Synchronized Multimedia Streaming on the iPhone Platform with Network Coding

Vingelmann, Peter; Fitzek, Frank; Pedersen, Morten Videbæk; Heide, Janus; Charaf, Hassan

Published in:
I E E E Communications Magazine

DOI (link to publication from Publisher):
10.1109/MCOM.2011.5783997

Publication date:
2011

Document Version
Early version, also known as pre-print

Synchronized Multimedia Streaming on the iPhone Platform with Network Coding

Péter Vingelmann, Budapest University of Technology and Economics and Aalborg University
Frank H. P. Fitzek, Morten Videbæk Pedersen, and Janus Heide, Aalborg University
Hassan Charaf, Budapest University of Technology and Economics

ABSTRACT

This work presents the implementation of synchronized multimedia streaming for the Apple iPhone platform. The idea is to stream multimedia content from a single source to multiple receivers with direct or multihop connections to the source. First we look into existing solutions for video streaming on the iPhone that use point-to-point architectures. After acknowledging their limitations, we propose a solution based on network coding to efficiently and reliably deliver the multimedia content to many devices in a synchronized manner. Then we introduce an application that implements this technique on the iPhone. We also present our testbed, which consists of 16 iPod Touch devices to showcase the capabilities of our application.

INTRODUCTION

Multimedia content distribution has received a lot of attention lately in the mobile world. New ways to convey multimedia content to mobile devices are discussed after the recent failure of DVB-H and DVBM. Besides the bare technology there is also the question how mobile users are looking at multimedia content. So far the main architecture was designed such that the overlay network with its highly centralized architecture is providing the content, and the mobile users are consuming it. But more and more users are starting to generate and collect their own content that they would like to distribute among each other in local area networks.

Apart from this discussion in this work we investigate the possibility of sending multimedia content from one device to many devices in closer proximity. In our previous work we have shown that it is possible to share photos and audio files among mobile devices even across different platforms [1]. Therefore, in this work we address mobile video as the next logical step.

Synchronized video playback can be used among friends to show their latest videos to each other. Exchanging music videos is especially interesting at social events if everybody can play them at the same time. Another fascinating application is for home entertainment: we can deploy a simple server that broadcasts a live video stream (e.g. a sporting event) that is accessible on every mobile device in the household.

In this article we not only present a mobile application supporting the described use cases, we also advocate the use of network coding in order to address the channel characteristics of wireless networks and the limited energy of mobile devices.

SHORTCOMINGS OF EXISTING SOLUTIONS

There are several applications that can stream multimedia content to the iPhone, for example AirVideo and TVersity. Basically these applications run a webserver to which the iPhone media player can connect. A TCP connection is established and the player issues standard HTTP range requests, then the webserver sends raw file data with HTTP headers in response. This approach has the clear drawback that with an increasing number of receivers the bandwidth of a given cell or access point will become the bottleneck due to the use of unicast connections.

In order to prove this, first we used a single iPod Touch to connect to an AirVideo server running on an iMac to play a video that was previously transcoded to a suitable format for the iPhone platform (Xvid and AAC codecs with an overall data rate of 500 kb/s). The video playback was fine, so this approach is sufficient for a single device playing a single media file. As the next step, we tried the same experiment with five iPod Touches as receivers, and we often experienced stuttering in the video playback. We also monitored the network traffic with Wireshark running in promiscuous mode.

The server sending rates in the two experiments are shown in Fig. 5 as captured by Wireshark. We observed that the overall throughput reached 16 Mb/s in a few seconds, and it remained this high for several hundred seconds. We can conclude that the video playback on the iPod Touches was not satisfactory due to the insufficient incoming data. Thus if we connect to the same server with multiple devices at the same time, we can quickly saturate the wireless network. This is an inherent drawback of unicast connections.

If we intend to efficiently deliver the same content to several devices, multicast transmissions provide a favorable solution. In [2] it was shown that multicast video streaming is feasible on the iPhone platform with network coding. The limitation of this work was the lack of synchronized playback and the reliance on jailbroken components on the iPhone. Moreover, the architecture design was limited to point-to-multipoint communication, whereas network coding implies the possibility of recoding packets at the intermediate nodes. This feature is particularly useful if the source and the receiver do not have a direct link, that is we have a multihop network.

