Acta Universitatis Sapientiae
Electrical and Mechanical Engineering, 5 (2013) 5-20

Network Based Caching for Near Real-Time Streaming Video

Csaba SIMON, Markosz MALIOSZ, Balázs BARANYAI

Department of Telecommunications and Media Informatics,
Budapest University of Technology and Economics,
1117 Budapest, Magyar T. krt. 2., Hungary,
e-mail: {simon; maliosz}tmit.bme.hu

Manuscript received March 29, 2014; revised September 15, 2014.

Abstract: Nowadays more and more live events are being consumed via the web or smart phones rather than legacy broadcast services. In this paper we present a scenario – focusing on wireless smartphone-based nodes – that covers this trend. Starting from this scenario we introduce a novel service that can be offered on top of classical streaming media services to the users of these mobile handheld devices. Considering that the wireless infrastructure is heavily loaded by the traffic of the real-time stream itself, we propose a distributed caching solution to offload it, improving the scalability of the system. We define two heuristic algorithms to place the caching elements within the network in an optimal way – the goal was either to minimize the required number of caching nodes or the size of the cache. In our earlier work we have investigated some individual aspects covered by the scenario and built a testbed that implements these proposals. In this paper we present the extension of this integrated testbed suitable for network coding based communication, which supports the measurement based evaluation of the presented scenario and the evaluation of the implementation details. Finally we offer a measurement based assessment of the features implemented in the testbed.

Keywords: network coding, video streaming, caching.

1. Introduction

The economic success and the market share of Android devices [1] led to the situation where a significant part of the population carries a plentiful of sensors and short-range communication devices with them. These general purpose devices have an unprecedentedly high computational power, allowing the execution of complex operations in the background. The smartphones also offer the possibility to experiment with a wide variety of wireless networking issues. In our earlier work we have started to develop a generic testbed environment, as introduced in [2] and [3]. In this testbed the Android devices may act as harvesters (mobile data collectors), communicating nodes and real-time media endpoints in distributed wireless networks. We primarily used this testbed to research the applicability of network coding techniques in sensor networks and on forwarding streaming video over wireless links [3].

In [2] we also presented several scenarios that motivated our research of network coding. The scenarios cover wireless sensor networking (WSN - [4]) communication and distributed wireless communication research issues, network coding application alternatives, real-time media streaming and communication management. For the details of the respective scenarios please refer to [2].

In this paper we extend the scenarios, introducing a novel service that can be offered on top of a classical streaming media service. The scenario focuses on crowded events, when lots of users follow the same live content. The new service is the replay of recent highlight of this real event. Considering that the wireless infrastructure is heavily loaded by the traffic of the real-time stream itself, we propose a distributed caching solution to offload it, improving the scalability of the system. We will discuss the details of such novel service and will present a model that will allow us to analyze its behavior later in Section 3. Although media distribution might use multicast in order to use efficiently the network resources, most of the devices and applications still use unicast. Therefore in the following we will focus only on unicast solutions. Because the scenarios are used as motivations for our research, which are validated by testbed experiments, the testbed development was a natural follow up of the scenario definition and analysis. We implemented both the media streaming and network coding support in our testbed that can be used to demonstrate the presented scenarios and the test implementation details. The details of this work are presented in Section 5.

As explained in this paper, the implementation of this scenario is based on network coding mechanisms, already used in earlier scenarios. Network coding (NC) mechanism re-codes packets of flows within the network at various nodes, promising overall higher throughput and/or reliability [2], [5]. Here we only shortly review the main advantage of the NC, using the widely used butterfly topology (see *Fig. 1*): "if the packets of a stream are distributed over parallel paths (flows *a* and *b*) and packets are encoded together using network coding, the original packets can be restored at the destination. This approach can be used to increase the network capacity. Let us assume that both *a* and *b* flows require the same bandwidth. In *Fig. 1* the common link carries the encoded *a+b* flow instead of carrying both *a* and *b* flows. Now, using NC techniques, the required capacity of *a+b* is the same as for the individual *a* (or *b*) flow. Thus we can achieve a 50% cost saving over the common link in terms of bandwidth capacity” [2].

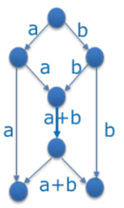

Figure 1: Network coding in a network with butterfly topology.

