MicroCast: Cooperative Video Streaming on Smartphones

Lorenzo Keller
School of I&C
EPFL, Lausanne, CH
lorenzo.keller@epfl.ch

Anh Le
CS Dept & CalIT2
UC Irvine
anh.le@uci.edu

Blerim Cici
Networked Systems & CalIT2
UC Irvine
bcici@uci.edu

Hulya Seferoglu
LIDS
MIT
hseferog@mit.edu

Christina Fragouli
School of I&C
EPFL, Lausanne, CH
christina.fragouli@epfl.ch

Athina Markopoulou
EECS Dept. & CalIT2
UC Irvine
athina@uci.edu

ABSTRACT

Video streaming is one of the increasingly popular, as well as demanding, applications on smartphones today. In this paper, we consider a group of smartphone users, within proximity of each other, who are interested in watching the same video from the Internet at the same time. The common practice today is that each user downloads the video independently using her own cellular connection, which often leads to poor quality.

We design, implement, and evaluate a novel system, MicroCast, that uses the resources on all smartphones of the group in a cooperative way so as to improve the streaming experience. Each phone uses simultaneously two network interfaces: the cellular to connect to the video server and the WiFi to connect to the rest of the group. Key ingredients of our design include the following. First, we propose a scheduling algorithm, MicroDownload, that decides which parts of the video each phone should download from the server, based on the phones' download rate. Second, we propose a novel all-to-all local dissemination scheme, MicroNC-P2, for sharing content among group members, which outperforms state-of-the-art peer-to-peer schemes in our setting. MicroNC-P2 is designed to exploit WiFi overhearing and network coding, based on a local packet broadcast framework, MicroBroadcast, which we developed specifically for Android phones. We evaluate MicroCast on a testbed consisting of seven Android phones, and we show that it brings significant performance benefits without battery penalty.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*

Keywords

Video Streaming, Wireless Networks, Smartphones, Network Coding

MobiSys'12, June 25–29, 2012, Low Wood Bay, Lake District, UK.

1. INTRODUCTION

Today's smartphones are equipped with significant processing, storage and sensing capabilities, as well as wireless connectivity through cellular, WiFi and Bluetooth. They provide ubiquitous Internet access, primarily through their cellular connection and secondarily through WiFi, and enable a plethora of new applications. Among those applications, video (including consuming and creating/posting video content) is increasingly popular. However, meeting the growing demand for high quality video is currently a challenge in cellular networks.

Indeed, cellular traffic is growing exponentially (tripling every year), with a share of video traffic increasing from 50% now to an expected 66% by 2015 [7]. Credit Suisse reported in [13] that 23% of base stations globally have utilization rates of more than 80 to 85% in busy hours, up from 20% last year. This dramatic increase in demand poses a challenge for 3G networks, which is likely to remain in 4G networks as well. Furthermore, the data rate of the cellular connection may fluctuate over time (*e.g.,* throughout the day); the service loss rate can be as high as 50% [36]; and coverage can be spotty depending on the location and user mobility.

In this paper, we are interested in the scenario where a group of smartphone users, within proximity of each other, are interested in watching the same video at the same time. Watching the video on one phone screen is not comfortable for more than two people; users may also prefer to watch the video on their own screens. The default operation today is that each user with a cellular connection downloads the video independently from the server. However, each phone's individual cellular connection may not be sufficient for providing high video quality.

Consider, for example, that a user wants to show to her friends a YouTube videoclip while being in a bar; a group of friends who want to watch a live soccer match together on their phones while at a remote location (*e.g.,* a camping or skiing site); or a family who wants to watch a movie on their phones in the train or in the car; or a group of co-workers who want to watch a lecture using WiFi at a busy hotspot, such as the company cafeteria or a conference room. In all these cases, some or all of the users may have poor or intermittent cellular connectivity, depending on the coverage of their providers, or may face congestion in the local network (*e.g.,* when they use WiFi to download).

Fortunately, because the users engage in a group activity, this scenario naturally lends itself to cooperation. Furthermore, when every phone has multiple parallel connections (*e.g.,* 3G, WiFi, and Bluetooth), there are even more available resources that, if properly used, can further improve the user experience.

A special case of the general scenario is when the video content is stored locally on one of the phones and the user wants to share it

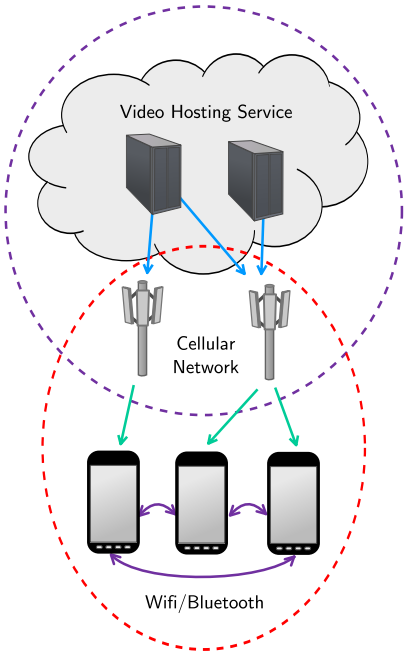


Figure 1: *MicroCast scenario.* A group of smartphone users, within proximity of each other, are interested in watching the same video at the same time. Each smartphone connects to the video source (*e.g.,* UStream, YouTube, or Netflix) using its cellular (3G or 4G) connection. The base station may be the same or different for different users, depending on the provider they use. Each smartphone can receive packets from the source as well as from other smartphones in the neighborhood through device-to-device (*e.g.,* WiFi) links.

with the other members of the local group. For example, a teacher wants to share a video with the students in a class; or a group of friends want to watch a video recorded of a group activity stored on one of the phones. In these cases, cooperation can also help.

We propose a novel cooperative scheme, called *MicroCast*[1], for video streaming to a group of smartphones within proximity of each other. Each phone utilizes simultaneously two network interfaces: one (cellular) to connect to the video server and download parts of the video; and the other (WiFi) to connect to the rest of the group and exchange downloaded parts. Our scheme optimizes the usage of the cellular links to ensure that all the available bandwidth is used when channel conditions are good so as to compensate for potential long periods of bad channel conditions. It also optimizes the dissemination in the local area so as to ensure that even in presence of heavy interference from other networks, the phones can still collaborate. In our scheme, each phone downloads the video much faster than it is played out when the conditions are good, in order to reduce the likelihood of buffering during the playback when the radio conditions significantly deteriorate, *e.g.,* it downloads at 2 Mbps or more on current cellular networks videos that are typically encoded at 750 Kbps.

Our system consists of the following key components.

[1]"Micro" indicates locality: there is a small number of users and they are all within proximity of each other. "Cast" indicates a multicast scenario: all users in the group are interested in the same content sent by a single source, and that we use local broadcast.

I. *MicroDownload:* This is a simple yet effective scheduler that decides which parts of the video each phone should download from the server and distribute on the local network. The decision is based on feedback from each phone and takes into account the cellular download rate. Compared to prior work, *e.g.,* [20], [29], [25], [26], [38], where each phone tries to download all the data and the local links are used for error recovery, our scheme *jointly* utilizes two interfaces for data delivery. Compared to cooperative mobile systems for offloading cellular networks, *e.g.,* [17], [14], [42], our goal is to improve the 3G downlink utilization of a group of smartphones. Compared to [35], our scheme tries to maximize the rate at which the content is downloaded to prevent buffer underflow, which leads to freezing of the video.

II. *MicroNC-P2:* This is a novel all-to-all dissemination scheme for locally sharing content among group members within proximity of each other. It leverages the combination of network coding and WiFi overhearing (provided by the MicroBroadcast mechanism, described below) to efficiently share the downloaded parts of the video locally. We show that, in our setting, MicroNC-P2 significantly outperforms state-of-the-art P2P schemes, including BitTorrent [8] and the network coding-based R2 [41]. This is because MicroNC-P2 is explicitly designed to exploit the capabilities of the underlying MicroBroadcast to introduce less redundancy than the previous algorithms [41]. MicroNC-P2 is not only a key component of our MicroCast system but also a standalone contribution that can be used independently, *e.g.,* for local all-to-all dissemination.

III. *MicroBroadcast:* We develop a component that enables high rate packet broadcast, building on WiFi of Android phones. Although the wireless medium has inherently the broadcast capability, this cannot be fully exploited using standard WiFi broadcast, due to various problems, including low rates. lack of reliability etc. Pseudo-broadcast on WiFi ad-hoc mode, which was previously used in [18], cannot be used on Android phones, as we discuss later. To the best of our knowledge, MicroBroadcast is the first Android-based system to fully exploit the potential of broadcast. MicroBroadcast is a contribution on its own right and also a necessary component for the high performance of the entire MicroCast system: *e.g.,* MicroNC-P2 exploits its capabilities.

