Spring 2023

Ubiquitous Application Data Collection in a Disconnected Distributed System

Deepak Munagala
San Jose State University

Ubiquitous Application Data Collection in a Disconnected Distributed System

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Deepak Munagala

May 2023

The Designated Project Committee Approves the Project Titled

Ubiquitous Application Data Collection in a Disconnected Distributed System

by

Deepak Munagala

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Ben Reed	Department of Computer Science
Dr. Genya Ishigaki	Department of Computer Science
Dr. Navrati Saxena	Department of Computer Science

ABSTRACT

Ubiquitous Application Data Collection in a Disconnected Distributed System

by Deepak Munagala

Despite some incredible advancements in technology, a significant population of the world does not have internet connectivity. These people lack access to crucial information that is easily available to the rest of the world. To solve this problem, we implement a Delay Tolerant Network (DTN) that allows users in disconnected regions access to the internet. This is enabled by collecting all data requests on the users' phones and passing them to a device that can carry them to a connected region. This device can then collect the necessary information and give it back to the users in the disconnected region. This work will focus on how modern applications on Android phones in disconnected areas can make use of this DTN architecture to send and receive data with their respective destinations.

ACKNOWLEDGMENTS

I would like to thank Dr. Ben Reed for his knowledge on the topic and enthusiasm in discussing this problem. The weekly meetings for the CS systems group motivated me to think about the problem in detail and come up with several approaches. I would also like to thank the members of the group, Shashank, Aditya, Anirudh, and Abhishek for the discussions regarding methods for different parts of this problem and for setting the scope for this project.

TABLE OF CONTENTS

CHAPTER

CHAPTER 1

Introduction

Over the past few decades, the Internet has become increasingly essential in our day-to-day tasks. The Internet is heavily used to connect with other people and get information in different formats like text, audio, and video. Even with wide internet access around the world, around 32.1 percent of the world is still not connected to the internet [1]. Lack of internet means that people do not have access to important information which puts them at a severe disadvantage.

We also lose access to the internet during natural disasters. This results in severe difficulty in executing relief and rescue operations. An efficient way to solve this problem is to form a network that connects people in danger with those at relief centers so that they can find the number of people affected by the disaster and accordingly carry out operations [2].

These problems we see so far can be solved with a type of resilient network known as a Delay Tolerant Network (DTN) [3]. It allows devices in disconnected regions to connect to the internet and gain access to information. This is a store carry and forward approach that makes use of intermediate devices, known as transports, that are responsible for carrying and passing messages between the client device and the source containing information. This source can be the internet or another computer that holds information. In this kind of network, communication between clients and transport devices can happen when they are in proximity to each other via wireless communication options like WiFi, NFC, or Bluetooth.

In modern applications, the users of the DTN can be an application on the user's smartphone, which has to talk to a server on the cloud, usually called an application server, to upload or download data for the client. In the absence of network connectivity, this communication between the application on the user's

smartphone and the application server needs to be handled by the DTN. Figure 1 below shows a typical system of the entities in a DTN and how data flows among them.

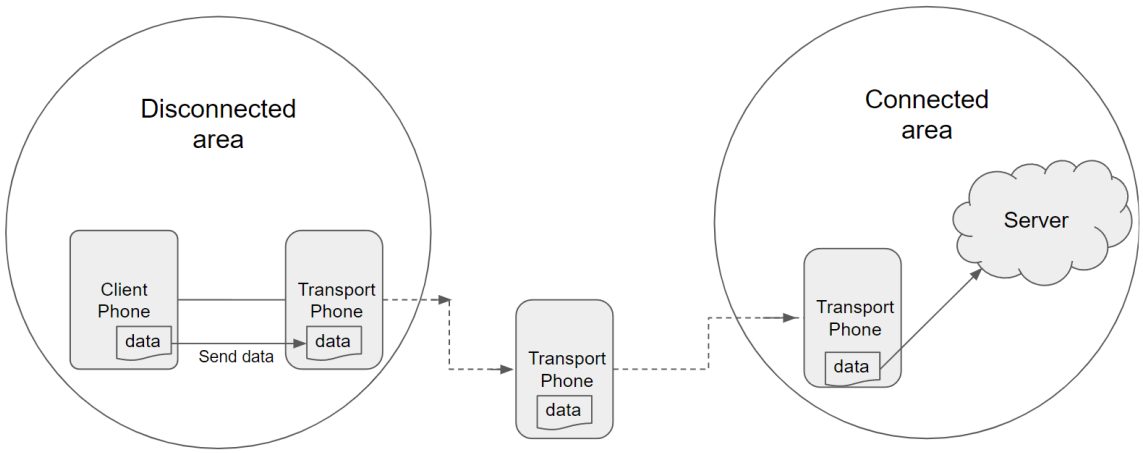

Figure 1: DTN architecture

The types of applications that can make use of this type of network include those that can tolerate a relatively high delay between the sender and receiver of data. For example, messaging, document sharing, and video downloading/uploading, which generally do not have strict requirements on round trip times, are well suited to this type of network. Some applications that can work with this network include texting applications, web browser applications, and video viewing/uploading applications. Any application that requires real-time communication between two users will not work with this type of network. Some of these examples include calling applications and live-streaming applications.

As shown in figure 1, an important part of this architecture is the transport. It is responsible for moving between different regions of the city, and communicating with multiple clients in disconnected regions to store user data. The transport eventually reaches a destination, which can be an internet-connected region. Here, the transport can upload user data and store any data that has to be sent back to the users. In this

type of network, a transport can be a device on some mode of transport such as a bus, train, or a ship.

Since the mode of communication between the transport and the users is unreliable, previous solutions make use of special hardware so that data can be moved easily between the transport and the user's device. As these types of DTN systems require special hardware, which is not scalable. To handle this problem, we need to make use of existing hardware that is already being used in these environments.

In this paper, we will describe an architecture built on top of Android-based smartphones that acts as a bridge between applications on client phones and their servers. Here, transport will be smartphones on public transport vehicles, which might include buses and trains. Our system will require a custom application on the user's phone that needs to be responsible for storing data that other applications on the phone want to send across to the internet. This custom application should also be able to deliver data to the transport device in the proximity of the user's device. For these reasons, we introduce an application, called **Bundle Client**, which has to be installed on the user's phone so that it can collect data from applications on the phone and send them to their destinations via the transport device. Each application on the user's phone has a server hosted on the internet that it communicates with, which is referred to as an **application server**.

When this data reaches the internet via transport, we will send them to our server, called **Bundle Server**, which is a part of the DTN architecture. The Bundle Server will forward them to the respective application's server. Here, each application will have a corresponding application server adapter which will be responsible for handling data it receives from the client via the Bundle Server. This adapter will also recognize that the client is making use of the DTN system and prepare any data that needs to be sent to the client via this system.

1.1 Using Signal Android application

Making a DTN system accessible in devices with the Android operating system means that it should be able to execute an application's interactions with the internet seamlessly. This includes:

1. application receiving notifications for data it receives over the internet
2. application sending data to the internet
3. application or server getting notified if the send operation was successful

Implementing a DTN architecture with these features on a widely used Android application will demonstrate its effectiveness. In this paper, we will demonstrate the working of our DTN implementation by using the Signal Android application, a highly popular application, with over 100 million downloads as of December 2022 [4]. Signal is an open-source multi-platform instant messaging service that strongly emphasizes end-to-end encryption.

The scope of this paper will be to implement an application that interacts with Signal Android to handle the operations mentioned above for Signal when the device is in a disconnected setting. Here, the data considered is primarily text messages and additional options of voice and video messages.

1.2 Other components

The focus of this paper will be to address issues in client application - bundle client communication, and Bundle Server - Application Server Adapter communication. Some other potential challenges in a DTN system, which will not be addressed here, include:

- registration of users with our DTN system
- preparation of data on the client device for send operation,

- encryption to protect application data,
- coordination and communication between client and transport device,
- maintaining the order in which data are sent and received

This project will assume that the above components work and focus on the issues mentioned in section 1.1.

CHAPTER 2

Related Works

This chapter discusses previous work on designs and implementations of DTNs that we will build upon. Specifically, we will go over works on the design of the data collection application on the users' phones, which we name the Bundle Client. We will first take a look at the considerations of how data is managed on a user's phone in a disconnected region. In section 2.3, we will then focus on how previous works process the data they receive from other applications.

2.1 Data Storage

The Bundle Client application on the user's device is responsible for storing data it receives from different applications on the phone and from the transport device. Before sending this data to their respective destination, the data needs to be organized so that the maximum amount of data can be delivered. A lot of existing implementations make use of custom implementations to showcase the working of their DTN architecture. [3] implemented the data storage of the Bundle Client by making use of a priority queue on client devices to decide which messages should be given priority in the case of lower data limits for transmission. High priority was given to smaller-sized messages with a low lifetime. On the other hand, [5] implemented the Quality of Service/Quality of experience feature on data being sent over DTN, which included giving priority to data that had an early deadline and dropping them if they can not be delivered before a specified timeline. In addition to this, [6] and [7] used a set of policies to decide which data to keep and which to drop from the queue of data messages, such as Drop Last (DL) to drop the most recently added message, Drop First (DF) to drop the oldest message in the queue, drop oldest (DO) to drop a message with the shortest TTL.