Scenario and Solution

In our scenario a server *S* wants to reliably transmit the same media file to several nearby receivers, $t_1, t_2, \ldots, t_N$, via a wireless link. This basic scenario is depicted in Fig. 1. As mentioned earlier, the traditional point-to-point data distribution paradigms (i.e., unicast transmissions) provide poor utilization of the available network resources for one-to-many services. Since all receivers are interested in the same content, we can efficiently utilize the wireless channel with broadcast transmissions.

Under ideal channel conditions all broadcast packets are delivered to all nodes simultaneously. In real-life wireless networks packet losses frequently occur [4]; thus, some sort of retransmission is necessary to ensure reliability (i.e., to correct packet losses at the receivers). A simple solution would be that the individual nodes request all missing packets from the original source. This would imply that every lost packet is transmitted again, and if packet losses are uncorrelated, most retransmissions will not be useful to many receivers since they have received those packets in the first place. To put it differently, it is likely that a single retransmission will only benefit a single receiver.

A shrewd way to maximize the impact of each retransmission is by using network coding [3, 4]. Researchers have shown that network coding can provide several advantages: improved throughput, robustness, security, and lower complexity in communication networks [5].

Network Coding

Network coding differs from channel or source coding, because it is not limited to end-to-end communication, but allows on-the-fly recoding of information whenever needed. Another important fact is that network coding breaks with the store-and-forward policy of existing communication systems. It has been widely accepted that in packet-based communication networks, all packets that enter a node will also leave the node in one way or another (packet drops due to buffer overflow are neglected for ease of illustration). In contrast, a network coding-enabled communication node is able to recode incoming data to tailor it to the needs of the outgoing channels. Network coding was introduced by Ahlswede *et al.* in [3] for fixed networks, and it was adapted by other researchers for wireless and mobile networks [6].

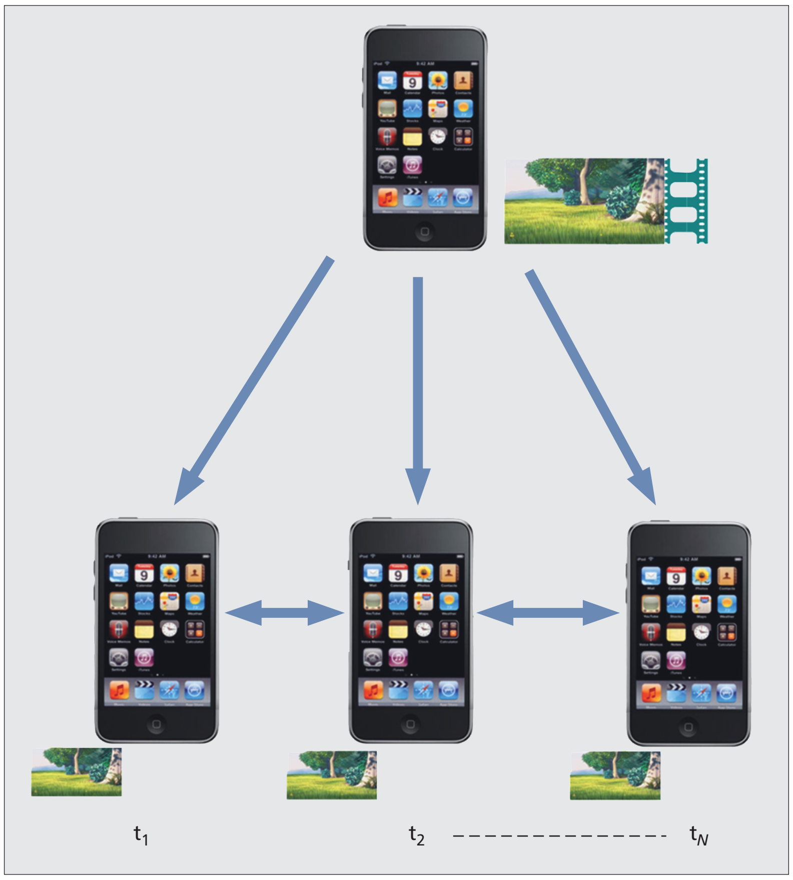

Figure 1. *A server* S *transmitting data to* N *receivers* $t_1, t_2, \ldots, t_N$.

Figure 2 gives a basic overview of the operations performed in a network coding system. If we intend to encode a large file, it should be split into several chunks, also called generations, each consisting of *g* packets [4]. Otherwise, the computational complexity of the encoding and decoding operations would be prohibitively high.