In our work we follow an alternative (Random Linear Network Coding - RLNC - [6]), where nodes assign coefficients to each packet randomly. All nodes using random network coding are independent and randomized, without the need of any knowledge of the rest of the network, assuring high quality transmission [7]. Early works focused on the case when the network is populated by large enough number of flows (inter-flow NC) [8], then also considered the intra-flow case [9]. Later works then further extended the area of applicability [10], [11], [12].

2. Caching of media streams – related work

The traffic volume of streaming media had exceeded that of any other traffic type, including peer-to-peer or web access and researchers tried to reduce its bandwidth demand by various methods, including caching. There is a vast available literature in this field. In what follows we highlight the most important ones.

Early caching strategies focused on the deployment of a dedicated proxy close to the consumers, and a god overview of these solutions divided them in four categories: sliding-interval caching, prefix caching, segment caching, and rate-split caching [13]. The sliding-interval caching [14] stores data at the first appearance of it (as it is arriving from the server), and delivers it to the subsequent requests. The prefix caching [15] always stores the first segments of the media, thus the proxy can immediately serve any request and starting to fetch the rest of the data in the meantime. The segment caching [16], [17], [18] offers a more advanced solution, because it assigns utilities to the segments, and caches them accordingly. The rate-split caching [19] breaks with the time-based segmentation of the stream and splits it along the rate. The lower rate component may arrive directly from the server, while the rest of the data, ensuring the premium quality of the service, uses only local networking resources.

The advent of the peer-to-peer (P2P) networks influenced this research area, as well. Several P2P based solutions were proposed: some of them are more generic, as rely on the logical P2P overlays [20], or ones that use the specific architecture of the access network and cannot be directly applied in other networks [21].

As Content Delivery Networks (CDN) gained in popularity, researchers applied the streaming video caching mechanisms on this technology, too [22]. Similarly, caching in clouds is a promising research direction [23]. Nevertheless, both directions require heavy investments in the infrastructure. Moreover, in our scenario the problem is the overload of the access, which connects the cloud to the users, thus we still have to address this issue. The combination of network coding methods with P2P solution is a noteworthy novelty [24], but in this paper we focus on a specific service, not supported in earlier papers.

Finally we mention the new networking paradigm, the Information Centric Network, which has an advanced caching mechanism [27], integrated with its novel data forwarding mechanism. The challenging research issues gave birth to several interesting papers [25], [26] involving multicast scenarios, but our work deals with classical IP unicast networks.

The reader interested in further details of streaming video caching is directed to a thorough overview of this field [28].

3. Near-real time streaming supported by caching

A. Basic Scenarios

Before presenting our new scenario we summarize first the basic scenarios that motivated our work, because it leads to a better understanding of our testbed and also introduces the reader to Section 3.B. For a detailed description the reader is redirected to [2]. The scenarios built on the fact that the cheap short-range wireless devices (RFID and NFC tags) favor the deployment of large distributed wireless sensor networks (WSN) [4]. Nevertheless, the collection of the sensory data from such large deployed networks will be both an economical and a logistic issue. We proposed to use the ubiquitous Android smartphones to do this task and envisaged a cooperative wireless mesh to convey the sensory data to a management center. Apart of sensory data, the smartphones had to forward live video and control traffic, both required to manage and control the wireless nodes. The network coding was the technology of choice to improve the reliability and performance of the communication over the wireless mesh.

In this paper we build on the basic scenarios, reusing the ideas of forwarding live streaming video streams. Although we reuse many building blocks developed during our earlier work, this new scenario can be described and analyzed as a standalone one. Note that this common background will be more evident when we describe our testbed, as mentioned later in Section 5. As a consequence the following sub-section contains the definition of the scenario itself, along with the motivation behind it.

B. Near real-time streaming service

In their quest to gain competitive advantage on the travel and leisure market, many cities organize large public events: city festivals or thematic events (e.g., concerts, art festivals, sporting events). In parallel with this trend more live events are being consumed via the web or smart phones rather than broadcast; the “second screen” is gaining prominence and people are multi-tasking on their devices; attendees of live events continue using their smart devices (e.g. for social media). Current access networks are hard pressed to provide the required QoS.

