ChronoChat on Android

Tyler Vernon Smith, Alexander Afanasyev, Lixia Zhang

***Abstract*—In this report I discuss my Master's capstone project, *ChronoChat-android*, an implementation of the ChronoChat instant message application for Android mobile devices *[7]*. ChronoChat-android allows communication in a chatroom cooperatively hosted by ChronoChat clients. As in other ChronoChat implementations, chatroom participants may be connected via an NDN hub; ChronoChat-android also supports ad-hoc communication between mobile devices running NFD-android with Wi-Fi Direct support *[1]*. This report will review the design and features of ChronoChat-android, focusing on aspects that are of particular importance to the Android platform. It will also compare ChronoChat-android to another Android app—NDN Whiteboard *[3]*—and to the web-based ChronoChat-js implementation *[11]*.**

1 Introduction

The motivating use case for ChronoChat-android is to provide an example of ad-hoc messaging utilizing NDN and Wi-Fi Direct. Contrary to my initial expectations, supporting this type of ad-hoc connection was not the primary challenge in development: because recent versions of NFD-android already support Wi-Fi Direct connections to other devices, ChronoChat-android only needed to support connections through a local NFD-android instance to achieve ad-hoc chatroom support. However, adapting the programming model provided by ChronoChat-js *[11]* to the Android platform proved more difficult than anticipated.

As my introduction to ChronoSync usage *[12]*, *[8]*—and, in fact, as my introduction to Android programming in general—I referred to NDN Whiteboard *[3]*. However, I came to decide that the architecture of NDN Whiteboard could be improved; I aimed to structure ChronoChat-android in a more "Android-style" way, which this report will detail.

2 Using ChronoChat-android

First I will give a high-level overview of ChronoChat-android from the end user's perspective. To begin using ChronoChat-android, the user must provide a screen name, a chatroom name, and an NDN prefix for chatroom data (Fig. 1). (ChronoChat-android always connects though a local NFD-android, rather than asking the user for a hub URI as in ChronoChat-js. The user should ensure that the NFD daemon has been started in the NFD-android app before signing in.)

After login, the user enters the chatroom (Fig. 2). After ChronoChat-android connects to the chatroom, other connected clients will appear to "join" the chat. The user

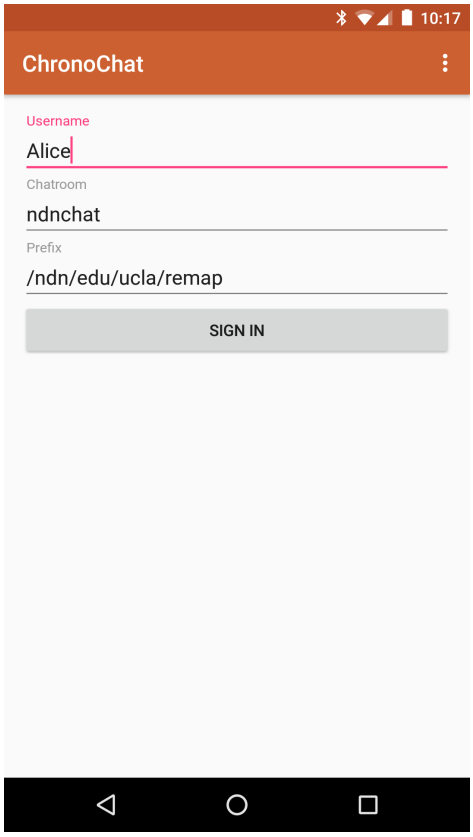

Fig. 1. Login screen.

can send messages using the field at the bottom of the chatroom window. The toolbar at the top may be used to leave the chatroom (by tapping the arrow at left), view the roster (second icon from right—see Fig. 3), or quit the app (accessed through the menu at right).

ChronoChat-android displays an ongoing notification while connected to a chatroom (Fig. 4). This helps prevent the Android OS from terminating the app while it is running in the background. The app will also alert the user via a notification when messages are received while the app is running in the background.

