting the buffer map of the second peer and a particular chunk, if the contacted peer possesses a missing chunk. When the peer has successfully downloaded the first chunk, it may itself serve chunk requests of other peers too. One can observe that we have “outsourced“ as much of the overlay maintenance functionality as possible to the dedicated network infrastructure. RapidStream even avoids the standard keep-alive message exchanges between the peers to keep the signaling overhead small. The scarce resources of the mobile devices are mainly used for the dissemination of video data. From the perspective of requiring a lean and sustainable P2P streaming protocol, large chunk sizes are also necessary to yield a small overhead rate. Otherwise, if the chunks are too small, there is a lot of signaling overhead due to the continuous connection establishments and the necessary negotiations among the peers. However, if the chunk size is chosen too large, the receiver will have to wait longer for the reception of a chunk leading to an increased play back delay. In the presented version of RapidStream we have opted for chunk sizes of up to 2 MB to reduce signaling traffic and the energy consumption by limiting the transmission phases of the air interface (compare with the stepwise increase of the received traffic in 7). As each peer receives data from a multitude of peers, the optimal number of concurrent data transmissions is an important parameter to reduce the resource usage too. Currently, each peer downloads from at most 5 peers in parallel. All of these parameters were chosen according to our experimental investigations, but for the best possible video experience they need to be validated analytically. However, we leave this open for future work.

D. Pitfalls on Android

Android provides a multimedia framework that includes codecs for the most common audio and video formats. We use the *MediaPlayer* of this API for the play back of the received video data in order to avoid writing our own video player. As one might expect from such a smartphone platform, like Android, the Java implementation of the MediaPlayer should be the same on all the particular devices. However,