We implement a prototype of our system as an application for Android phones. An attractive feature of our architecture is that it is modular: one can change each component (*e.g.,* the link layer, the algorithms for local distribution or download, the streaming protocol, etc.) independently of the others. We plan to make the application and its related libraries publicly available [1]. In this paper, we present first the system design and then the performance evaluation results from experiments on a testbed consisting of seven Android phones. We evaluate each of the three components separately as well as the combined system as a whole and we demonstrate that there are significant performance benefits (in terms of decrease download time and increase per-user download rate) compared to alternative approaches, without significant battery cost. A video demonstration and supporting materials can be found in [1].

The rest of the paper is structured as follows. Section 2 presents related work. Section 3 describes the MicroCast architecture. Section 4 presents the MicroDownload and MicroNC-P2 algorithms and their interaction. Section 5 discusses implementation details, including challenges specific to Android phones and design choices

made to address them. Section 6 presents performance evaluation results on our Android testbed for a range of scenarios. Section 7 discusses limitations and extensions. Section 8 concludes the paper.

2. RELATED WORK

Cooperative Mobile/Wireless Systems. When several users are interested in the same content and they are in proximity of each other, some of them may be able to use device-to-device connections, *e.g.,* through WiFi or Bluetooth, to get the content in a cooperative and/or opportunistic way. Opportunistic device-to-device communication is often used for the purpose of offloading the cellular network. For instance, [17], [14], and [42] consider a scenario in which device-to-device and cellular connections are used to disseminate the content, considering the social ties and geographical proximity. Instead of offloading cellular networks, our goal in this paper is to use cellular and local connectivity so as to essentially allow each user to enjoy the aggregated downlink rate.

Furthermore, cooperation between mobile devices for content dissemination or in delay tolerant networking, possibly taking into account social ties [16, 3], has extensively been studied. However, dissemination of content stored on a mobile device is only a special case of our framework, which uses only the local links, but not the downlinks. More importantly, we exploit single-hop broadcast transmissions, as opposed to multi-hop peer-to-peer communication that exploits mobility (at the expense of delay, which is crucial in our setting), but ignores broadcast.

The idea of using multiple interfaces of mobile devices has been used before but not in the same way as in this paper. For example, [37] effectively exploits cellular and WiFi interfaces simultaneously to create multiple paths to mobile devices. [34] improves the mobile networking experience using concurrent WiFi connections from multiple WiFi hot-spots. [28, 5] exploit the diversity of multiple interfaces on the same device to achieve better connectivity. Instead, we use the cellular connections of multiple smartphones to improve the download rate and jointly utilize local connections. [2, 35] address the same problem as we do, and use a similar strategy. However, they use only unicast communication among peers, as opposed to our (micro)broadcast. Therefore, they are more sensitive to local bandwidth fluctuations.

The recently announced Android Ice Cream Sandwich (Android 4.0) devices provide peer-to-peer (P2P) connectivity using WiFi Direct [43]. WiFi Direct is an industrial standard that prompts WiFi-equipped devices to establish ad-hoc peer-to-peer connections, while maintaining the infrastructure mode connection to the Internet. This is achieved by having one of the devices acts as a *virtual AP* to provide infrastructure to the rest. In MicroCast the mobile devices create P2P connections in a similar way, but in our case, we try to reduce the number of frames (in the channel) required for each data packet exchanged between the peers. This is done by "pseudo-adhoc", which is part of the MicroBroadcast component. The core idea of MicroBroadcast is to exploit the broadcast nature of the wireless channel, while the core idea of WiFi Direct is to offer ad-hoc connectivity to mobile devices.

Network Coding in Cooperative/Wireless Systems. Cellular links (3G or 4G) as well as WiFi suffer from packet loss due to noise and interference. One possible solution to this problem is to have several devices in a close proximity help each other with retransmissions of lost packets. Network coding is particularly beneficial as it can make each retransmission maximally useful to all nodes. Rate-distortion optimized network coding for cooperative video system repair in wireless peer-to-peer networks is considered in [21]. Wireless video broadcasting with P2P error recovery is proposed in [20]. An efficient scheduling approach with network coding for wireless local repair is introduced in [29]. [38], [26], and [25] propose a system, in which a group of smartphone users (which are connected to the Internet via cellular links) help each other for error correction, while base stations *broadcast* packets, which is not implemented in current systems; in contrast, we consider unicast (between a base station and a mobile device). A cooperative video streaming system is implemented over mobile devices (PDAs) in [27]. Compared to prior work, *e.g.,* [20], [29], [25], [26], [38, 15], where each phone tries to download all the data, and the local links are used for *error recovery*, our scheme *jointly* utilizes two interfaces, *i.e.,* 3G and WiFi, for data delivery.

Network Coding Implementation. There are WiFi testbeds that implement network coding, such as the COPE testbed [18]. COPE is a practical scheme for one-hop network coding across unicast sessions in wireless mesh networks. [24] proposes a cooperative IPTV system with pseudo-broadcast to improve reliability. Medusa [31] considers a scheme in which multiple unicast flows (video streams) are transmitted from a base station to clients with network coding. This scheme considers rate adaptation and video packet scheduling jointly.

The practicality of random network coding over iPhones is discussed in [32]. A toolkit to make network coding practical for system devices from servers to smartphones is introduced in [33]. [25] implements network coding on mobile devices and presents the performance in terms of throughput, delay and energy consumption. [26] extends [25] for picture transmission. A gesture broadcast protocol is designed for concurrent gesture streams in multiple broadcast sessions in [10] over smartphones using inter-session network coding. [38] proposes a system, in which a group of smartphone users (which are connected to the Internet via cellular links) help each other for error correction. A cooperative video streaming system is implemented over mobile devices (PDAs) in [27].

To the best of our knowledge, our work is the first to implement network coding and overhearing on the Android platform. These tasks are much more challenging on Androids than on laptops [18]. Network coding has been implemented before on other types of phones, [32, 25, 26], although not on Androids, but it has not been combined with overhearing, which is key in our setting.

Network coding for peer-to-peer systems. Network coding has been applied to P2P networks for content distribution [12, 39, 40] and live streaming [41, 22] (an excellent review is presented in [19]). We will show that, in our "micro"-setting, MicroNC-P2 significantly outperforms state-of-the-art P2P schemes, including the widely used BitTorrent [8] as well as the network coding-based R^2 [41]. This is because MicroNC-P2 is explicitly designed to exploit WiFi overhearing and network coding, and to avoid fundamental weaknesses of previous algorithms in our setting, such as the redundancy in R^2.

Network Optimization. Our system design is loosely inspired by the network utility maximization of the problem and its distributed solution [30]. For example, our preliminary analysis showed the benefit of cooperation and broadcast, which motivated the design of MicroNC-P2 and MicroBroadcast. It is worth noting that our experimental results closely match the trend obtained by the theoretical analysis, as explained in Section 6. Our MicroDownload is also inspired by the distributed rate control at [30].

Contributions. The contributions of this work lie (i) in the individual components, namely MicroDownload, MicroNC-P2, MicroBroadcast; each of which is novel, outperforms baselines, and can be used standalone; and (ii) in their combination into the overall MicroCast system. Our MicroCast architecture is modular, thus allowing for swapping various components (*e.g.,* different wireless

technologies, streaming protocols, scheduling and dissemination algorithms). We optimized the mechanisms used by each component, in our particular setting, so as to outperform alternatives and work in a synergetic way. For example, MicroNC-P2 exploits network coding and high-rate broadcast that is made available at the system-level by the MicroBroadcast; MicroBroadcast makes local WiFi broadcast at high rates possible, for the first time, on Android devices; MicroNC-P2 reduces the congestion at the local network, thus simplifying the design of MicroDownload; and cooperation is used jointly on the downlink and local links.

3. MICROCAST ARCHITECTURE

3.1 Setup and Assumptions

We consider the scenario presented in Fig. 1: a group of smartphone users, within proximity of each other, are all interested in downloading and watching the same video at the same time. To achieve this goal, we use cooperation among the users. Furthermore, each phone simultaneously uses two interfaces: the cellular interface (3G) to connect to the server, and the local interface (WiFi) to connect to all other users in the group. Each phone downloads segments of the video from the server and shares these with the rest of the group locally.

We optimized our system under certain assumptions that hold for our setup of interest: First, there is a small number of users (up to 6-7). Second, these users know and trust each other, as it is the case in the motivating examples we provided in the introduction. Third, all users are within proximity of each other and all local links have similar rates on average. This is important as we need to use only single-hop, broadcast transmissions (although we do not need multihop transmissions due to close proximity, we could still incorporate them in our system – as we do in our Bluetooth-based implementation that supports multihop; see Section 7). Fourth, in every phone, we use the cellular connection for the downlink, and WiFi to establish local, device-to-device links. (Alternatives, including WiFi for the downlink or Bluetooth for the local link, are discussed in Section 7.) This has the implication that we can use the two connections (cellular and WiFi) on each phone simultaneously and independently; this would not be possible otherwise, as we discuss later. Note that cellular and WiFi connections are used in parallel, but one connects directly to the server (3G), while the other connects to the other users (WiFi); we do not use both of these connections to connect a user *directly* to the server through two different paths.

