Demo Abstract
DISSEMINATE: A Demonstration of Device-to-Device Media Dissemination

Venkat Srinivasan, Tomasz Kalbarczyk, and Christine Julien
The Center for Advanced Research in Software Engineering
The University of Texas at Austin
Email: {venkat.s, tkalbar, c.julien}@utexas.edu

***Abstract*—In this demonstration, we showcase DISSEMINATE, a protocol and accompanying Android application for supporting device-to-device dissemination of data that fulfills shared interests of devices in pervasive computing environments. DISSEMINATE is a novel publish-subscribe based protocol that breaks large (likely media heavy) data items into smaller *chunks* and opportunistically shares the chunks among co-located mobile devices. Interested devices issue subscriptions for the chunks they need and advertisements for the chunks they can share. Our publish mechanism uses a computation of a chunk's *uniqueness* to determine whether or not to broadcast it to the locally connected devices. The DISSEMINATE app provides an implementation of the protocol in an Android application for sharing photographs via WiFi-Direct broadcast communication.**

I. INTRODUCTION

In pervasive computing environments, our demand for information at our fingertips is rapidly overtaxing our traditional communication capabilities. Specifically, such information access is commonly provided through cellular data connections, and it has been empirically demonstrated that, especially during crowded events, our demands on such infrastructure exceeds the capacity [3]. There are, however, many situations in which co-located users in these environments seek the same or similar data. For instance, witnesses to an urban event such as a festival or parade may wish to share and access video or other media created by other attendees at the event. Even when the data is downloaded from the infrastructure, the data demands may be overlapping. Consider, for example, a large number of commuters all experiencing the same traffic jam. These users may all immediately turn to their mapping application, requesting map information so they can plan a different route. Cooperating to download the map data by sharing some of the data among the devices' device-to-device connections can alleviate some of the burden on the infrastructure of everyone immediately attempting to individually download the map data.

In this demonstration, we present DISSEMINATE, our approach to providing device-to-device support for disseminating data that fulfills shared interests. In our previous work on the MadApp application [5], [6], we showed an application that enabled devices in a shared situation to share media (i.e., photographs) about their shared situation. However, as the media shared between the devices grows increasingly large, sharing entire files over the inherently unreliable device-to-device medium becomes very difficult and error prone. Further, when many of the nearby devices share and distribute the same information, blindly sharing all of the media files a device has with every other connected device wastes precious network and system resources (e.g., bandwidth and energy). In this demonstration, we showcase a protocol called DISSEMINATE that would run underneath an application like MadApp, enabling more efficient and effective dissemination over device-to-device communication links.

Related Work. Our approach fits in the emerging domain of mobile data traffic offloading [1], including similar work in the publish-subscribe domain [4], [7]. Existing approaches, however, focus on abstracting multicast communication and situations where information must be routed from a specific source to a specific destination. Other approaches expect that the the infrastructure mediates the device-to-device interactions, dictating which peer devices are used to supplement backend connections [2]. We perform all decision making *in the network*, allowing the devices to individually determine when and how to share data. Further, devices only participate in the collaborative dissemination of data that is also directly useful to the device or its user.

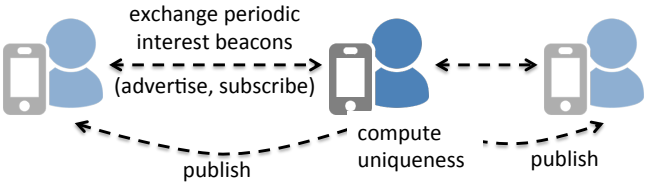

Fig. 1: Overview of DISSEMINATE protocol

Protocol Overview. DISSEMINATE splits each data item into smaller *chunks* that can be delivered out of order and reassembled at the receiving end. When the devices exchange these chunks instead of whole data items, a single lost chunk is much less catastrophic than losing the entire data item. Further, a given device can collect chunks from multiple different directly connected devices instead of having to receive the entire data item from a single device. Our approach to exchanging chunks is a publish-subscribe style protocol that

uses a combination of *advertising*, *subscribing*, and *publishing*. Fig. 1 shows an overview of the steps in DISSEMINATE. We accomplish advertising and subscribing in a single step. Specifically, devices periodically send beacons with a very lightweight representation of (1) the data items they are interested in and (2) the chunks of those data items they have already collected. The latter is an advertisement: these chunks represent the content that the device can disseminate. The two pieces together are a subscription: a device receiving the beacon can discern which chunks need to be delivered to the sending device. We derive a scheme for determining what data to publish based on a metric we term *uniqueness*: intuitively, our publish mechanism attempts to balance two aspects: (1) sending chunks that are not widely available in the immediate neighborhood and (2) sending chunks that are likely to be received by the intended subscriber(s). For the former, we simply measure the degree of redundancy of a particular chunk and choose chunks that are less redundant. For the latter, we favor chunks needed by neighbors whose connections have with high *link quality*, as measured by RSSI. In DISSEMINATE, each device operates independently, making the best apparent decision based on the information it has from the neighboring devices' beacons.