This scenario aims at providing various mechanisms to support the goal of increasing QoE of live media streaming while at the same time decreasing the network load. The target service is for (near) real-time streaming media distribution. Note that participants at events often want to see replays of the key moments or re-watch some recent performance, often only a few seconds after it happened. We build on the trend that attendees of live events will use their smart devices during the event – for email, social media and, more and more, for recording and consuming video.

The load in the wireless access is decreased by the use of network coding and stretching the lifetime of the network encoded packets somewhere in the network distributed in end devices or well-placed points in the distribution/ access domain. The cached distribution primarily is implemented on the user's devices.

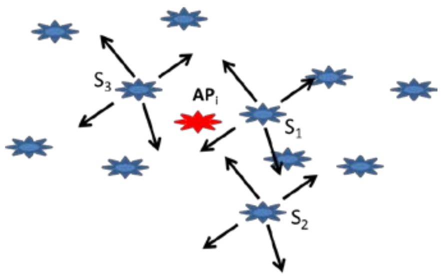

Figure 2: Media streaming with caching.

This scenario is depicted in *Fig. 2*. The streaming media is distributed by the AP_i Access Points. The nodes receiving the data encode it and cache it locally (S_1, …). Any time a node (blue stars) want a replay, they will have the data chunks readily available in their local mesh network.

The above scenario, although seems a quite limited one at first glance, it proves to be a complex one. The complexity comes from the fact that it uses many technologies and during the analysis of the scenario we have to deal both with the technical details to be solved in order to implement the scenario and with the interactions among these particular technologies. In our earlier work we have investigated some individual aspects covered by the scenario.

In our earlier paper [2] we evaluated candidate radio technologies to support the scenarios. In [3] we investigated the aspects of distributed streaming video communication in our testbed, focusing on the performance of a simplified logical ring that formed a peer-to-peer overlay over the Android devices within the testbed. In [5] we presented a simulation based performance evaluation of caching strategies. In this paper we add new elements to these series of results, addressing the particularities of this scenario introduced above. We propose two alternative algorithms that offer an optimized solution to our scenario. Then in Section 5 we present an extension of our earlier testbeds that can support the measurement based evaluation of our scenario.

4. Proposed caching model optimization

In this section we take a closer look at two alternatives to optimize the cache placement within the scenario presented in Section 3.B. The optimization problem is not trivial, because not all nodes should necessarily act as caches and even those that do, do not necessarily store every packet in their cache memory. The starting point is the all the nodes are attached to one AP_i from the set of a set of $\{AP_i\}$ Access Points and the covered area is large enough to make sure that nodes cannot communicate directly with everyone within the whole network. We also consider that the nearest caching nodes and cache hit statistics can be obtained by the requesting nodes [5].

A. Minimizing the number of caches

In the first model we search for the number of caching nodes to serve those nodes that are attached to the same AP and we allow to store unlimited number of segments in the cache. We can give the required number of caching nodes $|S_i|$ as the rounded up value of the total size of required data chunks (g_w segments are required for D_i users) divided by the direct link capacity c:

$$|S_i| = \left\lceil \frac{D_i g_w}{c} \right\rceil \tag{1}$$

Note that this solution does not allow that a node attached to AP_i to request data from a caching node attached to AP_j. *Our goal is to get the minimal number* *of caching nodes, a subset of all N nodes*. c_i is the total capacity of the direct link from node i (we consider that the total incoming and outgoing capacities are equal). In [5] we gave a 0-1 Integer Linear Programming (ILP) to solve this optimization problem, but this type of ILPs are known to be NP hard [29].

Therefore we propose the following heuristic algorithm (see *Table 1*).

Table 1: The cache placement algorithm.

```
Foreach $AP_i$
    Compute $|S_i|$ based on eq. (1)
    $X_i$ := arbitrary selection of $|S_i|$ nodes from the vicinity of $AP_i$
$S = \bigcup X_i$

Foreach $AP_i$
    Partition all the nodes with demands based on distance from the coverage
    area $AP_i$ among $X_{ij}$
    $C_{ij}$ = total expected demand on $X_{ij}$

Do
    Flag = 0
    Foreach $X_{ij}$
    If $C_{ij}$ < Capacity($X_{ij}$) then
        Flag = 1
        Elect an additional X from the vicinity
            Re-partition all the nodes with demands based on distance from
            the coverage area $AP_i$ among $X_{ij}$
Until Flag == 0
```

The above heuristic algorithm takes all APs and makes an initial selection of caches as given in (1) for the simplified scenario. Then it refines this selection, introducing new caching nodes in those areas, where the existing ones cannot answer all the demands. If there are no more unanswered demands, the algorithm stops.