3.2 Architecture

We implement a prototype of our system as an application on Android phones, with the architecture shown in Fig. 2. In the remaining of this section, we give an overview of the building blocks and the interaction between them. More specifically, MicroCast consists of the following main components.

MicroDownload runs only on one of the phones (in a group) that initiates the download. It instructs the requesters of all phones which segments to download from the server. The download algorithm is described in Section 4.1.

MicroNC-P2 is responsible for distributing segments using the local wireless network. (We describe the algorithm in detail in Section 4.2.) MicroNC-P2 specifically exploits the broadcast capability provided by its lower layer MicroBroadcast to distribute segments quickly and efficiently.

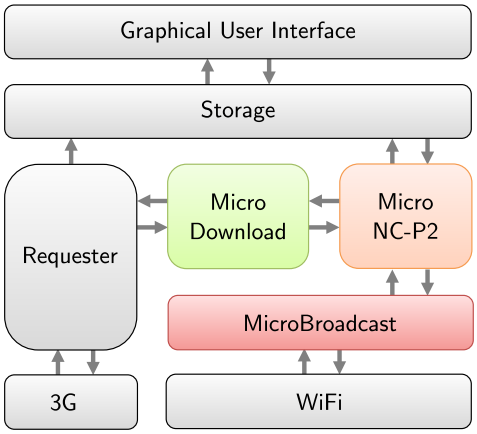


Figure 2: Architecture of MicroCast.

MicroBroadcast implements a comprehensive networking stack, which currently operates on wireless technologies including WiFi 802.11 and RFCOMM Bluetooth. The most important functionality that MicroBroadcast provides is the ability to pseudo-broadcast over WiFi. Nonetheless, MicroBroadcast also supports unicast, reliable and un-reliable message exchange between the network participants over both WiFi and Bluetooth. It also includes multi-hop routing, network-wide flooding, and peer discovery.

Requester retrieves segments of the video from the video source. Notice that depending on the video source location, only a subset of the phones might be able to request segments. For instance, if the video is locally available on one of the phones, only the phone having the file requests segments (from its local memory where the file is stored). If the video is on a remote HTTP server, only the phones with cellular connections request segments.

Storage is used to cache the received segments for successive playback.

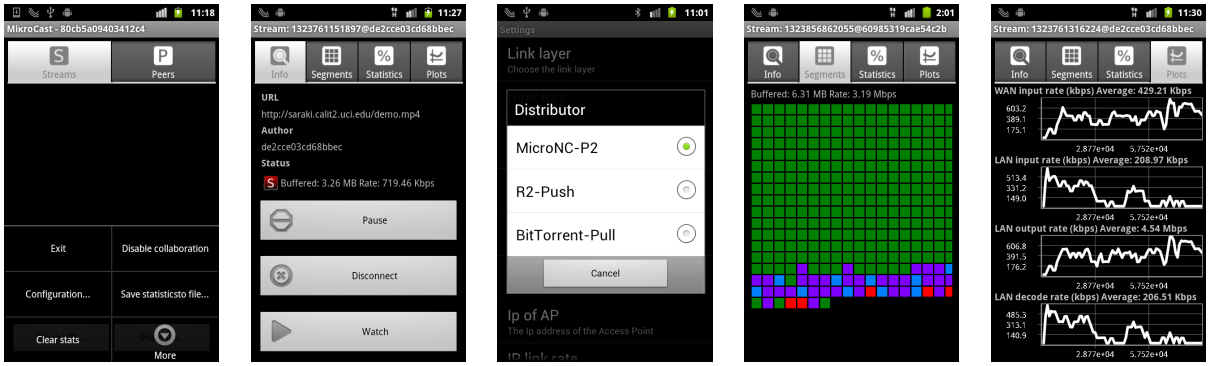
Graphical User Interface (GUI) provides an interface to users to create video streams, share local videos, start/stop downloading video files, join existing video streams, and play/pause video. In addition to these basic features, the GUI lets users discover and connect to other devices, specify the wireless interface that should be used for the local dissemination, and decide whether the phone should collaborate for video downloading. The GUI also allows users to select system modules as well as run-time parameters for experimental purposes. The GUI provides live feedback during the experiments, and provides detailed statistics after the experiments. These functionalities of the GUI facilitate field tests. We provide some screen shots of our GUI in Fig. 3.

4. MICROCAST ALGORITHMS

In this section, we present the algorithms that we chose to implement in each component.

4.1 MicroDownload Algorithm

The video is divided into segments of fixed size. MicroDownload handles the scheduling of which phone should download which segment. Its main idea is that the next segment to be downloaded is assigned to a phone which has the smallest backlog (*i.e.,* the set of segments a phone has to download from its cellular connection). The MicroDownload algorithm is summarized in Alg. 1 and in the following.



(a) Main Options (b) Information (c) Distributors (d) Downloading (e) Statistics

Figure 3: MicroCast Graphical User Interface.

Algorithm 1 MicroDownload Algorithm

```
1:  while there are segments to assign do
2:    Find the phone with the smallest backlog
3:    if the backlog of the phone is smaller than K then
4:      Schedule the phone to download the next segment
5:    else
6:      Sleep until new feedback is received
7:    end if
8:    if feedback from phone indicates a failure then
9:      Schedule the phone to download another segment
10:     Add the segment that failed to the list of segments
11:   end if
12: end while
```

MicroDownload has a list of segments that should be assigned to the phones. Initially, it assigns a fixed number (K) of segments to each phone. The phones try to download one after the other the segments that are assigned to them. If a phone downloads a segment successfully, it notifies MicroDownload. Otherwise, it reports failure. MicroDownload re-assigns the failed segments (based on the backlogs). This mechanism ensures that no segment remains trapped in phones which have bad cellular connectivity. Also, this mechanism adapts segment download to the varying rates and conditions of cellular links. For example, if a phone has a bad cellular connection, the requests being handled by it will be re-scheduled by assigning them to other phones, which hopefully have better connections. However, MicroDownload will still assign some segments to the phone with the bad cellular connection so that it can start downloading immediately as soon as its channel quality improves.

4.2 MicroNC-P2 Algorithm

Each segment downloaded by a phone, is divided into packets, and distributed to the remaining phones using the local network. To do so, we design a custom dissemination scheme that exploits the benefits of overhearing and network coding. At a high level, our scheme takes advantage of pseudo-broadcast, *i.e.,* unicast and overhearing, to reduce the number of transmissions. Furthermore, instead of disseminating regular packets, our scheme disseminates random linear combinations of packets of the same segment, *i.e.,* dimensions of subspaces or network coded packets. This is to maximize the usefulness of overheard packets [11]. We desribe in detail how we use network coding in Section 5.3. As we will show in Section 6.2, existing distribution protocols, such as BitTorrent [8] and R2 [41], do not exploit overhearing and thus have worse performance.

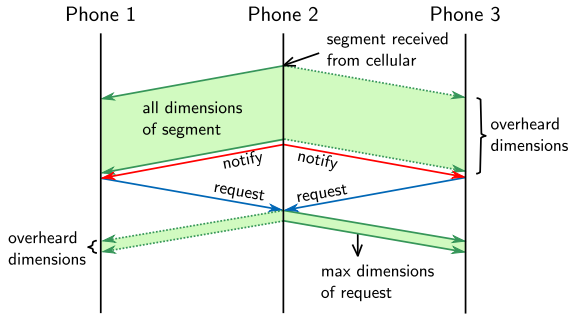
We term our dissemination scheme MicroNC-P2, where P2 refers to an *initial Push* and *subsequent Pulls*. Fundamentally, MicroNC-P2 is built based on traditional pull-based P2P dissemination schemes. In MicroNC-P2, a phone, A, periodically advertises the segments that it currently has to its neighbors. Then, a neighbor, say phone B, requests segments that it does not have based on the advertisement. Upon receiving the request, phone A sends the requested segments to phone B. More specifically,

- When phone B requests a segment s from phone A, it takes into account previously overheard dimensions of the subspace representing segment s. In particular, it explicitly indicates in the request how many additional dimensions it needs to receive to decode s. This reduces the number of dimensions to be sent.

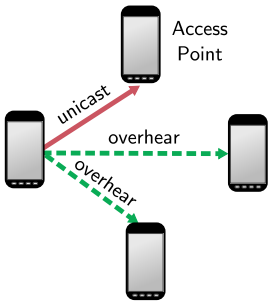
- When phone A is about to serve a segment s requested by phone B, it first checks if there are pending requests for the same segments from other neighbors. If there are, it finds the maximum number of dimensions requested among these requests. Denote this maximum dimension by d. If there is none, d is the number of dimensions requested by B. Afterwards, A serves d dimensions of segment s to B. The other phones, which need up to d dimensions of s, should be able to get the dimensions through overhearing.

