# XD (Exchange-Deliver): A Middleware for Developing Device-to-Device Mobile Applications

Tomasz Kalbarczyk and Christine Julien
The University of Texas at Austin
{tkalbar, cjulien}@utexas.edu

## ABSTRACT

In this demonstration, we showcase the XD middleware, a framework for expressive multiplexing of application communication streams onto underlying device-to-device communication links. XD allows applications to remain agnostic about which low-level networking stack is actually delivering messages and instead focus on the application-level content and delivery parameters. The IoT space has been flooded with new communication technologies (e.g., BLE, ZigBee, 6LoWPAN) to add to those already available on modern mobile devices (e.g., BLE, WiFi-Direct), substantially increasing the barrier to entry for developing innovative IoT applications. XD presents application developers with a simple publish-subscribe API for sending and receiving data streams, unburdening them from the task of selecting and coordinating communication channels. Our demonstration shows two Android applications, Disseminate and Prophet, running using our XD middleware for communication. We implemented BLE, WiFi Direct with TCP, and WiFi Direct with UDP communication stacks underneath XD.

## 1. INTRODUCTION

New and varied types of IoT (Internet of Things) devices are popularizing a variety of communication technologies (e.g., BLE, ZigBee, 6LoWPAN) and introducing proprietary ones. Modern mobile devices (e.g., tablets and smartphones) are following a similar trend in supporting device-to-device (D2D) communication using technologies like BLE and WiFi Direct. In the IoT, supporting D2D communication mechanisms will become critical, as the nature of interactions among devices take on intuitively local and device-to-device characteristics. To successfully build sophisticated mobile IoT applications that use such a wide array of technologies requires a way to seamlessly communicate across the diverse communication technologies. Our XD (Exchange and Deliver) middleware allows just this, while also enabling applications to be agnostic of the implementation's use of specific underlying communication technologies.

*ICSE '16 May 16-17 2016, Austin, TX, USA*
© 2016 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4178-3/16/05.
DOI: http://dx.doi.org/10.1145/2897073.2897706

To show mobile IoT applications' need for XD, we use a concrete scenario to highlight some of the challenges in adding D2D support to existing networked applications. Imagine an application for securely viewing medical health records that avoids storing health records at some central server (cloud or otherwise), instead allowing individual patients to make their own records available only on a need-to-know basis. When a patient walks into a doctor's office, the nurse needs to see a subset of his records while the doctor needs to see a more comprehensive set. Further, the patient's policy for releasing records may require the doctor (or nurse) to be co-located (e.g., in the same exam room) as the patient.

From an IoT / D2D perspective, we can reduce the scenario to contain four agents: the patient's device (e.g., smartphone), the nurse's tablet, the doctor's tablet, and a location tag within the exam room. For simplicity, assume the patient has his own medical records on his device. When the patient walks into the exam room, his medical records become available to the tablet of the doctor or nurse if they are co-located. In this example, it is reasonable that identification information would be transmitted over a lightweight communication technology such as BLE since this information may be continuously broadcast to enable discovery of neighboring doctors, nurses, and patients promptly. Once a target (e.g., the nurse's or doctor's tablet) is identified, the needed medical records are transferred using a technology designed for higher throughput such as WiFi Direct. This represents an ideal situation, where multiple communication technologies are available among the three devices. However, imagine that the patient is using a device that only has WiFi Direct available for D2D interactions. We still want the patient to be able to use the application, so we need the tablets to transfer identification information over WiFi Direct instead of BLE. As another complicating factor, the location tag in the exam room may broadcast over ZigBee, requiring a yet different communication capability.

To implement this flexibility, the application developer has to incorporate the communication stacks for BLE, WifiDirect, and ZigBee directly into the application. Further, the developer has to create the logic that selects which pieces of content are exchanged using which communication technology, which must also depend on the capabilities of the devices in the surroundings. When two devices have multiple matching technologies, the developer has to choose which one is best (e.g., BLE for identification information, and WiFi Direct for medical records). Mismatches must be identified (e.g., lack of BLE), and a secondary technology must be chosen (e.g., use WiFi Direct for both identification information and medical records). XD takes these burdens from application developers and provides a middleware that seamlessly adapts to the available communication capabilities based on the applications' high level requirements (e.g., the specific coordination required and the data being shared) and the state of the networked environment (e.g., the available communication technologies, other ongoing communication that may interfere). In this way, XD prevents the developer from being mired in the weeds of D2D communication and instead allows him to focus on what data to send and how to interpret data that is received.