B. Minimizing the cache size

In Section 4.A we tried to reduce the number of caching nodes, without restricting the individual cache sizes. *Now we will try to minimize the size of the cache, but we do not restrict the number of caching nodes*. Let us keep the same notations we introduced earlier in this section, and additionally let us note the number of chunks *stored* at node i with k_i. We proposed and ILP in [5] to solve this optimization problem, but the same reasoning presented in Section 4.A applies here, as well. Therefore we provide a heuristic algorithm that solves this problem. First we give the number of chunks a node should store, if the chunks are evenly distributed among all nodes attached to the AP_i. We keep the notation used for eq. (1), and we note with N_i the total number of attached nodes.

$$k_i = \left\lceil \frac{D_i g_w}{N_i} \right\rceil \tag{17}$$

The algorithm is given in *Table 2* below.

Table 2: The cache placement algorithm.

```
Foreach $AP_i$
    Compute $k_i$ based on eq. (17)
$k$ = $\min_i (k_i)$
Store $k$ chunks in each node

Foreach leacher $l$
    Assign the closest nodes (caches) to it
        Can't assign more than $c_l$
    $g_l$ = total available chunks at these caches

Do
    Flag = 0
    Foreach leacher $l$
    While $g_l < g_w$ do
        Flag = 1
        $k_l$++ for the caches linked to $l$
            //this  can be done in parallel – increase the cache only once
            //during a single round
Until Flag == 0
```

The algorithm computes the minimal number of chunks that should be available at the nodes (i.e., caches, since every node is a potential cache) in order to satisfy the lowest demand at any AP. Then we find those leachers, which have unsatisfied demand and increase the cache size by one for its caches. The algorithm stops, if there are no new unsatisfied leachers left.

5. Video streaming environment with Android smartphones

In this section we present a testbed to support the evaluation of the proposed scenario and algorithms presented in this paper.

A. Testbed for streaming video services

Earlier in this paper we have already motivated our testbed development work by the need to obtain a platform suitable to assess our scenarios. The common point in our scenarios is the presence of a mobile node capable of communicating over IP. This node was implemented as a remote controlled vehicle, where the computational and communication functions are executed by an Android smartphone, implementing an extended distributed data network. The interface between the Android device and the motors moving the vehicle is done by a dedicated microcontroller board, the IOIO [30].

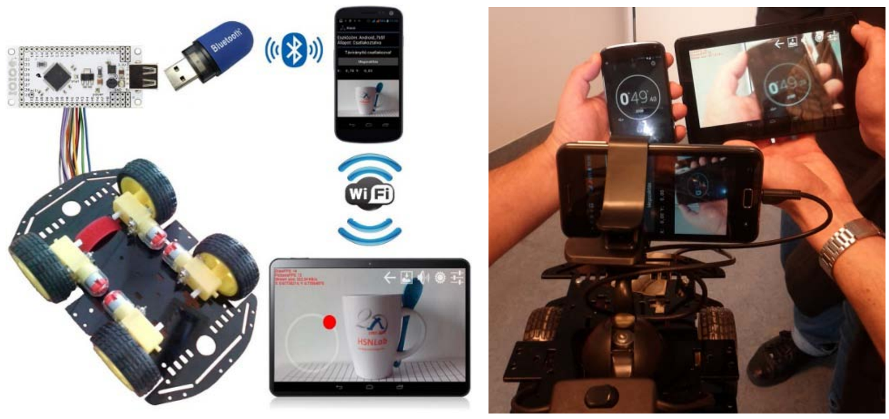

Figure 3: a) The elements of the mobile node and b) the implemented vehicle.