Demonstration Overview. The goal of this demonstration is not to evaluate the performance of DISSEMINATE. Instead, we showcase the ability of DISSEMINATE to distribute data related to shared interests of co-located devices. We show that it is feasible to implement the approach on stock Android phones, without any need to modify or "root" the phones or use any device-native features. Our application will run effectively on any WiFi-Direct enabled Android device.

II. IMPLEMENTATION

The implementation showcased in this demonstration is in two pieces. First (and most visibly) is the DISSEMINATE app, which performs device-to-device sharing of the chunks of the data items as determined by our novel publish-subscribe protocol and its uniqueness computation. Second (and transparent to the user) is a significant set of Android support components necessary for the demonstration to function; these components are interesting in their own right.

A. *The* DISSEMINATE *App*

The DISSEMINATE app embodies the novel publish-subscribe protocol that we built for the purpose of disseminating shared interests using device-to-device connections. We assume situations such as those described in the introduction, where multiple directly-connected devices share an interest in retrieving the same "heavyweight" (e.g., media) data items. We also assume that the data items are already present in the network, e.g., because one or more of the devices downloaded the data from the infrastructure or, as was the case in MadApp [5], because the devices in the network generated the data themselves.

To make the demonstration simple, the app currently assumes four pieces of heavyweight data shared between all connected devices. The app also assumes that, by a user launching the app, the user expresses an interest in all four data items. This means that the data items are immediately added to the device's beacons, and chunks for the data items are started to be collected if they are being published. On the other hand, the app does not assume that the data items are immediately available in the network to be shared. Instead, to share a data item one, or more of the users must explicitly tap the item and select "Download" from the menu that appears. This simulates either downloading the data item from the infrastructure or sharing a data item that is available on local storage. The app allows simulating low quality infrastructure connections in this process; the user can "tune" the data rate for the download, making the chunks of the data item appear more slowly or quickly, depending on the desired situation. As soon as at least one of the devices in the WiFi-Direct network initiates the download, the received chunks are shared via any available device-to-device connections available to other devices also running the DISSEMINATE app.

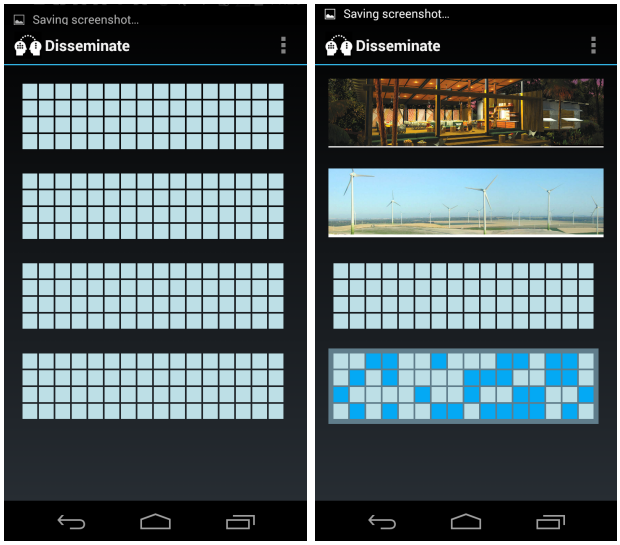

Fig. 2: Screenshots of the DISSEMINATE App

Fig. 2 shows a pair of screenshots of the DISSEMINATE app. On the left, the user has not initiated any downloading; the device is subscribed to receive all four data items, but it has not yet received any chunks. On the right, two downloads have completed and one is in progress. Within the app, each data item is divided into 64 chunks[1]. To show progress in collecting the chunks, the app's user interface shows the chunks as bricks that make up the eventual footprint of the file's thumbnail; as chunks are received, the associated bricks darken.

B. *Android Support Layers*

To support DISSEMINATE on Android, we built a communication layer that uses WiFi-Direct to support the device-to-device connections. This layer broadcasts data that is received

[1]This is a limiting design decision, taken for simplicity. It has impact on the performance of the publish-subscribe protocol but not on demonstrating the feasibility and nature of the approach.

from the DISSEMINATE app to all connected peers. We use two FIFO queues that communicate with the upper layer; we use two separate threads to process these queues, which run concurrently with the application layer's main thread.

Broadcast. The broadcast thread performs a form of busy-wait synchronization as it waits for packets to transmit. As soon as a packet arrives in the outgoing queue, it is broadcast using the WiFi-Direct broadcast address[2]. Packets sent by the DISSEMINATE app include the beacons that contain the advertisement and subscription information and also the published chunks (determined by the uniqueness computation that is part of the DISSEMINATE protocol). We expect that, in the future, we will be able to push the lightweight beacons onto dedicated device discovery approaches that are about to emerge on today's smartphones. Such beacon mechanisms will be made available with programming interfaces that allow application layers to "set" small amounts of payload data.