An important problem that a DTN architecture generally faces is the long duration

of round trip time. When an application generates some data and sends it to the Bundle Client, it can take a long time for Bundle Client to be in contact with a transport device that can send data to the server. Before the client device waits to receive confirmation from the transport device that the data was delivered successfully, which can be several hours or even days, there might be a lot of cached data that is waiting to be sent. To handle this problem, [8] maintained an upper limit on the data that can be stored for transmission via DTN. This was done by storing them in volatile memory, and if the data store application receives some data that crosses the memory limit, it will be discarded. When a transport arrives, this data can be written into file storage, where it can be prepared to be transmitted. To store in the file system, it makes use of a matchbox file system [9], which focuses on data corruption detection, low memory consumption, and low write overhead. Some problems with this approach are that volatile memory (RAM) can get overflowed and that the data can be lost if the data store application crashes before it is stored in the file system.

Another approach to solving this large round trip time is to always store application data in the file system without depending on the cache. On top of this, the Bundle Client application can allow users to specify an order of preference for a different type of action in applications [10]. In this approach, a resource management module can be used with the Bundle Client, which calculates a priority based on the cost-benefit analysis using user preference and device resources consumed, like battery power, bandwidth, and data size. Based on these calculations, the transmission module can decide which data needs to be prioritized.

2.2 Communication between Bundle Client and user applications

A device in a disconnected region can have limited battery life and restricted connection with the transport device. To ensure that the DTN architecture can make

the best use of these limited opportunities, the Bundle Client application on the user's phone needs an efficient protocol for moving data between the applications on the user's device and the transport devices. Looking at implementations of communication protocols in DTNs, [11] used file storage to store bundle data as it allows us to store large amounts of data and SQLite to store metadata so that we can easily query information about bundles. To ensure that the probability of these files being delivered is high, [12] split files into smaller chunks and gradually made the chunk size larger if the smaller chunks were delivered successfully. This was done to consider the variable connection times between the client and transport devices. Similarly, [13] collected knowledge of contact durations between these two types of devices to calculate how much data can be transferred between them and used this information to fragment the data. If it finds that the client and transport remain connected only for short durations, smaller fragments will be created so that they can be transferred in this small duration. A problem with this approach is that the receiver of these fragments needs to handle the merging of these fragments before sending them to their destination.

In contrast, [14] used the concept of a publisher-subscriber system for the implementation of DTN in natural disaster recovery operations, where clients are publishers of data and transport devices are subscribers. The transport devices collect information from a limited set of clients and delivered them to their destinations.

To communicate between client applications and the DTN application, [15] made use of an android binder, a low-level mechanism for inter-process communication. Since this is a low-level feature, it requires a lot of configuration to set up and allow applications to communicate with each other but gives a good design that can be followed while creating the Bundle Client application. In the future chapters, we will discuss another feature on the Android platform called the Content Provider, which

builds on top of Android Binder and is easy to set up and use. On the other hand, [16] implemented a web server in the DTN application on the client device that accepts app data from both web applications and mobile applications. The web server is able to cater to different types of applications because it introduces the Service-Adaptation Middleware module. This module is responsible for receiving data from applications written in different programming languages and storing their data in a form that can be used by the DTN to package and send to other devices. A problem that this research identified with the web server approach is that web applications need some manual deployment for connecting this application with the DTN architecture.

Another research, [17] found from experiments that a pushing mechanism to synchronize data between mobile devices and servers is an energy-efficient approach. This can be used in this research to communicate between client applications and the Bundle Client to notify each other about new data. Most of the implementations seen so far make use of a simplified version of the client application or run clients over a computer terminal. Since smartphone technology is rapidly evolving and a wide variety of communication protocols are used by many Android applications, we need to test the DTN on the latest Android OS devices with large-scale applications. Designing a DTN that can communicate with these different types of applications will provide a general standard that all applications can use to enable the Internet in disconnected areas.

2.3 Application data format

Application Data Unit (ADU) is the smallest unit of data generated by an application that is used by DTN protocol [18]. One major approach to sending ADUs within a DTN is to consider each ADU as an individual object (or bundle) that can be sent between nodes [10]. This follows the design principle of application layer framing

[19], where we do not depend on communication protocols such as HTTP, SMTP, POP, FTP, or IMAP for allocating identifiers to ADUs. With this in mind, [10] added additional information to ADUs such as keywords for describing the data, data management information (application identifier, creation date, modification date), and security-related information (encryption method and key, user and application access permissions). When this ADU passes through several intermediate nodes before reaching the destination, more data like a list of nodes it has passed through, a list of forwarding hints, and priority level as mentioned by the original sender of the ADU. To ensure that this data was not modified before it reaches the destination, the sender adds a hash of the ADU as a parameter [20].

On a different note, [21] made use of the HTTP standard to design the format of blob that can be used to send data between devices in the DTN. Since a lot of applications make use of HTTP to communicate with their server, these HTTP requests can be modified to add additional data for communicating between devices. To add additional metadata for the receiver, headers can be used. These headers can be removed by the bundle server or the application server adapter before sending the ADU to the application server.

CHAPTER 3

System design

This chapter describes the different components of the delay-tolerant network and how they communicate with each other. Section 3.4 and later will focus mainly on the inner workings of the Data Store Module and Application Data Management module.

3.1 Background

In a client smartphone, we have a bundle client application that receives data from several applications and sends data across to the transport's smartphone as a blob containing a list of messages, called a **bundle**. As this transport goes through several remote villages collecting data from different clients, the transport stores them and then sends collected data to the DTN server once reaching the connected region. The bundle server sends these messages to the respective destinations for further processing. When the transport is ready to collect data for a client, the bundle server will ask all the applications for data pertaining to this client, group the received data, and send it back to the transport.

In our delay-tolerant network system, we will discuss three crucial modules, as described in figure 2. These modules include:

Data Store Module (DSM): This module is responsible for receiving ADUs from the application and preparing them for delivery to transport and vice versa.

Application Data Manager (ADM): This module resides in the Bundle Client. In the client application, this is responsible for sending messages that the client application wants to send across the internet to the application data manager module in the bundle client application. This module also generates and stores the **client identifier**, which is used by the server to identify the client device.

Application Server Adapter (ASA): This module is implemented as part

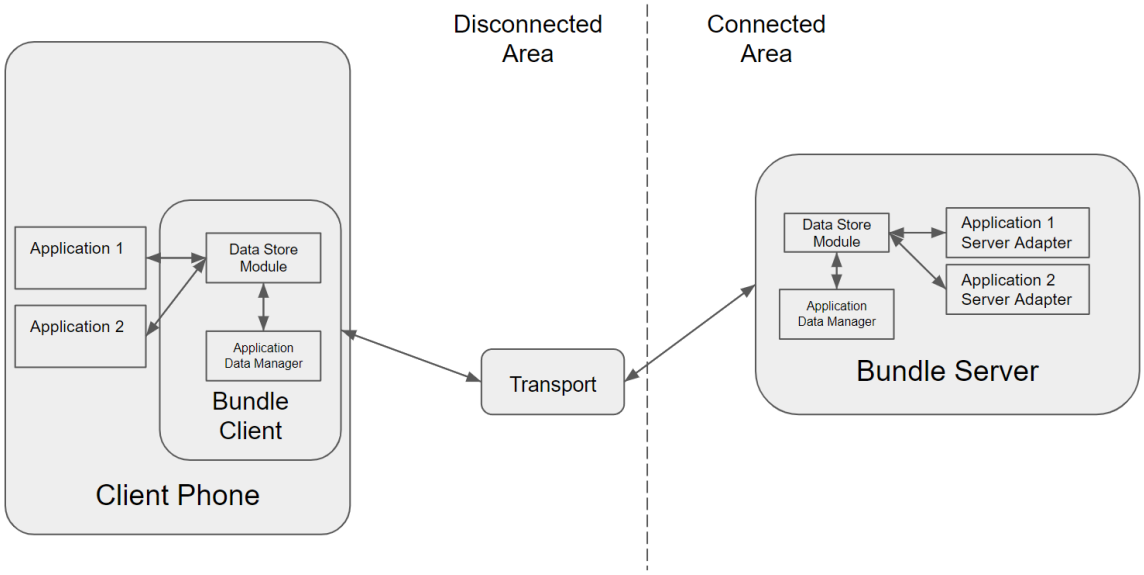

Figure 2: System Diagram

of the application's server. This module is an adapter [22], which is responsible for passing data between Bundle Server and the application server.

Another component is the **Application Server**, which is a component implemented by the Application. This module gets data from clients, processes it, and may send data back to the client. This component may also push data for a specific client.