In employing XD, the content that applications share is divided into *context* and *data*. Context information can vary in semantics, but it is typically a lightweight summary of a device's situation or knowledge. In XD, context is broadcast periodically to all neighboring devices. Data, on the other hand, is the actual application-level content being intentionally shared among devices. Data is expected to be heavyweight, and when and to whom it is sent is determined by the application logic (potentially based on the context). Based on the technologies available on the sending device and other nearby neighbors and the nature of a particular piece of content, XD automatically chooses one or more communication mediums over which to transmit. To delegate this behavior to XD, applications employ a simple publish/subscribe interface. Specifically, the application *subscribes* to content it is interested in, allowing XD to then coordinate with other neighboring devices to retrieve that information. The application triggers periodic publication of context information by setting the context in XD and publishes data on-demand using XD's send interface.

To demonstrate XD, we have implemented the middleware on the Android operating system and used it to support our own Disseminate application [8], and also the Prophet DTN Router [5]. In our prototype implementation, we have used Bluetooth Low Energy (BLE), WiFi Direct + TCP, and WiFi Direct + UDP as underlying communication technologies. Conceptually, XD could also incorporate other common IoT communication techniques (e.g., Zigbee, 6LoWPAN) or even proprietary protocols, allowing IoT applications that integrate devices from multiple vendors under a single umbrella. This demonstration showcases XD's ability to raise the level of abstraction of mobile IoT application programming, easing the burden on developers for this emerging marketplace of applications.

## 2. RELATED WORK

The XD middleware fits in the space of frameworks and architectures for opportunistic networks, including work involving data distribution using device-to-device communication [7, 8, 4]. Most middleware facilitating dD2D communication assume a single underlying communication technology. MobiClique [6], for instance, uses social information to exchange D2D information solely using underlying Bluetooth connections. Yarta [10] and CAMEO [1] focus on the social patterns of interaction while shielding the application from the details of handling low-level communication technologies. In fact, these middleware presuppose that something like XD is available to intelligently exploit the available opportunistic connections that the models rely on.

Specifically, Haggle [9] is designed to separate application logic from communication bindings, so that, similarly to XD, applications are agnostic of underlying communication technologies employed. Our work with XD is orthogonal to Haggle, and could even be integrated into the Haggle framework. Similary, delay-tolerant networking stacks [2] abstract away the underlying communication technology and handle challenges such as opportunistic connections. Other frameworks for building applications in opportunistic, D2D networks focus on abstracting the network structure so that it is simpler to develop applications that can communicate in D2D environments. For example, DTP [3], a distruption-tolerant, reliable transport layer protocol masks communication issues present in a typical D2D network, and provides the application a more traditional view of the network, providing the semblance of TCP semantics.

Generally, like XD, these approaches facilitate the exchange of content using a heterogeneous set of D2D protocols. However, they lack the essential logic necessary to seamlessly share contextual information among neighbors and to use this context in device discovery and in selecting the appropriate communication technology for exchange.

## 3. APPLICATIONS

To demonstrate the use and effects of XD, we developed two versions of two applications. For each application, one version uses the XD middleware to facilitate all device-to-device communication, while in the other, the application directly implements communication using a single selected communication technology.

The first application is a rework of our own device-to-device communication application, Disseminate [8]. Because we built the original version of this application, reimplementing it with XD allows us to show a direct comparison of the application's utility (as far as being able to use a wider range of communication technologies) and performance (in terms of transmission rates). Disseminate allows Android applications to share large media files downloaded from the Internet by breaking the (very) large data items into small chunks and then opportunistically sharing the chunks among co-located mobile devices. Interested devices issue subscriptions for the chunks they need and advertisements for the chunks they can share. In mapping Disseminate onto XD, deciding what content should be periodic context versus bulky data is straight-forward. The subscriptions and advertisements for chunks are packed as context information that is continually broadcast to neighboring peers. Peers then use this context information to initiate transfers of the actual chunks of media data.

Our second application is an implementation of the Prophet routing protocol for Delay Tolerant Networks (DTNs) [5]. Prophet is a widely used, probablistic routing protocol in which each device maintains a history of successful content delivery to known destinations in the network; that is, for each potential peer, a Prophet node maintains a *delivery predictability.* Whenever a device encounters another device, they exchange and update their known delivery predictabilities. Devices then use this predictability information to determine whether a neighboring peer is a good candidate for forwarding a piece of application data targeted to another peer in the network. Simply, if a Prophet node determines that one of its peer devices has a higher delivery probability for the destination of a piece of data, the node forwards the application data to the peer with the expectation that it will reach its final destination through that neighbor. Developing Prophet on XD is also straightforward, since it is again straightforward to split content into context (delivery predictabilities) and data (forwarded application packets).

This style of split of content into *context* and *data* is not specific to these two applications. In fact, we find that many device-to-device applications have some small bits of control data that are exchanged frequently and some larger pieces of application data that are exchanged much less frequently and often only with specific targeted neighbors. As described next, XD takes advantage of this split in the application content to effectively and efficiently leverage available underlying communication technologies.

