Fall 2019

Design and implementation of applications over delay tolerant networks for disaster and battlefield environment

Karthikeyan Sachidanandam

DESIGN AND IMPLEMENTATION OF APPLICATIONS OVER DELAY TOLERANT NETWORKS FOR DISASTER AND BATTLEFIELD ENVIRONMENT

by

KARTHIKEYAN SACHIDANANDAM

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2019

ABSTRACT

In disaster/battlefield applications, there may not be any centralized network that provides a mechanism for different nodes to connect with each other to share important data. In such cases, we can take advantage of an opportunistic network involving a substantial number of mobile devices that can communicate with each other using Bluetooth and Google Nearby Connections API(it uses Bluetooth, Bluetooth Low Energy (BLE), and Wi-Fi hotspots) when they are close to each other. These devices referred to as nodes form a Delay Tolerant Network (DTN), also known as an opportunistic network. As suggested by its name, DTN can tolerate delays and significant loss of data while forwarding a message from source to destination using store and forward paradigm. In DTN, it is of critical importance that the network is not completely flooded and also the message is not tampered or corrupted and readable only to the destined node.

Three algorithms have been implemented in the Android platform. The first algorithm [1] focuses on intelligent data transfer based on each node's interest and encourages each node to participate in data transfer by providing incentives and keeping track of the trustworthiness of each node. The second algorithm [2] focuses on the security of the transferred data by fragmenting both data- and key-shares with some redundancy and the destination node can resurrect the original data from the predefined minimum key- and data-shares. The third algorithm focusses on using object detection models and interest-based authorization using [3] to securely transfer and access data across DTN. The corrupted nodes are identified by using one-way keychain hashes created by source/relay nodes for a message which are validated at the destination node.

ACKNOWLEDGMENTS

I would like to deeply express my gratitude to my advisor Dr. Sanjay Madria for pointing me in the right direction for my research and helping me throughout the master’s program. He pushed me to learn new technologies and helped me to refine the implementation of various projects. I would also like to thank Dr. Ricardo Morales and Dr. Venkata Sriram Siddhardh Nadendla for being part of my thesis committee and taking time to review my thesis and provide their valuable feedback.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

In recent years, there has been a tremendous increase in the number of mobile devices, and these devices have the capability to communicate with each other using peer-to-peer technologies such as Bluetooth, BLE, and Wi-Fi hotspots. In disaster/battlefield scenarios there may not be any centralized network like LAN to facilitate connection among these devices. In such cases, it is necessary to pass on important information from the source device to the destination using wireless links. In such cases, the mobile devices can form their own wireless opportunistic network referred to as Delay Tolerant Networks (DTN). DTN is formed by a collection of nodes and each node interacts with other nodes only when they are within a short distance. Unlike traditional end-to-end connections, DTN has intermittent connections only when two nodes are nearby each other. Instead of seeing the intermittent connection as a disadvantage, DTN utilizes it for sharing relevant information between the nodes to forward the message to the destination node. Thus, message transfer from a source node to the destination may pass on through various intermediary nodes resulting in delay and significant data loss.

In this thesis, we focus on the implementation details of three algorithms; two are based on [1] and [2] and the third one is newly proposed. Reputation and Incentive-based DTN [1] focuses on intelligent data transfer between nodes, based on the interests set by the node and interests accumulated as the result of interaction with other nodes. Incentives are provided to encourage nodes to participate in data transmission. Distributed Reputation Metric (DRM) is like a local database maintained by each node, which contains a value corresponding to the trustworthiness of each encountered node based on the message received from it, the quality of the message, the message's annotation relevance, etc. When the nodes connect, the DRM is shared between them. This scheme reduces flooding of the network, creates multiple copies of the original data based on the interests between two connected nodes, and ensures that all the nodes participate in the data transmission and also try to suppress corrupted nodes.

The secure forwarding through fragmentation in DTN [2] overcomes delay, data loss, and insecure data transfer between nodes by fragmenting and creating redundant data- and key-shares. Thus, the destination node can resurrect the original data if it contains a minimum number of predefined key- and data-shares.

Lastly, in the newly proposed DTN algorithm, we have used Attribute-Based Encryption(ABE) [3] to design interest-based authorization to securely transfer and access data across DTN. We also keep track of intermediate nodes in the message path to identify malicious nodes using µTESLA algorithm [4]. Furthermore, we have used deep learning object detection models on an Android device to reduce the message size by identifying objects of interest from the device's camera.

The thesis is organized in the following way. Section 2 provides an overview of related work in the field of data routing in DTN and implementation of DTN and image object detection in the Android platform. Section 3 provides a brief introduction to various APIs and libraries that have been used in the implementation. Section 4 provides the background to understand the actual implementation, this involves explaining the algorithm and overall architecture of the system, as well as the implementation of DTN and object detection in the Android app with various modules, message formats, and screenshots. Section 5 provides a conclusion for the thesis.

2. LITERATURE REVIEW

2.1. CHITCHAT

Doug et al. [5] proposed a novel solution to achieve a high message delivery rate without flooding the DTNs. In this paper, each node has a set of interests called *Direct Interests*, which represents the interests of a node. *Transient Interests* of a node represents the set of interests that the node accumulated by meeting other nodes. When two nodes interact with each other, a decay function is applied to the interests. Next, they exchange the interests, and the growth function is applied to the modified new set of interests. Each created message is annotated with a set of keywords or interests. Once the growth function has been completed, the routing protocol is applied to decide whether a message with certain interests needs to be transferred to the connected node. This enables the message to have a high probability of reaching the set of users who are interested in that message.

2.2. REPUTATION AND CREDIT-BASED INCENTIVE MECHANISM

Himanshu et al. [1] proposed an incentive and reputation model on top of ChitChat algorithm [5]. In this scheme, the messages are annotated with keywords describing them and the relay nodes are encouraged to add more keywords so that the message contains real time information. This is especially useful in volatile situations like disaster/battlefield environment. The incentive model encourages users to participate in message transfer without turning into selfish nodes to preserve energy and storage capacity. The reputation model prevents the nodes from adding inappropriate keywords to the message.

2.3. MOBICENT

Chen et al. [6] proposed an incentive mechanism on top of existing DTN routing protocols. By providing incentives, even selfish nodes are encouraged to participate in relaying the message to the destination. Mobicent assumes that the underlying system uses cryptographic techniques like Onion Routing to prevent free riding, any strategies by participants to override the system, and disputes among relays and clients. A Trusted Third Party (TTP) calculates the payment that will be charged to the client, and two algorithms are available to the client to choose from. One minimizes the message delay and the other minimizes the payment. One of the limitations of Mobicent is that it uses TTP for both key management and payment services to relays.

2.4. RELICS

Uddin et al. [7] proposed a novel scheme that considers the trade-off between energy consumption and the cooperation of nodes for data transmission in DTN. When a message is delivered to the destination node, the destination node assigns a rank or real value to a set of relay nodes for that message and it will be shared with other devices. This rank directly impacts the node in a way that relay nodes favor to forward message from a high-ranking node than the ones with a lower rank. This study [7] also provides an algorithm for intelligent transfer of a certain number of messages based on energy consumption. Since most of the nodes are either mobile or sensor devices it is really important to give utmost importance to the energy savings (as they have limited energy) rather than giving top priority to forwarding the messages in the network.

2.5. MOBIGAME

Wei et al. [8] proposed a user-centric, reputation-based incentive protocol for DTN. This scheme uses game theory to decide whether a node should receive or forward a given message. Each message has a limited time to reach a destination, which is called *Time To Live(TTL)*. If a message received by a relay node has expired it is called a bad bundle. Each relay node gets a reward only if it successfully forwards the message to the other relay nodes or destination and provides proof of that to its predecessor. Using game theoretical analysis, [8] proves that each node transfers only the good bundle to the relay nodes and avoids being a malicious node to receive rewards.

2.6. MULTI-PARTY ENCRYPTION(MPE)

R. Cabaniss et al. [9] proposed a novel scheme for security in DTN. In traditional end-to-end connection, public-key cryptography is easily achieved by using a trusted key repository to verify which node is associated with a particular public key. However, in DTN, a node may not have access to the public key of all nodes and there is no trusted key repository. Thus, [9] proposes two schemes to ensure security: i) Chaining and ii) Fragmentation. Chaining is similar to onion encryption, in which the source message is encrypted with multiple public keys of relay nodes, and when each relay node receives the message, it removes its layer of encryption and replaces it by encrypting using the public key of the destination node if it is available. The node otherwise replaces it by encrypting using the public key of any other relay node. This increases security but reduces delivery time. In fragmentation, multiple copies of the original data are created. Threshold encryption is used in which *n* keys are created and *k* keys are required to retrieve the original message, where $k<n$. Each message copy is encrypted with a key-share and passed along the network. When the destination receives k key fragments, it can retrieve the original data.

2.7. SECURE SECRET SHARING

Parakh et al. [10] proposed a secure secret sharing scheme on top of Shamir's secret sharing algorithm. The proposed method can hold k-2 secrets of size b into n key-shares. The destination requires at least k key-shares to retrieve the secrets, where $k < n$. This method is especially useful in the DTN environment as the data loss is high and the connection may be interrupted at any time. The secrets may store the private key to decrypt the original message or other vital information related to the message.

2.8. INFRASTRUCTURE MODE BASED DTN ON ANDROID DEVICES

Ippisch et al. [13] explored the use of Wi-Fi hotspots in Android devices to create a DTN network from a set of devices. Since the Android system requires user intervention to accept connections in Wi-Fi Direct and Bluetooth, this study uses Wi-Fi hotspots, which can connect and transfer data automatically without user intervention in the background. The Spray and Focus routing protocol are used with TTL for each message. The application also makes use of GPS services, which helps to formulate location-based rules to enable hotspots that in turn help to conserve battery. When two devices are connected they exchange meta-information that contains information about the network, and the meta-information helps to make optimized message-forwarding decisions. Finally, the study [13] focuses on analyzing energy consumption, transmission range and speed of Wi-Fi hotspots, and the encryption and decryption speed in the varying devices and data sizes.

2.9. OPTIMIZING CAUCHY REED SOLOMON-CODES

Cauchy Reed Solomon (CRS) codes were already introduced in the 1990s; Optimized CRS [11] tried to improve its performance drastically so that a small number of computations are needed to perform, which in turn saves energy. CRS codes are used to create n fragments of a message, and from that, we need at least k fragments to resurrect the original message, where $k<n$. This data fragmentation comes in handy especially in DTN, and this study helps to reduce the energy consumption in creating fragments and retrieving the original data as the mobile devices have limited battery power at any given time.

2.10. SECURE FORWARDING THROUGH FRAGMENTATION IN DELAY-TOLERANT NETWORKS

Shudip et al. [2] proposed a scheme to forward messages in a secure way in DTN, especially in disaster/battlefield environment where multiple parties may work together. In such scenarios, it is difficult to use public-private cryptography as one group may not have access to the public key of other groups as there will be no central infrastructure. To overcome this limitation, the study proposed to encrypt the message and fragment both key and encrypted message with some redundancy before passing along the DTN. Since DTN has high data loss, this redundancy helps the destination node to retrieve the original message with fewer key and data fragments than created at the source.

2.11. DTN ON ANDROID PHONES

Yanggratoke et al. [12] were one of the first to implement DTN in the Android platform. They assume a simple DTN routing protocol and explain the overall system architecture for Android implementation. The architecture includes a background service, a user interface to interact with the service, and an Android TCP/IP stack which interacts back and forth with the background service. The authors discuss the challenges of implementing DTN in the Android system. Performance in terms of bandwidth and battery power consumption is also analyzed.

2.12. CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

The study [3] proposed a scheme in which messages are encrypted with policy and it can be decrypted if the attributes or condition specified in the policy are satisfied at the destination. Each user has a private key representing a set of attributes and the message is encrypted with a policy over these attributes. The authors provide various optimization strategies to reduce decryption time. The authors guarantee that this scheme is resistant to collusion attacks. This scheme is closely similar to traditional access control protocol such as Role-Based Access Control(RBAC).

2.13. AI BENCHMARK: DEEP NEURAL NETWORKS ON ANDROID

The authors [14] explored various mobile hardware chipsets such as Qualcomm, HiSilicon Chipsets, and MediaTek Chipsets which provide dedicated APIs to run deep neural networks using hardware acceleration. Various neural network frameworks for mobile devices such as TensorFlow Lite, Theano, and Caffe2 are explored and their advantages and disadvantages are described briefly. Android Neural Network API(NNAPI) provides a layer of abstraction, enabling developers to run neural network operations on different chipsets without having to write separate code for each chipset using its designated API. NNAPI provides a unified framework for working with hardware for neural network operations. Finally, various deep learning tests such as image recognition, face recognition, and segmentation are performed with different convolutional neural networks such as Inception-V3, and MobileNet on different mobile devices with varying RAM and hardware chipsets. A detailed comparison report is presented at the end.

3. API'S AND LIBRARIES

Some of the important APIs and libraries that helped to fasten the implementation and improve the efficiency of the overall system and core components of the implementation are discussed here.

3.1. GOOGLE NEARBY CONNECTIONS API

Google Nearby Connections is an API developed by Google for making peer-to-peer networking with Android devices simple and secure and performing data transfers at a high bandwidth. Traditionally, developers need to write separate code to make use of Bluetooth, BLE and Wi-Fi hotspots. It is a very tiresome task and end users' intervention is needed to accept connections from a nearby device. To solve that problem, this API leverages the strength of Bluetooth, BLE and Wi-Fi hotspots while circumventing their respective weaknesses. It performs advertising, discovery, and connections with nearby devices in a fully offline peer-to-peer manner. Furthermore, based on a connection with a nearby device, it automatically decides whether to use Bluetooth or Wi-Fi Hotspots and end-user intervention is not needed to switch them on. Using this API, it is possible to send a large volume of data between two Android devices at a high speed.

3.2. JPBC AND JAVA REED-SOLOMON LIBRARY

Java Pairing-Based Cryptography(JPBC) [15] is an open source library and in our implementation, it is used to create public and private keys using Elliptical Cryptography and to perform proxy re-encryption which will be explained in section 4. Java Reed-

Solomon Library helps to fragment a data into *n* blocks and at least *k* blocks are required at the destination to retrieve the original data, where $k<n$.