3.2 Data delivery Assumptions

Let's consider a smartphone in a disconnected area that has ADUs from a DTN application A with data IDs A1, A2, and A3 when the transport arrives in its vicinity. After we send these ADUs as a bundle to this transport, two more ADUs with ID A4 and A5 are received from application A. When any new transport arrives at this point, we will send ADUs A1 to A5. When we receive an acknowledgment that previous data have been delivered successfully, we will discard them from phone storage and send the remaining unacknowledged ADUs. For example, if we receive acknowledgment for ADUs with IDs A1, A2, and A3, we will remove them from storage and send A4

and A5 to subsequent transports.

Both the bundle client application and the bundle server place additional constraints. (1) ADUs can not be fragmented, i.e., either the entire ADU is sent or none at all. (2) The limit on the total size of data that can be placed per application in a bundle. If we receive any data unit from an application or application server adapter having a size greater than the set limit, we will discard the data. This ensures that each application can send some data without hitting the storage limit on the transport device. Applications will hold the responsibility of specifying the sequence in which data should send data based on some priority.

3.3 Acknowledgments

With the large-scale deployment of this DTN, it is possible that multiple applications on a user's phone might send data to the data collection application, which we call the Bundle Client. Since storage on the user's phone is a limited resource, the Bundle Client should minimize the use of phone storage as much as possible. One way to do this is to make sure that the phone does not store any data that is not required. In this DTN, storage can be optimally used by keeping track of application data received in Bundle Client and deleting them when they are no longer required. The ADUs in the Bundle Client application that were sent by applications on the device are deleted when the Bundle Client receives a confirmation that the ADUs were sent successfully to their destination. This is done with the help of **acknowledgments** that is sent by the receiver of data so that the sender can stop sending the same data again and clear up its memory. This is applicable in both cases where data is sent from Bundle Client to Bundle Server and vice versa. The upcoming sections will show how acknowledgments are handled at both Bundle Client and Bundle Server.

3.4 Data handling at Bundle Client

As shown in figure 2, the Bundle Client application is an interface between applications on the device and the transport. The Data Store Module component of this application exposes a service to the applications on the device so that they can send data to the Bundle Client. To forward this data to the application servers, this application regularly monitors for nearby transport devices and connects to them, if available.

The first step in the flow of data is that the bundle client should receive data from the transport, and then generate a bundle that can be sent to the transport device. The following sections will describe how the Bundle Client handles data from applications and from transport.

3.4.1 Push versus Pull

The communication between the Bundle Client and other applications should occur in a quick and energy-efficient manner. This is necessary because mobile devices have limited resources like battery life. Previous research [17] has shown that a push-based mechanism is more energy efficient as compared to a pull-based mechanism. Another reason why a push-based mechanism is preferable is that the sender can send data in real time, without waiting for the receiver to ask for data. This also helps save time when the Bundle Client wants to send data to the transport device because it does not have to spend time pulling data from all DTN-enabled applications on the device.

3.4.2 Sending Data to transport

When a DTN-enabled application wants to send some data to its server and it detects that there is no internet connection, it will send the data to the Bundle Client. The Bundle Client makes sure that any data that applications send does not cross

the size limit. This check allows the Bundle Client to allocate equal memory to all applications that want to send data. If any application's data does cross the limit, the application will be responsible for splitting it into smaller chunks and ensuring that the Bundle Client has received the data. The Bundle Client then assigns an identifier to each piece of data it receives, which is called an Application Data Unit Identifier, or **ADU_ID**, in short. The ADU also has an **application ID** linked to it, which is unique for each application on the Android OS. This ADU is then stored in the Bundle Client's storage.

When a transport device is ready to receive data from the client device, all the data stored in the Bundle Client application is grouped to form a bundle. Based on the size limit configured for the entire bundle, ADUs are added to the bundle until the size limit is reached. If some of the ADUs could not be sent, the Bundle Client waits until existing ADUs have been successfully delivered (which we know via acknowledgments), after which the rest of the ADUs can be sent. Figure 3 summarizes the actions for sending ADUs to the transport.

3.4.3 Receiving Data from transport

Upon the start of the bundle client service, the application on this client device regularly checks for transport devices in proximity. After connecting with a transport, this application first asks the transport to send any bundle it has for this client device. After the bundle is unpacked, the ADM gets the acknowledgments and a list of Application Data Units (ADUs). When DSM receives these acknowledgments and ADUs from the ADM, the DSM will inform the respective applications that the data was delivered successfully to the servers. Similar to the previous section, this communication with the application is done with the help of the Application ID that is present for each ADU. The acknowledgments extracted from the bundle mean that

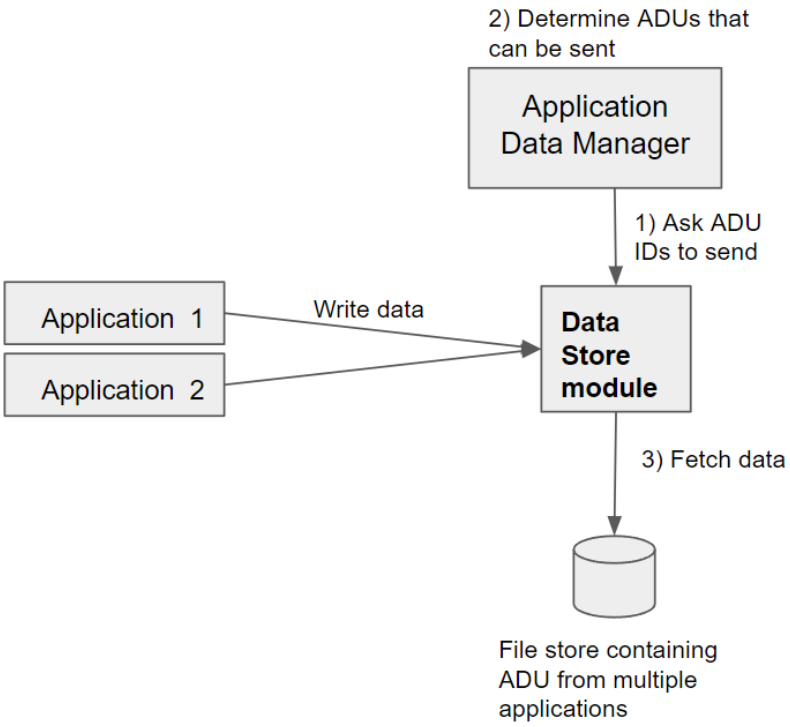

Figure 3: Flow of data for bundle client sending data to transport

the DSM will not have to send the same data again to this or any later transport device and that the data can be deleted from the Bundle Client's local storage.

After acknowledgments are processed, the ADM will share the ADUs with the Data Store Module (DSM), which will then store the ADUs in the file system, and asynchronously deliver the ADUs to the respective applications. Once the Bundle Client knows that the ADUs have been successfully delivered to the respective applications, the ADUs will be deleted from the storage. Figure 4 summarizes the actions for receiving ADUs from the transport.

3.5 Application data handling at bundle server

The way ADUs are handled at the bundle server is different compared to its handling at client devices seen in section 3.4. This is mainly because the server needs to handle data for many clients and should connect to different application servers to

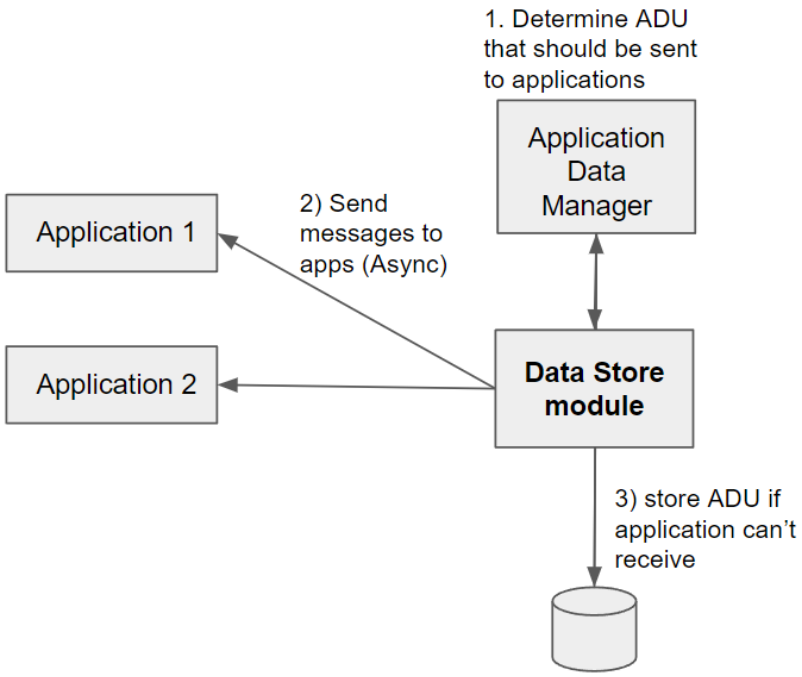

Figure 4: Flow of data for Bundle Client receiving data from transport

send and receive application data on behalf of clients.