3 Software architecture

ChronoChat-android consists of two Android *Activities* and one *Service* as its primary components. The central component—i.e., the component launched when the user first opens ChronoChat-android, and which generally coordinates most user-facing tasks—is *MainActivity*. When first launched,

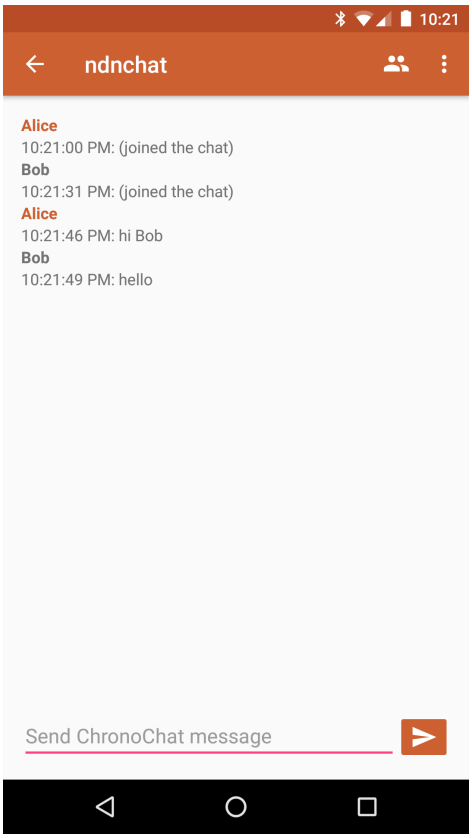

Fig. 2. Chatting.

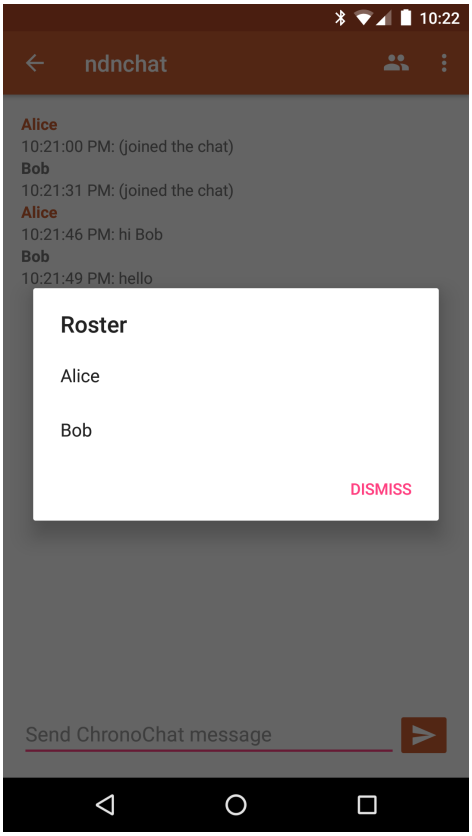

Fig. 3. Viewing the chatroom roster.

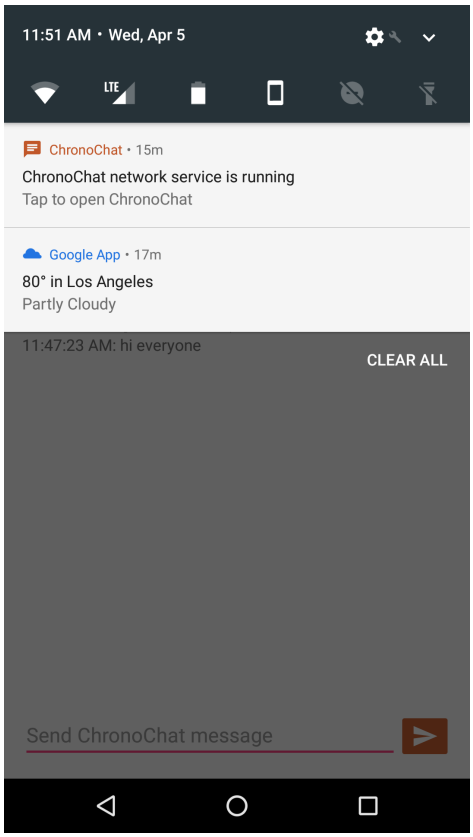

Fig. 4. Ongoing service notification.