3.3. TENSORFLOW OBJECT DETECTION API

The Tensorflow Object Detection API [16] was built by Google to make the process of multiple object detection in a single image as easy as possible. It is built on top of the TensorFlow framework and simplifies the process of training and deploying object detection models. This API provides a wide range of configuration files that depict the layers of Convolutional Neural Networks(CNN) such as MobileNet and Inception models. The developer can modify these files to set the training and validation size, threshold value, batch size, etc. This API also provides an easy way to perform transfer learning which helps to reduce the training time. Once an object detection model is trained, it can be converted to the relevant format such as the .pb format or .tflite format. Tflite format stands for TensorFlow Lite, which is especially useful for deploying the models in mobile or embedded devices.

3.4. TENSORFLOW LITE

TensorFlow Lite is a framework which helps to run inference on TensorFlow models in mobile, IoT, or embedded devices. It is designed to be lightweight and optimized for small devices. The pre-trained models should be converted to TFLite format. TFLite format reduces the size of the model and helps to perform inference in limited memory constraints. TensorFlow Lite internally uses NNAPI, which in turn helps to optimize the process of solving computationally intensive tasks by distributing the workload across on-device processors, such as dedicated neural network hardware, graphical processing units (GPUs), and digital signal processors (DSPs). In case such dedicated hardware is not available in a mobile or embedded device, TensorFlow Lite then requests NNAPI to optimize the code to execute requests on the CPU.

3.5. JAVA REALIZATION FOR CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION

Java realization for ciphertext-policy attribute-based encryption [17] provides java implementation of [3] for generating public and master key from a set of attributes e.g. [‘soldier’, ‘army’, ‘disaster’, ‘flooding’]. The private key is generated from the subset of previously used attributes such as [‘army’, ‘soldier’]. The message is encrypted with a policy. The policy contains attributes and k out of n feature, where $k<n$ such as [‘army soldier 1of2’]. Thus, the encrypted message can be decrypted if the private key was generated with either ‘army’ or ‘soldier’ as one of its attributes.

4. DESIGN OVERVIEW & IMPLEMENTATION

This section provides a design overview and implementation details of three studies of DTN algorithms that have been implemented in the Android platform.

4.1. STUDY 1: REPUTATION AND CREDIT-BASED INCENTIVE MECHANISM FOR CONTENT ENRICHMENT IN DTN

4.1.1. Background. Himanshu et al. [1] proposed an incentive scheme in DTN in order to encourage all the nodes to participate in relaying the information to the destination nodes. This is important because in real-time scenarios the nodes may turn selfish in order to conserve battery and memory, which are limited in mobile devices. Each message has a set of keywords or annotations describing it. The incentive provided for each received message is calculated based on various factors like the reputation of the source device, interest comparison between the message and the receiving device, and the quality of the message. Himanshu's method [1] allows relay nodes to improve the message description by adding more content as needed. This is especially suitable in battlefield/disaster scenarios, e.g. in a disaster situation an image of a dangerously broken building may be sent to the fire department as the destination via relay nodes so that necessary actions can be taken by the fire department. As time evolves, relay nodes can add the relevant keywords while forwarding this message so that the destination can have appropriate knowledge about the current situation.

In order to prevent nodes from becoming malicious by adding inappropriate keywords for a message, [1] introduced the Distributed Reputation Model (DRM), in which each relay or destination node can rate the source and other relay nodes based on the quality of the message, relevance of the keywords, etc.

The incentive and reputation models are built on top of the DTN routing algorithm called ChitChat [5]. ChitChat has been designed for users with small devices like Android or iOS smartphones that have the capability to connect with nearby devices. Each user can list a set of preferences or interests that describes the user well by listing keywords such as "ice-cream", "trekking", etc. These interests are called the direct interests of the user. Each user also accumulates a set of interests called transient interest, which is the aggregation of interests obtained by meeting other nodes and learning their interests, too. This is really helpful to get an idea about what kind of users each user meets and this information is useful in making message-forwarding decisions. A device is a destination if the message keywords are in sync with the direct interests of the device (i.e. user). In ChitChat algorithms, two modules play an important role one is the Real-Time Transient Social Relationship (RTSR) module and the other is the ChitChat Routing module. Each interest is associated with weights, and direct interests always have a minimum weight of 0.5. All the interests cannot have a value greater than 1. When two devices with the ChitChat app gets connected, the RTSR module is first invoked. The RTSR module performs decay function on both direct and transient interests to better reflect their recent relationships with other nodes.

4.1.1.1. Decay algorithm. When two devices, U and V, are connected with each other, the RTSR module is invoked, which in turn executes the decay algorithm in the respective devices. Once the interests are updated, the latest interest and their weights are exchanged between the devices. The decay algorithm is shown in Figure 4.1.

```
Decay algorithm in device U

For all interest I in device (both direct and transient interests):

    If any of the currently connected device with U has the
    same direct interest as I:
        $W_n = W_p$
    Else:
        If I is a direct interest of U:
            $W_n = (W_p – 0.5) / (\beta * (T_c – T_l)) + 0.5$
        Else:
            $W_n = (W_p) / (\beta * (T_c – T_l))$
End
```

Figure 4.1. Decay Algorithm.

W_n – New weight, W_p – Previous weight, β – Decay constant, T_c – Current timestamp, T_l – previous timestamp at which device with interest *I* was connected.

This decay function is formulated in such a way that interests with high weights reflect that the node frequently meets other nodes with that interest and low weights corresponds to the node rarely meeting other nodes of that interest. Once the interests are shared, the RTSR module then executes the growth function to update the weightage of its interests based on the shared interest obtained from the other device.

4.1.1.2. Growth algorithm. The growth algorithm takes 3 factors into account: one is the weightage of the interests in the device U at time T_s, the time duration for which the 2 devices are connected $T_c – T_s$, and finally, a growth dampening factor ψ that is based on the relationship of interest I in devices U and V. Interest I may be a direct interest in both U and V or a direct interest in U and a transient interest in V, while ψ can take a value between [1, 6] depending on various cases such as those mentioned above. Since the

Weights can only be between 0 and 1, the min function is used to prevent the interest weights from exceeding 1. The Growth algorithm is shown in Figure 4.2.

Δ – Change in weight, $W_v(I)$ – Weight of interest I in device V, T_v – Time at which V established the connection, T_s – Time at which device U and V are connected, ψ – Growth dampening factor, W_n – New weight of interest I in node U, W_p – Previous weight of interest I in node U.

```
Growth algorithm in device U

For all interest I in the device (both direct and transient interests):
        $\Delta = 0$
        If device V is connected:
                $\Delta \mathrel{+}= (W_v(I) * (T_c - T_s)) / \psi$
$W_n = min\{1, \Delta + W_p\}$
End
```

Figure 4.2. Growth Algorithm.

4.1.1.3. ChitChat routing module. Once the RTSR module is done, the ChitChat routing module is executed to identify relevant messages to forward to the connected device. This prevents the device from forwarding messages that have a low probability of reaching the destination via the connected relay. It helps to conserve battery in both the devices. Each message is annotated with a set of keywords or interests. The sum of weights of those interests is calculated in device U and the interest obtained from device V during the decay phase. A message is forwarded only when device $S_v > S_u$.. S_v refers to sum of interests for message M in device v and S_u refers to sum of interests for message M in device u.

4.1.1.4. Incentive mechanism. The incentive mechanism is credit and reputation-based. Both of these factors affect the incentives given to a device. This section explains the credit-based part. Each node is initialized with a value of incentive tokens that can be used for message dissemination and content enrichment. When a source or a relay node forwards a message to another relay node, it promises a certain number of incentive token from a destination node on the successful delivery of the message. A relay follows the same process that a source follows. If a relay A transfers message to relay B, A receives a fraction of incentive from B, if B has a high probability of delivering the message to the destination. The study [1] explains various factors involved in credit-based incentive calculation.

4.1.1.1. Distributed Reputation Model (DRM). The motivation for developing this model is to reduce the number of malicious users who might try to add irrelevant tags to the message in the pursuit of acquiring more incentive tokens. To overcome this problem, a recipient node (another relay node or destination node) can rate the source or other relay nodes of a message. The source is rated based on the message quality and added tags while the relay node is rated based on any of the tags added as part of content enrichment. Rating of a node is the average of ratings of messages received from that node. The device shares this rating with the next hop in the path of message traversal to the destination. The destination utilizes these ratings to decide the number of incentive tokens to be awarded to the delivering node. The study [1], explains in detail about the reputation part in incentive calculation.

4.1.1.2. Data flow between connected devices. Figure 4.3. displays the data flow between two connected devices, Alice and Bob. Once connected, the RTSR Module is invoked, it executes the decay algorithm and shares the interests along with the reputation of the other devices. Once the interests are shared, the growth algorithm is executed and relevant messages are transferred to Bob's device by invoking the message router. These executions are performed in Bob's device as well. Once the messages are received, the users can enrich the message by adding relevant tags and also provide ratings for source and relays nodes (DRM) with respect to the received message.

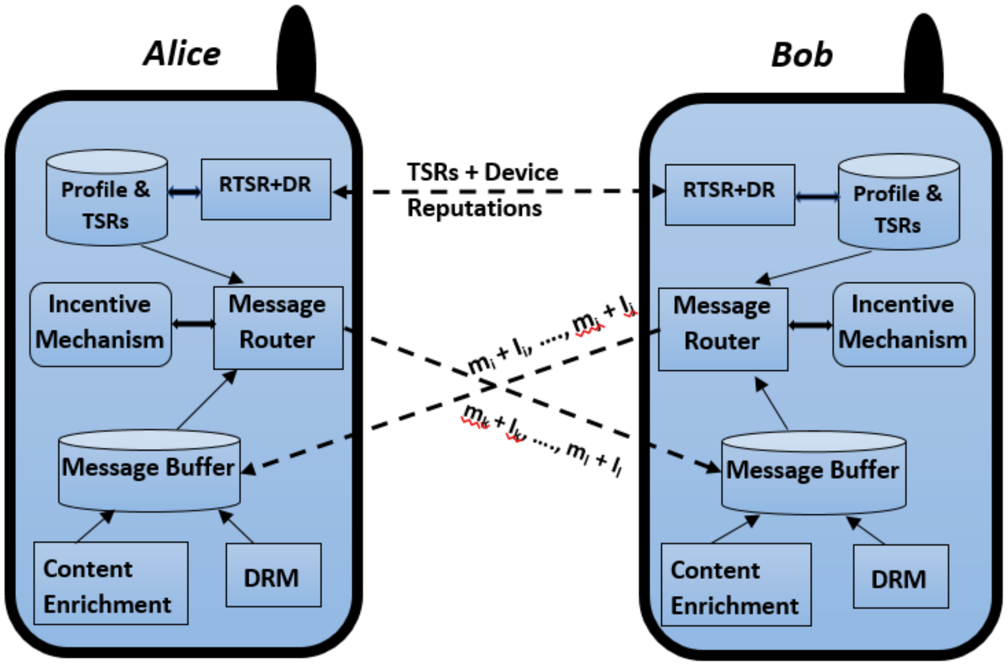


Figure 4.3. Flow of Information between the Connected Nodes [1].

4.1.2. Implementation Details. The DTN application [1] is implemented using the Android operating system and makes use of Google Nearby Communications and Bluetooth to transfer message between devices. It works on devices running on Android's KitKat (API 19) to Oreo (API 26) version.

Figure 4.4 displays the home screen of the app. The "Messages" icon allows the user to view the created messages and the messages received from other devices. The

"Interest" icon allows the user to list their set of interests and also view their transient interests. The "Send to IP" icon allows the user to send all the images and their metadata to the specified IP address. The "Device Ratings" icon lets the user view the DRM database and modify it. The two icons on the right end allow the user to switch the connection mode on and off. When the connection mode is switched on, the device is in a transfer-ready mode which allows messages to be transferred and received from nearby devices.

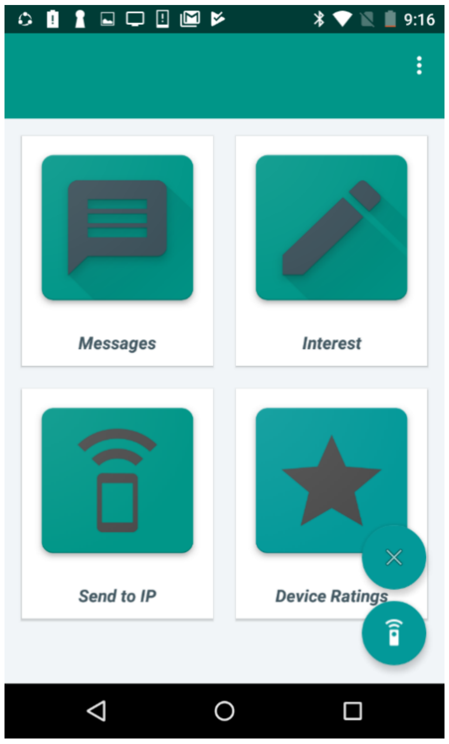
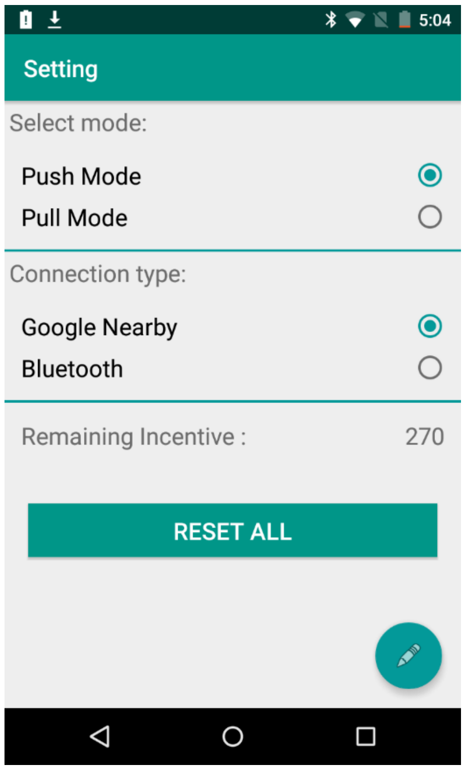


Figure 4.4. Home Screen of the DTN App.

Figure 4.5 displays the app's setting screen. The users can set their preferences to only receive messages from a particular location, and set of tags using the "Pull" mode, and "Push" mode to receive all messages based on the ChitChat algorithm. The "edit" button on the bottom right corner of the screen in Figure 4.5 allows the user to set location, a radius around a given location, and preferred interests for setting up the "Pull" mode. The user can choose the type of connection for peer-to-peer data transfer, either Google Nearby Connections API or the Bluetooth connection exclusively. Each user is given a credit of 300 when the app is first installed, thus allowing them to participate in the DTN by spending some credits to receive the messages. The app can be reset to default by clicking on the "Reset All" button.