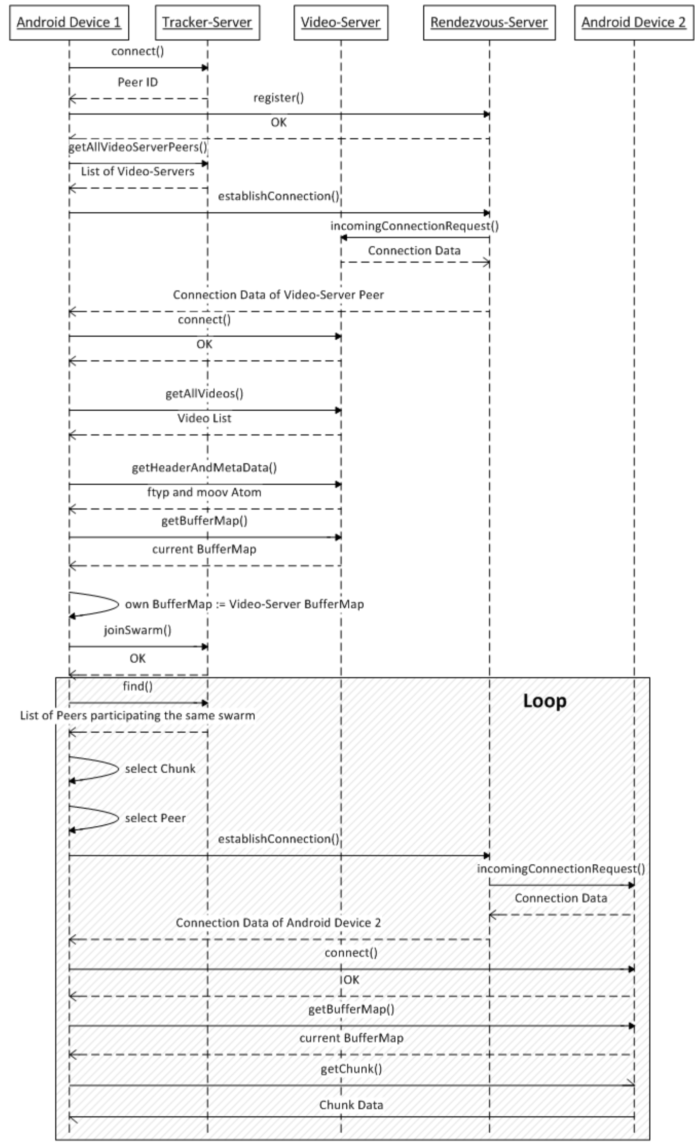


Figure 4. UML Sequence Diagram of the P2P streaming player

since the MediaPlayer is relying on native implementations of the video codecs, i.e. C/C++ code provided by the particular device manufacturer, we encountered a different behavior of the MediaPlayer instances with regard to different smartphone manufacturers. In order to get the certification "Android compatible" for a particular device, it must be tested by Google’s Android Compatibility Program. Only in this case, it may participate in the Android ecosystem, e.g. have access to the Android market. Despite this certification process by Google, we encountered serious problems with Samsung devices. The state diagram of Android’s MediaPlayer has been specified by Google (compare [8]). Yet, for reasons that have to be clarified, the method call of *prepare()* may lead in some cases directly to the state *Playback Completed* on Samsung devices. According to the state diagram in [8] this transition should not be possible, respectively, it is not allowed. To circumvent this error, our current implementation loops over the *prepare()*

Table I
HARDWARE BASE IN THE TEST RUN

ID	Vendor	Model
1	Samsung	Galaxy Gio (1)
2	Samsung	Galaxy S (1)
3	Samsung	Galaxy Gio (2)
4	Samsung	Galaxy S (2)
5	Samsung	Galaxy Ace
6	HTC	Desire
7	Sony Ericsson	MT15i
8	LG	Optimus 2X
9	Samsung	Galaxy Tab

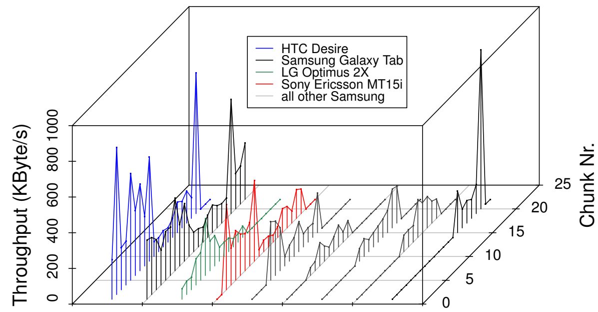


Figure 5. Download Throughput (Test Run)

method until the state *Prepare* is finally entered. We did not encounter this nuisance on devices of any other manufacturer. To be completely independent of such limitations, we would need to provide our own implementation of a media player with the related codecs. As such a task was not the focus of our work, we will await for a porting of other media players, like the VLC player, to the Android platform in future versions.

Android (up to version 3) supports the two container formats 3GPP and MPEG-4 (Part 12) for video codecs. The provided MediaPlayer is even capable to support streaming with RTP/RTSP. Therefore, the general suitability for live streaming in the P2P context is, in principle, given. Such an approach would need an implementation of a RTSP server on every terminal device. This RTSP server would be used to “feed” the MediaPlayer with the reassembled video data. However, we could not find a publicly available Java based implementation of the RTP/RTSP stack. Therefore, we decided not to implement the needed RTSP functionality, as this was not the focus of our work either. Yet, we decided to use the MediaPlayer after all, as one can rely on the fact that this player is available by default on every Android compatible device. To make the MediaPlayer capable of progressive streaming, the video server needs to manipulate the container format. Most encoders write the meta data of the MPEG-4 container, called *moov atom*, at the end of the video file and therefore, downloading the entire file is required in order for the MediaPlayer to be able to read the meta data of the video and start the play back. When the moov atom is relocated to the front of the file and its offsets are adjusted accordingly, the MediaPlayer is able to start playing the video, even if the whole file is not yet available. For this reason, the video server extracts the moov atom and provides it together with other meta information (the top-level *ftyp* atom) to the peers. Upon joining a particular swarm, the peer downloads at first the video meta data from the video server and creates a temporary file to buffer the video data as it is downloaded from the peers. The initial temporary file is empty apart from the meta data at the beginning. But it will be continuously filled with the received chunks, which have to be stored at the right position of the file. If a certain threshold of received video data is reached, the video play back starts.