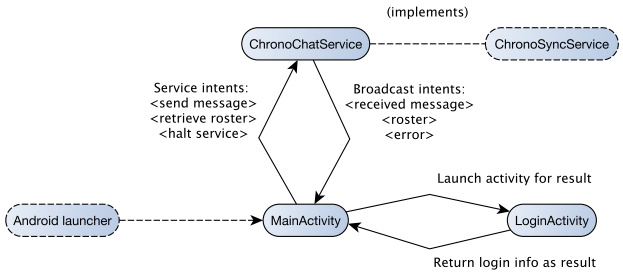

Fig. 5. Overview of primary application components.

MainActivity will discover that no login information has been set, and launch *LoginActivity* to request it from the user. This will display the screen shown in Fig. 1. When the user signs in, LoginActivity will return the information to MainActivity, which will then display the chatroom screen shown in Fig. 2.

Network tasks are handled by the *ChronoChatService* component. Communication between MainActivity and ChronoChatService consists of Intents, which are the usual means of interprocess (and "inter-component") communication in Android. When MainActivity signals ChronoChatService to send a message, the service starts up, establishes a connection to the chatroom, and publishes the data via ChronoSync. It remains running as a *foreground*

service—indicated by the ongoing notification shown in Fig. 4—to prevent the Android OS from killing the ChronoChat-android process whenever the user is not interacting with the app. The service maintains its connection to the chatroom and relays any received messages to MainActivity via Intents, while also keeping track of the chatroom roster and periodically sending "heartbeat" messages to the chatroom. MainActivity may also send Intents to retrieve the current roster from the service, or direct it to shut down.

Contrary to what the terminology may suggest, an Android "Service" such as ChronoChatService does not exist in a separate process or thread *[5]*. To avoid blocking the main "UI thread," ChronoChatService spawns a "network thread" on which it performs nearly all of its work aside from communication with MainActivity.

ChronoChatService is structured as a concrete implementation of the abstract *ChronoSyncService* class; I tried to place code that was not ChronoChat-specific into ChronoSyncService such that future ChronoSync-based Android applications could be built on ChronoSyncService.

4 Comparing the ChronoChat-android architecture to NDN Whiteboard

Like ChronoChat-android, NDN Whiteboard also uses an Activity as its "central" component, but unlike the former, NDN Whiteboard gives ownership of various persistent objects—namely, its equivalent of the "network thread" and the NDN *Face* object—to the Activity instead of to a Service. I opted to use a Service for this purpose because Activity objects are routinely destroyed by the Android OS. For example, switching an Android device from portrait to landscape orientation causes a foreground Activity to be destroyed and re-created; maintaining state in these cases is normally tedious and sometimes impossible. Services, however, can be configured to remain resident, and if run as a foreground service—as ChronoChatService is—the service should only be destroyed if the enclosing process is terminated by the system under extreme memory pressure from a foreground process *[6]*, *[4]*.

NDN Whiteboard uses a chain of *AsyncTasks* to initialize its ChronoSync implementation. While AsyncTasks are useful to prevent blocking the UI thread, NDN Whiteboard launches its initial AsyncTask from an Activity, meaning that if the Activity is destroyed, the entire chain of AsyncTasks will be lost. I found this chaining arrangement to be confusing and seemingly fragile, so I opted to perform all ChronoSync setup tasks in the same network thread that performs ongoing ChronoSync tasks post-setup. This arrangement also follows the instructions given by the NDN-CCL documentation *[9]*, which seems to warn of thread-safety issues.[1]

5 Comparing ChronoChat-android to ChronoChat-js

ChronoChat-android is designed to be interoperable with ChronoChat-js and have generally equivalent features. ChronoChat-android uses a Google Protocol Buffer ("Protobuf") definition taken from jNDN *[10]* for compatibility with the protocol used by ChronoChat-js. The app's Gradle script pulls in a Protobuf build plugin from Maven *[2]* which in turn compiles the protocol definition to a Java class.

ChronoChat-android reorders any messages received out-of-order (by timestamp[2]) before displaying them to the user; this doesn't seem to be a part of the ChronoChat protocol currently implemented by ChronoChat-js.