Select mode:

[✓ Push Mode]
Pull Mode

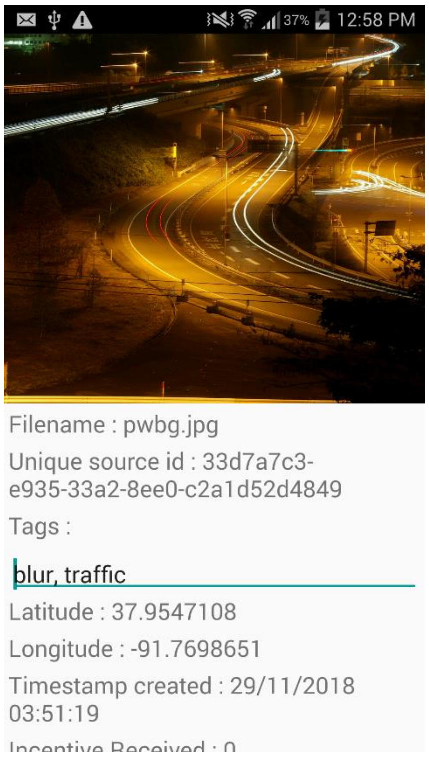
Connection type:

[✓ Google Nearby]
Bluetooth

Remaining Incentive : 270

Figure 4.5. Setting Screen.

Figure 4.6 shows the message creation, where users can select an image from the gallery or capture an image through the device's camera, and Google Cloud Vision API generates a list of tags describing the image. From these tags, the top two tags are set as keywords. However, users can still edit or add more tags to the list. Each message is associated with a list of metadata such as GPS coordinates of where the image was taken; timestamp of the image; the incentive paid, received, and promised for that message; the unique ID of that message; and the filename of the image.



Filename : pwbg.jpg
Unique source id : 33d7a7c3-e935-33a2-8ee0-c2a1d52d4849
Tags :
blur, traffic
Latitude : 37.9547108
Longitude : -91.7698651
Timestamp created : 29/11/2018 03:51:19
Incentive Received : 0

Figure 4.6. Message Creation Screen.

When two devices with our app are in communication range, they get connected, and the connected device's name is displayed to the user. The ChitChat and Incentive algorithm runs in the background to determine which messages and how many messages to send to the connected device. Once the transfer is complete, the devices get disconnected and search for other devices. The message format remains the same for each message across the network as shown in Figure 4.7. The “Message” part contains the multimedia message along with its unique Msg ID, keywords describing it, and the format type of the message. “Meta-information” contains data such as latitude, longitude, and timestamp. Finally, the “Node information” stores the node ID that generated the message, and the Recipients ID stores the list of node IDs through which the message was passed on.

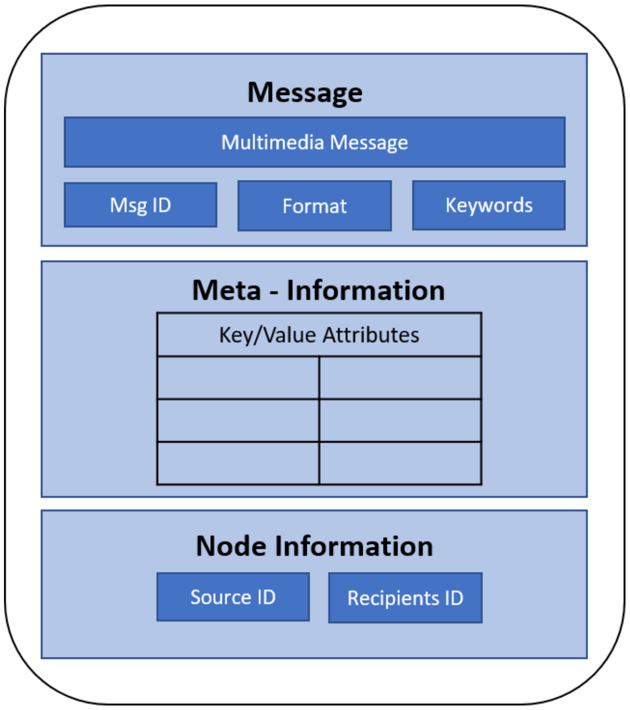
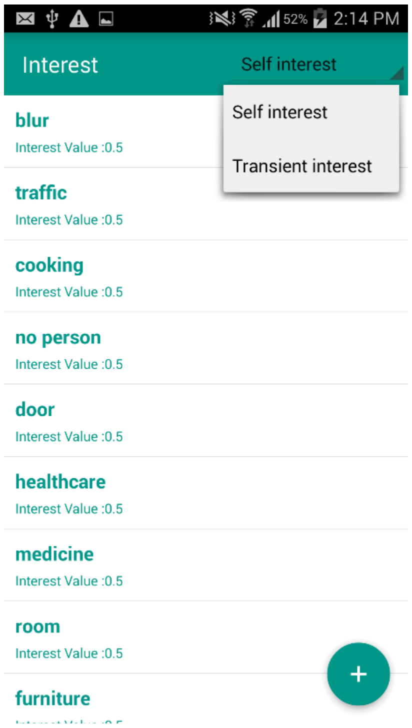


Figure 4.7. Message Format for DTN.

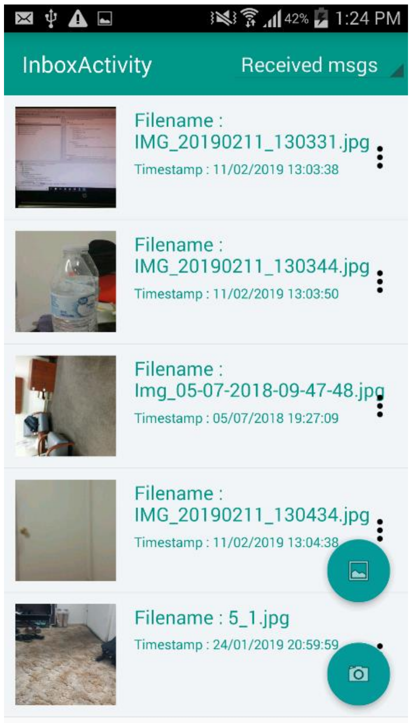
Figure 4.8 shows the “Interests” screen, where the user can view the created interests. Initially, the default value is 0.5 for each created interest and varies as the device participates in the message transfer. The user can also view the accumulated interests by toggling the spinner to select “Transient interest” at the top right end of the screen. The user can delete any interest by just swiping them to the left, and add new interests by clicking on the “plus” button at the end of the screen as shown in Figure 4.8.



blur
Interest Value :0.5
traffic
Interest Value :0.5
cooking
Interest Value :0.5
no person
Interest Value :0.5
door
Interest Value :0.5
healthcare
Interest Value :0.5
medicine
Interest Value :0.5
room
Interest Value :0.5
furniture

Figure 4.8. Interests Screen.

Figure 4.9 displays the list of received messages for a device. On clicking each image, a user can view the metadata associated with the message, including the amount of incentive paid for that message. The user can rate the intermediate nodes for that message, which will later be exchanged with other devices. This, in turn, affects the reputation and incentives given to the intermediary nodes. Thus, all the intermediary nodes are encouraged to enrich the content, and not to add irrelevant tags. Clicking the three dots at the right end of each message allows the user to rate the intermediary nodes, open Google Maps to view the location where the received image was taken, or delete the message. The two rounded buttons at the bottom right end allow the user to create a message either by selecting an image from the gallery or opening the camera to capture a new image.



Filename : IMG_20190211_130331.jpg
Timestamp : 11/02/2019 13:03:38

Filename : IMG_20190211_130344.jpg
Timestamp : 11/02/2019 13:03:50

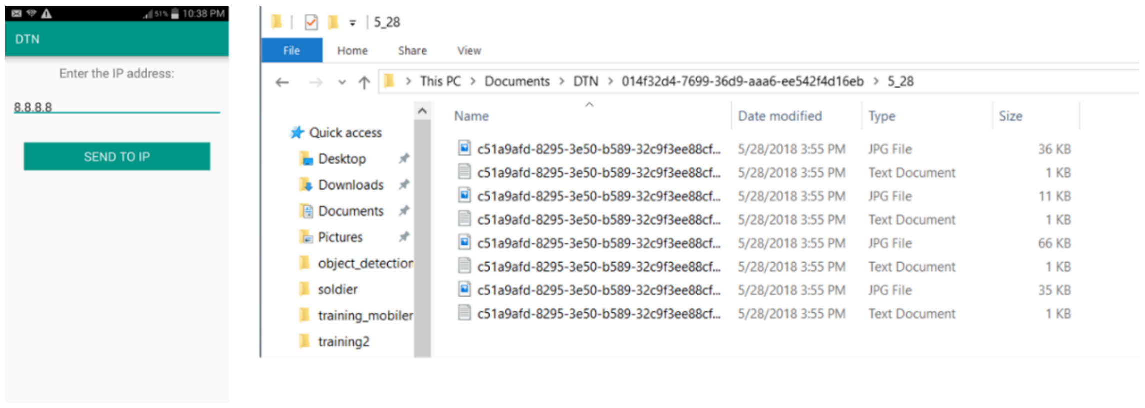
Filename : Img_05-07-2018-09-47-48.jpg
Timestamp : 05/07/2018 19:27:09

Filename : IMG_20190211_130434.jpg
Timestamp : 11/02/2019 13:04:38

Filename : 5_1.jpg
Timestamp : 24/01/2019 20:59:59

Figure 4.9. Received Messages.

Apart from these, there is also a feature to send all the messages along with their metadata and intermediary nodes information to the base station/server using an IP address with designated port. The left image in Figure 4.10 displays the screen where the user can enter the IP address and transfer all the data by clicking on the “Send To IP” button. For this feature to work, the device must be connected to the internet. The image on the right in Figure 4.10 displays the received messages stored in the server (basic java socket programming is used for the implementation). The received messages are stored in the file path in the pattern of /DTN/Device_ID/Month_Data/Unique_Message_ID. For each message, two files are created on the server. One is the actual image and another one is a text file (.txt) that contains the metadata for that message.



Enter the IP address:

8.8.8.8

SEND TO IP

This PC > Documents > DTN > 014f32d4-7699-36d9-aaa6-ee542f4d16eb > 5_28

Name	Date modified	Type	Size
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	JPG File	36 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	Text Document	1 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	JPG File	11 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	Text Document	1 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	JPG File	66 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	Text Document	1 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	JPG File	35 KB
c51a9afd-8295-3e50-b589-32c9f3ee88cf...	5/28/2018 3:55 PM	Text Document	1 KB

Figure 4.10. Send to IP Screen.

4.2. STUDY 2: SECURE FORWARDING BY FRAGMENTATION IN DELAY-TOLERANT NETWORKS

4.2.1. Background. Shudip et al. [2] proposed a scheme that can be used in volatile situations like disaster/battlefield zones where multiple groups under different authorities are operating and need to share information securely. There may not be any central network infrastructure in such volatile situations, and as an alternative, opportunistic networks like DTN can be used by making smartphones act as nodes. To avoid malicious intent, messages need to be secured within and across different groups.

Traditional public-private key cryptography will not work in this situation due to the non-availability of the infrastructure. Encrypting data with a symmetric key and then fragmenting both the key and data before disseminating through the DTN helps to significantly reduce the probability of a message being compromised by some unauthorized node. However, in this fragmentation process, few redundant fragments are created to increase the probability of a message being delivered to the destination node in such an environment. Thus, for each message, *x* number of key-shares are generated with hidden information related to the decryption of the original message. Out of *x* key-shares, we need at least *y* key-shares to obtain the hidden information where $y < x$. Similarly, the encrypted message is fragmented into *n* data-shares that contain some parity blocks to improve the delivery rate of the message. Out of *n* data-shares, at least *k* data-shares are needed to obtain the original message where $k<n$. In order to verify that each key- and data-share is not corrupted before decryption at the destination, an integrity check is also set up which helps in conserving the device's battery. Each device also has a unique ID that it receives from the cloud server by connecting to the internet before it can perform any other operations.

The four important modules are:

- Key handler
- Data handler
- Key Integrity Checker
- Data Integrity Checker

An overview of all these modules is presented in the following sections:

4.2.1.1. Key handler. This module generates a public-private key using Elliptical Cryptography (EC) when the application is run for the first time on the device and it remains the same for consecutive runs. The public-private key is of size 1024 bits and 160 bits respectively. Next, whenever a device creates a new message, it creates a random symmetric key of size 256 bits using Advanced Encryption Standard (AES) which will be used to encrypt the message. Using the Recursive Secret Sharing (RSS) [10] algorithm over Shamir’s scheme we generate 8 key-shares with the AES key, the public key of the source node (SN), and any other information related to the message as secrets. At the destination node (DN), we need at least 4 key-shares to retrieve the secrets.

The number of key-shares for each message is constant throughout the network and each key share is 32 bytes. The SN generates a number of proxy re-encryption keys using its own private key and the public key of the intermediary node. Each of the key-shares is encrypted with one of these proxy re-encryption keys. Thus, INs on receiving key-share proxy re-encrypted using its public key can act a source to that key-share. Instead of decrypting, the IN re-encrypts the key-share with proxy re-encryption keys generated using its own private key and the public key of the DN (if available) or another available IN. Only the DN decrypts the key-shares once it obtains a minimum of 4 key-shares and retrieves message-related secrets. An overview of the algorithm to send message from one device to another is shown in Figure 4.11.