After serving B, A notifies all phones that requested some dimensions of segment s. Upon receiving the notification, these phones check if they received all the necessary dimensions to decode s. If not, they send requests for additional dimensions. This is necessary, because overhearing is not guaranteed for all dimensions sent by A and for all phones. Finally, the scheme gives higher priority to requests that are closer to the playback time when serving them. Overhearing and unicasts effectively allow for pseudo-broadcast. As described, the amount of traffic saved by pseudo-broadcasting segment s depends not only on the quality of the overhearing but also on the number of requests of segments s from other phones that A processes at the time of broadcasting.

To be concrete, consider a local network consisting of four phones: A, B, C, and D. After finishing downloading segment s using 3G, A advertises it to B, C, and D. B, C, and D then send requests for this segment to A. For simplicity, assume perfect overhearing. First, consider the case where all requests arrive at A at a similar time. In this case, when A serves, *e.g.,* B's request for segment

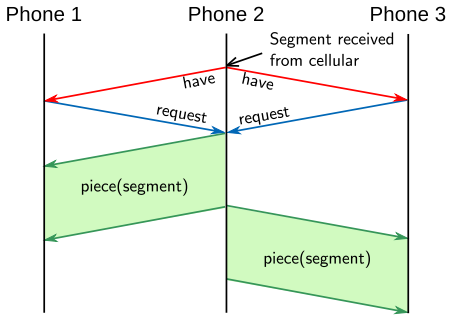


(a) MicroNC-P2

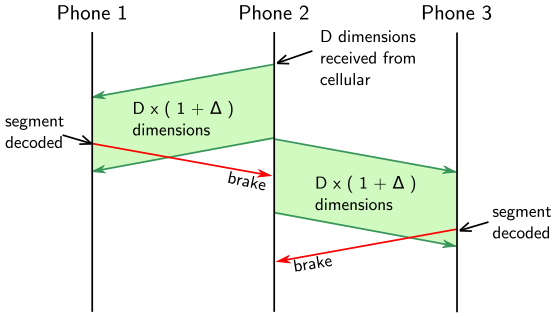


(b) MicroBroadcast

Figure 4: Space-time Diagram of MicroNC-P2 and MicroBroadcast.



(a) BitTorrent-Pull



(b) R2-Push

Figure 5: Space-time Diagrams of BitTorrent-Pull and R2-Push.

s, A removes C and D's requests for s. Effectively, A serves all B, C, and D using a single transmission of s. Now, consider the case where the request from D arrives later than the time A initially serves s to B and C. The late arrival of D's request could be due to various reasons, such as large receiving and sending queues

Algorithm 2 MicroNC-P2 Algorithm

when a new segment s is received
  if s is received by the requester **then**
    // initial push
    Send all dimensions of s to a neighbor
  end if
  Add s to the list of segments to be advertised
end when

when a packet p is received from A
  if p is an advertisement
    or notification containing s **then**
    // subsequent pulls exploit overhearing
    Request A for the missing dimensions of s
  else if p is a request for d dimensions of s **then**
    Add this request to the request queue
  else if p is a dimension of s **then**
    Progressively decode s using p
  end if
end when

when there is a request for d dim. of s from A
  // pseudo-broadcast
  if there are other similar requests **then**
    Let d be largest requested dimension
    Remove these requests from the request queue
  end if
  Send d dimensions of s to A
end when

of D. A now has to serve D all dimensions of s even though D may overheard some dimensions initially sent by A to B (or C). Apparently, A needs to send more than needed.

To address this issue, we propose an initial push of segment s. Specifically, when A finishes downloading segment s, it sends all dimensions of s to a randomly selected neighbor before advertising the segment. By doing so, A ensures that the initial dissemination of segment s is taken into account in subsequent requests of segment s (if any) of A's neighbors. This effectively creates a *perfect synchronization* of the reception of the initial requests of segment s. We provide the pseudocode of MicroNC-P2 distribution algorithm in Alg. 2 and the space-time diagram of MicroNC-P2 in Fig. 4(a).

Last but not least, in order to address loss of request and notification packets, which could lead to missing segments, MicroNC-P2 includes a *recovery thread*. This thread periodically re-requests segments that were requested after a certain amount of time but never received.

4.3 Reception Rate

Our system aims to allow each phone to receive at a rate equal to the sum of the 3G/4G download rates of all phones in the cooperative group. This is the best rate we can hope to achieve since this is the maximum rate at which our network gets new information from the server. This rate may be higher than the playback rate of the video; this is useful to reduce the probability of a buffer underflow during playback. Downloading at a rate higher than the playback rate requires caching of the stream locally. This is not a problem on modern phones which come with large storage space. For instance, a Nexus S can store up to 16 GB of data. Assume for simplicity that each phone can receive rate R_c from the cellular network and rate R_l from the local network. If we have N phones, this implies that we need to maintain $R_l \geq (N-1)R_c$, since each phone is

expected to receive through the local network the segments that the other $N-1$ phones have downloaded. In our set-up of interest, this was possible thanks to our efficient use of the local network through network coding and overhearing: our bottleneck (min-cut) has been between the phones and the cellular network.

5. IMPLEMENTATION DETAILS

In this section, we describe the implementation details, the major challenges we faced specifically on Android phones, and the design choices we made to address them.

5.1 Architecture Details

Our software is developed mostly in Java, with some minor parts in C, and runs on Android 2.3 and 4.0 and Java 2 SE. Both the Android and Java versions share the same structure and code, except for the graphical user interface (GUI) and the code that uses the local network wireless API. In this paper, we restrict our attention to the Android version.

Requester. It internally uses components called producers to retrieve segments of the video from the source of the stream. Our current implementation contains producers for three types of sources: HTTP, file, and content. The first one (HTTP) loads segments from an HTTP server using range requests. The second one (file) loads video from locally available files. Finally, the third one (content) retrieves data using the ContentProvider API of Android (*e.g.,* videos can be captured with the phone camera). The implementation allows to easily add new producers. The HTTP producer is agnostic to the actual networking technology used to access the server. Therefore, it can work not only when the phones use a 3G network to access the video server, but also when they use an infrastructure mode WiFi network. The overhead of using range requests measured in bytes is relatively small, around 3% when using segments of 22500 bytes. The download of segments is affected by round-trip time, but if the HTTP server hosting the video supports persistent connections, all requests can be carried out in a single TCP connection, thus saving some overhead. To further reduce the impact of round-trip time, our code supports the usage of multiple parallel TCP connections on each phone.

Storage. The segments are stored in the internal flash memory of the phone to keep the application memory requirements low. It is possible to access the segments from the storage either using a Java API, as done by the requester and MicroNC-P2, or via an embedded HTTP server that we have developed. This second interface allows us to play the video stream using the native Android media API. In order to support playback of non-streamable video, for instance, MP4 files with `moov` atom at the end of the file, our embedded HTTP server supports range requests; thus, the Android Media Player can perform random access of the video stream. If a range that has not yet been downloaded is requested, the HTTP server waits until it receives the full range before answering.

Graphical User Interface (GUI). The GUI automatically displays the locally reachable peers to form a group, and which streams (if any) is being downloaded in the group. The user only needs to select the desired stream and join it. The phones can play the video in a synchronized manner or at their own pace. Playback and download are decoupled thanks to our storage mechanism: a phone could be participating in the download while its video player is paused. To render the video, the application uses the media playback API included in Android, which supports various containers and video formats, such as H.264 in a MP4 container. The video can be displayed while MicroCast is still downloading; therefore, live streaming is supported.

MicroBroadcast. In order to facilitate porting of the application to different wireless technologies, MicroBroadcast contains an application layer implementation of a networking stack. Depending on the wireless technology used, features of MicroBroadcast are either implemented using a native mechanism or emulated. For instance, the Bluetooth implementation re-uses the native peer discovery mechanism while WiFi nodes run a custom peer discovery protocol. We give the implementation details for pseudo-broadcast in Section 5.4.

5.2 Multiple Network Interfaces

Each phone needs to use an interface as downlink (*e.g.,* 3G or WiFi) and another interface (*e.g.,* WiFi or Bluetooth) for the local cooperation. For the connection to the server, we chose 3G over WiFi mainly because it provides ubiquitous Internet access. A second reason is that Bluetooth and WiFi share partially overlapping parts of the spectrum and are often implemented in the same chip, while 3G is usually implemented on a different chip and uses a different part of the spectrum. This suggests that using WiFi (for the connection to the server) together with Bluetooth (for the local network) may noticeably decrease the transmission rate, which was indeed the case during our initial experiments. 3G is independent from both Bluetooth and WiFi, so the combination of 3G and either WiFi or Bluetooth does not reduce the transmission rate. For the local connection, we use WiFi instead of Bluetooth because it can support a larger number of connections at higher rates.

We also note that the Android connectivity manager imposes additional challenges when 3G is utilized simultaneously with WiFi. In particular, in order to improve the battery life, every time the WiFi interface is activated, Android turns the 3G data connection off. We solve this problem using an undocumented API that forces routing of packets for the HTTP server through the 3G interface, and therefore preventing the interface from being shut down.