The top component in Fig. 2 is the encoder that generates and transmits linear combinations of the original data packets in the current generation. Addition and multiplication are performed over a Galois field; therefore, a linear combination of several packets will have the same size as a single packet. Note that any number of encoded packets can be generated for a single generation. The middle layer in this system is the wireless channel, where packet erasures may occur depending on the channel conditions. The network nodes receive a series of encoded packets that are passed to the decoder (the bottom component in the figure), which will be able to reconstruct the original data packets after receiving at least *g* linearly independent packets.

Recoding is an additional operation of net-

An obvious benefit of using network coding is that a network node is no longer required to gather all data packets one by one; instead, it only has to receive enough linearly independent encoded packets. This is important for our scenario.

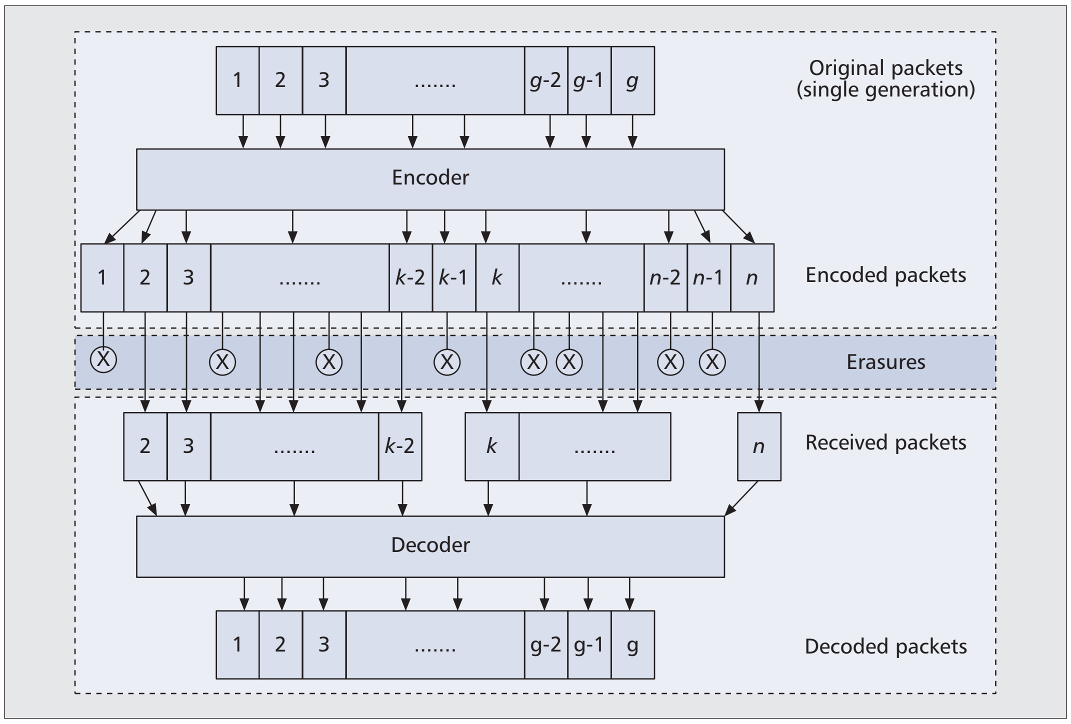

Figure 2. *Overview of network coding.*

work coding that is not directly shown in this figure. Recoding means that all network nodes are allowed to generate and send new encoded packets (i.e., new linear combinations of the packets they have previously received).

An obvious benefit of using network coding is that a network node is no longer required to gather all data packets one by one; instead, it only has to receive enough linearly independent encoded packets. This is important for our scenario since we can simultaneously serve multiple nodes with a single transmission by sending a linear combination instead of choosing a specific packet.

As does any other coding operation, network coding involves computational overhead, which might be prohibitive from a practical point of view. The authors in [7] proposed an efficient solution for mobile devices. We can use a systematic code to ensure reliability with low overhead. It is not necessary to always send encoded packets while using network coding. Uncoded packets can be considered primitive linear combinations, and thus can be processed by the decoder. A simple and efficient method is to transmit each generation in two stages. In the first stage, the source transmits all packets uncoded. Each of these packets will contain new useful information for the individual receivers. In the second stage, the source will generate and send random linear combinations of the original data in order to correct packet losses, which have occurred during the first stage. Note that a single encoded packet can potentially correct different losses at different nodes. With this approach we can maximize the number of nodes for which a packet is useful.