4.2.1.2. Data handler. When a message is created, it is encrypted using a random symmetric key generated using AES by executing the key handler module. The message is split into *n* data-shares, and the data shares are encoded using Cauchy Reed Solomon Erasure Code (CRS) [11]. Out of *n* data-shares, we need at least *k* data-shares at DN to retrieve the original message where $k<n$. When an IN receives a data-share it just forwards it to the next IN or DN. When a DN receives a data fragment, it first checks whether it has obtained a minimum number of key-shares to retrieves the secrets. Then it checks whether it has received a minimum number of data fragments. If both requirements are satisfied, the AES symmetric key (secret) is obtained from the key-shares, and then CRS decoding

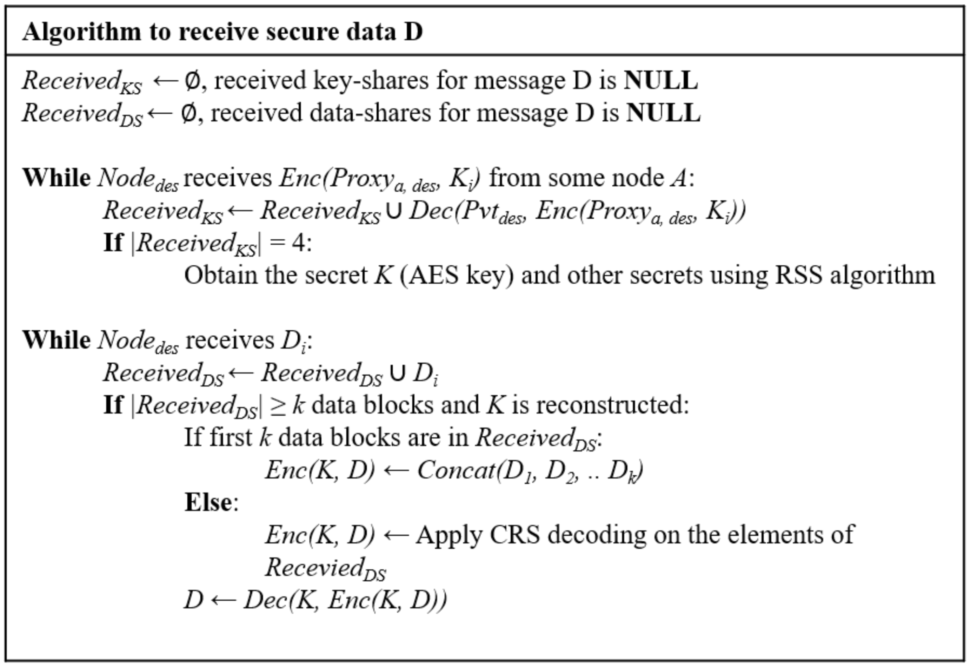
is performed on the data-shares. Using the AES symmetric key obtained from the key-shares, the encrypted message is decrypted to obtain the original message.

```
Algorithm to send Secure data D

$PubOthers_A$ ← Set of public keys of other nodes, Node A has obtained
$K$ ← AES symmetric key used for encrypting the message
$KS$ ← Generate n key shares using K, $Pub_{src}$ as hidden information using RSS

Procedure Send_Key_Share(Node a, Node b, Boolean flag, Key-Share $K_i$):
    $Proxy_{a,b}$ ← Proxy re-encryption key generated by $Pvt_a$ and $Pub_b$
    $Enc(Proxy_{a,b}, K_i)$ ← Encrypt key share KS with $Proxy_{a,b}$
    If(flag == true):
        Intermediate destination is Node b
    Else:
        Intermediate destination is NULL

For each key-share $K_i$ from KS:
    If $Pub_{des} \in PubOthers_{src}$:
        Send_Key_Share(src, des, false, $K_i$)
    Else:
        $Node_j$ ← Random node j from whose public key is available in
        $PubOthers_{src}$
        Send_Key_Share(src, j, true, $K_i$)

If $Node_A$ gets Enc($Pub_A$, $K_i$) and A is the intermediate destination of $K_i$:
    If $K_i$'s destination node's $pub_{des} \in PubOthers_A$:
        Send_Key_Share(a, des, false, $K_i$)
    Else:
        $Node_j$ ← Random node j from whose public key is available in
        $PubOthers_{src}$
        Send_Key_Share(a, j, true, $K_i$)

$DF$ ← Fragment $Enc(K, D)$ to $k$ data blocks
$DF$ ← $DF \cup (n - k)$ encoded blocks using CRS algorithm

For each data block $D_i$ in $DF$:
    $Node_j$ ← Random node $j$ from whose public key is available in $PubOthers_{src}$
    Send $D_i$ via $Node_j$
```

Figure 4.11. Algorithm to Send Data D.

The number of fragments *n* for a message and a minimum number of fragments *k* to retrieve the original message can be varied based on different scenarios as follows:

- **Static data fragmentation:** In this mode, *n* and *k* are constant across the network. Thus for each message, the number of fragments remain the same. Sometimes it is not useful, as even for a small message more fragments will be created.
- **Dynamic data fragmentation:** In this mode, the parameters *n* and *k* are decided based on the message size, where $k<n$. This is particularly helpful for small messages; however, it incurs extra computational overhead as the value of *n* and *k* needs to be sent as one of the secrets at the source.
- **Priority-based data fragmentation:** Each message has a priority value associated with it. The higher the priority value, the higher the number of fragments that are created for that message. This increases the probability of that message reaching the destination in a short time.



Algorithm to receive secure data D

```
$Received_{KS} \leftarrow \emptyset$, received key-shares for message D is NULL
$Received_{DS} \leftarrow \emptyset$, received data-shares for message D is NULL

While $Node_{des}$ receives $Enc(Proxy_{a,\ des}, K_i)$ from some node $A$:
    $Received_{KS} \leftarrow Received_{KS} \cup Dec(Pvt_{des}, Enc(Proxy_{a,\ des}, K_i))$
    If $|Received_{KS}| = 4$:
        Obtain the secret $K$ (AES key) and other secrets using RSS algorithm

While $Node_{des}$ receives $D_i$:
    $Received_{DS} \leftarrow Received_{DS} \cup D_i$
    If $|Received_{DS}| \geq k$ data blocks and $K$ is reconstructed:
        If first $k$ data blocks are in $Received_{DS}$:
            $Enc(K, D) \leftarrow Concat(D_1, D_2, .. D_k)$
        Else:
            $Enc(K, D) \leftarrow$ Apply CRS decoding on the elements of
            $Recevied_{DS}$
        $D \leftarrow Dec(K, Enc(K, D))$
```

Figure 4.12. Algorithm to Receive Data D at Destination.

There are *n* data fragments for a message and DN needs at least *k* data fragments to retrieve the original message. The first *k* fragments are the original data fragments and *n-k* fragments are additional coded fragments. DN can wait to receive the first *k* fragments to avoid data decoding which takes quadratic time to execute. This helps to conserve the battery. The destination node can detect a faulty key-share or data-share only after performing decrypting or decoding respectively. In order to avoid wasting battery power, the destination node first performs key and data integrity checks to ensure that all the received fragments are not corrupted due to transmission errors or malicious nodes. An overview of the algorithm to receive data D at destination is shown in Figure 4.12.

4.2.1.3. Key integrity check. If a key-share is corrupted, the DN will only know when it tries to retrieve the secrets from the key-shares using RSS. In order to save the computation time and cost, the source creates a Merkle Hash Tree (MHT) using the hash values of each key-share. Then the source appends the relevant MHT node values to each key-share so that the MHT root can be generated from it. Thus, when a DN has a minimum number of key-shares and tries to retrieve the secrets, it first generates the MHT root for each share and selects those key-shares whose root values are common and rejects the odd ones.

4.2.1.4. Data integrity check. Similarly, if a data-share is corrupted, the DN will know about it only after performing CRS and decryption using the AES key. In order to avoid the computation cost and time, we sign each data-share at the source with the source's private key. At DN, when it has a minimum number of data- and key-shares, before performing CRS, the signature of each data-share is verified using the public key of the source which is obtained as a secret from the key-shares.

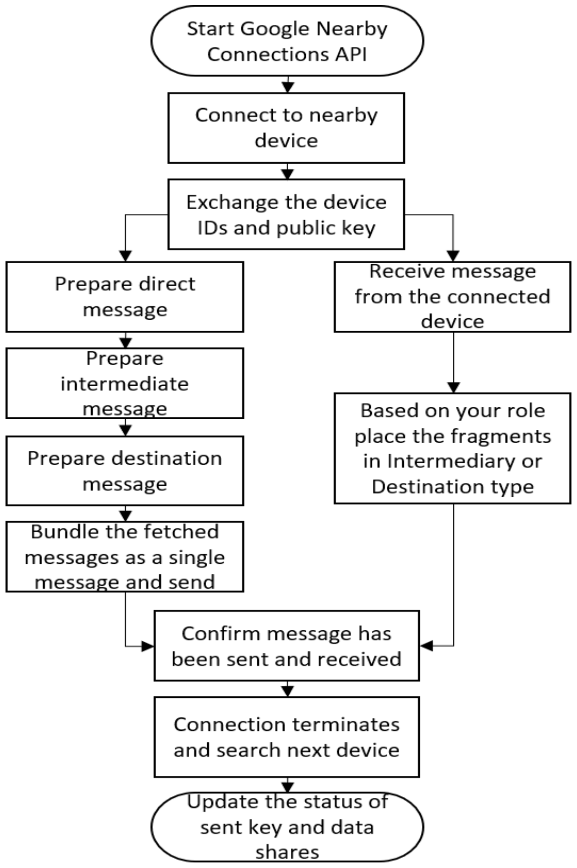

Start Google Nearby Connections API
Connect to nearby device
Exchange the device IDs and public key
Prepare direct message
Prepare intermediate message
Prepare destination message
Bundle the fetched messages as a single message and send
Receive message from the connected device
Based on your role place the fragments in Intermediary or Destination type
Confirm message has been sent and received
Connection terminates and search next device
Update the status of sent key and data shares

Figure 4.13. Secure Data Flow between Connected Devices.

4.2.1.5. Data flow between connected devices. Whenever two devices connect to each other, each device's unique ID and the public key are first exchanged. Message ID is a unique ID for each message, its format is *DeviceID_ithmessage_generated*. For example, if a device with ID 2 generates the 3rd message, its message ID will be *2_3*. Each message's data and key-share have a unique ID associated with them. The two devices exchange 3 types of messages.

- The **direct message** is sent only when the source creates a message and it meets the destination directly. In such a case, the original message prior to fragmentation is set as the direct message.

- From storage, the source device collects a key and data-share of all messages for which the connected device acts as an IN, i.e. for all the message for which the connected device is not the destination and no key- or data-share for that message has already been sent to this device previously. This collection is bundled together and set as an **Intermediary message**.
- **Destination message** is a collection of data and key-shares of all messages for which the connected device is the destination.

For each key-share, the user needs to create a proxy-key using the stored public key of any of the IN or DN. Only IN with which the proxy-key has been generated can perform proxy re-encryption and acts as a source for that key-share and only DN performs decryption of the key-shares and no IN needs to perform any decryption activity. The overall data flow between the two connected devices is shown in Figure 4.13.

4.2.2. Implementation Details. Secure Forwarding through fragmentation in DTN is implemented over the Android operating system and it makes use of Google Nearby Connections API to communicate between devices. It works on the devices running on Android's KitKat (API 19) to Oreo (API 27) version. For each device, the user must first connect to the internet and obtain a one-time unique device ID from Google’s Firebase Realtime Database. The user can start using the app only after receiving the device ID.

Using elliptical cryptography, the public-private key for the device is initialized and it remains the same for the lifetime of the app on that device. Figure 4.14 shows the home screen of the secure forwarding app. The user can create a new message by clicking on the

"Camera" button and capture an image through the device's camera. Once the image is captured, the user can choose the preferred mode of data fragmentation as specified in [2].

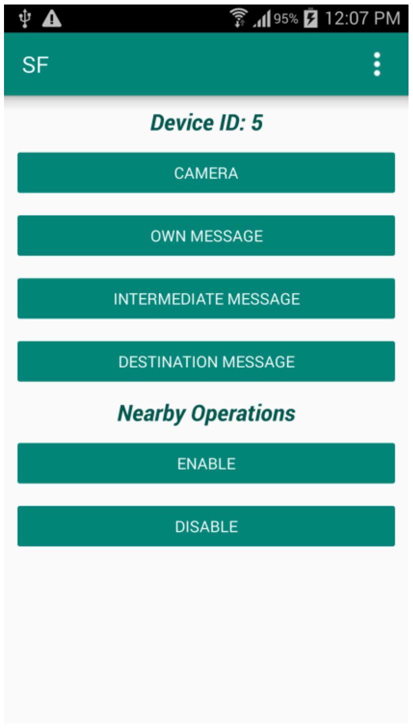
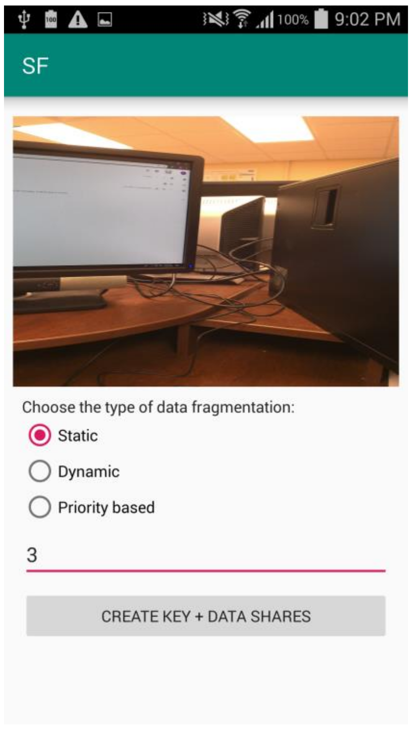


Figure 4.14. Home Screen for Device ID:5.

Figure 4.15 shows message creation with different modes of data fragmentation that the user can choose from. The user can input the destination device ID in the "EditText" field, which is available above the "Create key + data shares" button. Upon clicking the "Create key + data shares" button, the image is encrypted with the random AES key. They AES key along with necessary message-related information such as the *n* and *k* values of the data-shares ($k<n$) and the public key of the source node, is kept as the secret and 8 key-shares are created using RSS. Out of 8 key-shares, DN needs at least 4 key-shares to retrieve the secret. After creating the key-shares, the Key Integrity Module

will be invoked, and that creates an MHT with these key-shares. The Key Integrity Module appends relevant node values (hash values) to each key-share so that the MHT root hash value can be recreated from it. Using the CRS algorithm, data fragmentation is done based on user preferences.



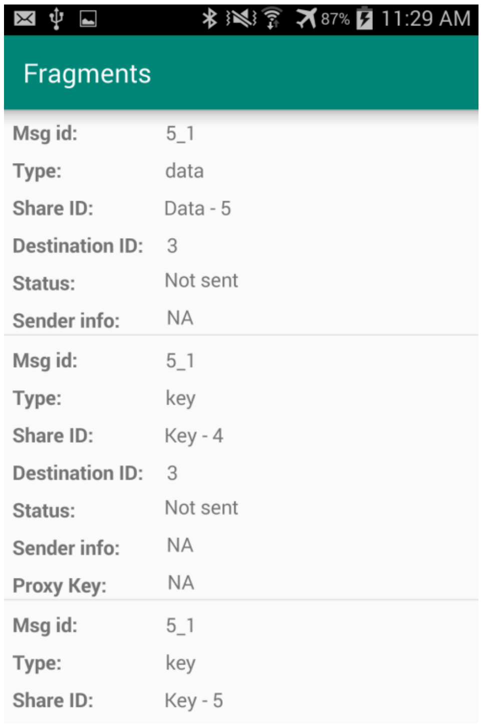
Choose the type of data fragmentation:

[✓ Static]
Dynamic
Priority based

3

Figure 4.15. Message Creation with DN 3.