3.5.1 Receiving messages from transport

The transport sends all bundles it has collected from clients to the bundle server when it gets network connectivity. For each ADU in the bundle, the ADM discards the ADU if it has already been received from an earlier transport. The ADM sends the remaining messages to DSM, which forwards them to the respective application server adapter for further processing. The application server adapters send these ADUs to the Application Servers and receive a response for each ADU, which will be sent back to the DSM. The DSM will store the ADUs in its' file storage until a transport arrives that can successfully deliver the data to the client. Figure 5 summarizes the actions for receiving ADUs from the transport.

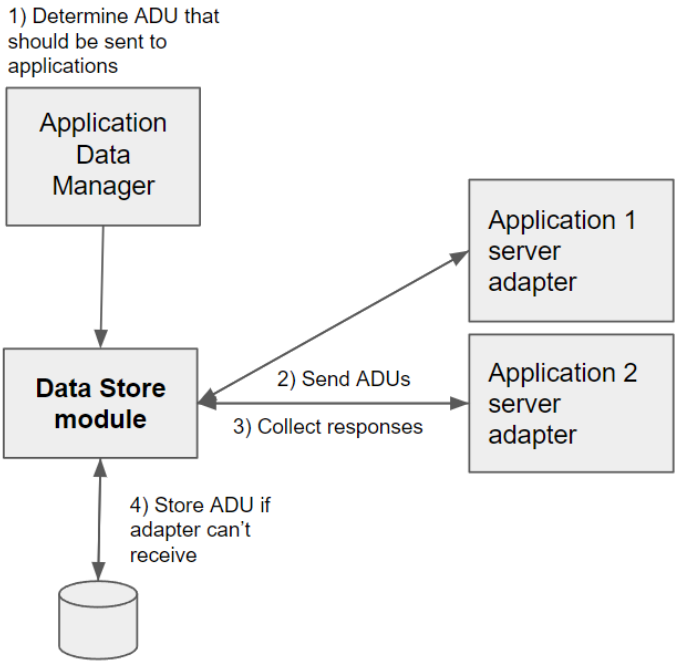

Figure 5: Flow of data for bundle server receiving data from transport

3.5.2 Sending messages to transport

A transport that is ready to receive client data will ask the bundle server for bundles. The BTM, which keeps track of the clients for which this transport has successfully delivered messages, will ask ADM if there is application data that needs to be sent to these selected clients. If there is additional data to be sent, BTM will initiate the process of creating a new bundle by generating a new bundle ID and asking ADM for all the data that has to be bundled. Once BTM finishes the creation and delivery of the bundle, it will store data in its file system so that they can be used later when another transport arrives. Figure 6 summarizes the actions for sending ADUs from the server to the transport.

3.6 Application data Storage

Several applications like Facebook, Whatsapp, Youtube, and Signal allow users to upload files whose size can range from a few KBs to a few GBs. To efficiently access

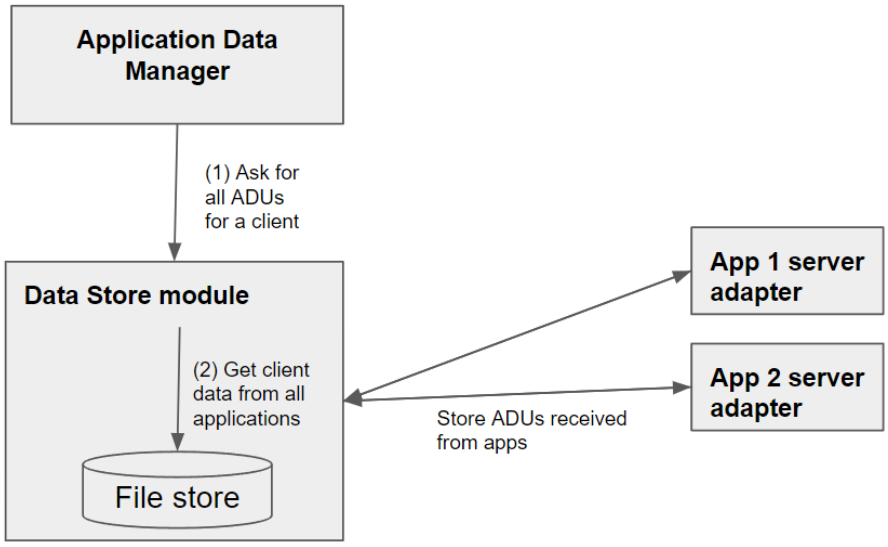

Figure 6: Flow of data for bundle server receiving data from transport

this data, the system makes use of the file storage system [23]. On the client device, a common storage system is required where applications can push their data, and the Bundle Client can read that data and send it to transport devices, whenever required. Also, before applications push their data, the Bundle Client should be able to add additional data like the ADU_ID, and Application ID. For these reasons, we store application data in the Bundle Client application's file storage. Here, we maintain two separate folders, one for the application to store ADUs that should be sent to the server, and another for the application to receive ADUs that the server has sent for the application. Each application will store some metadata such as the largest ADU_ID that has been used till now. Figure 7 shows the file structure in the Bundle Client application containing ADUs for multiple application IDs.

On the server, the storage structure is slightly different as the server has to take into account client information as well. Here, we have two folders based on the destination of the ADUs, which can go to either the client ('Send' folder) or the application adapter server ('Receive' folder). These folders contain a separate sub-

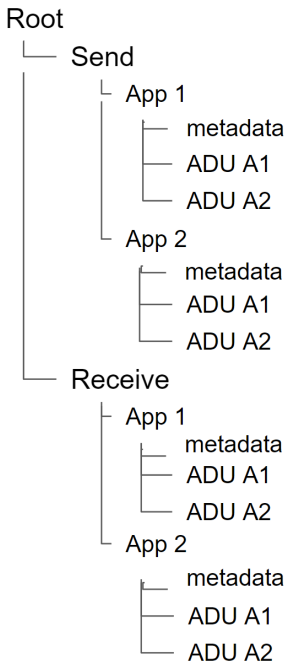

Figure 7: File structure for ADUs in client device

folder for each client, which in turn store application data grouped by the application identifier. Figure 8 shows the file structure in the bundle server containing ADUs for multiple clients and application IDs.

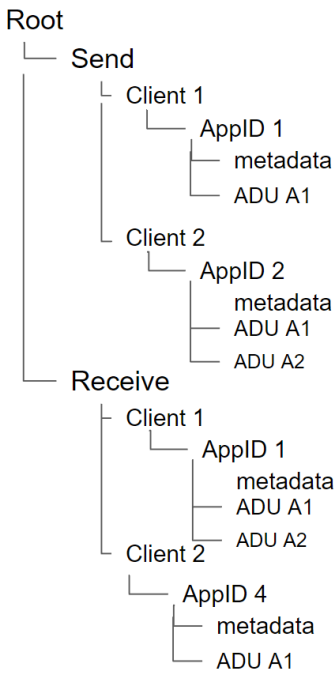

Figure 8: File structure for ADUs in bundle server

CHAPTER 4

Implementation Details

This chapter will describe the implementation details of a sample application, that acts as a client for our DTN system, and the way this application communicates with the Bundle Client application. This chapter will also describe how the bundle server handles this data and communicates with the application adapter server to send and receive client-specific application data. To show the efficacy of the DTN system, we will take a look at two kinds of client applications. First is a simple Android application that tests send and receive operations on simple plain text data, and the other is the open-source Android OS project for Signal [24]. Second, is the modification to the open-source Signal Android project to show how it can make use of this DTN to send and receive messages.

4.1 Content Provider

A content provider is an essential component of the Android OS that is used to manage access to a repository of data in an application. An application can expose a content provider by implementing a Java class that extends from the ContentProvider class. In this class, methods for main operations like Insert, Query, Update, and Delete have to be implemented. The data source on which these operations will be performed can be either a SQL database or the host application's file storage.

Any application can access data hosted on a content provider as long as they have read and write permissions for the content provider in their Manifest file. This means that any application that wants to make use of this DTN should have the permissions for the Bundle Client's content provider. This also means that the user does not have to do manual registration to allow applications to make use of Content Provider. To ensure that applications do not access data that is not meant for them, the content provider should implement the necessary validation in the methods for the operations

```
<provider
    android:name="com.ddd.datastore.providers.MessageProvider"
    android:authorities="com.ddd.datastore.providers"
    android:exported="true"
    android:multiprocess="true"
    android:grantUriPermissions="true"
    android:writePermission="android.permission.messagingproviderwriteperm"
    android:readPermission="android.permission.messagingproviderreadperm"></provider>
```

Figure 9: Manifest file for the client application to send data

discussed earlier. This is done by checking the Application ID of the application requesting data and only operating on the data for the specified Application ID.

Since content providers are used for sending large amounts of data between applications, it is favorable to use it in Bundle Client to host other applications' data. Figure 9 shows the Manifest file configuration for hosting the content provider on the Bundle Client.