In order to illustrate the advantage of network coding for the envisioned use case, we show Fig. 3, where the leftmost mobile device would like to share a video file with three other devices in close proximity. One of the three receivers has a larger distance to the originating device. Based on the distance, the packet error rate differs significantly. Here we assume that the first-tier neighbors have a 10 percent loss rate, while the second-tier device has a packet error rate of 50 percent. Those values are reasonable and were reported in [8]. If communication is only allowed on the blue links (i.e., the originating device is transmitting packets), the overall time until all devices receive the file depends primarily on the second-tier neighbor.

If we enable multihop relaying (the red links in Fig. 3), the first-tier neighbors can also try to forward packets to the second tier. The problem is that the relaying devices need to be coordinated in order to not convey redundant information (i.e., to prevent the same packet being relayed twice). Here network coding enables the relaying devices to recode previously received packets in such a way that redundant information is minimal.

IMPLEMENTATION

This section discusses the implementation of the application based on the ideas outlined above. Our primary target platform is the iPhone, where the Objective-C language is mandatory for graphical user interface (GUI) development, but we chose to write most of our application in C++ in order to facilitate its porting to other mobile platforms. GCC 4.2 is the default internal compiler in the Xcode development environment, and it can be used to compile C++ source files in Objective-C++ mode and link with the generated object files. Consequently, the GUI that has to be written in Objective-C can call regular C++ code, and a high degree of platform independence can be achieved. Note that we used iPhone OS v. 3.1.3 for development.

STREAMING

When the streaming server starts, it enumerates its network interfaces in order to find out its IP and broadcast address corresponding to the wireless network interface.

Upon user request it opens the selected video or audio file, then determines its media type and overall size. The packet payload size is set to 1024 bytes and generation size is 64. Upon creating a new data stream, the server reads a data chunk (64 kbytes) from the file to fill the input buffer for the first generation. It also calculates the total number of generations based on the overall file size. Then it begins to send uncoded packets and some metadata with a specified data rate, which is slightly higher than the native data rate of the media file. All packets are sent to the broadcast address of the wireless interface. After sending 64 uncoded packets, the server transmits several encoded messages in order to repair all packet losses of the individual receivers. The actual number of these extra encoded packets can be adjusted based on the current network conditions. Of course, this approach would require periodic feedback from several receivers. A simple solution is to always add a large overhead (e.g., 50 percent) to combat packet losses even under the worst conditions. After sending a specific number of encoded packets, the server moves on to the next generation. It fills its input buffer with a new data chunk that is read from the input media file. Then it begins to broadcast uncoded (and later encoded) packets for the current generation, and this process continues until we reach the end of the input file.

Note that the streaming server can easily be integrated into the client application to enable users to stream multimedia content from their mobile devices.

PLAYBACK

The most important question is how to play the incoming media stream on the iPhone. The built-in media player (an instance of the MPMoviePlayerController class) can play a local file in the application bundle or open an HTTP network stream at a given URL. We intend to initiate playback when only a small part of the entire file is received, and this media player cannot play an incomplete file because it tries to buffer up a significant amount of data in the beginning. Using any other media player is not recommended by Apple, so the only solution is to run a web server on localhost that can continuously feed the incoming data into the player itself.

The media player issues standard HTTP range requests, which will be processed by the embedded web server in our application. Each range request is served by a new POSIX thread. The range in the first request is always 0-1, which means the first 2 bytes of the file. The response will contain the size of the whole file, and the media player uses this information and the media file header to issue further range requests to the web server. It can decide to close the current socket before the actual range request is fully served, and the web server must be able to detect this behavior in order to avoid sending packets that will not be processed by the player.