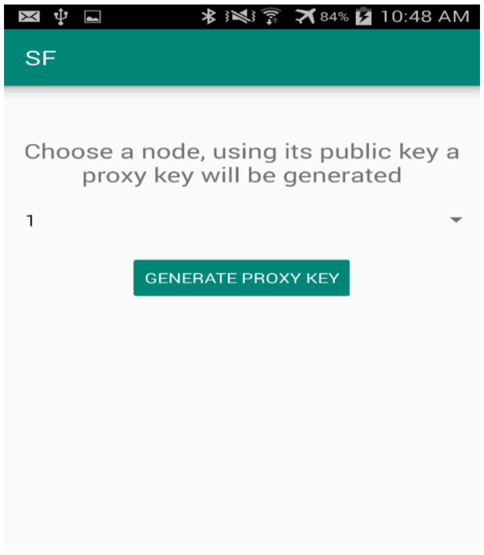
Figure 4.14 shows that when a user clicks on the "Own Message" button, the user can view the created messages and then clicking on a message, the user can view the data- and key-shares for that message as shown in Figure 4.16. For each key-share, the user needs to create a proxy-key using any of the available public keys of IN. It is to be noted that if the public key of DN is available, then all the key-shares are encrypted with the proxy-key that was created using the private key of SN and the public key of DN. Encryption of a key-share using the proxy-key of any available IN is done by clicking a key-share and that allows the user to select any of the devices' IDs from a drop-down list. Upon clicking on the "Generate Proxy key" button after selecting an IN's device ID, a proxy-key is generated with that IN and they key-share is encrypted with that proxy-key.



Msg id:	5_1
Type:	data
Share ID:	Data - 5
Destination ID:	3
Status:	Not sent
Sender info:	NA
Msg id:	5_1
Type:	key
Share ID:	Key - 4
Destination ID:	3
Status:	Not sent
Sender info:	NA
Proxy Key:	NA
Msg id:	5_1
Type:	key
Share ID:	Key - 5

Figure 4.16. Data- and Key-shares of a Message.

Each data- and key-share have certain attributes associated with them as shown in Table 4.1. Figure 4.17 shows the screen in which the user can select an IN to create a proxy-key and encrypt the key-share. If the device does not have any public key of other devices, it just sends a key-share to an IN without encrypting it.



Choose a node, using its public key a proxy key will be generated

Figure 4.17. Proxy-key Generation and Encryption for a Key-share.

Table 4.1. Data- and Key-share Attribute Description.

Attribute	Description	Key-share	Data-Share
Msg ID	Msg ID is of the form *DeviceID_i*th*MessageGenerated*. It denotes the message to which the corresponding fragment belongs to.	✓	✓
Type	Denotes whether a fragment is a key-share or a data-share.	✓	✓
Share ID	Key-shares have an ID in the range [4, 11] and data-shares have ID starting from 1.	✓	✓
Destination ID	Denotes the DN's ID for which the message was created.	✓	✓

Table 4.1. Data- and Key-share Attribute Description (Cont.).

Status	Denotes whether the fragment has been transferred to any IN or not.	✓	✓
Sender Info	It hold the ID of the device to which the fragment has been transferred, else it has the value of NA(Not Applicable)	✓	✓
Proxy-key	Displays the device ID using which proxy-key was created to encrypt the key-share	✓	✗

In Figure 4.14, upon clicking the “Enable” button, the application searches for nearby Android devices. Once they are connected, the public key and device ID are exchanged. Based on the connected device ID, the application checks whether the user has created any message for that device. If it finds such messages, it directly sends the original message to the connected device. For all data- and key-shares for which the connected device is not the DN, the application will first check each message whether any of the key- or data-share have already been previously shared with that device. If previously not shared with the connected device, it selects one of the key- and data-share of that message to send. Lastly, the device will send all the data- and key-shares of all messages for which the connected device is the DN.

As shown in Figure 4.18, all these messages are bundled together and sent to the connected device in a single transmission. The device also receives messages from the connected device. The data- and key-shares are stored appropriately in the storage based on the role of the device for that fragment (i.e. either IN or DN). For each data-share received for which the device acts as the DN, it checks whether 4 or more key-shares are present and whether the secret has been retrieved. If not, it waits for the minimum number of key-shares for that message before proceeding further. If 4 or more key-shares are received for a message, then the key-shares are decrypted. The integrity of each key-share is verified by generating the MHT root hash value. The common MHT root is taken as the valid root hash value and the remaining odd values are discarded as the corrupted ones. If 4 or more key-shares are present even after discarding these values, then the device executes the RSS algorithm over the key-shares to obtain the secret. The secret mandatorily includes the AES key that is used to encrypt the image and the SN's public key. This secret is stored in the internal memory of the DN until the original message is retrieved.

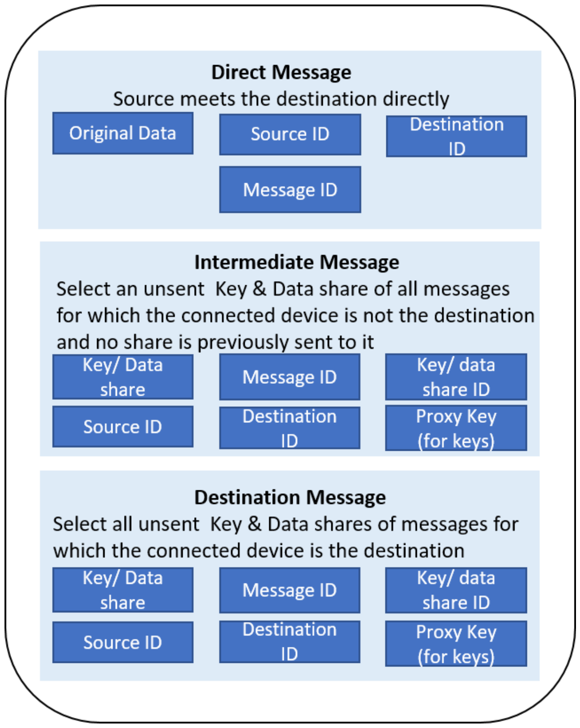
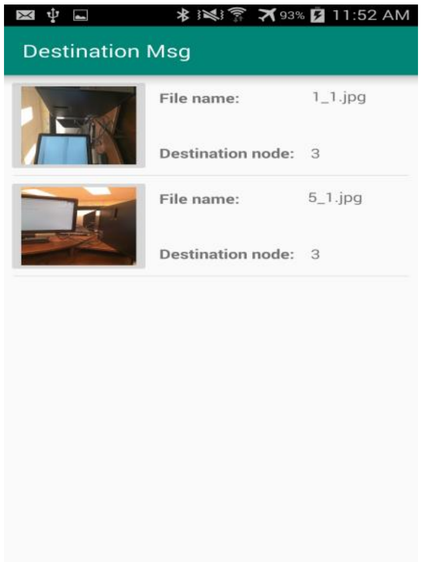

Direct Message
Source meets the destination directly
Original Data
Source ID
Destination ID
Message ID
Intermediate Message
Select an unsent Key & Data share of all messages for which the connected device is not the destination and no share is previously sent to it
Key/ Data share
Message ID
Key/ data share ID
Source ID
Destination ID
Proxy Key (for keys)
Destination Message
Select all unsent Key & Data shares of messages for which the connected device is the destination
Key/ Data share
Message ID
Key/ data share ID
Source ID
Destination ID
Proxy Key (for keys)

Figure 4.18. Bundled Message Format.

Next, the device checks whether *k* (the minimum number of data-shares required at the DN to retrieve the original message) or more data-shares are obtained for that message. If the answer is yes, then the device checks the integrity of each data-share by verifying whether it was signed by the SN using the SN's public key that was obtained as the secret. The corrupted data-shares are discarded. Even after discarding, if there are *k* or more data-shares, CRS algorithm is executed to merge the data-shares into a single encrypted image. Now the image is decrypted using the AES key obtained as the secret. Once the final message is obtained, it is stored in the internal memory so that the user can view it later along with its associated data- and key-shares.



File name: 1_1.jpg
Destination node: 3
File name: 5_1.jpg
Destination node: 3

Figure 4.19. Message 5_1 Retrieved at Device:3.

Figure 4.19. shows the messages received by Device 3 for which it is the DN. In the same figure, message 5_1 can be seen, which was created by device 5 with device 3 as the DN as shown in Figure 4.15. Figure 4.18 can be viewed by clicking on the "Destination Message" button present in Figure 4.14.

The study [2] has been simulated using "The One" simulator in Java programming language. In the earlier simulation phases, the testing environment included multiple virtual nodes on a single Java program. An elliptic curve used to generate the public and private keys is randomly generated and one instance of an elliptic curve is shared across all the nodes. However, due to the lack of synchrony across multiple Android devices, the random instances of elliptic curves are no longer identical. Thus, the app is bootstrapped with a predefined elliptic curve using curve parameters and elements that are written into a file (bundled with the app) to generate public and private keys and perform cryptographic operations from the same elliptic curve.

4.2.3. Performance Analysis. The Android studio provides a tool called "Profiler" which helps to keep track of the performance of the app. Figure 4.20 shows the performs of the app when computationally intensive tasks like encryption/decryption are performed. The encryption and decryption process involves Key Handler/Key Integrity Checker module and Data Handler/Data Integrity Checker module. Other operations such as peer-to-peer message transfer using Nearby Communications API are not tracked as the API takes care of the underlying connections and data management and hence cannot be modified. From this tool two major computationally intensive tasks encryption/decryption are analyzed for a single message. During such tasks, the app consumes almost 40% of the CPU utilization. The average memory used by the app is 60MB during such tasks. The tool describes energy consumption in three levels; light, medium and heavy. The energy consumption of the app varies from light to medium utilization. The analysis has been done on Pixel 2 virtual device running on Android Pie 9.0 (API 28).

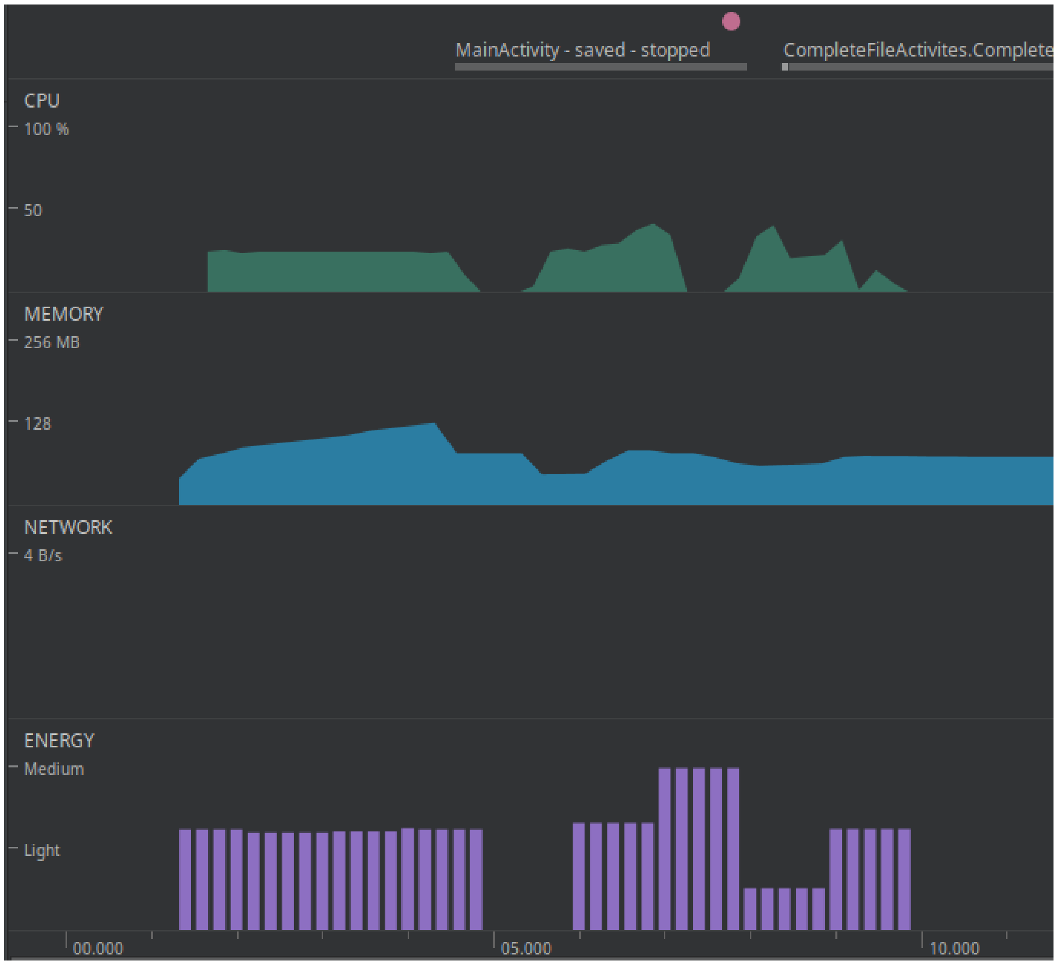


Figure 4.20. Performance Analysis of Secure Forwarding DTN App.

4.3. STUDY 3: INTEREST-BASED AUTHENTICATION USING CIPHERTEXT-POLICY ATTRIBUTE-BASED ENCRYPTION IN DELAY-TOLERANT NETWORKS

4.3.1. Background. In a disaster/battlefield environment, different parties may come along and work together and it is necessary that each information is sent to the concerned parties in a secure way. Each party is associated with a set of attributes or interests e.g. Airforce party may be described by [‘Fighter planes’, ‘Jet’]. In such a scenario, interest-based authentication becomes really useful to provide access to data in a way similar to traditional role-based access control. The messages are encrypted with policy and the destination node can decrypt only if the policy is satisfied. A policy contains a set of attributes (or interests) and a condition, stating the minimum number of attributes to be present in the private key (generated from a set of attributes) to decrypt the message.