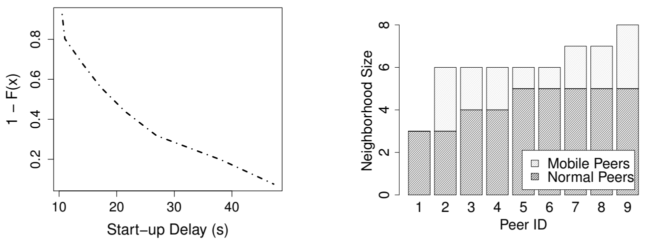
Android is a standardized platform for mobile devices, therefore, one might anticipate that this fact makes the life of a developer much simpler. Quite to the contrary, due to the broad, heterogeneous hardware base of Android compatible devices, we found that the developer needs to test his application on as many different devices as possible to ensure its functionality.

IV. MEASUREMENT RESULTS OF AN INITIAL FIELD TEST

Since we are especially interested in the performance that can be achieved in practice, we have evaluated the performance of RapidStream in the following experiment. To judge the general feasibility of the proposed system, we have conducted a small scale test run. Due to the fact that we have only a very limited number of Android devices, we recruited nine persons, who posses Android smartphones, to participate in the test run[1]. The resulting hardware base is depicted in Table I. Unfortunately, most of the participating devices were from Samsung. This fact is responsible for the relatively long start-up delays depicted in Figure 6 (a). The start-up delay is the time between the initial request of the video data and the time the video playback starts. As already mentioned, our implementation had to loop over the *prepare()* method until the state *Prepared* was finally reached on Samsung devices. This circumstance had a negative effect on the start-up delays. Yet, half of the peers could start watching the video in less than 20 seconds. On the HTC Desire smartphone we even measured start-up delays of less than 10 seconds, which is a really promising result. To start our experiment, the participants had to download and install RapidStream from a web server. Subsequently, they could enter the P2P network and join the test swarm. One initial video server provided the open content film "Big Buck Bunny"[2]. This video was encoded by H.264, a resolution of 320x180 pixels and 24 frames per second. The movie length is 9.56 min and it has a total data volume of 61.7 MB. Thus, to watch the video without disruptions, a device needs at least a download throughput of

[1] A short video of the test run is provided at: http://www.ktr.uni-bamberg.de/project/rapidstream.html

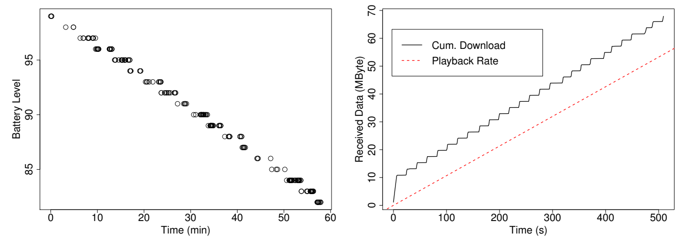
[2] http://www.bigbuckbunny.org/



(a) Start-up Delay

(b) Peer Neighborhood Distribution

Figure 6. Test Run Results



(a) Battery level

(b) Cumulative Download Traffic

Figure 7. Test Run Results (continued)

105.94 KByte/s respectively 847,5 KBit/s. Figure 5 depicts the achieved download throughput in the test run. It can be observed that nearly all the devices are capable to reach the necessary throughput rate. The different starting points of the particular graphs result from different swarm joining times, since some participants needed more time to download and install RapidStream on their device. To provide a more realistic scenario, we have setup some additional PCs running the peer application too. Figure 6 (b) illustrates the distribution of the peer neighborhood of the mobile devices. A *normal peer* represents a peer application running on a desktop machine. Since the PCs were started before the mobile devices did join the P2P network, they were able to serve successfully most of the chunk requests of the mobile peers. Figure 7 illustrates more measurement results taken on the Samsung Galaxy Tab. Figure 7 (a) displays the drainage of the battery during the test run. If one extrapolates the battery usage, one can see that the battery of this device is able to provide roughly 4 hours of video display with RapidStream. Figure 7 (b) illustrates the received download traffic and the playback rate of the video for the first 10 minutes of the test. As already mentioned, the stepwise increase of the cumulated down-link traffic is due to the large chunk sizes and the hereby induced on-off traffic pattern that aims to reduce the energy consumption of the air interface as much as possible.