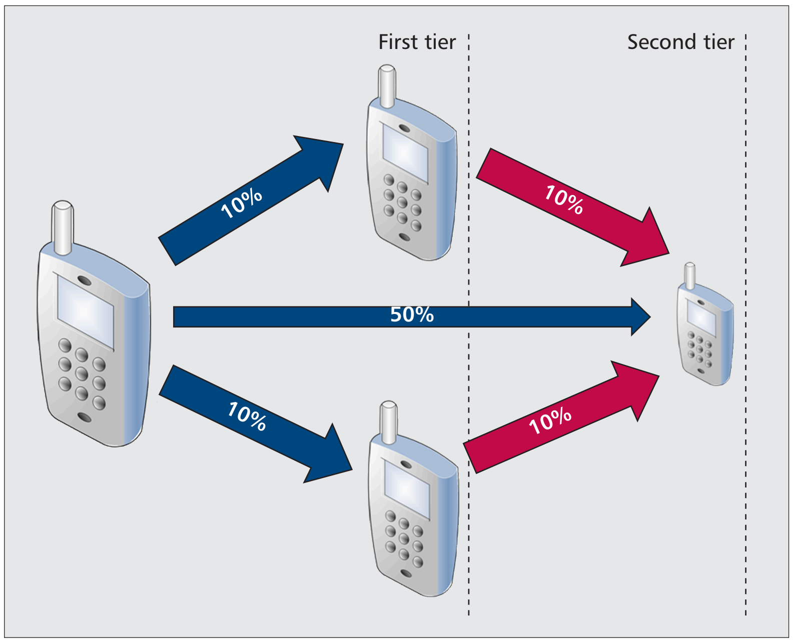

Figure 3. *Using network coding in a two-tier topology.*

The web server feeds the media player with raw data of a generation when it is completely decoded. Thirty seconds after transferring all bytes from a generation, it will be considered obsolete, and its data buffer will be deleted from memory. This way we can avoid memory leaks, which would quickly accumulate in our application during the playback of long media files.

The built-in media player can only play video files that use the QuickTime (.mov) container format and H.264 or Xvid video codecs with a suitable resolution for the iPhone/iPod Touch. Therefore, the usual AVI videos must be transcoded before streaming. This can be performed with open-source tools like ffmpeg.

The media player is configured with the control mode option Volume Only so that users cannot seek arbitrary offsets in the media stream. It is important to point out that our application has no control over the media player component after the playback is started. The player cannot be paused or stopped; there are only two callback functions that signal the host application when the media file is preloaded and when the playback is finished.

Starting the player imposes a significant load on the CPU, and during this period (1–2 seconds) a large number of incoming packets are lost on the receivers. A simple way to overcome this issue is to stop the streaming server for 2 seconds after sending out the very first packet. This will give time for the built-in media player to properly load up, and no packets will be lost on the client devices.

Note that the video and audio codecs are able to recover if some dirty data is transferred to the player. This can happen under extreme channel conditions, when the overhead sent from the server is not enough to repair all packet losses of the receivers. In this case we typical-

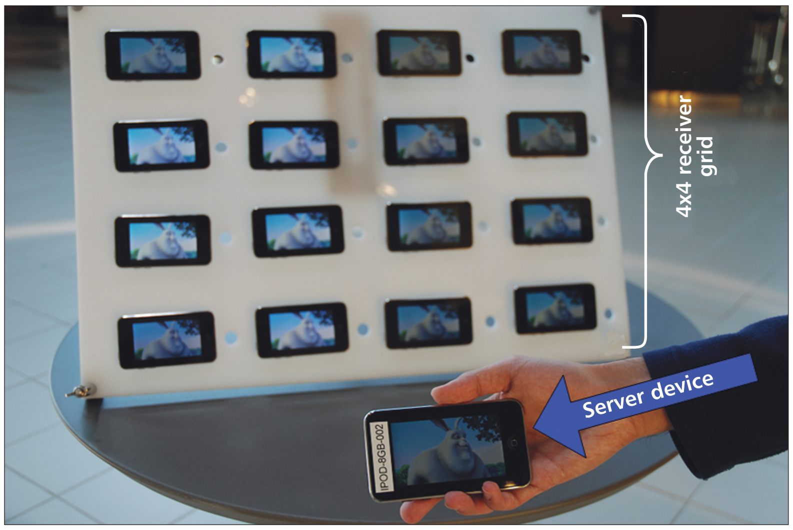

Figure 4. *The testbed consists of 16 third-generation iPod Touches, and the video is streamed from another iPod Touch.*

Processor	ARM CORTEX-A8
Clock rate	833 MHz (underclocked to 600 MHz)
Memory	256 MB DRAM
Flash memory	32 or 64 Gbytes
Display	320 × 480 px, 3.5 in
Playback time	Video: 6 hours, audio: 30 hours

Table 1. *Technical specifications of the 3rd generation iPod Touch.*

ly observe green boxes on screen and hear some audio glitches, but the playback continues and the errors disappear when channel conditions return to normal.

SYNCHRONIZATION

The playback synchronization among multiple devices is based on the assumption that every client device decodes each generation at approximately the same time. Slight variations are possible due to the differences in propagation and processing delay and due to different packet erasures. The cumulative effect of these phenomena translates to a variation that is always smaller than 0.05 seconds. On the other hand, the startup of the media player can cause much bigger deviations.