Receive. The receive thread waits for broadcasts received through WiFi-Direct. Once the thread receives new data on its open socket, it puts it into a second synchronous queue, which allows the DISSEMINATE app to retrieve received data as needed. Our implementation does not differentiate between receiving a chunk or a beacon; all of this logic is implemented in the DISSEMINATE app. The separation of the broadcast and receive threads allows for asynchronous broadcasting that frees up the DISSEMINATE app thread to perform other tasks.

Notable Aspects. Our implementation uses only stock Android APIs and does not require administrative privileges or any special device level code or functionality. It will work "out of the box" on any WiFi-Direct capable Android device. While the result is easy to use, the implementation was not straightforward. WiFi-Direct is far from a mature technology, and official support (including basic documentation) is scarce. We succeeded largely through trial and error. We have also overcome some of the major obstacles of working with WiFi-Direct (from an application's or user's perspective). For instance, we have made the WiFi-Direct group within DISSEMINATE "sticky," so the application and user are sheltered from intermittent connectivity issues, e.g., if the devices disconnect from the WiFi-Direct group, they automatically reconnect without user intervention. One major disadvantage to using WiFi-Direct to support device-to-device communication has to do with the way that WiFi-Direct groups are structured. Each group has a "group owner" that effectively serves as a router for the group members. Every packet (even broadcast packets) have to be routed through the group owner, so the performance in our Android prototype is not reflective of "true" device-to-device communication. However, this functions against us; in a true device-to-device implementation, we would expect the performance of DISSEMINATE to improve relative to the result reported below.

We have made both our application (with our implementation of DISSEMINATE) and our WiFi-Direct device-to-device abstraction layer available publicly, alongside a video showing the DISSEMINATE app in action[3].

III. DEMONSTRATION

The nature of the demonstration follows fairly intuitively from the descriptions in the previous sections. When the demonstration application starts, the user will see the screen depicted to the left of Fig. 2 on each device on which the app starts. If the user for one of the devices taps one of the grids of bricks, a "download" of that file from the (simulated) infrastructure will begin. The participants will be able to tune the rate at which the download progresses, simulating both high and low quality cellular data connections. Because all of the active devices are subscribed to all of the data items, as the chunks of the data item "download" they will immediately be shared with any connected devices. The participants will observe the grids representing the chunks for each data item fill in as chunks are received. When an entire data item has been received, the media file will be displayed.

Using the DISSEMINATE app, the participants will be able to experiment with different settings to observe when cooperating dissemination is effective and when it is not as effective. For instance, the participants will be able to test what happens when multiple devices download the same file from the infrastructure at very low cellular connection speeds and then share the chunks as they download (i.e., cooperative download will benefit *all* users).

We will have available devices the participants can use during the demonstration; alternatively, a user can use his or her own WiFi-Direct capable device, assuming the user loads the DISSEMINATE APK and joins the WiFi-Direct group.

IV. TECHNICAL REQUIREMENTS

This demonstration has no special technical requirements. We will supply our own Android devices. Visitors with WiFi-Direct capable Android devices should also be able to participate in the demonstration using their own devices. Access to power for charging our devices would be preferable.

REFERENCES

[1] B.Han, P. Hui, V.S.A. Kumar, M.V. Maranthe, J. Shao, and A. Srinivasan. Mobile data offloading through opportunistic communications and social participation. *IEEE Trans. on Mobile Computing*, 11(5):821–834, 2011.

[2] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragoul, and A. Markopoulou. MicroCast: Cooperative video streaming on smartphones. In *Proc. of MobiSys*, pages 57–70, June 2012.

[3] M.Z. Shafiq, L. Ji, A.X. Liu, J. Pang, S. Venkataraman, and J. Wang. A first look at cellular network performance during crowded events. In *Proc. of SIGMETRICS*, pages 17–28, June 2013.

[4] M. Skjegstad, F.T. Johnson, T.H. Bloenaum, and T. Maseng. Mist: A reliable and delay-tolerant publish/subscribe solution for dynamic networks. In *Proc. of NTMS*, pages 1–8, May 2012.

[5] V. Srinivasan and C. Julien. Demo: MadApp: Dynamic content support for delay-tolerant web applications. In *Proc. of PerCom (Demonstrations)*, 2014.

[6] V. Srinivasan and C. Julien. MadApp: A middleware for opportunistic data in mobile applications. In *Proc. of MDM*, 2014.

[7] J. Su, J. Scott, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton. Haggle: Seamless networking for mobile applications. In *Proc. of Ubicomp*, pages 391–408, Sept. 2007.

[2] On Android 4.4, the WiFi-Direct broadcast address is hardcoded at 192.168.49.255.

[3] http://mpc.ece.utexas.edu/research/disseminate