4.2 Android Intents

When the Bundle Client receives data from transport, it should be able to push the data to applications rather than waiting for applications to request data. This can be done on the Android OS with the help of intents. **Intents** are an inter-application communication technique in the Android platform. They are used by an application to send data to other applications or ask other applications to perform some action. In this DTN, we send data from the Bundle Client to other applications by specifying the Application ID. Since this ID is unique on the Android platform, the data will be sent to the intended destination and not another malicious application.

For an application to be able to process intents sent by other applications, it needs to implement an **Intent Service**. The class that is responsible for receiving intent-specific data from other applications should be specified in the application's Manifest file.

```
<service
    android:name=".ReceiveIntentService"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.dtn.SEND_DATA" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</service>
```

Figure 10: Manifest file for the client application to receive data

4.3 Client Application

In a simple scenario, if a client application wants to send data to its server in the absence of network connectivity, it can choose to send data to the Bundle Client application which is installed on the same phone.

On the Android platform, the Bundle Client application hosts a content provider named 'com.ddd.datastore.providers.MessageProvider'. Client applications can use this name to push data to the Bundle Client.

On the other hand, for the client application to receive data, it should implement an intent service that accepts data from the Bundle Client application by specifying the action 'android.intent.dtn.SEND_DATA'. Figure 10 shows the configuration in the manifest file for a client application to be able to receive intents from the Bundle Client.

4.4 Bundle Client Application

This application is the entry point for client applications to communicate with the DTN architecture. It exposes a content provider to the Android system which can be accessed by any other application as long they have the read and write permissions for it. The content provider exposes a few endpoints to view or modify data such as

'query', 'insert', 'update', and 'delete'. In this implementation, a file system is used to store application data, as described in 3.6.

Any data received by this application is associated with an application identifier. For this, the client application's package name is used because it has to be globally unique on the google play store. The content provider gives the Application ID of the calling application, which helps rule out any phishing attacks.

Any piece of data received by the Bundle Client is assigned a unique ID, known as ADU_ID, which helps keep track of the sequence of data that has been sent to transport or has been acknowledged by the server. When Bundle Client receives an acknowledgment from the bundle server that the data with an ADU_ID has been received by the server, we can delete it from the Bundle Client application storage.

Once this application receives data from the transport and stores it in the file system, it pushes data to the application using intents. By specifying the application ID, Android OS allows the Bundle Client to send data to the specific application. Once data is sent successfully, it is deleted from its local storage.

The Bundle Client keeps track of all operations on the ADUs with the help of a metadata file. This file maintains (1) the ADUs that have been deleted, (2) the latest ADU ID added, (3) the most recent ADU ID that has been processed by the application, and (4) the latest ADU ID that was delivered successfully to the server. Figure 11 below shows the structure of the metadata object.

It is possible that the client application might not be able to receive data from the Bundle Client. In such a case, the client can query the content provider for unseen data.

```
public class Metadata {
    //last messageId added by the app
    public long lastAddedMessageId;

    //last messageId sent successfully via DTN
    public long lastSentMessageId;

    //last messageId received via DTN
    public long lastReceivedMessageId;

    //latest messageId processed by the application
    public long lastProcessedMessageId;

    public Metadata(long lastAddedMessageId, long lastSentMessageId,
                    long lastReceivedMessageId, long lastProcessedMessageId){...}
}
```

Figure 11: Metadata file in each folder in Bundle Client storage

4.5 Bundle Server

The main role of the Bundle Server is to receive data from the transport device and send it to the respective application's server adapter. Since the communication for the Bundle Server with both the transport device and the application server adapters is done over the internet, we can use **gRPC**, which is an open-source high-performance remote procedure call framework. The Bundle Server implements two gRPC services, with one exposed to the transport, called BundleService, allowing transport devices to send and receive bundles of data. Another gRPC service, called DTNCommunication service, is exposed to the application adapter services, allowing them to register the server with the application ID.

When a transport connects with Bundle Server and asks for data, the server should be able to quickly find the client data that can be sent to it. Here, we do this by monitoring all ADUs it has received so far from the application adapters and storing their information in a MySQL database. As shown in the schema below, we

only store metadata related to the ADUs in the database. The actual data is stored in the file system, as described in figure 8.

Column Name	Type	Length	Is Null
app_id	varchar	100	not null
client_id	varchar	100	not null
adu_id	int unsigned	4	not null
direction_id	varchar	6	not null

Table 1: Structure for app_data_table

4.5.1 DTNCommunicationService

When a new application server wants to connect to DTN Bundle Server, it will send a registerAdapter request, passing in the network address and the application ID. The bundle server will store this mapping in the registered_app_adapter_table. To make sure that a malicious agent does not attempt to act as another application to get unauthorized data from the Bundle Server, the registration step should be done manually. Next time, whenever we receive client data from transport, we will look at the network address registered for the application and forward data to that server. Figure 12 below shows the design of the service running in the bundle server.

Column Name	Type	Length	Is Null
app_id	varchar	100	not null
address	varchar	200	not null

Table 2: Structure for registered_app_adapter_table

When transport brings some client data to the bundle server, the data store module (DSM) will first be notified that it should prepare data for the clientId. Once this bundle has been decrypted, DSM will store it as described in figure 8, in the "Receive" directory before forwarding them to the registered adapter via its gRPC. Based on the network address registered for an application, all data for the client will

```
service DTNCommunication {
  rpc registerAdapter (ConnectionData) returns (ResponseStatus) {}
}

message ConnectionData {
  string appName = 1;
  string url = 2;
}

message ResponseStatus {
  int32 code = 1;
  string message = 2;
}
```

Figure 12: GRPC service for application adapter server to register with bundle server

be sent to their adapter and their responses will be collected and stored in the "Send" directory.

4.6 Application Adapter Server

This adapter server is responsible for collecting data from the bundle server and forwarding them to the application's server. This server is hosted separately by each application. It is also responsible for storing any additional information regarding the client so that it can successfully connect to the application server and get data on behalf of the client.

The definition for this service is shown in figure 13. Here, PrepareData RPC is used by the bundle server to notify the application adapter that it needs to collect data for the specific client IDs and store them locally. We need this method because the process of fetching data from the application's server may be time-consuming. By calling this method as soon as the bundle server receives data from transport, the PreparaData method can run in parallel with bundle decryption and other computations done by ADM and Bundle Security module.

```
service DTNAdapter {
  rpc saveData (AppData) returns (AppData) {}
  rpc prepareData (ClientData) returns (PrepareResponse) {}
}

message AppData {
  string clientId = 1;
  repeated AppDataUnit dataList = 2;
  int64 lastADUIdReceived = 3;
}

message AppDataUnit{
  int64 aduId = 1;
  bytes data = 2;
}

message PrepareResponse {
  StatusCode code = 1;
}

enum StatusCode {
  SUCCESS = 0;
  FAIL = 1;
  CLIENT_NOT_REGISTERED = 2;
}

message ClientData{
  string clientId = 1;
}
```

Figure 13: GRPC service for application adapter server to receive data from bundle server

After the bundle is decrypted and stored in the file system, the bundle server asks for data by calling saveData RPC. The adapter will send data to the application server and collect their response. The response and the data generated during the PrepareData method call will be returned to the bundle server.

If there is no data to be sent in the SaveData call, it can just send the client ID list and get the data to be sent to the client. Figure 14 summarizes the sequence of steps performed at the bundle server when it receives a bundle from the transport

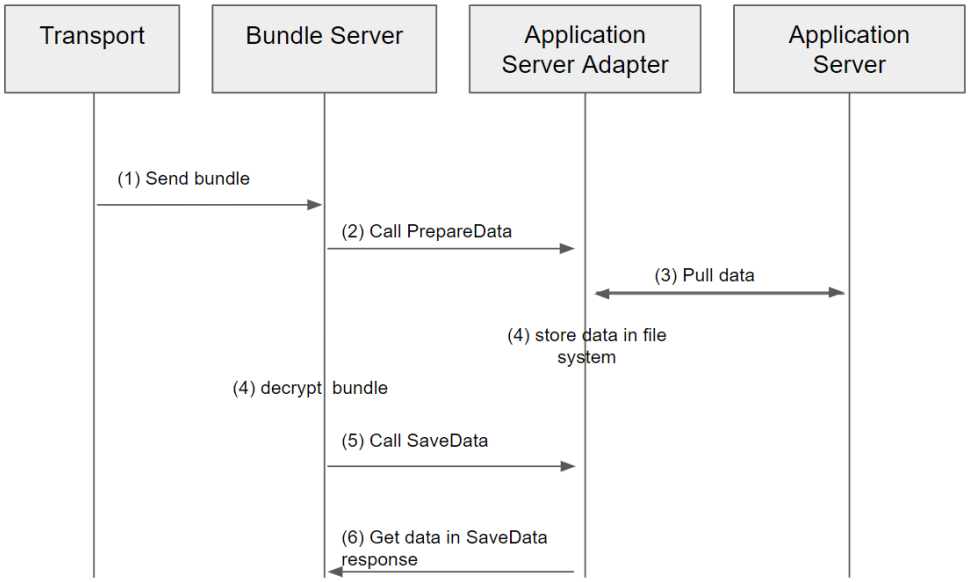

Figure 14: Flow at Bundle Server when it receives bundle from transport device

device.