As mentioned before, a generation cannot be fed to the player until it is completely decoded. Each generation is protected by a mutex that is initially locked by the main application thread. When the generation is decoded the mutex is unlocked, and the threads that are currently serving the HTTP range requests can proceed with reading the raw data of the decoded generation.

It is crucial that every player starts the playback at the same time. If a player starts later than the others, it will not be able to catch up, even though it decodes all subsequent generations at the right time. On the other hand, if a player starts early, it will resynchronize with the others (i.e., it will be stalled), since it cannot decode generations earlier than the others.

The synchronization can be further improved by performing explicit clock synchronization among the individual receivers. For example, all receivers can synchronize with the server's clock, and specific timestamps can be used to precisely control the moment when the generations are unlocked. Nevertheless, such high precision is not required in our setup, and its implementation would involve additional overhead.

TESTBED

We have assembled a testbed to demonstrate the capabilities of our application. The receiver grid consists of 16 third-generation iPod Touch devices, and can be seen in Fig. 4. The iPod Touch technical specifications are shown in Table 1.

For demonstration purposes, we chose the 10-minute-long "Big Buck Bunny" animation movie that was released under the Creative Commons Attribution 3.0 license. The original HD movie was transcoded to a resolution of 480 × 320 pixels using the Xvid codec for video (bit rate: 372 kb/s) and the MPEG-4 AAC codec for audio (bit rate: 128 kb/s). The transcoded video can be played on the iPhone/iPod Touch using the built-in media player with a smooth playback and fine quality. In [1] the full-blown testbed is shown while playing this video using our application.

The iPod Touches were connected to an ad hoc wireless network that had been created using a Nokia N95 mobile phone, as the iPhone OS does not support the creation of ad hoc networks. The video stream was sent from another iPod Touch that was running our application in server mode. Although the streaming server has been integrated into our application, unfortunately it cannot access the video and audio files stored on the iPhone due to Apple's limitations. An application can only access files within its own application bundle. At the moment, the only solution is to store all media files we want to share in the application bundle.

In Fig. 5 we present the bandwidth usage of our streaming server compared to the unicast solutions mentioned earlier. All plots were generated by Wireshark during the playback of the same video file. As seen on these plots, the bandwidth usage is constant and never exceeds 100 kbytes/s, which is significantly lower than the others. Note that adding additional receivers will not have a negative impact on the performance of our application, since we only use minimal feedback from the clients. Moreover, the low bandwidth usage infers that streaming multiple videos at the same time might be a feasible idea. In theory, 10 simultaneous video channels can be broadcast if we assume a net throughput of 10–16 Mb/s in a typical wireless network.

FUTURE WORK

The application can be extended in the future to support user cooperation and multihop ad hoc networks.

The number of extra encoded packets sent by the server can be significantly reduced by using cooperation among the receivers. Assuming that the packet losses are uncorrelated, the cooperating devices can exchange missing packets with each other; thus, the server can send less overhead. It has been shown in [9] that using network coding in such a cooperative cluster is an efficient way of realizing packet exchange. The clients simply broadcast recoded packets (for the current generation), which most likely convey information that is useful to the other receivers in case of uncorrelated packet erasures. However, if we implement this approach, precautions must be taken to avoid massive collisions in the network. We can use carefully chosen backoff timers in the client applications to solve this issue. For example, these timeouts can be adjusted so that the receiver with the most packet losses is given priority to request missing packets from the others. Then the node with the most knowledge will reply first with a series of recoded packets.

So far we have only considered single-hop ad hoc (IEEE 802.11b/g) networks, where we have the convenience of the MAC layer performing a fair division of available channel capacity if multiple nodes are sending packets and can sense each other. In general, this is not true in multihop networks where the hidden node problem is responsible for many collisions. Request/clear to send (RTS/CTS) acknowledgment and handshake packets cannot be used for multicast transmissions; therefore, all nodes should follow the same cooperative protocol to facilitate efficient data dissemination.