## 4. ARCHITECTURE / IMPLEMENTATION

XD divides the components of an IoT interaction into the content, the application, and the communication technology. Since XD sits between the application and the communication technology, it provides two sets of APIs. The first is a public publish-subscribe interface that allows an application to receive and send content. The second is an internal interface that each communication technology is required to implement in order for it to be selected for content distribution. In the section, we discuss how XD organizes the content, how it allows applications to operate on this content, how it transmits the content using multiple communication technologies, and the logic necessary to bridge the applications and the communication technologies. Figure 1 provides a visualization of the architecture that shows the high level modules that make up XD, and depicts how the middleware interacts with applications above and communications technologies below.

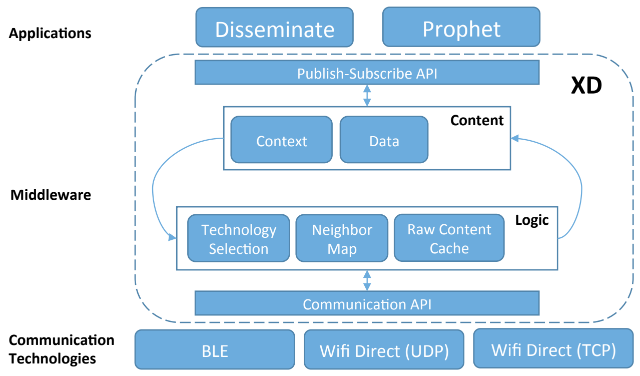

**Figure 1: XD Middleware Architecture**

**Content.** As motivated above, XD separates content into two categories: context, which is lightweight information that is continually exchanged among peers, and data, which is the actual heavyweight, application content that is delivered to specific peers (or subsets of peers). Applications commonly use the context to help select the data to send. We allow the application to perform three primary actions on each type of content: specifying optional parameters (e.g., maximum content size, broadcast frequency, transmission rate), determining the content to distribute, and implementing a callback to process the content that has been received. XD stores the received context and also uses it to update its internal Neighbor Map (see Figure 1). Data is the application-level information shared between devices. Our interface allows for the specification of what exact data will be transmitted and to whom, based on application logic that may reflect received context information.

**Application.** The application component consists of high level IoT or D2D applications that conform to a simple publish-subscribe interface. The application subscribes for content of interest and publishes content that it wants to share. The XD middleware filters incoming content so that the application only sees content to which it is entitled and for which it is subscribed. The published content may be user generated, automatically collected, or retrieved through other means, e.g., from other device-to-device applicaiton interactions. The API is broken down into four routines: `receiveContext`, `receiveData`, `sendContext(parameters, context)`, and `sendData(destination, data)`.

The API is best explained using an existing application like. In the "application layer", Prophet implemented on top of XD provides a `receiveContext` callback that subscribes the application to receive delivery predictabilities from other devices also running the Prophet application. Similarly, the application implements the `receiveData` callback in order to process all payloads received from other devices (i.e., messages targeted for the node or messages for which the peer has determined that the node is a better forwarder). The Prophet XD implementation uses received context to update its local representation of the known delivery predictabilities; it then uses the `sendContext` function to update the delivery predictabilities that are broadcast to neighbors. Finally, the Prophet implementation uses the `sendData` function to deliver payloads to specific devices, based on Prophet's internal logic. Note that while XD does not explicitly manage forwarding data through D2D links, the API provides a simple way for an application to do so[1]. In the case of Prophet, all that needs to be done to forward a payload is to call the `sendData` function from within the implementation of the `receiveData` callback. This makes XD suitable even for complex IoT applications that might involve multihop communication.

**Communication Technology.** XD uses an internal interface to facilitate the addition of an underlying communication technology that can be used for context information sharing, data distribution, or both. Depending on the type of technology, the stack can include capabilities such as the maximum payload size, the range of broadcast frequencies, the range of transmission rates, and the ability to send and/or receive content. For our demonstration, we have implemented three communication stacks in Android: Bluetooth Low Energy (BLE), WiFi Direct over TCP, WiFi Direct over UDP.

Starting with Android 5.0 Lollipop, Android devices have begun supporting BLE operation in peripheral mode, meaning that they can send broadcast beacons in addition to receiving beacons. Our BLE implementation has two threads running simultaneously: one scans for incoming BLE beacons, and the other broadcasts BLE beacons to all neighbors. The scan thread receives beacons and places them into a synchronous queue (Raw Content Cache in Figure!1), that is subsequently polled by a receiver thread that wraps each beacon and presents it to the application as either context or data. The broadcast thread transmits a beacon at

[1]In this discussion we refer to Prophet as an "application" even though it is a routing protocol; this is XD's perspective; the Prophet implementation interacts with XD like any other "application" employing XD's interfaces

the interval indicated by the application parameters if BLE was chosen as a context distributor by the selection module.