5.3 Network Coding

Network Coding Scheme. We use generation-based network coding [6] over the field GF(2^8). Each segment is broken down into m packets $\hat{\mathbf{b}}_i$, which together form one generation (or segment), where m is the segment or generation size. Each packet contains n bytes, and we treat each byte as a symbol in GF(2^8). We also augment each packet with the m coding coefficients, each of which is selected uniformly at random from GF(2^8). Thus, each packet can be seen as a vector of length $n+m$ symbols from GF(2^8)

Phone A sends to phone B linear combinations of packets of the same segment, where the coding coefficients used to create linear combinations are selected uniformly at random from GF(2^8). Phone B can decode a segment upon receiving m linearly independent combinations of packets of the segment. Let M denote the matrix formed by m linearly independent packets: $M = [E \,|\, C]$, where E is of size $m \times n$ and C is the coefficient matrix of size $m \times m$. Original packets $\hat{\mathbf{b}}_i$ can be recovered by finding the inverse of C. In particular, $C^{-1} \cdot [E \,|\, C] = [B \,|\, I]$, where B is the matrix of size $m \times n$ whose row i is $\hat{\mathbf{b}}_i$ and I is the $m \times m$ identity matrix. Inverting C takes $\Theta(m^3)$ and multiplying C^{-1} with $[E \,|\, C]$ takes $\Theta(m^2(n+m))$ in terms of finite field multiplication. Thus, the decoding takes $\Theta(m^3 + nm^2)$ in total. Generating m randomly encoded packets can be done by generating a random coefficient matrix R of size $m \times m$ and multiplying R with $[B \,|\, I]$. Thus, the encoding of a segment also takes $\Theta(m^3 + nm^2)$ in total.

CPU Limitations. Network coding is a CPU intensive operation. In MicroCast, encoding and decoding must be performed efficiently,

at a rate matching that of the local network dissemination; otherwise, CPU risks to become the bottleneck of the video distribution. Therefore, in our implementation, we explored several ways to optimize the coding speed.

The first method to reduce the CPU usage is to limit the size of the coding generation. The smaller the number of packets in each segment, the smaller the coding complexity. Using smaller segment sizes, however, reduces the diversity of encoded packets, *i.e.,* packets are less likely to bring innovative information to their recipients. In Section 6.5, we give encoding and decoding rates as a function of segment size.

Second, we seek to optimize our implementation of network coding. In particular, we test two implementation approaches: pure Java and native code. In the first implementation, the encoding and decoding operations are performed by code that runs in the Dalvik virtual machine. In the second approach, the code runs natively on the phone CPU and is invoked through the Java Native Interface. The Java implementation has the advantage of being portable across different hardware platforms but is less efficient than the native version. In both implementations, we use table lookups to perform finite field multiplication and division, and we use the bit-by-bit XOR operation to perform addition and subtraction.

For packet length equal 900 (bytes), segment length equal $22,500$ (bytes), and (resulting) generation size equal 25, inverting the coefficient matrix C takes roughly 8% of the decoding time (measured on the native implementation). The rest of the time is used to recover the original vector by linearly combining the received packets (multiplying C^{-1} with $[E \mid C]$). The Java implementation can encode at 2.9 Mbps and decode at 4.3 Mbps (the significant difference in rate is due to a different memory usage pattern), while the native implementation can both encode and decode at 24 Mbps. The Java implementation is sufficient for low bit-rate videos while the native implementation can support even high-quality video streaming (indeed, with the native implementation, our experiments show that MicroCast can support 2.5 Mbps stream to a group of 7 phones). Both implementations, although sufficient for our needs, are not fully optimized and thus the rate could potentially be further improved.

5.4 Implementing High-Rate WiFi Broadcast

MicroBroadcast provides to to MicroNC-P2 an interface for high-rate local broadcast. To the best of our knowledge, this is the first system that provides this capability on top of WiFi on Android phones.

Although phones within proximity of each other can, in principle, overhear all transmissions, high-rate broadcast was not possible with the existing modes. The *unicast mode of 802.11* does not exploit broadcast: it (redundantly) transmits the same packets to each receiver separately. The *broadcast mode of 802.11* has its own disadvantages, [31]: (i) it lacks a back-off mechanism, which may harm the performance of other flows; (ii) its transmission rate is limited to the minimum (base rate, 1 Mbps); (iii) finally, unlike laptops, it is not always possible to adapt the broadcast transmission rate on Android phones due to wireless driver and firmware limitations.

A possible solution is to use *pseudo-broadcast*, *i.e., overhearing*, which combines the benefits of unicast and broadcast. Unicast is used as the transmission mode, but the phones overhear all transmissions in their neighborhood. Therefore, pseudo-broadcast combines the desirable properties of unicast (high rate, back-off) with overhearing, which makes it attractive. Although it has been implemented in several frameworks [18, 31, 4], when implementing pseudo-broadcast, we faced several challenges that are specific to Android phones.

First, the phones we used do not readily support the *promiscuous mode* due to the constraints imposed by the WiFi firmware and drivers. Therefore, we needed to update the WiFi drivers of all the Android 2.3 phones we used, and the firmware in some of them. In particular, we updated the WiFi driver and firmware by installing CyanogenMod 7 ROM [9] (a custom Android firmware) on the phones after testing various possible firmwares. With the CyanogenMod 7 ROM, promiscuous mode is available, but only in infrastructure mode.

Second, even with the promiscuous mode enabled, Android does not support pseudo-broadcast mode natively, *i.e.,* does not pass the overheard packets up to the application layer. We had to develop our own *overhearing API* for that purpose "under the hood of Android framework" by developing our own C library and a C binary executable program that runs as a daemon. This involved filtering out irrelevant overheard packets (*i.e.,* packets which do not belong to video data that the phones are interested in) so as not to overload the CPU.

Third, as we mentioned above, since overhearing is not available in ad-hoc mode, this pseudo-broadcast implementation works only in infrastructure mode. Using infrastructure mode has a major disadvantage compared to ad-hoc mode: when a phone transmits a packet to another phone, the packet has to be relayed by the access point, which results in double amount of traffic.

To avoid this disadvantage of infrastructure mode as well as to exploit the benefits of overhearing, we implemented a *pseudo-adhoc mode*, which is shown in Fig. 4(b). In this mode, one of the phones acts as an access point (AP), and all other phones transmit data to it. These transmissions are overheard opportunistically by all other phones. When the AP phone receives a packet, it *does not* forward it (as it would normally do in the infrastructure mode), since the other phones should already have received it via overhearing. In this manner, we are able to enable overhearing (which is only possible in infrastructure mode), while ensuring only one transmission per packet (which is the case in ad-hoc mode), thus the term *pseudo-adhoc mode*.

6. PERFORMANCE EVALUATION

In this section, we first evaluate the performance of MicroDownload and MicroNC-P2 and compare their performance to baseline, popular alternatives. We then evaluate the entire MicroCast system as a whole. We show that our schemes significantly improve the streaming experience in terms of download time and video rate, without introducing significant battery and CPU penalty.

In our evaluation, we perform experiments on an Android testbed consisting of seven smartphones: four Samsung Captivate and three Nexus S. All smartphones have a 1 Ghz Cortex-A8 CPU and 512 MB RAM. Six of them use Android Gingerbread (2.3) and one (Nexus S) uses Android Ice Scream Sandwich (4.0) as their operating systems.

6.1 Evaluation of MicroDownload

In this section, we present experimental results that motivate the necessity of implementing MicroDownload and we show its effectiveness. The setup is the following: we used three Nexus S connected to the same cellular network provider, and placed them within proximity of each other (the distances among them are approximately 2 cm) in an indoor environment. The phones were placed in their positions 5 minutes before the experiment started to eliminate any possible positive or negative bias due to mobility.

In our experiment, we disabled MicroNC-P2 and we measured

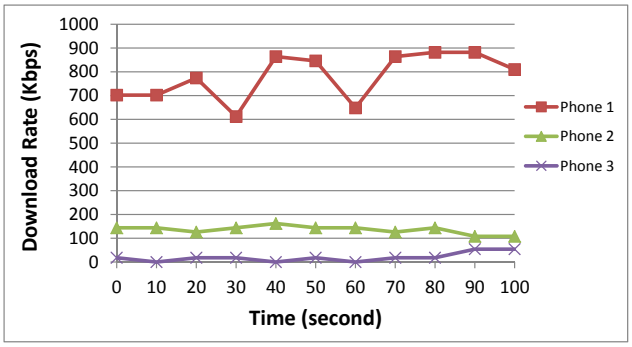


Figure 6: The cellular link rates of three smartphones in the same geographical area.