I originally designed ChronoChat-android to avoid retrieving "old" messages that were published before joining the chatroom. I was concerned that users might be confused by the sudden arrival of numerous "old" messages upon connecting to the chat. However, I ultimately reverted to retrieving these "old" messages the way ChronoChat-js does, because this is an expected behavior of the ChronoChat synchronization protocol. My concerns about user confusion were also mitigated by the fact that received messages are displayed in timestamp order.

6 Miscellaneous issues encountered

ChronoChat-js will sometimes repeatedly display the last message published by a ChronoChat-android client. I believe the problem lies with ChronoChat-js, specifically in how it handles "recovery" sync states, but I haven't been able to track down the exact cause.

During my testing, mysterious "broken pipe" error messages from ChronoSync would occasionally appear in the debug log, without any Java exception being thrown to the surrounding code. I suspect that ChronoSync may be losing its connection to NFD-android due to components of the latter being spontaneously destroyed by the Android OS. (I assume that NFD-android is susceptible to this because it doesn't run as a foreground service.)

The Gradle configuration I used for the Protobuf Gradle plugin differs from the instructions given by the plugin's maintainers *[2]*. I couldn't get the "javalite" versions of the plugins to work as described, so I found a way to make things work using the standard Java versions.

[1] It should be noted that despite such warnings, I did not encounter any obvious thread-safety bugs in my testing of NDN Whiteboard or alternative threading arrangements in ChronoChatService.)

[2] ChronoChat-android trusts the sender-specified message timestamp.

7 Conclusion: limitations and future additions

ChronoChat-android can only connect to one chatroom at a time, and lacks the chatroom discovery feature of the original ChronoChat. The security configuration is based on NDN Whiteboard—i.e., it lacks "proper" security.

Some users may wish to silence or disable the "new message" notifications displayed by ChronoChat-android, but the app currently lacks such configuration options. The ability to save chatroom history to persistent storage could also be useful.

Outside of ChronoChat-android, I would suggest providing a way to interact with NFD-android via Intents, instead of through a persistent Face object. I believe this would make it easier to develop NDN applications for Android, by providing a more natural and "Android-style" asynchronous programming model. Finally, as I previously alluded to, I believe NFD-android would be more reliable if configured to run as a foreground service.

I hope that ChronoChat-android will serve as a useful addition to the suite of NDN mobile apps, and an example of how to structure a ChronoSync-based Android application.

References

[1] Alexander Afanasyev. NFD-android. URL: https://github.com/named-data-mobile/NFD-android.

[2] Google. Protobuf Plugin for Gradle. URL: https://github.com/google/protobuf-gradle-plugin.

[3] Sumit Gouthaman and Alexander Afanasyev. NDN Whiteboard. URL: https://github.com/named-data-mobile/apps-NDN-Whiteboard.

[4] Android Open Source Project. Android API Guide: Processes and Application Life Cycle. URL: http://developer.android.com/guide/topics/processes/process-lifecycle.html.

[5] Android Open Source Project. Android API Guide: Processes and Threads. URL: http://developer.android.com/guide/components/processes-and-threads.html#Threads.

[6] Android Open Source Project. Android API Guide: Services. URL: https://developer.android.com/guide/components/services.html#Foreground.

[7] Tyler Vernon Smith. ChronoChat-android. URL: https://github.com/tylervernonsmith/ChronoChat-android.

[8] NDN Project Team. NDN Common Client Libraries API 0.5.1 Documentation: ChronoSync2013 Class. URL: http://named-data.net/doc/ndn-ccl-api/chrono-sync2013.html.

[9] NDN Project Team. NDN Common Client Libraries API 0.5.1 Documentation: Face Class. URL: http://named-data.net/doc/ndn-ccl-api/face.html#face-processevents-method.

[10] Jeff Thompson. ChronoChat Protobuf definition. URL: https://github.com/named-data/jndn/blob/master/examples/src/net/named_data/jndn/tests/chatbuf-proto.proto.

[11] Jeff Thompson. ChronoChat-js. URL: https://github.com/named-data/ChronoChat-js.

[12] Zhenkai Zhu and Alexander Afanasyev. Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking. In *21st IEEE International Conference on Network Protocols (ICNP 2013)*. October 2013. URL: https://named-data.net/wp-content/uploads/2014/03/chronosync-icnp2013.pdf.