Fig. 3. a) presents the components of the implemented mobile node. The vehicle and the driving motors are on the left bottom, the IOIO board is one the upper left part of the figure. The Android smartphone (pictured on the upper right part of the figure) is linked to the IOIO board by Bluetooth (but we can also use USB cables for this purpose). This device is also responsible with the control tasks. A central management entity (bottom right corner) has a suitable GUI to control the movement of the node (on the downlink direction) and is connected to it over variants of WiFi (see Section 5.C for further details). As a part of the control process, live video streams are sent from the mobile node to the control station (i.e., uplink), the implementation details of this streaming service being discussed later in Section 5.B. The implemented mobile node and the management station is pictured in *Fig. 3.* b).

B. Streaming media over Android

The multimedia stream in the testbed is offered by an application implemented from scratch by us.

A natural choice would have been the use of Real-time Transport Protocol (RTP) for such task [31]. RTP based streaming in Android is supported by the *libstreaming* library. The encoding for RTP transmission should be done using H263 codecs instead of H264, since the latter has higher resource demands. For decoding at the destination an external library can be used (we used vlc player, which was able to decode the stream [32]). Due to the above Android environment details (e.g., the delays introduced by encapsulation and control, video handling difficulties) we decided to go for different solution.

As a first option, in order to capture the stream from the camera of the smartphone, we use the camera preview images. Then these raw images are sent to an image encoding library, and this encoded JPEG image, which corresponds to a frame of the video stream, is sent over UDP to the recipient(s). This solution makes it easier to handle the frames at the receiving peer and debug our implementation: any disturbance in the encoding/decoding process is limited to a well identifiable frame, not to a sequence of frames. The fragmentation of the video frames and the header structure of the packets have been designed to be able to carry multiple streams and/or support multiple networks encoding along the path. In our first tests we checked the viability of our solution.

A second option was to use other, older encoders developed for TV transmission in a selfcontaining mode. E.g., MPEG-2 Transport Stream (MPEG-2 TS, M2TS) has been introduced in ISO/IEC 13818-1 standard in 1995 [33] . It has the advantage that video format info is in-band. Most of Android smartphones support the playout of such streams, but the encoding of such streams officially is not supported. Some forums reported that that in some devices with Android OS above version 3.0 have unofficial support for this format (e.g., Google Nexus 4). We used this smartphone to implement and test our solution.

Fig. 4. shows the logical structure of our solution. The live feed from the camera is encoded by the Media Recorder in real time. Instead of saving this stream locally, it is written into a socket, which is used to send the stream over the Internet. On the receiver side the ideal playout method would have been to feed the read socket directly to the player, but the available players require a seekable source. Therefore the socket is read in a loop, temporarily stored in a file and the player plays the video from this file. The file is continuously filled from the socket and read by the players. We just have to make sure that the new video content is filled fast enough into the file as the player never reaches the end of the file. We made some experiments and found that a 200ms buffer is enough for this goal.

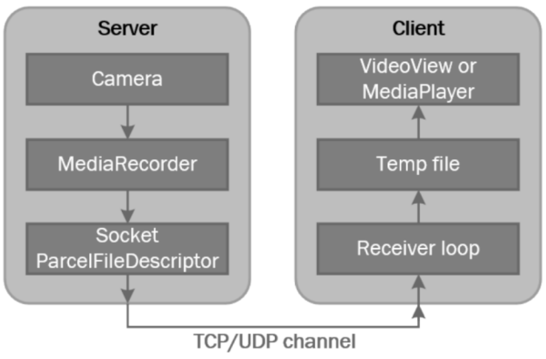

Figure 4: Client-server architecture of the MPEG-2 TS implementation.

We compared our two solutions and we found that there is a tradeoff on quality vs delay. The JPEG based solution has lower delays (avg. 300ms), but during fast moves it is blurred, while the MPEG-TS based solution offers usable picture during fast moves, too, but it has higher delays (avg. 500ms).

Finally, in *Table 3* we summarized the measured performance of the video transfer in our testbed. The first row corresponds to our first implementation, when we forward the series of JPEG still pictures captured at the source. Then we show the RTP-based solution, finally the MPEG2-TS based implementation. As we can see, the packet delays are extremely high in the case of the RTP based solution, while the other two provide similar results, with a clear advantage on the JPEG-based implementation. Note that for the MPEG based implementation we did not have packet-level statistics. The required bandwidth was similar for the first two solutions, but it came at the price of lower Quality of Experience (QoE). The highest disadvantage of the JPEG based solution was the low fps (frame per second) value. The QoE of the MPEG based solution outperformed both previous solutions.