The fundamental problem in multihop networks is that some nodes are not directly reachable by the source. Delivering the data stream to all receivers is only possible if some nodes, called relays, propagate the received data to other nodes that are farther away from the source. A relay node helps in the dissemination process by generating and transmitting re-encoded packets. Dynamically selecting these relays in an ad hoc network is not a trivial problem, as was shown in [10]. Currently the application is capable of generating and forwarding recoded packets. A severe limitation is that the physical data rate of broadcast transmissions is fixed to 1 Mb/s on the iPhone platform, so receiving and sending packets with a data rate close to 1 Mb/s (which is typical for a video) is not possible. At the moment, our solution is sufficient for propagating an audio stream (having a bit rate of 128 kb/s) in a linear multihop network, where the nodes are positioned to form a virtual line. But as the network topology becomes more complex or even dynamic, the nodes will broadcast a spate of packets if we continue to use this approach. This phenomenon was called the broadcast storm problem in [11].

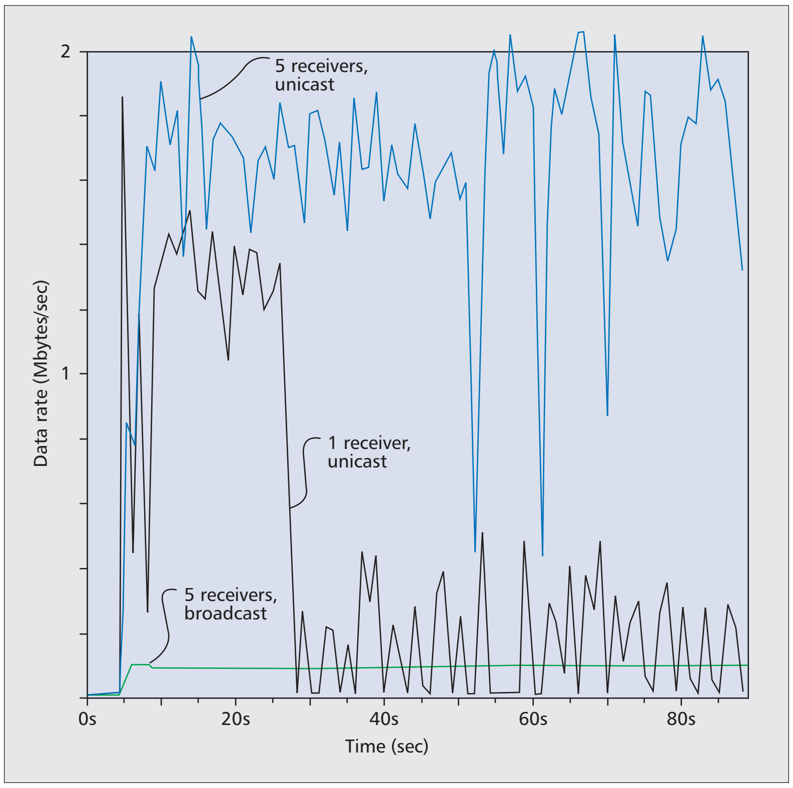

Figure 5. *AirVideo server sending rates with a single receiver (black) and with five receivers (blue) using TCP unicast connections in comparison with the bandwidth usage of our streaming server (green) using UDP multicast transmissions with network coding.*

CONCLUSION

In this article we have introduced a way to disseminate multimedia content in a synchronized manner. We propose a method based on network coding to efficiently deliver data from a single source to many receivers. An application running on Apple iPhone/iPod Touch devices has been presented to show the feasibility of this approach. We observe that the bandwidth usage of this application is remarkably low in comparison with other existing solutions. In [1] the full-blown testbed was shown while playing this video using our application. The first commercial implementation of this technology can be found at [12].

ACKNOWLEDGMENTS

This work was partially financed by the CONE project (Grant No. 09-066549/FTP) of the Danish Ministry of Science, Technology and Innovation as well as by collaboration with Nokia/Renesas, Oulu throughout the ENOC project.

REFERENCES

[1] Mobile Device Group of Aalborg Univ., Network Coding Videos, http://bit.ly/abcOrC, June 2010; collection of YouTube videos.
[2] H. Shojania and B. Li, "Random Network Coding on the iPhone: Fact or Fiction?," *ACM NOSSDAV 2009*, June 2009.
[3] R. Ahlswede *et al.*, "Network Information Flow," *IEEE Trans. Info. Theory*, vol. 46. no. 4, July 2000, pp. 1204–16.