V. Conclusion

To the best of our knowledge, this work presents the first academical P2P application operating on Android. In summary, we have introduced the general architecture of RapidStream, a proof-of-concept implementation of a P2P video streaming application for Android compatible devices. Furthermore, the paper presents a preliminary field test to evaluate the general feasibility of the proposed approach. Our first insights regarding the feasibility of P2P streaming in the domain of mobile devices and the results of our experiments are promising. Despite the fact that our test run was performed on a rather small scale, we were able to identify that Android devices are in principle capable to support the necessary streaming rates. In addition, we could identify the main challenges that need to be addressed in future work. The most important challenge for mobile P2P is given by the reduction of the energy consumption on battery powered devices. Thus, the goal is to be as energy efficient as possible. Further requirements that need to be addressed in future work consist of the optimal peer selection and the efficient data transmission in cellular networks. We will address the shown pitfalls and investigate more sophisticated data dissemination patterns specifically focused on mobile networks in future versions of RapidStream.

Acknowledgments

The authors are thankful to Marcel Großmann for his assistance with the measurements.

References

[1] T. Bova and T. Krivoruchka. Internet-draft - reliable udp protocol. 1999.
[2] Cisco Systems. Visual networking index cisco visual networking index: Forecast and methodology, 2010-2015. *White Paper*, 2011.
[3] D. Ciullo, M. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: A measurement study. *IEEE Transactions on Multimedia*, 12(1):54 –63, 2010.
[4] R. S. Cruz, M. S. Nunes, Y. Gu, J. Xia, D. A. Bryan, J. P. Taveira, and D. Lingli. Internet-draft - ppsp tracker protocol. 2011.
[5] H. Cycon, T. Schmidt, G. Hege, M. Wahlisch, D. Marpe, and M. Palkow. Peer-to-peer videoconferencing with h.264 software codec for mobiles. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks. (WoWMoM '08)*, pages 1–6, 2008.
[6] A. Diaz, P. Merino, L. Panizo, and A. M. Recio. Experimental analysis of peer-to-peer streaming in cellular networks. In *21st International Conference on Advanced Information Networking and Applications. (AINA '07)*, pages 784–791, 2007.
[7] P. M. Eittenberger, S. Kim, and U. R. Krieger. Damming the torrent: Adjusting bittorrent-like peer-to-peer networks to mobile and wireless environments. *Advances in Electronics and Telecommunications*, 2(3):14 – 22, 2011.
[8] http://developer.android.com/reference/android/media/MediaPlayer.html. State diagram of android's mediaplayer. 2011.
[9] M. Leung and S. G. Chan. Broadcast-based peer-to-peer collaborative video streaming among mobiles. *IEEE Transactions on Broadcasting*, 53(1):350–361, 2007.
[10] J. Noh, M. Makar, and B. Girod. Streaming to mobile users in a peer-to-peer network. In *Proceedings of the 5th International ICST Mobile Multimedia Communications Conference (MOBIMEDIA '09)*, pages 24:1–24:7, 2009.
[11] J. Peltotalo, J. Harju, L. Väätämöinen, I. Bouazizi, and I. D. D. Curcio. Rtsp-based mobile peer-to-peer streaming system. *International Journal of Digital Multimedia Broadcasting*, 2010.
[12] http://napa-wine.eu. Napa Wine.
[13] http://www.p2p-next.org/. P2P-Next.
[14] S. Venot and L. Yan. On-demand mobile peer-to-peer streaming over the jxta overlay. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies. (UBICOMM '07).*, pages 131 –136, 2007.

[15] J. Zhang, J. Niu, R. He, J. Hu, and S. Limin. P2p-leveraged mobile live streaming. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops. (AINAW '07).*, pages 195–200, 2007.