It is also useful to send an image message rather than a text message. A single image helps to convey the situation more clearly than many text lines. For this reason, while implementing DTN on Android, images along with associated metadata are considered as a message. However, due to battery and memory constraints, it is important to keep the messages as short as possible in the DTN environment. For example, images of a broken building or enemy soldiers need to be sent to the relevant interested parties. In such a case, it is a waste of memory and energy consumption to send an image with background information. In addition, it is also invaluable to provide a short description of the interesting object to the user. Thus, deep learning models are used to detect interesting objects. In recent times, performing object detection with Android devices has improved a lot, which provides low latency, low memory constraints, low battery consumption, and improved accuracy. Some of the best object detection algorithms are Faster-R-CNN (Regional Neural Network), YOLO (You Look Only Once) and SSD (Single Shot Detector). Out of these, we have used SSDLite MobileNet V2 for object detection on Android devices. One of the primary reasons for using this model is that it has few parameters than other object detection algorithms and the FPS (Frames per second processed) is relatively higher, which makes it suitable for real-time object detection [18].

The overall architecture of the system is shown in Figure 4.21. The important components of [3] in DTN are as follows:

- Master key generator and Edge Servers
- Delay-tolerant network nodes
- Trusted path measurement
- Object detection

4.3.1.1. Master key generator and Edge servers. At the start of the system, all the attributes that are associated with each edge server are collected. Using ABE, these attributes are used to generate public (7128 bits) and master (156 bits) key. The master key is used for the generation of the private (5464 bits) key.

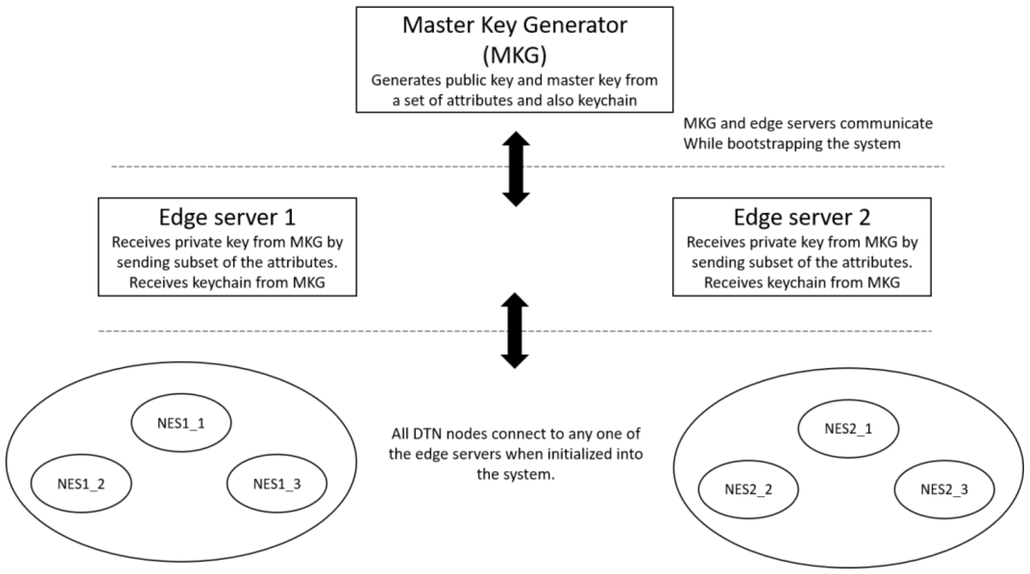

Master Key Generator (MKG)
Generates public key and master key from a set of attributes and also keychain
MKG and edge servers communicate While bootstrapping the system
Edge server 1
Receives private key from MKG by sending subset of the attributes. Receives keychain from MKG
Edge server 2
Receives private key from MKG by sending subset of the attributes. Receives keychain from MKG
NES1_1
NES1_2
NES1_3
All DTN nodes connect to any one of the edge servers when initialized into the system.
NES2_1
NES2_2
NES2_3

Figure 4.21. Architecture of ABE in DTN.

The edge servers are in the second layer of the architecture. When they are first initialized, they send a subset of attributes to the MKG and receive the private and public key from MKG. Edge servers distribute these keys to the DTN nodes when they are first initialized into the system.

4.3.1.2. Delay-tolerant network nodes. The DTN nodes are responsible for message encryption/decryption and message transmission from source to the destination. When a node is first initialized, it connects to the relevant edge server based on its interest and receives the public/private key. For example, an army edge server may have a private key generated from the attributes [‘army’, ‘soldier’]. A node connecting with this edge server will receive the private key from these attributes and will be able to decrypt messages encrypted with these attributes via a policy. A message may be encrypted with a policy such as [‘army fire 1of2’] which implies that the destination node should have a private key generated from a set of attributes containing either ‘army’ or ‘fire’ or both.

4.3.1.3. Trusted path measurement. In order to find the corrupted nodes in the path of message traversal from source to destination, the keychain-based approach is used. A keychain is a one-way backward chain of the hash outputs of a predefined hash function (SHA-256) where the first key k_n is created randomly. Then the next key k_{n-1} is created by the hashing of k_n and we keep hashing until we get k_0. The MKG creates a set of keychains for all the edge servers. When an edge server connects to the MKG, it receives keychains for its own and also receives the keychains of other edge servers. Instead of sending all the hash values, MKG sends only k_n and k_0 for each keychain. Similarly, when a node gets connected to the edge server it receives all the keychains from the edge server.

In Figure 4.22, the source node selects one of the keychains and obtains k_4 (i.e. k_n) from its storage before transferring a message created by it. Next, it creates a keyed hash using SHA-256 from k_4 and the IDs of source S and intermediate node I_1. It also sends k_3 and IDs of S and I_1 to I_1. When I_1 meets another intermediate node I_2, it creates a keyed hash from k_3 and IDs of I_1 and I_2. This keyed hash along with previous keyed hash values generated for this message is sent to I_2. k_2 along with IDs of I_1 and I_2 is also sent to I_2. The same process continues when I_2 meets I_3 and when I_3 meets the destination node. When the destination node receives k_0, it obtains k_4 from its storage which was previously obtained from the edge server on initialization. From k_4 and the pair of IDs associated with each hash, the destination node recalculates and validates all the hashes associated with the message. By validating, the destination node can identify the malicious nodes. Any node which might have contributed to altering any hash or path information will be kept out of consideration for further distribution of data.

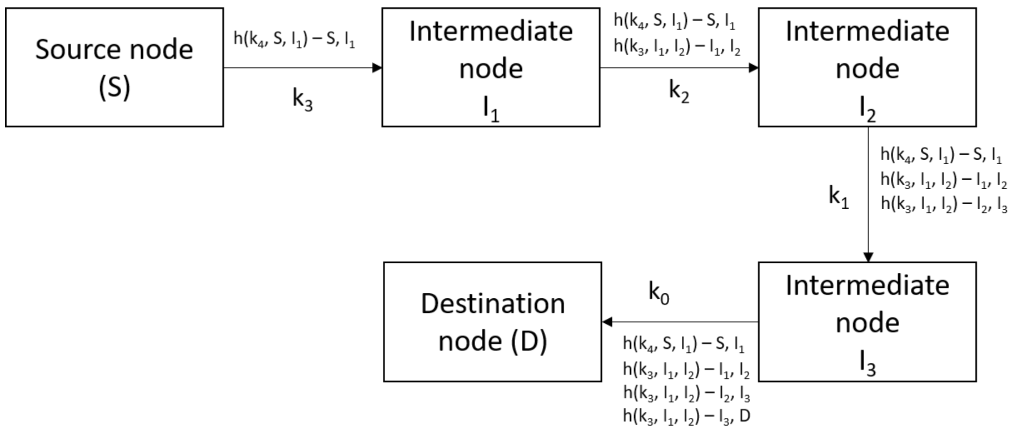

Source node (S)
$h(k_4, S, I_1) – S, I_1$
k_3
Intermediate node I_1
$h(k_4, S, I_1) – S, I_1$
$h(k_3, I_1, I_2) – I_1, I_2$
k_2
Intermediate node I_2
k_1
$h(k_4, S, I_1) – S, I_1$
$h(k_3, I_1, I_2) – I_1, I_2$
$h(k_3, I_1, I_2) – I_2, I_3$
Intermediate node I_3
k_0
Destination node (D)
$h(k_4, S, I_1) – S, I_1$
$h(k_3, I_1, I_2) – I_1, I_2$
$h(k_3, I_1, I_2) – I_2, I_3$
$h(k_3, I_1, I_2) – I_3, D$

Figure 4.22. Hash Chain Created by Source/Relay Nodes for a Message.

4.3.1.4. Object detection. SSDLite MobileNet V2 is an object detection model designed by Google to more efficiently run inference on this model on mobile and embedded devices. The base network of this model is MobileNet V2 which makes use of depthwise separable convolution and efficient handling of linear bottlenecks.

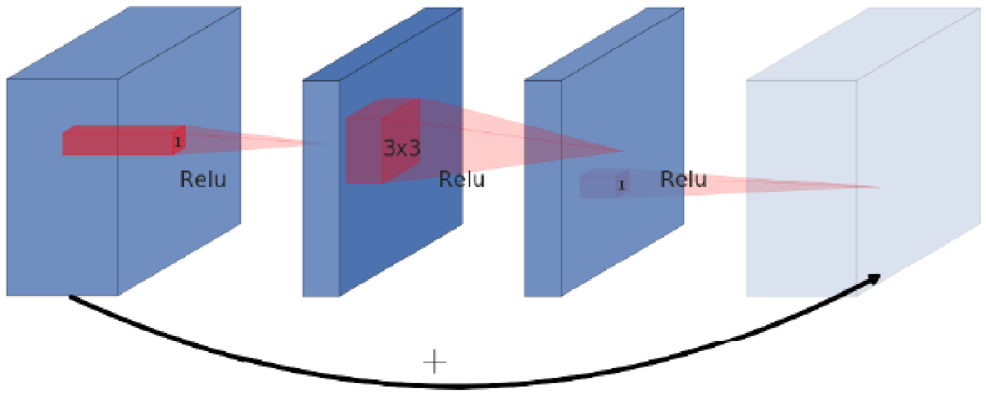


Figure 4.23. Residual Blocks [18].

Depthwise separable convolution helps to minimize the computation by reducing the number of addition and multiplication operations. In depthwise separable convolution, the convolution operation is split into two smaller functions. The first operation is depthwise convolution and it performs convolution to a single input channel at a time. For example, there are 3 channels for an RGB image, and depthwise convolution performs convolution by applying a filter or kernel to a single channel at a time. Thus, the number of kernels should be the same as the number of input channels. Depthwise convolution is succeeded by pointwise convolution. Pointwise convolution performs a linear combination on the output from depthwise convolution. It is a 1x1 convolution operation with N filters. From [18] it is known that depthwise separable convolution has k^2 reduced computation than standard convolution operation, and *k* is the size of the kernel in the depthwise convolution. MobileNet has *k=3*, which in turn reduces the computation cost up to 8 to 9 times compared to the standard convolution operation.

As the number of layers increases, deep neural networks can act as universal approximators, especially in computer vision problems. However, there is an upper bound on the number of layers added which results in improvement of accuracy. The increased number of layers may result in problems such as vanishing gradients and curse of dimensionality. All these problems are overcome by using residual networks in which output from one layer is skipped a few training layers and connected with another layer as shown in Figure 4.23.

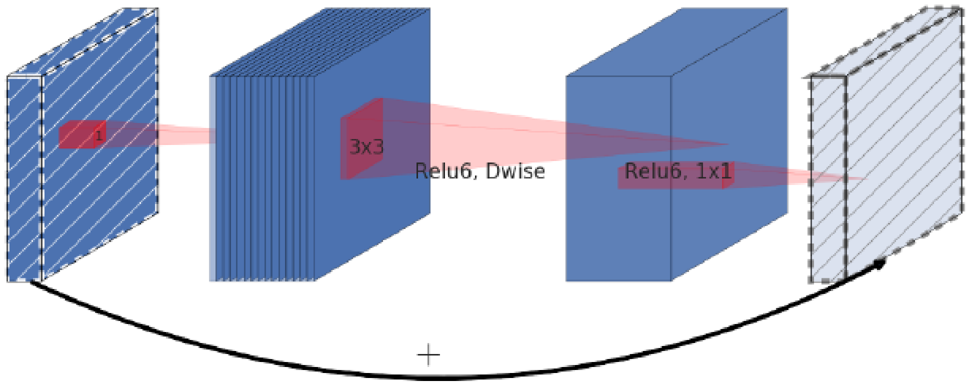
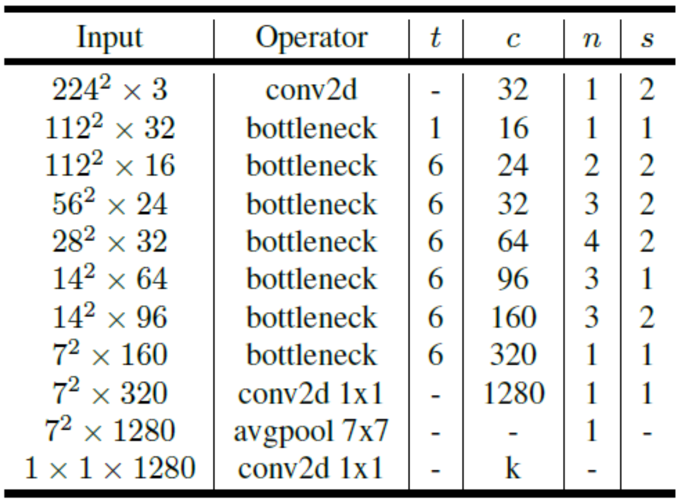


Figure 4.24. Inverted Residual Blocks [18].

In this way, the deeper layers have access to earlier activations that were not modified by the in-between convolutional blocks. From the diagram, it can be seen that the residual blocks are in the pattern of wider-narrow-wider in regards to the number of channels. This is the traditional pattern when residual blocks are used. However, MobileNet V2 uses inverted residual blocks as shown in Figure 4.24. Here the pattern follows a narrow-wider-narrow approach in regards to the number of channels in the input. The author specifies that, when both the patterns are implemented, it is observed that inverted residual blocks have fewer parameters than the traditional approach.