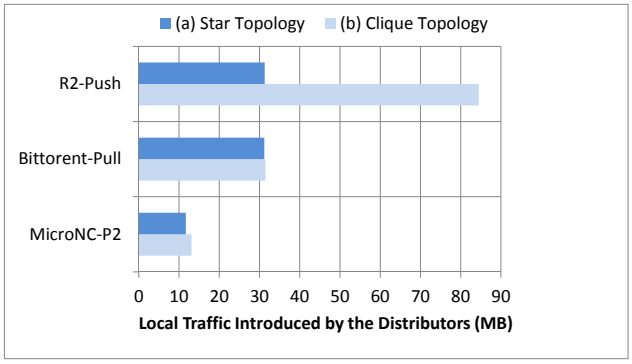


Figure 7: The amount of local traffic introduced by the phones when using different distributors. The file being downloaded is 9.93 MB. Bandwidth of the local network is sufficient to support the local dissemination. All phones receive the file at a similar rate 550 Kbps. MicroNC-P2 manages to introduce the least amount of traffic thanks to network coding and overhearing.

the download rates of the smartphones over 100 seconds. The results are presented in Fig. 6. The figure shows that despite being in close proximity and being connected to the same operator the phones experience significantly different average download rates. Phone 3 has a very low rate because it uses EDGE. The other two phones use the same 3G network but still have significantly different download rates. Moreover, phone 1 experiences significant rate variations. This variability in time and across phones is our motivation to develop MicroDownload to adaptively request data according to the downlink rates of cellular links, instead of making static decisions, such as splitting the requests equally among the phones.

Using these measurements, we can compare the effectiveness of MicroDownload algorithm to a simpler static strategy. We consider a scenario where the three phones download a 750 kB file, and MicroDownload makes a static decision, *i.e.,* each phone requests one third of the file. In this case, phone 3 (considering the same channel realization as in Fig. 6) is the bottleneck for downloading the file, and the total download duration is 80 seconds. However, if MicroDownload makes adaptive requests, as proposed in Section 4.1, then phone 3 is not a bottleneck anymore, and the total download duration is less than 10 seconds. This shows the importance of the adaptive request mechanism of MicroDownload.

6.2 Evaluation of MicroNC-P2

In this section, we compare the performance of MicroNC-P2 to a BitTorrent-based distributor [8] and an R2-based distributor [41]. We refer to these two distributors as BitTorrent-Pull and R2-Push, respectively. The performance metric of interest is the amount of local network traffic introduced by the phones when using different distributors to disseminate the same amount of information. We consider a clique and a star overlay topologies for local connectivity. Packets are exchanged locally using UDP.

We implemented the BitTorrent-Pull scheme based on the description of the BitTorrent protocol [8]. In particular, our implementation of the protocol supports three main types of messages: (i) *bitfield* and *have* messages, which are used by a phone to advertise the segments to its neighbors; (ii) *request* messages, which are used by a phone to request specific segments from its neighbors; and (iii) *piece* messages, which contain the actual data. The space-time diagram of BitTorrent-Pull is provided in Fig. 5(a). Fundamentally, BitTorrent is a pull-based P2P protocol: when a phone has downloaded a new segment, it advertises this segment to its neighbors. The neighbors then explicitly request the segments that they are missing. To account for the wireless loss rate (when using UDP), we implemented a recovery thread which periodically re-requests missing segments.

We implemented the R2-Push scheme based on the description in [41]. The R2 protocol was introduced to exploit the benefit of random network coding and random push. Following [41], our implementation of R2-Push supports two main types of messages: (i) *data* messages, which are random linear combinations of packets belonging to the same segment; and (ii) *brake* messages, which are used by phones to inform their neighbors that they successfully received and decoded specific segments. The purpose of brake messages is to ensure that the neighbors would stop pushing (unnecessary) linear combinations of the decoded data segments.

The space-time diagram of R2-Push is provided in Fig. 5(b). In contrast to BitTorrent-Pull, with R2-Push, the phones start pushing linear combinations of segments as soon as they receive them from either the cellular network or their neighbors. In our implementation, for a particular segment a phone is downloading, we limit the number of linear combinations that it can push to its neighbors to the rank of the matrix formed by the received packets plus a fixed amount of redundancy, Δ, to account for the wireless loss rate.

Fig. 7 shows the total amount of traffic introduced to the local network by four phones to disseminate a file when the phones are connected using star and clique topologies. Note that the difference between these topologies are that in the star topology, a hub is used as an access point (AP), and in the clique topology a random phone is chosen to serve as an AP. In both topologies, all phones overhear all the transmissions in the group, *e.g.,* from phones to the AP which is either a hub or a phone. MicroNC-P2 utilizes pseudo-adhoc as described in Section 5. The file size is 9.93 MB and is downloaded by a single phone using its 3G connection. The average rate of the 3G connection is measured at 550 Kbps. The

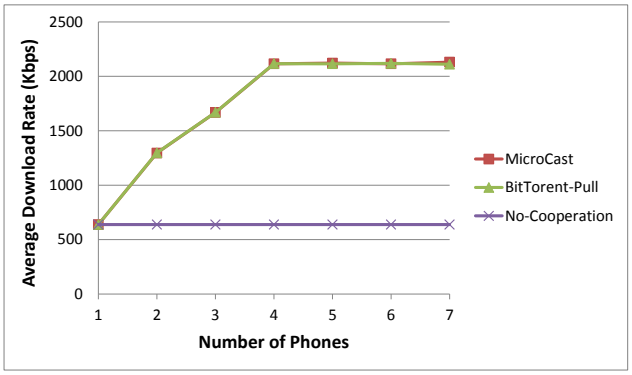


Figure 8: Average download rate as a function of number of phones when the local network bandwidth is 20 Mbps. Performance of MicroCast and BitTorrent-Pull almost coincide on the plots.

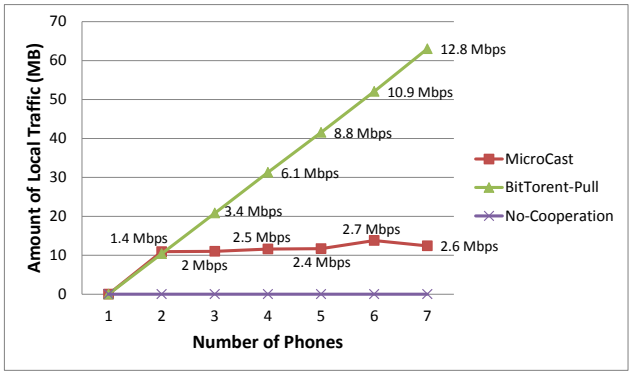


Figure 9: The amount of local traffic introduced by all phones.

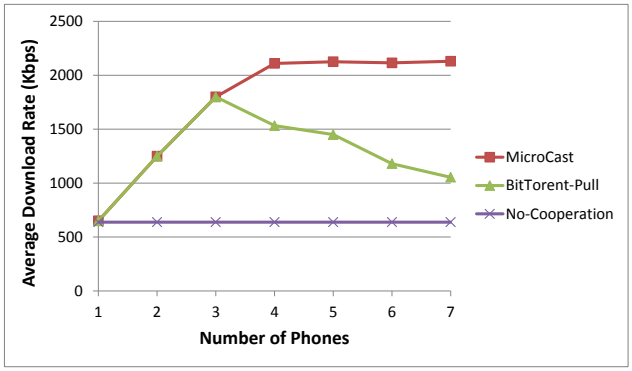


Figure 10: Average download rate as a function of number of phones when the local network bandwidth is 4 Mbps.

phone that downloads the file is chosen at random. For R2-Push, we choose $\Delta = 3\%$. The local network can support 20 Mbps UDP traffic, measured using *iperf* [23]. This bandwidth is much larger than what is needed to support the traffic introduced by the phones in both topologies. Since the local network bandwidth is sufficient, each of the phone receives at the rate similar to the phone which downloads the file through 3G. Each reported number is averaged over three experiments.

We first observe from Fig. 7 that the amount of traffic introduced by both BitTorrent-Pull and R2-Push are more than three times higher than that of MicroNC-P2. Intuitively, this is due to the fact that when using MicroNC-P2, a packet sent by a phone may be beneficial to three phones instead of one thanks to network coding and overhearing.

Fig. 7 also shows that in a clique topology, R2-Push introduces much more traffic as compared to the star topology, while BitTorrent-Pull and MicroNC-P2 introduce similar amount of traffic in both clique and star topologies. This is due to the fact that in a clique topology, a phone may simultaneously receive linear combinations of the same segment from multiple neighbors. When this happens, it is critical that the neighbors which are sending to this phone stop pushing linear combinations in a timely manner. This could only be achieved with a timely arrival of the brake (stop) messages, which is not always possible in the clique topology, or in a setup where additional traffic is very high. The authors of R2 also observed the problem and reported it in [22]. In their setting [22], the amount of redundancy could be reduced by using larger segment sizes. However, due to limited computational power of mobile devices, we cannot afford having large segment sizes as discussed in Section 5.

To summarize, the set of experimental results presented in this section clearly show that by exploiting the broadcast nature of the wireless medium, MicroNC-P2 manages to introduce less amount of traffic into the local network as compared to BitTorrent-Pull and R2-Push.