4.6.1 Data synchronization between adapter and bundle server

When data is sent between the adapter server and the bundle server, there is a possibility that the channel between them might go down due to network failure or an error in code. If this happens, the sender does not know if the data was delivered successfully or not. To prevent this, the sender of data will assign an ADU ID to each unit of data. The receiver of data keeps track of ADU IDs that have been received so far and should provide this information to the sender. This allows the sender to resend ADUs that were not received in a previous communication. Once the sender receives confirmation that the data was sent successfully, it can delete the data from the local storage.

For example, after the adapter gets client-specific data from the application server, it will store the data in the local file system until they are successfully sent to the application server. The adapter knows of this when the bundle server includes the

latest ADU ID it has for the client in the input for prepareData RPC.

4.7 Integration with Signal Application

The Signal application communicates with the Signal server for three main reasons; registering the user with the server, sending or receiving messages, and sending or receiving availability statuses. To send data to Signal Server, the Android application first checks if the device has network access, then encrypts data and sends data to the server over a web socket.

To allow the application to send data over DTN, we additionally set the network connectivity parameter to true if the Bundle Client application is installed (by checking if the content provider in it is available). In this case, we serialize the encrypted data and send it to Bundle Client using the content provider.

Until now, ADUs are considered to be an array of bytes. But, it is possible that some applications might want to send additional parameters, such as the type of request, user metadata, or data type. To send these types of data, the application can encapsulate it into the JSON format, which can then be converted into an array of bytes and sent to the DTN service.

4.7.1 Registration of a new user on to Signal

The first step of integration with Signal is to start with the registration process. When a user first opens the application, they press the "Get Started" button to proceed with registration by entering the phone number. Since the registration process requires verification of the phone number by using a One Time Password which is not possible in a disconnected setting, this process is being modified.

To register a new user on the signal application, the application should generate some public and private key pairs, and send the public part of the keys to the server so that it can keep track of clients and their keys.

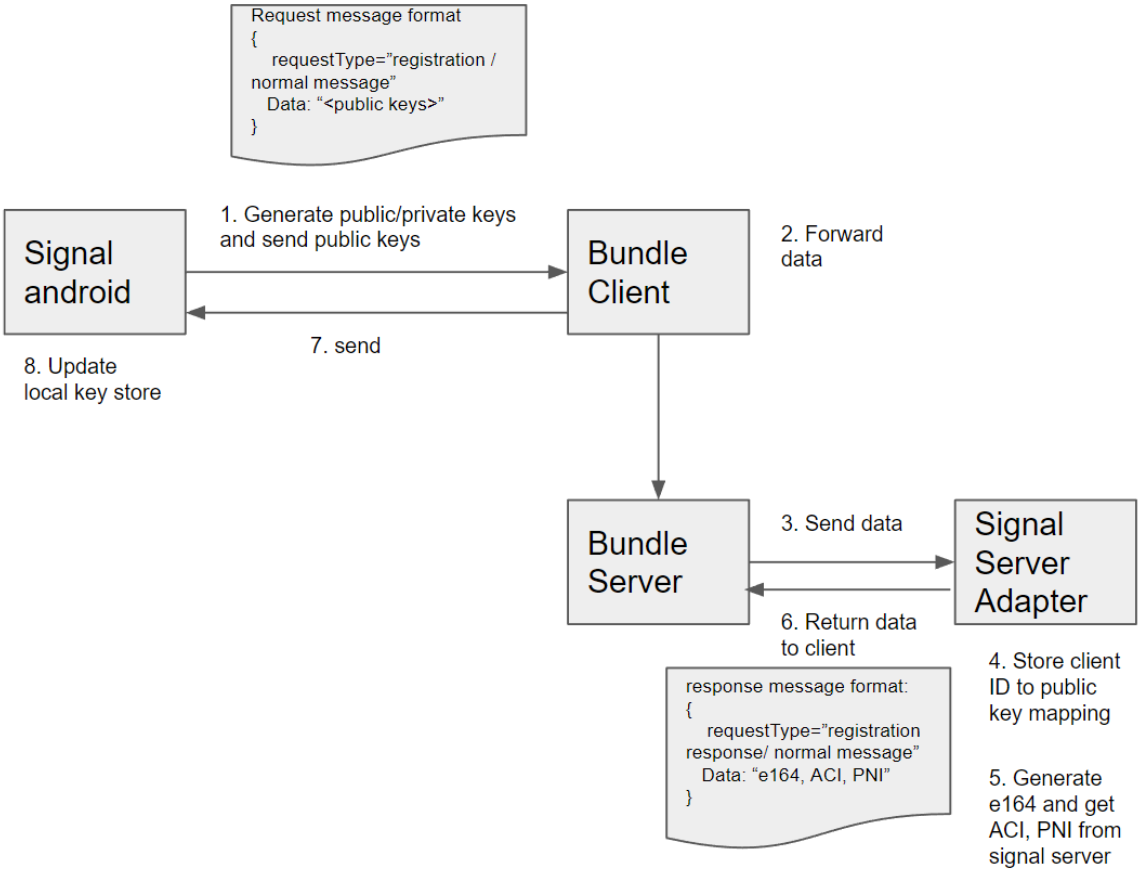

Figure 15: Data flow for Signal application

To send this data over DTN, we encapsulate this key data in a JSON object under the key 'Data'. We also include the request type as 'registration' so that the application adapter server will know which type of request should be called on the application server for the given data. When Signal Adapter Server sends this data to the Signal server, it will receive additional parameters like E164, ACI, and PNI, which will be used by the Signal application to encrypt any message that it wants to send. This flow of data is summarized in figure 15 below.

4.8 Integration with other modules

The implementation details described so far are part of the Data Store Module, as described in section 3.1. This section will describe how this module is placed within

the rest of the system.

4.8.1 Bundle Client

After the Bundle Client receives data from several applications, several steps need to be performed before they can be sent to the transport. The ADUs need to be placed in a bundle, and any ADUs that cross the size limit for a bundle should be kept aside at the moment. If we have previously received any other data from the server (via transport), we should also include a metadata file listing the ADUs that we have received so far. Once we have all the necessary data, they can be grouped into a bundle and sent to the transport.

Another thing to note is that the bundle generation process is triggered in the Bundle Client when it gets in contact with a transport device. It then asks for ADM for a bundle that should be sent to the transport. Here, ADM interfaces with Data Store Module to get the necessary ADUs for the bundle.

Similarly, receiving bundles is initiated with the connection of the transport device with the bundle server. When a bundle is unpacked by the ADM in the bundle server, the server receives acknowledgments along with new data. Based on acknowledgments for the data that the server has sent, ADM tells DSM to delete all ADUs up till a specific ID. The ADM module should only ask for a starting ADU ID because it keeps track of ADUs that have already been delivered successfully and the ADUs that have been sent previously but not acknowledged. All the APIs that we have discussed so far are described in figure 16.

4.8.2 Bundle Server

The integration on the server is similar to that in the Bundle Client as it too requires handling acknowledgments, and exchanging data with the transport device, and with the application. One additional factor that has to be handled here is the

```
class DataStoreModule {

    private FileStoreHelper sendFileStoreHelper;
    private FileStoreHelper receiveFileStoreHelper;
    private Context applicationContext;

    public DataStoreModule(String appRootDataDirectory){...}

    private void sendDataToApp(ADU adu){...}

    public void persistADU(ADU adu) {...}

    public void deleteADUs(String appId, long aduIdEnd) {...}

    private ADU fetchADU(String appId, long aduId) {...}

    public List<ADU> fetchADUs(String appId, long aduIdStart) {...}
}
```

Figure 16: Design for DataStoreModule in Bundle Client

Client ID. For any interaction with the data, the ADM module has to specify the client ID.

Another step that needs to be taken by ADM is to call the PrepareData RPC (as described in section 13) on all application adapters by passing in the client IDs for which it would like to collect data. This call needs to be made as soon as receiving a bundle from the transport device. The information regarding the list of client IDs whose data can be sent via this transport is stored with the Bundle Security module. The security module does this by keeping track of the transport device and the number of times it has successfully delivered to the client or received data from the client. All the APIs that we have discussed so far are described in figure 17.

```
class DataStoreModule {
    private FileStoreHelper sendFileStoreHelper;
    private FileStoreHelper receiveFileStoreHelper;

    public DataStoreModule(String appRootDataDirectory){...}

    public void deleteADUs(String clientId, String appId, Long aduIdEnd) {...}

    //get IP address and port for application adaptor server from database
    private String getAppAdapterAddress(String appId){...}

    //store all data for one app received from transport and send to app adapter
    public void persistADUsForServer(String clientId, String appId, List<ADU> adus) {...}

    private ADU fetchADU(String clientId, String appId, long aduId) {...}

    //check if there is adapter
    //create GRPC connection to adapter and ask for data for the client
    public void saveDataFromAdaptor(String clientId, String appId, AppData appData){...}

    public List<ADU> fetchADUs(String clientId, String appId, Long aduIdStart) {...}
}
```

Figure 17: Design for DataStoreModule in bundle server

CHAPTER 5

Experiments

This chapter will describe the tests performed on the Bundle Client, Bundle Server, and application adapter server.