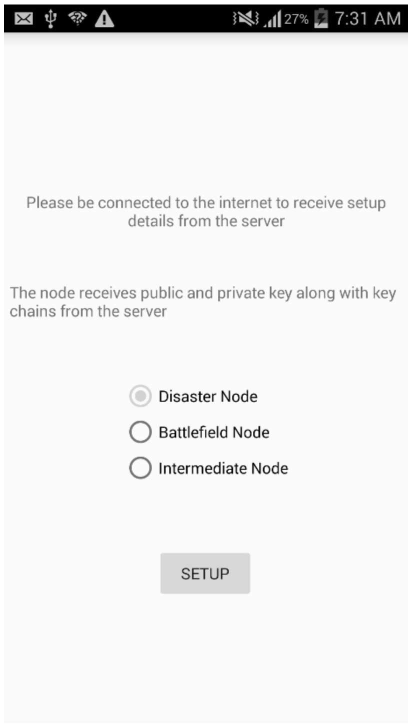
It is important to have non-linear activations in neural networks; it helps to make neural networks act as a good approximator. ReLU(Rectified Linear Unit) is one of the frequently used non-linear activation functions in deep neural networks which discard values smaller than 0. Thus, there is some kind of information loss that occurs in each layer. This is overcome by increasing the number of channels to increase the capacity of the network. However, in the inverted residual model, the layers are squeezed and skip connections are established. Thus the author proposed the idea of linear bottlenecks in which the last convolution of the residual block has a linear output before it is added to the initial activation from the skip connection. MobileNet V2 uses batch normalization after every convolutional block and it uses ReLU6 as the activation function.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure. 4.25. MobileNet V2 Architecture [18].

SSDLite MobileNet V2 contains an SSDLite layer at the top for object detection. SSDLite is a trimmed-down version of SSD. The overall architecture of MobileNet V2 is shown in Figure. 4.25. The first layer of SSDLite is attached to layer 15, and the rest of the SSDLite layer is added on top of the last layer of MobileNet V2, shown in Figure 4.25. The symbol *t* represents the expansion rate of the channel, *c* represents the number of input channels, *n* represents how often the block is repeated and lastly *s* represents the stride used.


Please be connected to the internet to receive setup details from the server

The node receives public and private key along with key chains from the server

[✓ Disaster Node] Battlefield Node Intermediate Node

Figure 4.26. Setup Screen of ABE DTN App.

4.3.2. Implementation Details. MKG server and edge servers are created using Java socket programming. The implementation details are explained based on the demo performed. For the demo, the system is bootstrapped with MKG server and two edge servers. The MKG server runs on port number 5000 and creates a public/master key based on the following attributes [‘army’, ‘soldier’, ‘disaster’, ‘flooding’]. One of the edge servers is Battlefield edge server and it is associated with the attributes [‘army’, ‘soldier’] and it runs on port number 5001, while the other one is Disaster edge server, its attributes are [‘disaster’, ‘flooding’] and runs on port number 5002. All the ABE related operations are implemented using [17].

When the system is first initialized, MKG server is run first and it creates a public/master key. Based on the number of edge servers it creates a set of keychains using SHA-256 for each edge server. Next, the edge servers are initialized. During initialization, they request the MKG server by passing their associated attributes to receive public/private key and also the keychains. Once the edge servers are initialized, DTN nodes can be initialized.

Figure 4.26 shows the setup screen for the Android app. The node can choose the edge servers it wants to connect to or choose an intermediate node option which restricts the node functionality to just receive and forward messages. In order to connect to the edge server, the node must be connected to the internet. On clicking the “Setup” button, the node receives the public/private key, unique device ID and keychain from the edge server which will be stored in the internal storage of the device. Figure 4.27 displays the “Homescreen” of the app. “Homescreen” displays the unique ID of the device and also the set of attributes from which its private key has been generated. Thus, a message encrypted with policy using these attributes can be decrypted by this node if it satisfies the policy conditions too.

In Figure 4.27. on clicking “Soldier detection”, the app displays a single screen where it converts the live video feed from the device’s camera into frames of size 640x480 and feeds the cropped frame of size 300x300 to the object detection model at the rate of 135FPS in Pixel 6 Android device (Virtual device). Once the objects are detected, the relative box coordinates are returned by the model that is used for drawing bounding boxes on the detected objects on the screen. The app also provides an option for the user to save detected objects in storage.

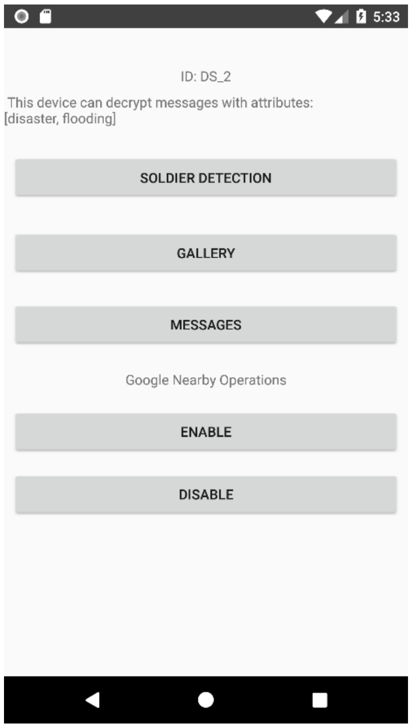


Figure 4.27. Homescreen of DTN ABE App.

On exploring various object detection models, SSDLite MobileNet V2 is the most effective in terms of accuracy, lesser parameters, and lesser computation (i.e. lesser addition/multiplication operations) [18].From Table 4.2. it is clear that another efficient mobile-friendly object detection algorithm YOLO v2, has 20 times more computation and 10 times more parameters than SSDLite MobileNet V2 when trained on COCO dataset.

MobileNet models provide a good tradeoff between resource utilization and accuracy. They are often coined as “mobile-first” architectures. In SSDLite MobileNet V2, the base MobileNet model use depthwise separable convolutions and inverted residual blocks which makes them insanely fast, small, and remarkably accurate, and makes them easy to tune for resource vs. accuracy.

In order to develop an object detection model, the first stage is obtaining the training and testing data. For this purpose, 650 images of soldiers were scraped off from the internet and 250 images of normal civilians are taken from INRIA Person Dataset [19]. The civilian images are added to prevent the model from detecting civilians as soldiers as both have almost the same features. The training data consists of 800 images and the evaluation data consists of 100 images out of the total of 900 images. For all the training/testing data, ground truth box for soldiers and civilians are created using LabelImg utility [20].

Table 4.2. Performance Comparison of SSDLite MobileNet V2 with Other Object Detectors on the COCO Dataset [18].

Network	mAP	Params	Madd	CPU
YOLO V2	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
Mnet V2 + SSDLite	22.1	4.3M	0.8B	200ms

Using TensorFlow Object Detection API, SSDLite MobileNet V2 model was configured and transfer learning was done from training on COCO dataset. Thus, the model already knows the features to detect humans in an image. This reduces training time and improves accuracy.

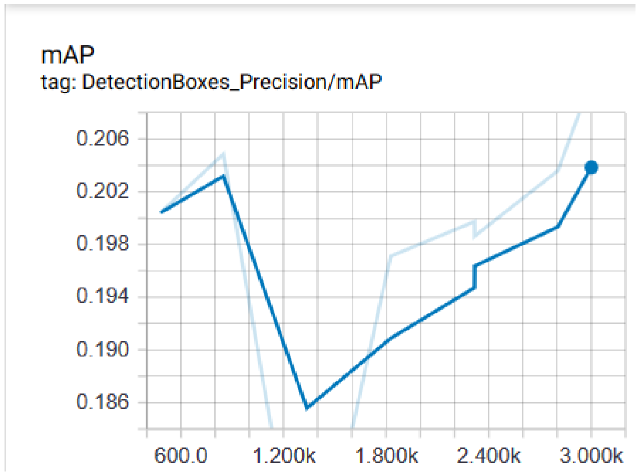


Figure 4.28. Mean Average Precision(mAP) Graph.

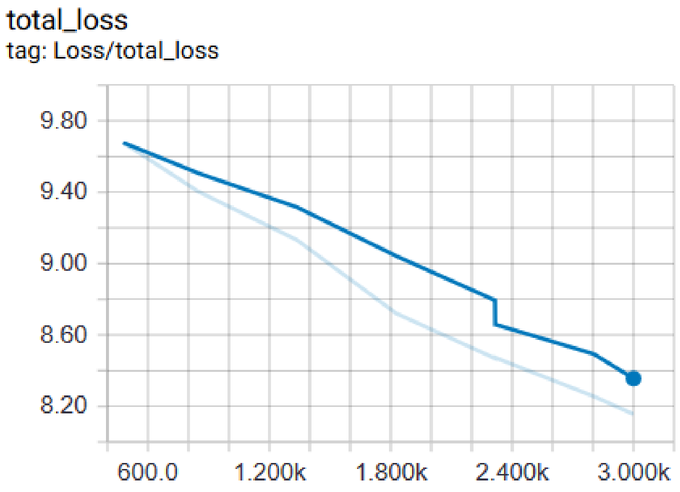


Figure 4.29. Total Loss Graph.

The training was continued until the mean average precision (mAP) was steadily increased and the overall total loss was very much reduced. From the graphs in Figure 4.28. and 4.29 from TensorBoard, we identified any training beyond this worsened the accuracy of the model as it was overfitting with the training data and performed poorly on the evaluation dataset. In Figure 4.28 the horizontal axis denotes the training steps and the vertical axis denotes the mAP values. Similarly, in Figure 4.29 the horizontal axis represents the training steps and the vertical axis denotes the total loss value. Once the model is trained, it is converted to a frozen graph in which information unnecessary for inference are removed, so that the size of the model is very much reduced. The frozen graph has .pb file format and the model's size was about 18 MB.

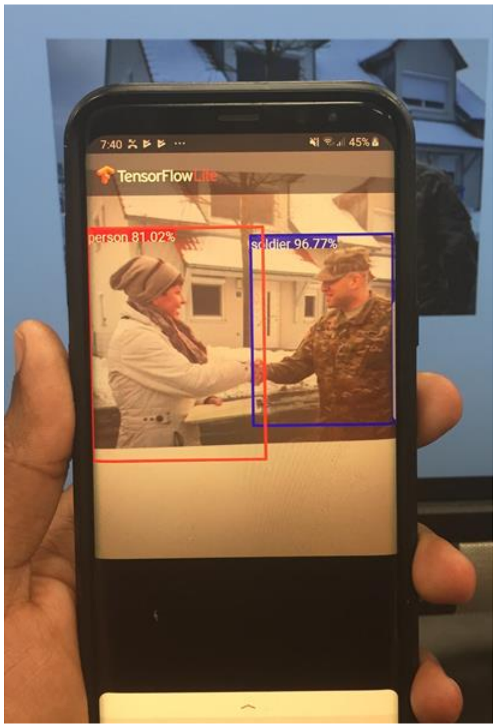


Figure 4.30. Soldier Detection in Android App.

The .pb file format is converted to TFLite format so that the Android device can make use of NNAPI to improve the runtime of inference on this model. Converting models to TFLite format reduces the model size and provide optimizations without reducing the accuracy drastically. An example of such optimizations is to convert the model to a quantized model which reduces the precision of values and operations within the model to reduce the time required for inference. The model size was 17 MB after converting to TFLite format. Figure 4.30 shows the screen where a soldier has been detected and displayed to the user.

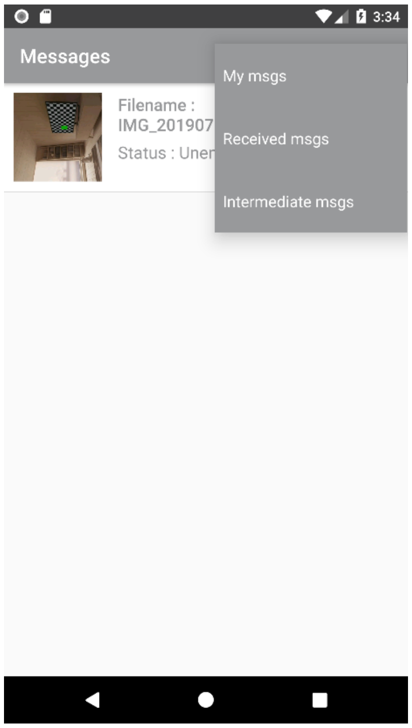
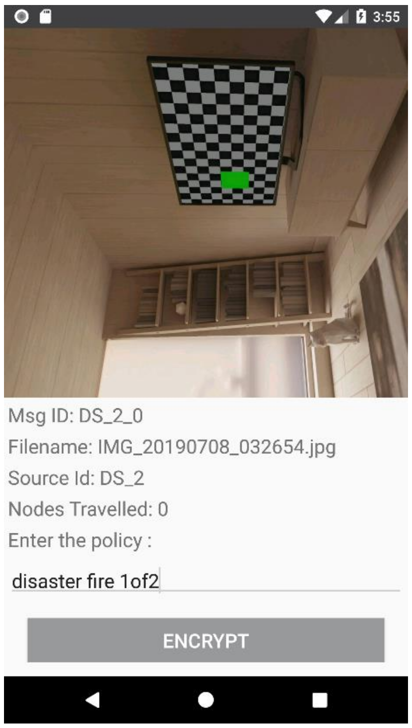


Figure 4.31. Messages Screen.

In Figure 4.27, on clicking the “Gallery” button, the user can select images from the device’s gallery to create a message. All the images selected will be added to the database which can be viewed on clicking “Messages” button. On clicking the “Messages” button in Figure 4.27, the app displays the “Messages” screen as shown in Figure 4.31. The user can view the created messages by selecting “My msgs” spinner in the ActionBar.

The user can view the decrypted messages (i.e. it the destination node for those messages) by selecting "Received msgs". For all those messages which cannot be decrypted by the device will be considered as an intermediate message and will be forwarded to other devices on contact. Intermediate messages can be viewed by selecting "Intermediate msgs" in Figure 4.31. The messages are displayed as a list containing filename and message status (i.e., encrypted or not).



Msg ID: DS_2_0
Filename: IMG_20190708_032654.jpg
Source Id: DS_2
Nodes Travelled: 0
Enter the policy :
disaster fire 1of2

Figure 4.32. Message Details Screen.