6.3 Evaluation of the MicroCast System

In this section, we present the performance evaluation of the entire MicroCast system. We compare the average download rates of MicroCast to two other schemes: no cooperation, which we will refer to as No-Cooperation, and the combination of MicroDownload and BitTorrent-Pull, which we will simply refer to as BitTorrent-Pull. Note that we do not include R2-Push as a baseline in this section due to its inefficiency in our setup, as explained in Section 6.2.

In our experimental setup, we used up to seven phones, the first four of which had 3G rates varying from 480 Kbps to 670 Kbps. The rest of the phones did not have 3G connections. Packets are exchanged locally using UDP. The local network can support up to 20 Mbps UDP traffic. We use the star topology as explained in Section 6.2. We also use pseudo-adhoc. The size of the video file is 9.93 MB. Each value reported in this section is averaged over three experiments.

Fig. 8 shows the average download rate versus the number of phones. We observe that both MicroCast and BitTorrent-Pull are able to improve the average download rate up to the sum capacity of the 3G links. Note that MicroCast and BitTorrent-Pull do not provide any improvement for more than four phones because only four phones have 3G connections in our setup, and the aver-

age download rate is limited by the sum capacity of the four corresponding 3G links. Fig. 9 shows the amount of local traffic versus the number of phones. Although in Fig. 8 we see similar average download rates for both MicroCast and BitTorrent-Pull, Fig. 9 shows that BitTorrent-Pull introduces a larger amount of local traffic (which increases linearly in the number of phones) as compared to MicroCast. This behavior of BitTorrent-Pull is detrimental in terms of the average download rate in congested networks. An important observation is that, as the number of phones increases, MicroCast rate does not increase. This indicates that, even with many nodes, overheard packets are lost very rarely.

We updated our experimental setup to evaluate the performance of MicroCast and BitTorrent-Pull in a congested network. In our new setup, the congested network is generated by introducing 16 Mbps background UDP traffic on the same 802.11 channel (note that there is also interference from other sources in the environment which contributes to the background traffic). Since the local network can support up to 20 Mbps traffic, the leftover traffic is less than 4 Mbps. Fig. 10 presents the average download rate versus the number of phones in this setup. We see that the average download rate of BitTorrent-Pull reduces when we have more than three phones. This is because BitTorrent-Pull introduces a large amount of local traffic (as illustrated in Fig. 9), which leads to congestion.

Note that the addition of the 5th, 6th, and 7th phones also increases the local traffic in BitTorrent-Pull, even though they do not have 3G connection. This is because they still need to receive the file in the local area, which contributes to local area traffic.

On the other hand, Fig. 10 shows that MicroCast still improves the average download rate up to the total capacity of 3G links (of four phones) in a congested network. This is because it introduces only a small amount of local traffic (*e.g.,* even for seven phones, MicroCast only introduces 2.6 Mbps traffic to the local network). It can be observed from Fig. 10 that the average download rate of MicroCast is more than three times higher than that of No-Cooperation. Also, the improvement of MicroCast over BitTorrent-Pull in terms of average download rate is as high as 75% (we observed even more improvement for different setups, *e.g.,* with 18 Mbps background traffic), which is significant. We also note that our experimental results are consistent with the theoretical findings in our earlier work [30], *e.g.,* Fig. 8 and Fig. 10 are consistent with Fig. 3 in [30].

6.4 Evaluation of Energy Consumption

In this section, we evaluate the energy consumption of MicroCast when compared to our baselines: BitTorrent-Pull and No-Cooperation. We consider a setup similar to the one in Section 6.3, *i.e.,* three phones are connected to a fourth one that acts as AP , the 3G rates vary from 450 Kbps to 700 Kbps, and the size of the video file is 95.4 MB. We used the *BatteryManager* class of the Android SDK for the power consumption measurements. Before the experiment, all four phones are fully charged. During the experiment, the battery states are recorded every 10 seconds. In the experiments that do not use the local network, the wireless interface is turned off. The experiments are repeated three times, and their average is reported.

Fig. 11 presents the battery state (100% corresponds to the fully charged battery) versus time. Note that "MicroNC-P2 Access Point" and "BitTorrent-Pull Access Point" show the battery consumption levels of the phone which is selected to act as an AP in MicroCast and BitTorrent-Pull schemes, respectively. On the other hand, "MicroNC-P2 Normal" and "BitTorrent-Pull Normal" show the battery consumption levels of a phone which is not an AP. We observe from Fig. 11 that the No-Cooperation scheme has less battery consumption compared to MicroCast and BitTorrent-Pull at a given time, *e.g.,* at 400 second. However, the time required to download the video file of No-Cooperation is very high (more than two times) as compared to MicroCast and BitTorrent-Pull.

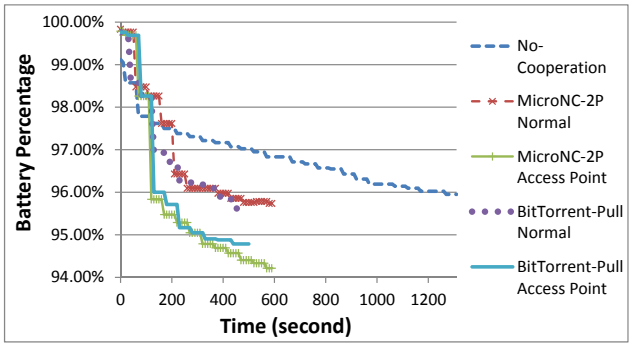


Figure 11: Battery drain when downloading the same file of size 95.4 MB using different schemes.

If we look at the battery levels of all schemes when the file transmission is completed, we see that the battery consumption levels of No-Cooperation, MicroCast, and BitTorrent-Pull are similar. This demonstrates that employing cooperation in the long term (to download a video file) does not bring any significant battery cost. Finally, the phones which act as APs consume more battery as compared to the other phones: this is expected because the AP phone has additional tasks. However, even in the worst case (for the AP phone), MicroCast consumed approximately 6% of the battery to download the whole file. Considering that the phones are downloading a large file (*i.e.,* 95.4 MB), this battery consumption level is reasonable. These considerations show that the rate benefit of MicroCast comes at no significant battery cost.

6.5 Evaluation of NC Implementations

In this section, we present the performance of the two implementations of network coding we developed: native (written in C) and Java-based coding. The two implementations are described in Section 5.3. Fig. 12 shows the highest achievable decoding and encoding data rates. The slowest encoding rate for the Java implementation is 1 Mbps, while for the Native implementation, it is 8 Mpbs, which is significantly higher than what is needed for video streaming applications considered in this paper. Fig. 12 also shows that the Java implementation is more than sufficient to stream today's typical Internet videos when using generation size equal 25.

7. LIMITATIONS AND EXTENSIONS

The system presented in this paper builds on the assumptions described in Section 3: there is a small number of trusted users within proximity of each other, using the cellular connection to download and WiFi (with broadcast and single-hop transmission) for local sharing. However, our architecture is highly modular and allows for easily swapping several components, transparently to the rest of the system.

First, we have already implemented and successfully tested a version of the system that uses Bluetooth in the local area, in-

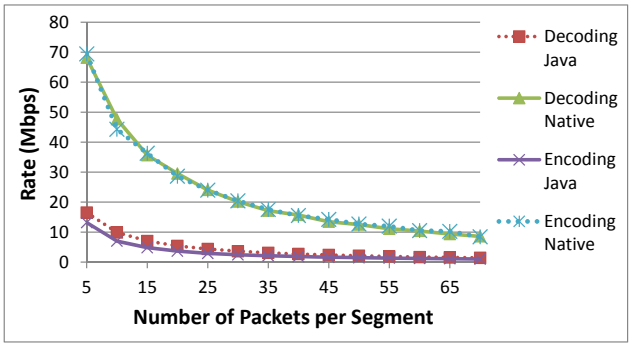


Figure 12: Coding and decoding throughput as a function of the generation length. Each packet is 900-byte long.

stead of WiFi. An attractive feature of Bluetooth is that it can be used without rooting the phone. Furthermore, it supports both single-hop and multi-hop transmission via piconets and scatternets respectively. On the downside, Bluetooth does not support broadcast, which was a necessary ingredient for achieving MicroCast's benefits. It would be interesting to study potential benefits of Bluetooth in multi-hop scenarios, *e.g.,* when not all phones can overhear each other sufficiently well. Second, one could use WiFi (instead of cellular) for downloading the content from the server: our architecture already supports that. However, in that case, the downlink and the local transmissions are no longer independent as they both use WiFi. We plan to explore this interaction in future work; we have already analyzed this problem theoretically (in a network utility maximization framework) in [30].

Our current implementation of MicroDownload uses a simple algorithm that could be improved in multiple ways. It currently does not take into account the local link quality between the nodes and does not try to download in parallel blocks on multiple devices when spare capacity is available (for instance, at the end of the stream). The current implementation of MicroDownload also cannot exploit broadcast capabilities of the WAN link. This is typically not available on cellular networks but could be used if, for instance, the phones are connected to an 802.11 AP. Notice that even when broadcast is available on the WAN link, it is still useful to perform collaboration: local dissemination can help correcting uncorrelated erasures experienced by the nodes.