Table 3: The performance of tested video transfer solutions.

Techno-logy of choice	Delay [ms]	Avg. delay [ms]	Band-width [kilo Bytes/ sec]	Avg. BW [kilo Bytes/ sec]	Avg. video q. [fps]	Still picture quality (QoE)	Moving picture (video) quality (QoE)
JPEG	170-440	290	100-300	170	15	Blurry	Blurry
RTSP	1090-1570	1300	100-200	150	20	Good	Checkered
MPEG-2 TS	-	500	-	250	-	Excellent (sharp)	Rarely checkered

C. Communication between the mobile nodes

The Android smartphone attached to the vehicle is able to control the vehicle, to collect sensory data and to capture live streams. Nevertheless, in order to exploit these capabilities we have to assure the delivery of the data to the management center. We proposed scenarios where this is achieved through a cooperative mesh between Android smartphones (see the testbed in *Fig. 5.*).

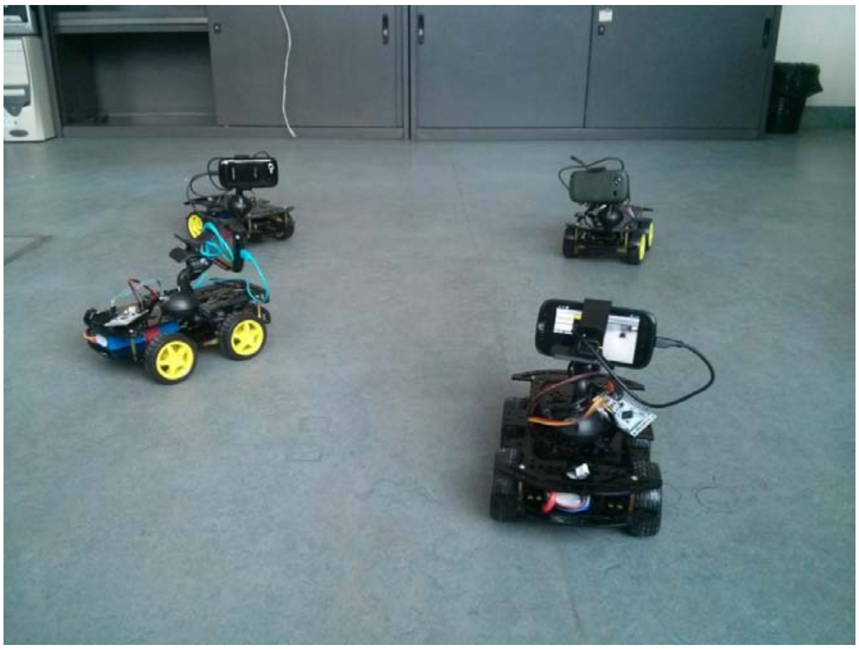

Figure 5: Mobile nodes in the distributed testbed.

The cooperative mesh is a logical construct at layer 3. In order to implement it in the testbed we used several WiFi variants. The Android SDK assures a complex API to handle WiFi related aspects [34]. Unfortunately, the WiFi ad hoc mode is not supported by Android. They introduced WiFi Direct [35] (earlier also known as WiFi P2P) instead, designed to provide seamless device-to-device WiFi connection. We implemented our testbed using this technology, the measurement results being generated with this setup. Nevertheless, we also used the infrastructure mode WiFi communication in our testbed and focus on the logical overlay to emulate distributed communication scenarios [3]. [2] introduces our RLNC implementation, and is not affected from the choice of which WiFi alternative is used.

As the proposed scenario is based on the network coding of streaming data, we evaluated the performance of the testebed by means of measurements. We used the unsuccessful display attempts during the playout process to assess the performance of the implementation. This is a complex parameter, aggregating the events of unsuccessful decoding attempts and the de-synchronization events. This latter happens when the encoded packet streams arriving over two branches are skewed, thus the decoder cannot get enough coded segments. Note that this event may be caused by the delays caused by a slow wireless link (e.g., slowed down by interference, resent packets, etc.).

Since decoding is a computationally intensive process, we tested devices from two different generations (Google Nexus S and Nexus 4). *Table 4* presents the average results obtained after 10 experiments. We present both the encoding and decoding performance. Note that in all cases the QoE of the played video was specific to the JPEG streaming solution (i.e., it allowed to successfully control the mobile node in real-time).