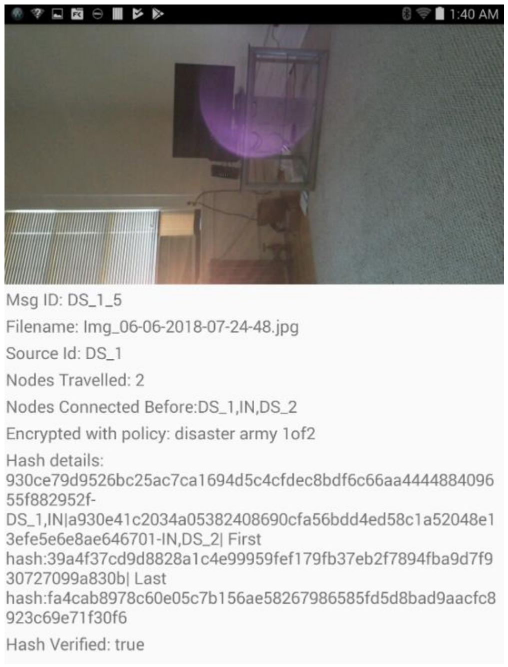
If the message was created by the device, the message will be displayed by selecting "My msgs" in Figure 4.31. On clicking on the message, the metadata associated with the message is displayed as shown in Figure 4.32. The user can enter the policy for the message to be encrypted in EditText. *Policy disaster fire 1of2* implies that the destination node should have a private key generated from *disaster* or *fire* or both. By clicking on “Encrypt” button the message will be encrypted and stored in the database. The metadata associated with each message is shown in Table 4.3.

Table 4.3. Metadata Associated with ABE DTN App Message.

Attribute	Description
Msg_ID	Msg ID is of the form *DeviceID_i*th*MessageGenerated*. It denotes the message to which the corresponding fragment belongs to.
Filename	Contains the image filename
Source_ID	Contains the ID of the device which created the message
Path	Contains the path where the image is stored in the device
Type	Refers to the type of message for that device. Can be own, intermediate or received message for a device
CipherPath	Refers to the path where the encrypted message is stored in the device
Policy	Contains policy used for encryption of the message
HashInfo	Contains the hashes created by each device the message traveled through (for trusted path measurement)
Num_nodes_travelled	Number of intermediate nodes + destination node traveled by the message
Connected_devices	IDs of devices the message traveled through

Table 4.3. Metadata Associated with ABE DTN App Message (Cont.).

Next_hash	If a source/intermediate node created a hash using k_n, it computes k_{n-1} and stores it in this attribute
isVerified	The destination node verifies the hash values created by source and various relay nodes of the message and assigns a boolean value to this attribute.



```
Msg ID: DS_1_5
Filename: Img_06-06-2018-07-24-48.jpg
Source Id: DS_1
Nodes Travelled: 2
Nodes Connected Before:DS_1,IN,DS_2
Encrypted with policy: disaster army 1of2
Hash details:
930ce79d9526bc25ac7ca1694d5c4cfdec8bdf6c66aa4444884096
55f882952f-
DS_1,IN|a930e41c2034a05382408690cfa56bdd4ed58c1a52048e1
3efe5e6e8ae646701-IN,DS_2| First
hash:39a4f37cd9d8828a1c4e99959fef179fb37eb2f7894fba9d7f9
30727099a830b| Last
hash:fa4cab8978c60e05c7b156ae58267986585fd5d8bad9aacfc8
923c69e71f30f6
Hash Verified: true
```

Figure 4.33. Message Metadata.

In Figure 4.27, upon clicking the “Enable” button, the application searches for nearby Android devices. Once they are connected, the device ID is exchanged. Based on the connected device ID, the application pulls all the encrypted messages from storage.

While forwarding each message it creates a hash using the keychain and the IDs of both the connected devices are computed and appended to "HashInfo" attribute. The "Next_hash" attribute is computed and stored. The connected device's ID is appended to the "Connected_devices" attribute and "Num_nodes_travelled" attribute is increased by one. Once all these attributes are updated for a message, it is transferred to the other device including the encrypted message as a stream.

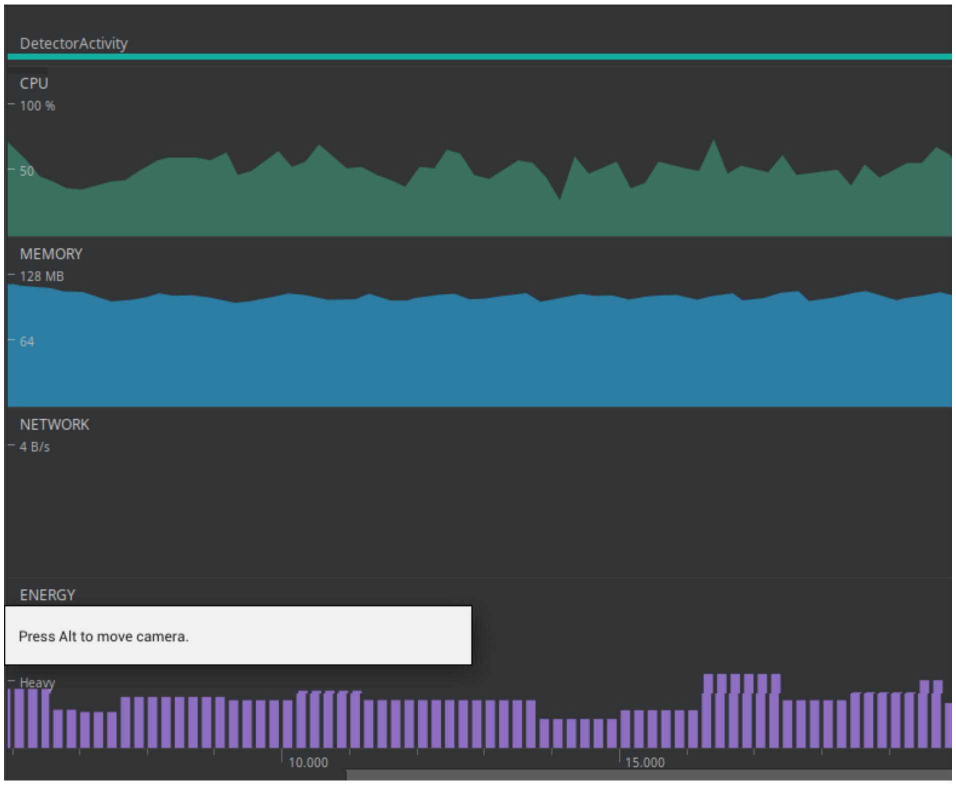


Figure 4.34. Object Detection Performance Analysis.

Similarly, the device also receives encrypted messages from the connected device. If the device is able to decrypt the encrypted message then it is the destination for that message and does not forward to any other device in the future. The retrieved image is stored in the device's storage and encrypted message stream is discarded. The decrypted message can be viewed by clicking on "Received msgs" from Spinner in ActionBar in Figure 4.31. On clicking a received message, the retrieved image and metadata are displayed as shown in Figure 4.33. The "Hash details" display "HashInfo" attribute of a message for the demo purposes. If the device is not able to decrypt an encrypted message received it stores them as an intermediate message and forwards it to other devices in the future.

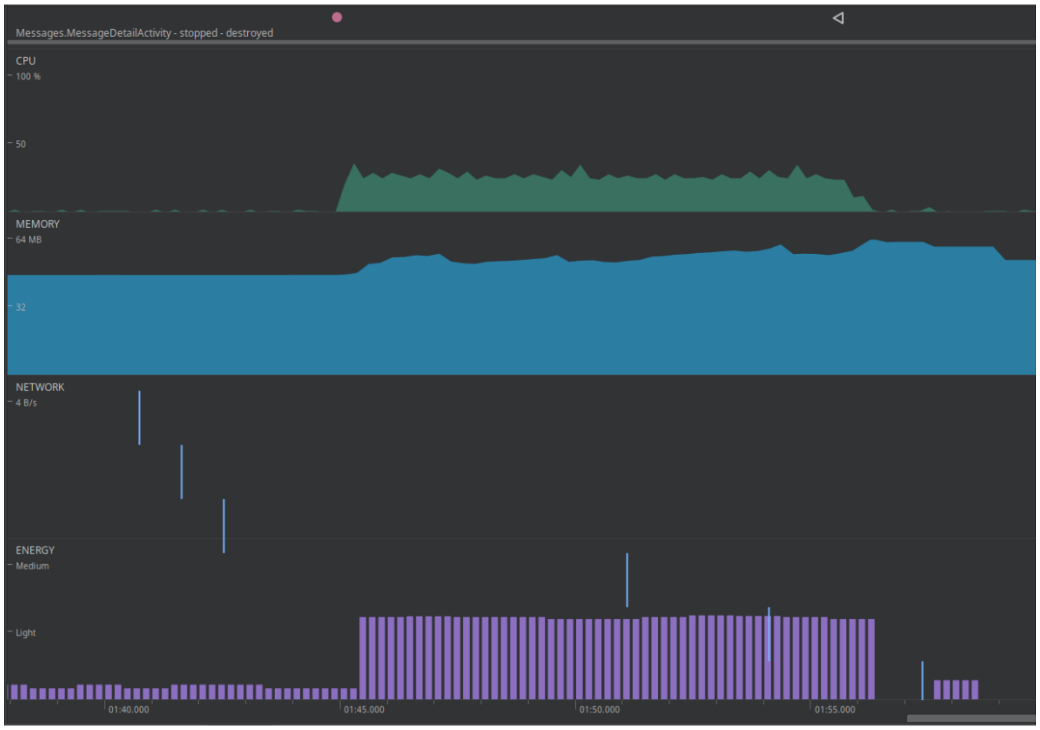


Figure 4.35. ABE Encryption/Decryption Analysis.

4.3.3. Performance Analysis. As mentioned in the implementation details of [2] the tool called "Profiler" in Android Studio IDE is used for performance analysis of the app. The app is analyzed for object detection module and encryption/decryption using [3] for images. Figure 4.34 displays the analysis of the object detection module of the app. The object detection module heavily consumes the device battery. On average it consumes 63% of the CPU utilization. It uses an average of 110MB of memory when its performing object detection on a live feed from the device's camera. Figure 4.35 displays the performance analysis of the encryption/decryption process using ABE.

The analysis has been done for encryption/decryption of a single message. The app consumes almost 45% of the CPU utilization and uses an average of 60MB for internal memory. The energy consumption is of "medium" level while performing those operations. The analysis has been done on Pixel 2 virtual device running on Android Pie 9.0 (API 28).

5. CONCLUSION

Three DTN protocols [1], [2] and [3] are implemented on the Android platform as an app. The apps use various available peer-to-peer networking technologies to communicate with nearby devices. The protocols [1], [2] and [3] in DTN are experimented in real-time scenarios by running tests with a number of Android devices with the respective apps and that shows the usability and effectiveness of these protocols in disaster/battlefield applications. These protocols provide better content management and cybersecurity in DTN environment. Furthermore, we have explored the models for object detection using deep learning and integrating them with the Android platform, and implemented one of the mobile-friendly object detection models SSDLite MobileNet V2 for soldier detection. The object detection helps in routing the appropriate data satisfying the access control policies in DTN environment. The codes for all the 3 projects are available at [21].

BIBLIOGRAPHY

[1] H. Jethawa and S. Madria, “Reputation and credit based incentive mechanism for data-centric message delivery in DTNs,” in *Proceedings - IEEE International Conference on Mobile Data Management*, 2018.

[2] S. Datta, S. Madria, J. Milligan, and M. Linderman, “Secure information forwarding through fragmentation in delay-tolerant networks,” in *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2019.

[3] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *Proceedings - IEEE Symposium on Security and Privacy*, 2007.

[4] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, “The TESLA Broadcast Authentication Protocol,” *RSA CryptoBytes Tech. Newsl.*, 2002.

[5] D. McGeehan, D. Lin, and S. Madria, “ChitChat: An Effective Message Delivery Method in Sparse Pocket-Switched Networks,” in *Proceedings - International Conference on Distributed Computing Systems*, 2016.

[6] B. Bin Chen and M. C. Chan, “MobiCent: A credit-based incentive system for disruption tolerant network,” in *Proceedings - IEEE INFOCOM*, 2010.

[7] M. Y. S. Uddin, B. Godfrey, and T. Abdelzaher, “RELICS: In-network realization of incentives to combat selfishness in DTNs,” *Proc. - Int. Conf. Netw. Protoc. ICNP*, pp. 203–212, 2010.

[8] L. Wei, Z. Cao, and H. Zhu, “MobiGame: A user-centric reputation based incentive protocol for delay/disruption tolerant networks,” in *GLOBECOM - IEEE Global Telecommunications Conference*, 2011.

[9] R. Cabaniss, V. Kumar, and S. Madria, “Multi-party encryption (MPE): secure communications in delay tolerant networks,” *Wirel. Networks*, 2015.

[10] A. Parakh and S. Kak, “Recursive secret sharing for distributed storage and information hiding,” in *2009 IEEE 3rd International Symposium on Advanced Networks and Telecommunication Systems, ANTS 2009*, 2009.

[11] J. S. Plank and L. Xu, “Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications,” in *Proceedings - Fifth IEEE International Symposium on Network Computing and Applications, NCA 2006*, 2006.

[12] R. Yanggratoke, A. Azfar, M. J. P. Marval, and S. Ahmed, “Delay tolerant network on android phones: Implementation issues and performance measurements,” *J. Commun.*, 2011.

[13] A. Ippisch and K. Graffi, “Infrastructure mode based opportunistic networks on android devices,” in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, 2017.

[14] A. Ignatov *et al.*, “AI Benchmark: Running deep neural networks on android smartphones,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11133 LNCS, pp. 288–314, 2019.

[15] A. De Caro and V. Iovino, “jPBC: Java Pairing Based Cryptography,” in *Proceedings - IEEE Symposium on Computers and Communications*, 2011.

[16] J. Huang *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3296–3305, 2017.

[17] J. Wang, “Java Realization for Ciphertext-Policy Attribute-Based Encryption,” 2012.

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[19] N. DALAL, “INRIA Person Dataset.” [Online]. Available: http://pascal.inrialpes.fr/data/human/.

[20] Tzutalin, “LabelImg. Git code (2015),” 2015. [Online]. Available: https://github.com/tzutalin/labelImg.If

[21] K. Sachidanandam, “DTN project code repository.” [Online]. Available: https://github.com/karsac93.

VITA

Karthikeyan Sachidanandam was born in Chennai, India. He received his Bachelor's degree in Information Technology from Anna University, India in 2014. After working in an IT firm for almost 3 years, he became a graduate student in the Computer Science Department at Missouri S&T. He did his thesis research under Dr. Sanjay Kumar Madria from January 2018 to July 2019. He received his master’s in Computer Science from Missouri S&T in December 2019.