Finally, the current implementation cannot support more than 7 concurrent devices (when an Android 4.0 device acts as the AP) or 6 devices (when an Android 2.3 device acts as the AP). This is due to the limitation of the softAP currently implemented in Android. In order to further increase the number of users, one could use more than one phones as softAPs, but this requires modification of the system. In particular, promiscuous mode does not allow for overhearing of packets associated with a different softAP. We will investigate such extensions as part of future work.

8. CONCLUSION

In this paper, we designed, implemented, and evaluated a novel system, MicroCast, that enables a group of smartphone users within proximity of each other to watch the same video from the Internet at the same time. MicroCast cooperatively uses the resources on all smartphones of the group, such as cellular links and WiFi connections, to improve the streaming experience. The system consists of three key components, namely: MicroDownload, which determines which parts of the video each phone should download from the server; MicroNC-P2, which exploits overhearing and network coding over WiFi; and MicroBroadcast, which provides, for the first time, high-rate broadcast over WiFi on Android phones. Experimental results demonstrate significant performance benefits in terms of per-user video download rate without battery penalty.

9. ACKNOWLEDGEMENTS

This work was funded by the following grants: ArmaSuisse Wissenschaft+Technologie (W+T) Project no. 8003413832, ERC Project NOWIRE (ERC-2009-StG-240317), AFOSR MURI award FA9550-09-0643, and NSF CAREER award 0747110.

10. REFERENCES

[1] Wireless network coding: from theory to practice, project wiki-page. http://odysseas.calit2.uci.edu/doku.php/public:muri09.

[2] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. A. Thekkath. COMBINE: leveraging the power of wireless peers through collaborative downloading. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 286–298, 2007.

[3] C. Boldrini, M. Conti, and A. Passarella. Exploiting users' social relations to forward data in opportunistic networks: The HiBOp solution. *Pervasive and Mobile Computing*, 4(5):633–657, Oct. 2008.

[4] R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye, R. Ramjee, and L. Ravindranath. DirCast: a practical and efficient Wi-Fi multicast system. In *Proceedings of the 17th IEEE International Conference on Network Protocols (ICNP)*, pages 161–170, Oct. 2009.

[5] J. Chesterfield, R. Chakravorty, I. Pratt, S. Banerjee, and P. Rodriguez. Exploiting diversity to enhance multimedia streaming over cellular links. In *Proceedings of the 2005 IEEE INFOCOM*, volume 3, pages 2020–2031, Mar. 2005.

[6] P. Chou and Y. Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5):77–85, Sept. 2007.

[7] Cisco Systems. Cisco visual networking index: Global mobile data traffic forecast update, 2011-2016. http://www.cisco.com.

[8] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[9] CyanogenMod Team. Cyanogenmod. http://www.cyanogenmod.com.

[10] Y. Feng, Z. Liu, and B. Li. GestureFlow: streaming gestures to an audience. In *Proceedings of the 2011 IEEE INFOCOM*, pages 748–756, Apr. 2011.

[11] C. Fragouli and E. Soljanin. *Network Coding Fundamentals*. Now Publishers Inc, Delft, The Netherlands, June 2007.

[12] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proceedings of the 2005 IEEE INFOCOM*, volume 4, pages 2235–2245, Mar. 2005.

[13] P. Goldstein. Credit suisse report: U.S. wireless networks running at 80% of total capacity. *FierceWireless.com*, July 2011.

[14] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, G. Pei, and A. Srinivasan. Cellular traffic offloading through opportunistic communications: a case study. In *Proceedings of the 5th ACM Workshop on Challenged Networks (CHANTS)*, pages 31–38, 2010.

[15] S. Hua, Y. Guo, Y. Liu, H. Liu, and S. Panwar. Scalable video multicast in hybrid 3G/Ad-Hoc networks. *IEEE Transactions on Multimedia*, 13(2):402–413, Apr. 2011.

[16] P. Hui, J. Crowcroft, and E. Yoneki. Bubble rap: social-based forwarding in delay tolerant networks. In *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 241–250, 2008.

[17] S. Ioannidis, A. Chaintreau, and L. Massoulie. Optimal and scalable distribution of content updates over a mobile social network. In *Proceedings of the 2009 IEEE INFOCOM*, pages 1422–1430, Apr. 2009.

[18] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the air: Practical wireless network coding. *IEEE/ACM Transactions on Networking*, 16(3):497–510, June 2008.

[19] B. Li and D. Niu. Random network coding in Peer-to-Peer networks: From theory to practice. *Proceedings of the IEEE*, 99(3):513–523, Mar. 2011.

[20] S. Li and S. Chan. BOPPER: wireless video broadcasting with Peer-to-Peer error recovery. In *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo*, pages 392–395, July 2007.

[21] X. Liu, G. Cheung, and C. Chuah. Rate-distortion optimized network coding for cooperative video stream repair in wireless peer-to-peer networks. In *Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, June 2008.

[22] Z. Liu, C. Wu, B. Li, and S. Zhao. UUSee: Large-Scale operational On-Demand streaming with random network coding. In *Proceedings of the 2010 IEEE INFOCOM*, pages 1–9, Mar. 2010.

[23] NLANR/DAST. IPerf. http://sourceforge.net/projects/iperf.

[24] Y. Park, C. Jo, S. Yun, and H. Kim. Multi-Room IPTV delivery through Pseudo-Broadcast over IEEE 802.11 links. In *Proceedings of the IEEE 71st Vehicular Technology Conference (VTC)*, pages 1–5, May 2010.

[25] M. Pedersen and F. Fitzek. Implementation and performance evaluation of network coding for cooperative mobile devices. In *Proceedings of the 2008 IEEE International Conference on Communications (ICC) Workshops*, pages 91–96, May 2008.

[26] M. Pedersen, J. Heide, F. Fitzek, and T. Larsen. PictureViewer - a mobile application using network coding. In *Proceedings of the 2009 European Wireless Conference*, pages 151–156, May 2009.

[27] M. Ramadan, L. El Zein, and Z. Dawy. Implementation and evaluation of cooperative video streaming for mobile devices. In *Proceedings of the 19th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–5, Sept. 2008.

[28] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. MAR: a commuter router infrastructure for the mobile internet. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 217–230, 2004.

[29] J. B. Saleh, D. Qiu, and A. K. Elhakeem. Performance of an efficient scheduling approach to network coding for wireless local repair. *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications*, Jan. 2011.

[30] H. Seferoglu, L. Keller, B. Cici, A. Le, and A. Markopoulou. Cooperative video streaming on smartphones. In *Proceedings of the 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 220–227, Sept. 2011.

[31] S. Sen, N. K. Madabhushi, and S. Banerjee. Scalable WiFi media delivery through adaptive broadcasts. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[32] H. Shojania and B. Li. Random network coding on the iPhone: fact or fiction? In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 37–42, 2009.

[33] H. Shojania and B. Li. Tenor: making coding practical from servers to smartphones. In *Proceedings of the 2010 International Conference on Multimedia (MM)*, pages 45–54, 2010.

[34] H. Soroush, P. Gilbert, N. Banerjee, M. D. Corner, B. N. Levine, and L. Cox. Spider: improving mobile networking with concurrent wi-fi connections. *SIGCOMM Computer Communication Review*, 41(4):402–403, Aug. 2011.

[35] M. Stiemerling and S. Kiesel. A system for peer-to-peer video streaming in resource constrained mobile environments. In *Proceedings of the 1st ACM Workshop on User-provided Networking: Challenges and Opportunities (U-NET)*, pages 25–30, 2009.

[36] M. Sullivan. A day in the life of 3G. *PCWorld.com*, June 2009.

[37] C. Tsao and R. Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 337–348, 2009.

[38] P. Vingelmann, M. Pedersen, F. Fitzek, and J. Heide. On-the-Fly packet error recovery in a cooperative cluster of mobile devices. In *Proceedings of the 2011 IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, Dec. 2011.

[39] M. Wang and B. Li. How practical is network coding? In *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS)*, pages 274–278, June 2006.

[40] M. Wang and B. Li. Lava: A reality check of network coding in Peer-to-Peer live streaming. In *Proceedings of the 2007 IEEE INFOCOM*, pages 1082–1090, May 2007.

[41] M. Wang and B. Li. R2: Random push with random network coding in live Peer-to-Peer streaming. *IEEE Journal on Selected Areas in Communications*, 25(9):1655–1666, Dec. 2007.

[42] J. Whitbeck, M. Amorim, Y. Lopez, J. Leguay, and V. Conan. Relieving the wireless infrastructure: When opportunistic networks meet guaranteed delays. In *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1 –10, June 2011.

[43] WiFi Alliance. Wi-Fi direct. http://www.wi-fi.org.