5.1 Modifications on client device

In this section, we will describe the experiments done on an Android device containing the Bundle Client and a sample client application.

5.1.1 Bundle Client

To make sure that the Bundle Client is able to receive data and push data to applications, we create a sample application that can take in user input, and send this input to the Bundle Client using its content provider. We also check that the sample application is also able to receive data from the Bundle Client using the intent service.

To simulate the Bundle Client receiving data for an application, we create a simple UI where the user can enter a message and specify the application ID to which the message should go. We can then see this message being sent to the application and stored in its file system.

By entering a message, specifying the application ID, and clicking on send, we were able to store a simple message and send it to the application. If the send operation is successful, the data is deleted from the Bundle Client file storage. To check if the file is present or not, Android Studio's Device Manager can be used for checking the file system on the phone.

5.1.2 A sample client application

Before we test the DTN architecture on the Signal application, we test it with a sample application, which we call MySignal. Here, we simulated an Android application that wants to make use of the DTN architecture to send and receive data from its application server. When it wants to send some data, it checks if the Bundle

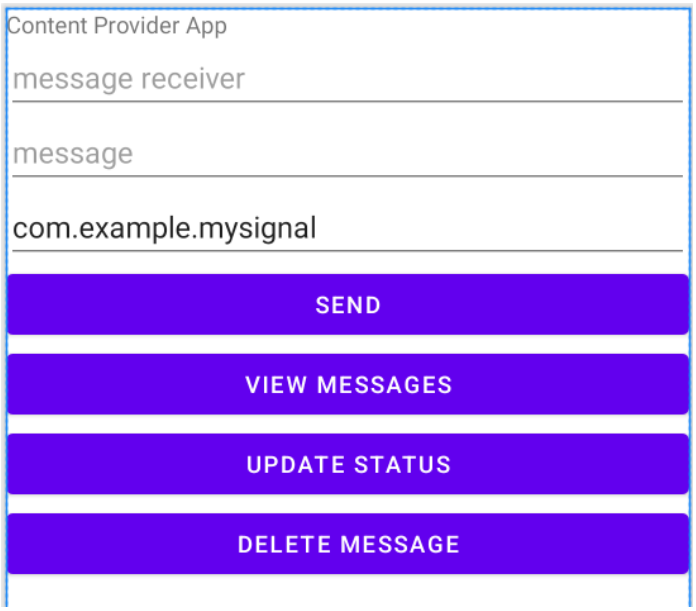

Figure 18: UI for Bundle Client application

Client is installed, and then decides to send data to it. If MySignal sends data of size crossing a configured limit, the Bundle Client will return an error message. When the Bundle Client receives the first message from this application, the file structure described in earlier sections is generated and a file named "1.txt" is created. We also create an entry in 'REGISTERED_APP_IDS.txt', which is used later by the ADM to fetch ADUs for creating the bundle. Any subsequent messages sent by MySignal are stored in sequential order in the same directory. We can also see that the metadata file too reflects the correct values based on these insert operations.

This application also implements a Receive Intent Service, which receives all data generated by the application server from the Bundle Client. To test this, we go to Bundle Client and enter a message and the package name of the MySignal application. When we click on the 'Send' button, the data is sent to the Receive Intent Service of MySignal, which then stores the file in its storage.

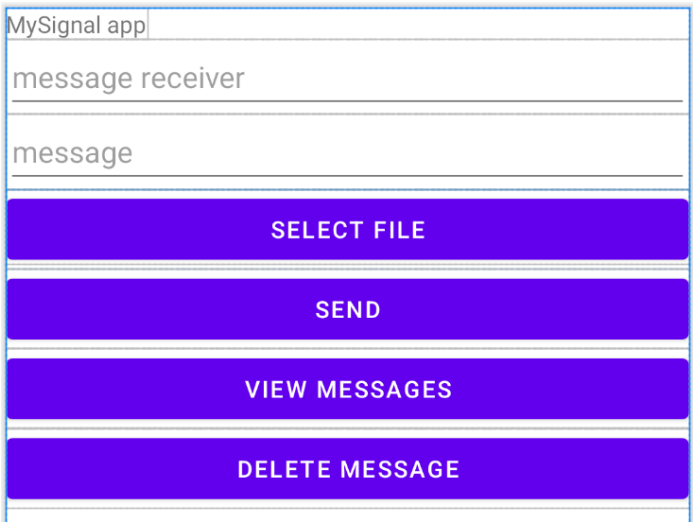

Figure 19: UI for a sample client application

5.1.3 Signal Implementation

The first step of integration with Signal is to start with the registration process. When a user first opens the application, they click on the "Get Started" button to proceed with registration. On this button press, all necessary keys are generated and stored in a file. The generated data is sent to the Bundle Client and stored as an ADU file.

5.2 Integration of data store module with Bundle Client

All experiments explained so far are done only with the Data Store Module, as described in 3.1. This section will detail the experiments on the integration with the other parts of the application.

5.2.1 Bundle Client

As seen earlier, the process of generating and sending a bundle starts when the Bundle Client detects the presence of a transport device and connects with it. The first step is to receive any bundles that the transport has for this client's device. When the ADM module receives the bundle, it will unpack the bundle and check

for acknowledgments. It shares this information with the Data Store module, which deletes these ADUs and informs the respective applications that the specified ADUs have been delivered successfully. Now, the Data Store module processes the ADUs in the received bundle. The processing is successful because we can see in the application the ADUs that were sent from the transport device.

After all the ADUs have been processed, the Bundle Client starts the bundle generation process that needs to be sent to the transport device. the ADM module collects ADUs from the Data Store module and starts generating a bundle. To test if the operation was successful, we store the generated bundle on the Bundle Client storage to see the contents of the bundle that was delivered by the Bundle Client to the transport device. Here, we can see that the messages sent by the MySignal client application were correctly present inside the bundle folder.

5.2.2 Bundle Server

Similar to the Bundle Client, when a transport device connects with the Bundle Server, the transport should first send any bundles it has in its storage. After the ADM module unpacks the bundle and sends the ADUs to DSM, the DSM sends these ADUs to the respective application adapter servers. As an experiment, when we generate a sample bundle and send it from transport, we can see that the ADUs were received successfully by the application adapter server because they were found in its file storage.

Now, we need to test if the Bundle Server generated the bundle and was able to send it to the transport device. To test this, we create some sample client data in the application adapter server (to simulate receiving of data from the actual application server). Here too, we store the generated bundle in the Bundle Server file system so that we can clearly see the bundle contents. After the bundle is delivered successfully

to the transport device, we can see that the bundle contains the sample data that was generated in the application adapter server.

5.3 DTNCommunication Service

As described in section 4.5, this service is responsible for maintaining the location of application servers. To test the working of this service, a new application adapter needs to register to this DTN by calling the registerAdapter RPC. After execution of this RPC, a new entry is created in the registered_app_adapter_table table. If the same method is called again on the same application ID, an error is returned saying that an adapter for the specified application is already registered.

5.4 Application adapter server

The central part of this server is to receive and send data to the Bundle Server. This is done when the Bundle Server calls the SaveData RPC on the application adapter service. We test this by creating a GRPC client for this service and passing some simple plain text data, a client ID, and an application ID. After execution, we can see that the adapter server has processed this message by writing it to a file and sending an acknowledgment message back to the Bundle Server.

CHAPTER 6

Future Work

In this chapter, we will take a look at how this project can be extended.

6.1 Intermittent connectivity on client device

People in disconnected regions might experience intermittent connectivity while traveling within disconnected regions. During this brief period of connectivity, the data that could not be sent previously will be sent to the respective application server. Since the application has already sent this data to the Bundle Client, this application will need to inform Bundle Client that the data has already been sent the data via the internet and that the Bundle Client can discard this data. The Bundle Client should also share this information with Bundle Server so that the server can discard this data when it arrives from the transport devices. Figure 20 shows this scenario in detail.

6.2 Data fragmentation

A lot of modern applications such as Youtube, Facebook, and Netflix transfer large amounts of data over the internet. Since there are data limit constraints on how much the Bundle Client and the transport device can store, additional policies are required for sending these vast amounts of data over this DTN. A simple option would be to fragment a large piece of data into several ADUs before sending it to the transport device. Once all fragments are received on the other side, they can be merged to form the whole data and then processed.

Another option might include setting lower priority and higher data limits on data-heavy applications. A user interface can also be created on the Bundle Client to allow the user to set a higher priority on some applications so that data for these applications are delivered first.