Table 4: The performance of network coding of media streams.

Smartphone model	Encoding loss rate [%]	Decoding loss rate [%]
Google Nexus S	4.8	19.1
Google Nexus 4	4.1	11.5

We can see that the encoding process has significantly lower impact. Also note that the older device has only slightly lower performance than the Nexus 4. During the decoding phase the Nexus S cannot keep the pace, the performance gap between the two becoming significantly larger. A practical conclusion of these measurements is that current state-of-the art Android devices (which are 3 years newer by design compared to the Nexus 4) provide enough resources to support real-time RLNC decoding.

6. Conclusions

The spread of both WiFi capable Android devices makes feasible the design and implementation of general purpose distributed heterogeneous data networks. The high processing capacity of the smartphones makes them ideal to experiment with streaming video distribution and related services. In this paper we presented a novel streaming video service, and a method using network coding based caching to offload the wireless infrastructure. We proposed a heuristic algorithm to place the caching elements within the network.

We have built a prototype testbed based on Android smartphones which will serve as a demonstration platform for our proposal. We have implemented a mobile node, the live streaming video solution over the Android system and the random linear network coding based communication between the Android devices. We measured the performance of the implemented technologies. We showed that the implementation of RLNC based streaming video is feasible in modern Android devices. In our future work we plan to analyze the different aspects of caching strategies in our proposed scenario.

Acknowledgements

Csaba Simon’s research work was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4.A/1-11-1-2012-0001 National Program of Excellence (NKP).

Markosz Maliosz’s research work was carried out as part of the EITKIC_12-1-2012-0001 project, which is supported by the Hungarian Government, managed by the National Development Agency, financed by the Research and Technology Innovation Fund and was performed in cooperation with the EIT ICT Labs Budapest Associate Partner Group (www.ictlabs.elte.hu).

References

[1] Google Android developers guide – available online from: http://developer.android.com/guide/components/index.html

[2] Soos, A., Simon, Cs., Maliosz, M., “Network Coding Based Data Collection in Distributed Sensor Networks”, *The 4th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics, MACRo 2013*, Târgu Mureş, Romania, 2013.

[3] Zalatnay, Zs., Simon, Cs., Maliosz, M., Terza, B., “Managing streaming services in a distributed testbed”, *The 5th International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics, MACRo 2015*, Târgu Mureş, Romania, March 2015.

[4] Akan, O. B., Isik, M. T., Buyurman, B., “Wireless passive sensor networks”, *IEEE Communications Magazine*, vol. 47, nr. 8, pp. 92-99, 2009.

[5] Simon, Cs., Maliosz, M., “Network Coding Based Caching for Near Real-Time Streaming Media”, To appear in: *Infocommunications Journal: a publication of the Scientific Association for Infocommunications (HTE)*, vol. 7, nr. 1, pp. 7-14, 2015.

[6] Li, B., et al, “Random network coding in peer-to-peer networks: From theory to practice”, 2011.

[7] Gajic, B., Riihijrvi, J., Mhnen, P., “Performance evaluation of network coding: Effects of topology and network traffic for linear and xor coding”, *Journal of Communication*, vol. 4, nr. 11, pp. 885-893, 2009.

[8] Chachulski, S., and Katti, S., “Trading structure for randomness in wireless opportunistic routing”, in *Proc. of ACM SIGCOMM*, 2007.

[9] Katti, S., Katabi, D., Hu, W., Rahul, H., and Medard, M., “The importance of being opportunistic: Practical network coding for wireless environments”, in *Proc. 43rd Annual Allerton Conference on Communication, Control, and Computing*, 2005.

[10] Gkantsidis, C., and Rodriguez, P. R. “Network coding for large scale content distribution”, in *Proc. of IEEE INFOCOM 2005*, vol. 4, 2005.

[11] Traskov, D., Lenz, J., Ratnakar, N. and Médard, M., “Asynchronous Network Coded Multicast”, in *Proc. of ICC Communication Theory Symposium*, 2010.

[12] Zhang, X., Neglia, G., Kurose, J., "Network Coding in Disruption Tolerant Networks, Network Coding: Fundamentals and Applications", Elsevier Science (Ed.), 2011.

