Demo: MadApp: Dynamic Content Support for Delay-Tolerant Web Applications

Venkat Srinivasan and Christine Julien
Center for Advanced Research in Software Engineering
The University of Texas at Austin
Email: venkat.s@utexas.edu, christine.julien@mail.utexas.edu

***Abstract*—This paper showcases MadApp, an application-level development framework that supports the expressive development and flexible deployment of applications for web applications in delay-tolerant networks. MadApp allows web applications to be developed in two pieces. First, a static piece that contains unchanging content can be downloaded from a traditional web server. This static piece may specify “holes” that designate places in which dynamically collected content can be integrated into the web application as this content is opportunistically collected from the pervasive computing environment. MadApp supports both the development of these web applications by making is easy to specify this missing content and how it is dynamically integrated and the deployment of the web applications by providing middleware support for collecting and integrating content on the fly. This demonstration showcases how MadApp can be used to support such opportunistic web applications through a webpage integrated with the demonstration event itself. Users will be able to download a webpage that contains a static map and a listing of the demonstrations. As users move through the demonstration space, they can generate content, e.g., photos of demonstrations, which their app will then share opportunistically with others in the exhibition space.**

I. Introduction

> *You arrive at a conference exhibition event and download (from the Internet) a schedule and map of the venue. As attendees wander around the event, they generate photographs and reviews of the available demonstrations. As you wander, opportunistically encountering other attendees, your view of the schedule and map is dynamically updated to include new information, including these snapshots and reviews, allowing you to adjust your plan to target the exhibitions that are of the most interest to you.*

Scenarios like this abound in pervasive computing, but building an application that can tolerate such dynamic content shared via opportunistic peer-to-peer connections using today’s programming tools is complex. We have built MadApp [11], a middleware that supports the development of this style of delay-tolerant application. Using MadApp’s programming constructs, developers can easily create applications that seamlessly handle frequent disconnections, significant uncertainty in data and resource availability, and extreme delays in content arrival. MadApp allows web applications to specify the types of *content* of interest, and, as data arrives, the MadApp middleware demultiplexes it to already executing applications. MadApp provides natural and intuitive programming constructs integrated into common web application development frameworks. These programming constructs make it simple for developers to enable the application-layer to flexibly incorporate dynamic information received on-the-fly with unpredictable delays. While its conceptual model is platform-independent, our implementation of MadApp is integrated with the Django web framework, making it simple and straightforward for application developers to use.

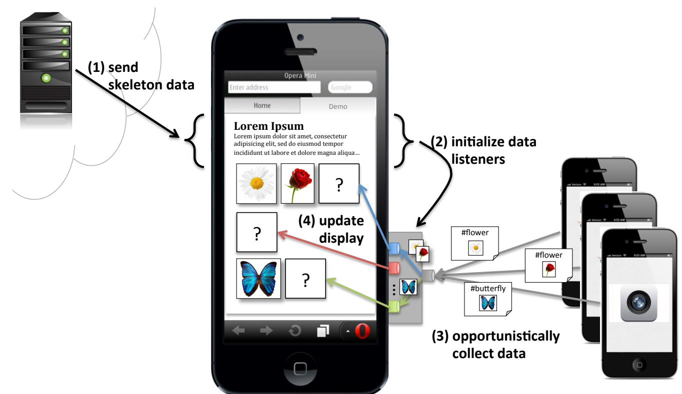

Fig. 1. MadApp Conceptual Model

Fig. 1 shows MadApp’s conceptual model. The user’s device first downloads the static content from an ordinary web-server in the Internet. This static content (e.g., a webpage) comes with “holes” that indicate the type of expected information. MadApp then listens to receive this information from any opportunistically connected “peer” devices that may become connected in the immediate environment. As these peers share relevant information, MadApp sends the received content along to the registered applications, which can, for example, update the content displayed to the user.

In addition to our example scenario, the MadApp approach is widely applicable, for example in developing regions, where asynchronous web page interactions have been shown to be preferable when connectivity to the Internet is slow and unreliable [5], [6]. MadApp sits in the context of existing approaches applied to opportunistic networks, for example approaches that use a combination of existing infrastructure and peer-to-peer interactions [2], [3] or that use proxies and prefetching to support browsing in delay-tolerant networks [1],

[4], [10]. Other approaches have shown how to use peer-to-peer connections to opportunistically *route* (usually multimedia) content [9], [12]–[15] or to manage connections when the opportunistic network experiences a high degree of churn [8].

In this context, the key innovation behind MadApp is that the content available to the web application can be received across multiple *delivery vectors* without the application itself having to handle or even be aware of the different modalities. This entails novel contributions at two levels: (1) MadApp uses widely adopted web application strategies to support collecting content from multiple delivery vectors and (2) MadApp provides simple and intuitive programming constructs that focus on enabling average web programmers to create these highly flexible applications. In the most likely case (and the one used as part of this demonstration), the basic (static) information for a web page can be received in the traditional manner by downloading it from an ordinary web server in the Internet, while dynamic media content (e.g., pictures, videos, and live updates) can be acquired directly from peer devices that are encountered in the physical environment.