[4] C. Fragouli, J.-Y. Le Boudec, and J. Widmer, "Network Coding: An Instant Primer," *SIGCOMM Comp. Commun. Rev.*, vol. 36, no. 1, 2006, pp. 63–68.
[6] S. Katti *et al.*, "Xors in the Air: Practical Wireless Network Coding," *SIGCOMM Comp. Commun. Rev.*, vol. 36, no. 4, 2006, pp. 243–54.
[7] J. Heide *et al.*, "Network Coding for Mobile Devices — Systematic Binary Random Rateless Codes," IEEE Wksp. Cooperative Mobile Networks at ICC'09, June 2009.
[8] J. Heide *et al.*, "Know Your Neighbour: Packet Loss Correlation in IEEE 802.11b/g Multicast," *4th Int'l. Mobile Multimedia Commun. Conf.*, Oulu, Finland, July 2008.
[5] T. Ho and D. Lun, *Network Coding: An Introduction*, Cambridge University Press, 2008.
[9] F. H. P. Fitzek and M. Katz, Eds., *Cooperation in Wireless Networks: Principles and Applications — Real Egoistic Behavior Is to Cooperate!*, Springer, Apr. 2006.
[10] P. Vingelmann, F. H. P. Fitzek, and D. E. Lucani, "Application-Level Data Dissemination in Multi-Hop Wireless Networks," *IEEE ICC 2010 — CoCoNet Wksp.*, May 2010.
[11] S.-Y. Ni *et al.*, "The Broadcast Storm Problem in A Mobile Ad Hoc Network," *ACM/IEEE MobiCom '99*, New York, NY, 1999, pp. 151–62.
[12] Steinwurf ApS, http://www.steinwurf.com

BIOGRAPHIES

PÉTER VINGELMANN (peter.vingelmann@gmail.com) received his M.Sc. in technical informatics from Budapest University of Technology and Economics in 2009. He is currently pursuing a Ph.D. in wireless communication at the same university. Since January 2009 he has been working with the Mobile Device research group at Aalborg University, Denmark. His main research interests are programming, mobile networks, mobile applications, cooperative communication, network coding, and network performance evaluation.

FRANK H. P. FITZEK is a professor in the Department of Electronic Systems, Aalborg University, heading the Mobile Device group. He has visited various research institutes including Massachusetts Institute of Technology (MIT), VTT, and Arizona State University. In 2005 he won the YRP award and received the Young Elite Researcher Award of Denmark. He was selected to receive the NOKIA Champion Award five times in a row from 2007 to 2011. In 2008 he was awarded the Nokia Achievement Award for his work on cooperative networks. In 2011 he received the SAPERE AUDE research grant from the Danish government. His current research interests are in the areas of wireless and mobile communication networks, mobile phone programming, cross-layer as well as energy-efficient protocol design, and cooperative networking.

MORTEN VIDEBÆK PEDERSEN received his B.Sc. in electronics engineering in 2007 and M.Sc. in wireless communication in 2009, both from Aalborg University. He is currently pursuing a Ph.D. in wireless communication from Aalborg University. Since January 2006 he has been working in the Mobile Devices research group at Aalborg University, where his primary focus has been on implementation and performance evaluation of network coding algorithms, and cooperative networking protocols and methods. He has co-authored and published several peer-reviewed journal and conference papers, and multiple book chapters. In 2010 he was appointed Forum Nokia Champion. His main research interests are mobile programming, cooperative communication, network coding, and network performance evaluation.

JANUS HEIDE received his M.Sc. in electrical engineering with specialization in wireless communication engineering in 2009 from Aalborg University. He is currently a Ph.D. Fellow within the research section of Antennas, Propagation, and Radio Networking (APNET) in the Department of Electronic Systems at Aalborg University. Since July 2007 he has been working in the Mobile Devices research group at Aalborg University. His main research interests are protocol analysis and design, wireless network systems, characteristics and models, cooperative communication, network coding, and data distribution in ad hoc and meshed networks.

HASSAN CHARAF came to Hungary in 1987 to study at the Electrical Engineering Faculty of Budapest University of Technology and received his M.Sc. degree in 1992, and later on his Ph.D. degree in 1998. He also earned an M.Sc. degree in economics at Budapest University of Economics. Presently he is an associate professor at the Department of Automation and Applied Informatics of Budapest University of Technology and Economics, where he is head of the IT Group. He holds a number of positions in professional bodies: Microsoft Regional Director in Hungary (1998–2004), IEEE Computer Society, John von Neumann Computer Science Society, Automation and Computer Science Committee of the Hungarian Academy of Sciences, Informatics Committee of the Hungarian Academy of Sciences, and Electrotechnics Committee of the Hungarian Academy of Sciences. He is the author of several books and lecture notes.