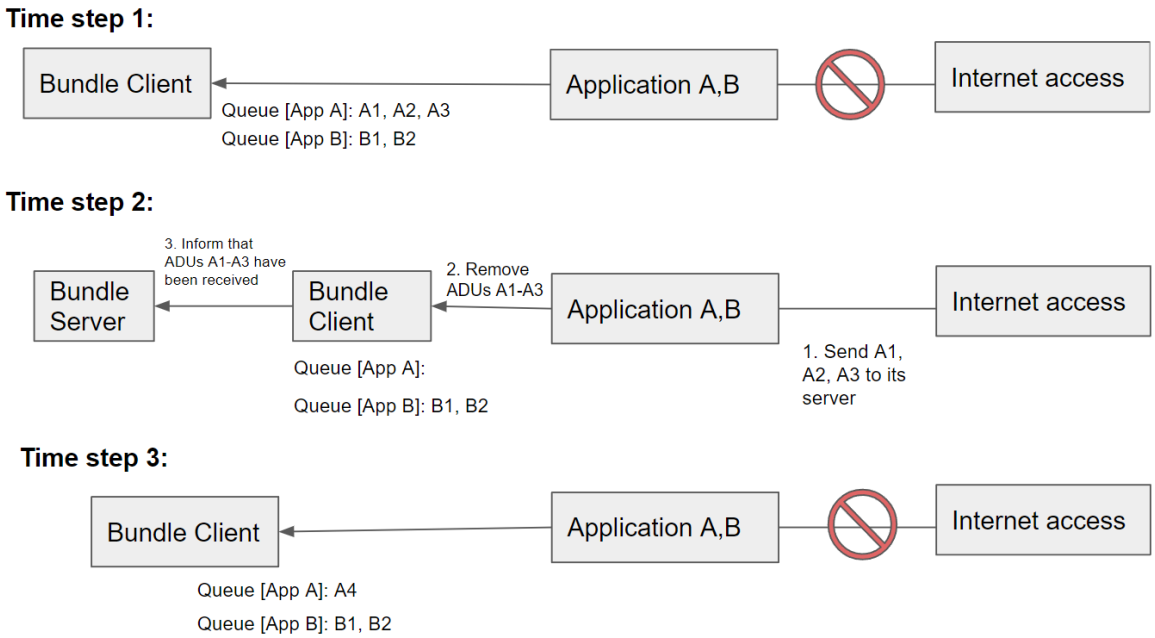

Figure 20: Scenario in case of intermittent connectivity

6.3 Improving security

In this entire DTN, the communication between the Bundle Server and the Application Adapter Servers is the only channel that is not encrypted. Even though applications ensure that the data is encrypted before sending from the user's phone or the servers, some additional information like sender, receiver, and type of data remains exposed. This allows malicious agents to eavesdrop on the gRPC communication between Bundle Server and Application Adapter Server. Encryption of data before sending can improve the overall security of the DTN.

CHAPTER 7

Conclusion

Several research has shown that a large percentage of the population is still disconnected from the internet. We took a look at this problem and designed an architecture that builds on existing tools, without depending on any additional hardware. By using the Android platform, we show that it is feasible to connect a user in a disconnected setting with the internet. This is made possible with the use of "transport" devices that act as carriers of data between a user in a disconnected region and the internet. We have also shown that it is possible to collect application data on the users' phones so that they can be sent to transport devices. We have also shown that the data that the users' phones receive from the transport can be sent back to the respective applications.

In the connected region, we show how the Bundle Server can take the application data from users' phones, and send them to their respective destinations. We then show how the Bundle Server collects new data for the users and send them back to the transport devices.

We also show that the open-source Signal application can make use of this DTN architecture to perform user registration and pass messages between the user and the Signal server. This demonstrates the feasibility of this DTN with modern applications and provides a design that other applications can follow to enable connectivity in disconnected regions.

LIST OF REFERENCES

[1] I. W. Stats, ‘‘https://internetworldstats.com/stats.htm,’’ 2022.

[2] L. Zhen, K. Wang, and H.-C. Liu, ‘‘Disaster relief facility network design in metropolises,’’ *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 5, pp. 751--761, 2015.

[3] K. Fall, ‘‘A delay-tolerant network architecture for challenged internets,’’ in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 27--34.

[4] Google, ‘‘play.google.com/store/apps/details?id=org.thoughtcrime.securesms,’’ 2022.

[5] M. Li, P. Si, and Y. Zhang, ‘‘Delay-tolerant data traffic to software-defined vehicular networks with mobile edge computing in smart city,’’ *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9073--9086, 2018.

[6] A. Krifa, C. Barakat, and T. Spyropoulos, ‘‘Optimal buffer management policies for delay tolerant networks,’’ in *2008 5th annual IEEE communications society conference on sensor, mesh and ad hoc communications and networks*. IEEE, 2008, pp. 260--268.

[7] A. Lindgren and K. S. Phanse, ‘‘Evaluation of queueing policies and forwarding strategies for routing in intermittently connected networks,’’ in *2006 1st International Conference on Communication Systems Software & Middleware*. Ieee, 2006, pp. 1--10.

[8] R. Patra and S. Nedevschi, ‘‘Dtnlite: A reliable data transfer architecture for sensor networks,’’ *CS294-1: Deeply Embedded Networks (Fall 2003)*, 2003.

[9] D. Gay, ‘‘Design of matchbox, the simple filing system for motes,’’ 2003.

[10] J. Scott, J. Crowcroft, P. Hui, and C. Diot, ‘‘Haggle: a Networking Architecture Designed Around Mobile Users,’’ in *WONS 2006 : Third Annual Conference on Wireless On-demand Network Systems and Services*. Les Ménuires (France): INRIA, INSA Lyon, Alcatel, IFIP, Jan. 2006, pp. 78--86, http://citi.insa-lyon.fr/wons2006/index.html. [Online]. Available: https://hal.inria.fr/inria-00001012

[11] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, ‘‘Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation,’’ *Electronic Communications of the EASST*, vol. 37, 2011.

[12] C. E. Palazzi and A. Bujari, ‘‘Social-aware delay tolerant networking for mobile-to-mobile file sharing,’’ *International Journal of Communication Systems*, vol. 25, no. 10, pp. 1281--1299, 2012.

[13] C. Caini, P. Cornice, R. Firrincieli, M. Livini, and D. Lacamera, ‘‘Dtn meets smartphones: Future prospects and tests,’’ in *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*. IEEE, 2010, pp. 355--360.

[14] P. Jiang, J. Bigham, E. Bodanese, and E. Claudel, ‘‘Publish/subscribe delay-tolerant message-oriented middleware for resilient communication,’’ *IEEE Communications Magazine*, vol. 49, no. 9, pp. 124--130, 2011.

[15] H. Ntareme, M. Zennaro, and B. Pehrson, ‘‘Delay tolerant network on smartphones: Applications for communication challenged areas,’’ in *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition*, 2011, pp. 1--6.

[16] K. Sankaran, A. L. Ananda, M. C. Chan, and L.-S. Peh, ‘‘Dynamic framework for building highly-localized mobile web dtn applications,’’ in *Proceedings of the 9th ACM MobiCom workshop on Challenged networks*, 2014, pp. 43--48.

[17] S. Carvalho, R. N. de Lima, and A. G. da Silva-Filho, ‘‘A pushing approach for data synchronization in cloud to reduce energy consumption in mobile devices,’’ in *2014 Brazilian Symposium on Computing Systems Engineering*. IEEE, 2014, pp. 31--36.

[18] K. Scott and S. C. Burleigh, ‘‘Bundle Protocol Specification,’’ RFC 5050, Nov. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc5050

[19] D. D. Clark and D. L. Tennenhouse, ‘‘Architectural considerations for a new generation of protocols,’’ in *Proceedings of the ACM Symposium on Communications Architectures Protocols*, ser. SIGCOMM ’90. New York, NY, USA: Association for Computing Machinery, 1990, p. 200–208. [Online]. Available: https://doi.org/10.1145/99508.99553

[20] M. Skjegstad, F. T. Johnsen, T. H. Bloebaum, and T. Maseng, ‘‘Mist: A reliable and delay-tolerant publish/subscribe solution for dynamic networks,’’ in *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, 2012, pp. 1--8.

[21] L. Wood, P. Holliday, D. Floreani, and I. Psaras, ‘‘Moving data in dtns with http and mime,’’ in *2009 International Conference on Ultra Modern Telecommunications Workshops*, 2009, pp. 1--4.

[22] V. Alves and P. Borba, ‘‘Distributed adapters pattern: A design pattern for object-oriented distributed applications.’’

[23] R. Sears, C. Van Ingen, and J. Gray, ‘‘To blob or not to blob: Large object storage in a database or a filesystem?’’ *arXiv preprint cs/0701168*, 2007.

[24] S. Ronglong and C. Arpnikanondt, ‘‘Signal: An open-source cross-platform universal messaging system with feedback support,’’ *Journal of Systems and Software*, vol. 117, pp. 30--54, 2016.