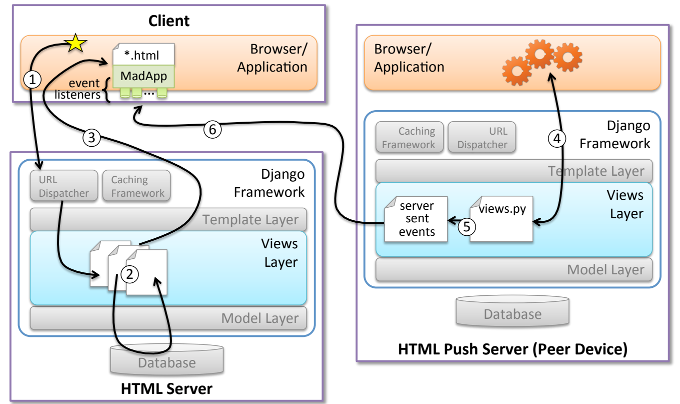

Fig. 2. Architecture of Django-based implementation. A client requests a MadApp enabled webpage (step 1), which is retrieved via Django in the traditional way (step 2). The webpage is delivered to the client and loaded on top of MadApp (step 3). Asynchronously, peer devices' push serves generate content, which is passed into the peer device's Django views layer (step 4), encapsulated in a server-sent event (step 5) and sent to the client device using the server-sent event implementation (step 6).

II. Implementation

Our implementation of MadApp builds on the widely used Django web framework to enable smooth transition for otherwise ordinary web developers. In the implementation we demonstrate, MadApp assumes a statically defined set of peer devices. We do not assume that the client device is persistently connected to all peers all the time; instead MadApp connects to peers as they become available and can disconnect and reconnect to peers as mobility dictates. This is for demonstration purposes; the MadApp middleware builds on approaches in the literature that can discover available connected devices in the pervasive computing environment, e.g., via Bluetooth or Wi-Fi connections [7], which can easily replace our static neighbor definition component.

Fig. 2 shows the concrete implementation of the MadApp architecture, which is distributed across three types of physical components. The *Client* (in the upper left of the figure) provides the user-facing web application, in our example and demonstration, via any browser app on the device. The *HTML Server* (in the lower left of the figure) is an ordinary Django web server running in the Internet. The *HTML Push Server* runs on a peer device and provides dynamic content to fill in the client's browsing experience. A MadApp instance may consist of many clients and push server devices all opportunistically connecting and disconnecting from each other.

A MadApp interaction begins on the client, who requests a MadApp-enabled webpage. This webpage is downloaded in the standard way from the HTML server through the Django framework. However, when the MadApp-enabled webpage arrives at the client, the MadApp middleware detects any "missing" content in the received webpage. This missing content is indicated by the web application developer through the use of *hashtags* that indicate the type of content desired and are associated with *callbacks* that execute when any content matching the hashtag is received. In the simplest case (and the one used in our demonstration application), this callback simply adds the received content to the displayed webpage in a location and manner designated by the webpage developer.

Users of the push server devices may opportunistically generate content (e.g., by taking photographs of an event). The user then makes this content available to the MadApp push server through its sharing interface. Then the MadApp HTML push server implementation, upon discovering a peer, simply pushes this content to the newly connected peer. On the peer (i.e., the client), the received information is then passed along to the registered application.

MadApp enabled webpages can be viewed on most common web browsers; thus the "client" depicted in Fig. 2 can be hosted on any device without requiring any specialized functionality. The HTML server in Fig. 2 is a standard Django web server. In our demonstration (described in more detail in Section III), we host our own web server that serves up MadApp enabled webpages. For the purposes of this demonstration, we created an implementation of the MadApp push server as an Android application. When this demonstration application launches, it displays a graphical interface to the user to allow him to create and share content; in our demonstration, a user can take a photo with the device's camera, tag the photo with a keyword, and designate it to be shared.

To achieve the full push server functionality, the application must also launch the Django web framework in the background. This is necessary to support the server sent events[1] that must move from the peer device directly to the client device. Basically, the Android application becomes a small web server that can push shared content over local wireless network connections. To accomplish this, we used the SL4A[2] python interpreter. When it starts, the MadApp

[1] https://github.com/niwibe/sse

[2] http://code.google.com/p/android-scripting

Android application launches both the graphical user interface described above and the python interpreter. It loads the Django python implementation into the interpreter, and, when new content is generated, the `views.py` implementation grabs the new content and sends it to any connected peer devices. The push server also keeps track of what content it has sent to which other client devices. When a new client device connects, the push server sends that client any content that it has not yet sent to that client.

On the client side, the javascript definitions embedded in the MadApp enabled webpage displayed in the client's browser automatically catch any received content and distribute it to any "listeners" in the webpage that are waiting for content with the received tags. The client's webpage then updates to display the newly received content. As described previously, the power of MadApp comes from the fact that the live content sharing does not require devices to coordinate across the Internet; instead the peer-to-peer content sharing can be supported directly by technologies like Wi-Fi Direct[3] or LTE Direct[4]; in our demonstration, we use 802.11 ad hoc connections for this peer-to-peer content sharing simply because this modality is supported by the largest number of currently available devices.