While BLE is often a good choice for exchanging lightweight context, the beacons are much too small for bulky data delivery. We also include WiFi Direct as a communication technology. WiFi Direct using UDP sockets is also often a reasonable choice for broadcasting context information, though it can be inefficient due to the way that WiFi Direct groups are structured around 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, drastically reducing broadcast data throughput. WiFi Direct is much better suited towards delivering bulky data by forming temporary one-to-one connections through TCP sockets. Similarly to the BLE beacons, all content that is received over WiFi Direct is placed into XD's Raw Content Cache for wrapping and presentation to the application. The communication stacks which we implemented for usage with XD are far from the only ones, and wrapping other stacks for use by XD is straightforward. The complexity lies in writing the communication stacks themselves.

**Middleware Logic.** The primary logic within the middleware is the Technology Selection Module, which is responsible for determining over which communication technology a particular piece of content (context or data) should be sent. Two sets of information are used to make this decision: the optional parameters specified by the application (the size of the content, frequency of broadcast and desired transmission rate) and the technologies currently in use by devices in the neighborhood. To facilitate determining what these technologies are, each device periodically sends an address beacon (initially this is transmitted using the least expensive communication technology in terms of energy usage). The beacon contains a set of tuples (communication technology, address) whose purpose two fold: to allow the selection module to choose a communication technology and to provide the address for transmitting bulky data to specific neighbor (or set of neighbors) once the technology has been chosen. Each instance of the XD middleware maintains a neighbor map that is continuously updated with the tuples transmitted by such address beacons.

## 5. DEMONSTRATION

To showcase the XD middleware, we will show participants two versions of the Disseminate application and the Prophet application. One version uses XD while the other uses only WiFi Direct over UDP. Prior to each run of each application, the participant will be encouraged to toggle either Bluetooth or WiFi on or off. Depending on which communication technologies are available, the participant will observe visible changes in the applications' communication rates (media exchange rates in Disseminate; packet forwarding successes in Prophet) when using the version of the application that uses XD.

This visible download rate only scratches the surface in terms of showing how XD improves the process of developing device-to-device applications. We will also provide a larger display that shows real-time performance statistics such as energy consumption and transmission rates, giving participants a more complete picture of the benefits to using XD. We will also display the application code for both versions so that participants can see the substantial difference in code complexity when using XD and when not.

## 6. FUTURE WORK

Our current implementation of XD has room for extension and improvement. While we include a technology selection module, it currently only uses application parameters and address beacons to select which communication technology to use. The selection algorithm does not yet incorporate features such as broadcasting context over multiple technologies, adjusting beacon frequencies to prevent congestion, or buffering data so that fewer D2D connections need to be made. Additionally, more intelligent selection could be performed by passively sensing contextual information that might not be explicitly relate to the XD-based application. This is particularly relevant since the IoT is predicated on the presence of many sensors providing such context. Finally, although our prototype of XD is implemented as a middleware, it could (and likely should) be integrated into the operating system as an abstract system communication service, making it even more efficient. For example, currently on Android, to use Bluetooth or WiFi Direct, XD is forced to use application layer interfaces that reduce efficiency since information needs to travel up and down the stack. Moreover, only out of the box communication technologies are available meaning that any kinds of lower layer security features (for example, masking the MAC address of a beaconing device) are not possible.

## Acknowledgments

This work was funded in part by the NSF, Grant #CNS-0844850 and the DoD, Grant #H98230-12-C-0336.

## 7. REFERENCES

[1] V. Arnaboldi, M. Conti, and F. Delmastro. Cameo: A context-aware middleware for opportunistic mobile social networks. In *Proc. of WoWMoM*, 2011.

[2] M. Doering et al. Ibr-dtn: An efficient implementation for embedded systems. In *Proc. of CHANTS*, 2008.

[3] Y. Go, Y. Moon, G. Nam, and K. Park. A disruption-tolerant transmission protocol for practical mobile data offloading. In *Proc. of MobiOpp*, 2012.

[4] L. Keller et al. MicroCast: Cooperative video streaming on smartphones. In *Proc. of MobiSys*, 2012.

[5] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7(3):19–20, July 2003.

[6] A.-K. Pietiläinen et al. Mobiclique: Middleware for mobile social networking. In *Proc. of WOSN*, 2009.

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

[8] V. Srinivasan, T. Kalbarczyk, and C. Julien. Disseminate: A demonstration of device-to-device media dissemination. In *Proc. of PerCom (Workshops)*, 2015.

[9] J. Su et al. Haggle: Seamless networking for mobile applications. In *Proc. of Ubicomp*, 2007.

[10] A. Toninelli, A. Pathak, and V. Issarny. Yarta: A middleware for managing mobile social ecosystems. In *Proc. of GPC*. 2011.