[13] Liu, J., and Xu, J., "Proxy caching for media streaming over the Internet", *IEEE Communications Magazine*, vol. 42 nr. 8, pp. 88-94, 2004.

[14] Tewari, R., Vin, H. M., Dan, A., and Sitaram, D. "Resource-based caching for Web servers", in *Proc. of SPIE/ACM Conf. on Multimedia Computing and Networking* (*MMCN'98*), San Jose, CA., 1998.

[15] Sen, S., Rexford, J., and Towsley, D., "Proxy prefix caching for multimedia streams", in *Proc. of IEEE INFOCOM'99*, New York, NY, 1999.

[16] Wu, K. L., Yu, P. S., and Wolf, J. L., "Segment-based proxy caching of multimedia streams", in *Proc. of WorldWideWeb Conference* (WWW10), Hong Kong, 2001.

[17] Miao, Z., and Ortega, A., "Scalable proxy caching of video under storage constraints", *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1315-1327, September 2002.

[18] Fahmi, H., Latif,M., Sedigh-Ali, S., Ghafoor, A., Liu, P., and Hsu, L., "Proxy servers for scalable interactive video support", *IEEE Computer*, 43(9): 54-60, 2001.

[19] Zhang, Z.-L.,Wang, Y., Du, D., and Su, D., "Video staging: A proxyserver-based approach to end-to-end video delivery over wide-area networks", *IEEE/ACM Transactions on Networking*, vol. 8 nr. 4, pp. 429-442, 2000.

[20] Zhang, M. et al., "A Peer-to-Peer Network for Live Media Streaming - Using a Push-Pull Approach", in *Proc. of ACM Multimedia*, 2005.

[21] Kőrösi, A., Székely, B., Császár, A., Lukovszki, Cs., "High quality P2P-Video-on-Demand with download bandwidth limitation", in *17th IEEE International Workshop on Quality of Service*, Charleston, USA pp. 1-9, July 2009.

[22] Adhikari, V. K., et al., "Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery", in *Proc. IEEE INFOCOM*, 2012.

[23] Wu, Y., et al.,"CloudMedia: when Cloud on Demand Meets Video on Demand", in *Proc. IEEE ICDCS*, 2011.

[24] Wang, M., Li, B., "R2: Random Push with Random Network Coding in Live Peer-to-Peer Streaming", *IEEE Journal on Selected Areas in Communications*, 2007.

[25] Wong, W., Wang, L., and Kangasharju, J., "Neighborhood search and admission control in cooperative caching networks", in *IEEE Global Communications Conference* (GLOBECOM), pp. 2852-2858, 2012.

[26] Montpetit, M.-J., Westphal, C., and Trossen, D., "Network coding meets information-centric networking", in *ACM MobiHoc workshop NOM'12*, pp. 31–36, June 2012.

[27] Sourlas, V., Gkatzikis, L., Flegkas, P., and Tassiulas, L., "Distributed cache management in information-centric networks", *IEEE Transactions on Network and Service Management*, vol. Early Access Online, 2013.

[28] Li, B., Wang, Z., Liu, J., and Zhu, W., "Two decades of internet video streaming: A retrospective view", *ACM Transactions on Multimedia Computing, Communications and Applications* (TOMCCAP), 9(1s), 33. 2013.

[29] Karp, R. M., "Reducibility Among Combinatorial Problems", in *Proc. Sympos. Complexity of Computer Computations*, IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y. New York: Plenum, pp.85-103, 1972.

[30] IOIO library basic documentation, https://github.com/ytai/ioio/wiki/IOIOLib-Basics, Sparkfun 2012.

[31] RTP: A Transport Protocol for Real-Time Applications, IETF RFC 3550, available from: http://tools.ietf.org/html/rfc3550.

[32] Official page of the VLC media player, available form: https://www.videolan.org/vlc/index.html.

[33] International Telegraph Union - Telecommunication Standardization Sector, Recommendation H.222.0 (07/95), available from: http://www.itu.int/rec/T-REC-H.222.0-199507-S/en.

[34] WiFi package for Android, available from: http://developer.android.com/reference/android/net/wifi/package-summary.html.

[35] Wi-Fi Alliance, Wi-Fi Direct, available from: http://www.wi-fi.org/discover-and-learn/wi-fi-direct.