III. Demonstration

We have studied our implementation of MadApp using a simple "tourism" webpage that displays photos of attractions shared by nearby tourists [11]. To demonstrate its capabilities and potential, we developed a second demonstration that uses the conference venue and events directly. The demonstration will execute during the conference's demonstration session and consist of three components:

1) We will run (locally, on a laptop) a Django web server that will host a MadApp enabled webpage that lists the demonstrations and posters that can be viewed during the session. If the information is available, the webpage can also provide a map of the contributions' display locations. Because it is a MadApp enabled webpage, the page will be "missing" content, with the intent that it will be filled in dynamically, as it is generated.
2) Any device with a web browser (regardless of operating system) should be able to download and display this webpage. Therefore any of the conference visitors can access the live, dynamic page providing details about the available demonstrations and posters. However, to showcase the true use cases of MadApp, these devices will have to join a local device-to-device network (via Wi-Fi) that will allow the content to be distributed without the support of any infrastructure (i.e., wireless access points or other means of accessing the Internet).
3) We will have available a handful of our own Android devices that visitors can pick up and use as they walk around the conference venue. These devices will be dedicated push servers, and the users of these devices will be expected to take photographs of the contributions to be shared with other client devices displaying the MadApp webpage in their browsers. We will also make the MadApp Android application available for download so that other users with Android devices can launch the push server on their own devices if they desire to.

[3]http://www.wi-fi.org/discover-and-learn/wi-fi-direct
[4]http://www.qualcomm.com/research/projects/lte-direct

IV. Technical Requirements

While the demonstration will be targeted to run using a locally defined network that does not require the Internet, the availability of a wireless access point would be an enabler of a richer demonstration experience (showing downloading of the MadApp enabled webpage from the Internet) and allowing visitors to download the MadApp Android application to use on their own device if they choose.

Beyond networking capability, we need a reasonably sized and prominently placed space to attract users and explain the project. There are no other particularly special requirements.

Acknowledgments

This work was funded, in part, by the National Science Foundation (NSF), Grant #CNS-0844850 and the Department of Defense (DoD), Grant #H98230-12-C-0336. The views and conclusions herein are those of the authors and do not necessarily reflect the views of the sponsoring agencies. The authors would like to thank Agoston Petz for his initial work on the project and input on the implementation.

References

[1] A. Balasubramanian, B. Levine, and A. Venkataramani. Enhancing interactive web applications in hybrid networks. In *Proc. of MobiCom*, pages 70–80, 2008.
[2] A. Balasubramanian, Y. Zhou, W. Croft, B. Levine, and A. Venkataramani. Web search from a bus. In *Proc. of CHANTS*, 2007.
[3] S. Chava, R. Ennaji, J. Chen, and L. Subramanian. Cost-aware mobile web browsing. *IEEE Pervasive Computing*, 11(3):34–42, 2012.
[4] B. Chen and M. Chan. MobTorrent: A framework for mobile internet access from vehicles. In *Proc. of INFOCOM*, pages 1404–1412, 2009.
[5] J. Chen, S. Amershi, A. Dhananjay, and L. Subramanian. Comparing web interaction models in developing regions. In *Proc. of DEV*, 2010.
[6] J. Chen, L. Subramanian, and J. Li. RuralCafe: Web search in the rural developing world. In *Proc. of WWW*, pages 411–420, 2009.
[7] T. Clausen, C. Dearlove, and J. Dean. Mobile ad hoc network (manet) neighborhood discovery protocol (nhdp). IETF RFC 6130, http://xml2rfc.tools.ietf.org/html/rfc6130, 2011.
[8] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: Vehicular content delivery using WiFi. In *Proc. of MobiCom*, 2008.
[9] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. PROMISE: Peer-to-peer media streaming using CollectCast. In *Proc. of MM*, 2003.
[10] F. Malandrino, C. Casetti, C. Chiasserini, and M. Fiore. Content downloading in vehicular networks: What really matters. In *Proc. of INFOCOM*, pages 426–430, 2011.
[11] V. Srinivasan and C. Julien. MadApp: A middleware architecture for delay-tolerant web applications. Under Review, PerCom 2014.
[12] D. Tran, K. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proc. of INFOCOM*, pages 1283–1292, 2003.
[13] J. Wu, Z. Lu, B. Lu, and S. Zhang. PeerCDN: A novel P2P network assisted streaming content delivery network scheme. In *Proc. of ICCST*, pages 601–606, 2008.
[14] M. Zhang, J.-G. Luo, L. Zhao, and S.-Q. Yang. A peer-to-peer network for live media streaming using a push-pull approach. In *Proc. of MM*, pages 287–290, 2005.
[15] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proc. of INFOCOM*, pages 2102–2111, 2005.