NAIST-IS-DD1461012

Doctoral Dissertation

Disaster Response Systems using Distributed Computing across Delay-Tolerant Networks

Edgar Marko Trono

March 12, 2017

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of Engineering

Edgar Marko Trono

Thesis Committee:

Professor Keiichi Yasumoto	(Supervisor)
Professor Shoji Kasahara	(Co-supervisor)
Associate Professor Yutaka Arakawa	(Co-supervisor)
Assistant Professor Hirohiko Suwa	(Co-supervisor)
Assistant Professor Manato Fujimoto	(Co-supervisor)

Disaster Response Systems using Distributed Computing across Delay-Tolerant Networks*

Edgar Marko Trono

Abstract

Disaster response teams perform many tasks during their operations including mapping the disaster area and Family Tracing and Reunification (FTR). Responders can perform tasks more efficiently by using systems that can automatically generate digital maps and locate missing persons. However, disasters can damage network infrastructure, leaving the affected area without access to the Internet and Cloud-based computing resources. In this study, we present the designs, implementation, and evaluations of systems that aid responders in disaster area mapping and FTR. Realizing these systems is challenging because they must be able to (1) send and receive without continuous, end-to-end networks and (2) handle heavy computing loads without access to Cloud-based resources. We address these challenges by (1) using Delay-Tolerant Networks and data ferries for communication and (2) distributing computing tasks to the available devices in the disaster area. We show how our mapping and FTR systems work, including functions for data collection and delivery, computing load balancing, and output delivery. We found that the improvement in processing latency from load balancing offsets the communication latency. Our mapping system with load balancing decreases the time needed to generate and deliver pieces of disaster area maps by approximately 2 hours in cases where large amounts of data have to be processed. The 2-hour reduction is a large benefit for disaster operations where the speed of generation and arrival of information are critical. Furthermore, initial evaluations of our FTR system show that it can execute accurate face recognition in 7 seconds, thus it is capable of quickly handling the computing requirements of FTR.

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, NAIST-IS-DD1461012, March 12, 2017.

Keywords:

Disaster Response, Delay-Tolerant Networks, Distributed Computing

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

In recent years, natural disasters have been occurring more frequently. In 2010, a magnitude 7.0 earthquake struck Haiti, causing more than 200,000 fatalities and causing 1.5 million people to move to evacuation camps at its peak. In 2013, Typhoon Haiyan hit the Philippines, displacing approximately 600,000 people. Most recently in 2016, a series of earthquakes struck the Kumamoto prefecture of Japan, and more than 44,000 people had to be evacuated.

In the immediate aftermath of such disasters, local governments and civilian volunteers commence disaster response operations, which include a variety of tasks. One such task is disaster area mapping. After the disaster strikes, the layout of the affected areas drastically change. Many roads and thoroughfares become impassable because they are damaged or flooded. Buildings and infrastructure collapse and become unrecognizable. Because of the altered landscape, generating maps of the affected areas becomes an integral step in response operations. The generated maps aid in operations by guiding evacuees through unfamiliar and hazardous terrain to safe refuges and by providing the response team leaders with information that help in their decision-making.

However, disasters can interrupt or totally destroy communication infrastructures [42]. Commonly-used end-to-end communication networks (i.e. the Internet) become unavailable, which renders the Cloud-based resources and services, which are required by current mapping systems to store data and generate maps, inaccessible. Because of this, response teams are limited to sketching and using paper maps which are, unlike their digital counterparts, difficult to disseminate and replicate. Also, paper maps cannot be used to calculate the fastest routes, for instance, from a responder's current location to a point-of-interest in the area.

Furthermore, digital maps can use a least-cost path algorithm (e.g. Dijkstra's algorithm) to calculate such paths.

Another task that disaster response teams perform is Family Tracing and Reunification (FTR). During response and evacuation operations, many families become separated. Relatives end up in different shelters or get lost. FTR is the process by which these separated families are reunited. During the FTR process, separated persons are placed in the care of response teams, which then proactively search for their families or guardians.

Similar to disaster area mapping, the FTR process is limited by the lack of network infrastructures. Digital FTR systems require Cloud-based resources for storing and processing large amounts of evacuee data [57]. These resources however, are inaccessible in disaster areas. Moreover, while traditional methods can expedite the process, they rely on paper-based registries and notice boards, which cannot be easily disseminated nor automatically match missing family member queries with existing records. Also, FTR systems require personal information, which some registrants, such as children, the elderly, or People with Disabilities (PWDs), may be unable to provide [21].

While in recent years, smartphones have become more ubiquitous and have improved in terms of computing power, they still may not be able to handle the heavy computing load of disaster area mapping and FTR systems.

The aforementioned limitations to the current methods of mapping and FTR during disaster response operations serve as motivation for this research. In this work, we aim to realize digital applications that aid the disaster area map generation and FTR processes. We address various technical challenges to realizing our applications and evaluate our proposed solutions using simulations and experiments.

1.2 Problem Statements

The shortcomings of current methods of disaster area mapping and FTR motivated us to answer the following problems in this thesis:

1. **"How can we realize a digital mapping system that functions in a disaster area without continuous, end-to-end communication in-**

frastructures?" Our goal in answering this problem to satisfy the computing and communication requirements of a digital mapping system. (1) The system must have functionality that collects the required data to generate the map. (2) It must be able to handle the execution a map inference algorithm to process large amounts of data to generate the map, without access to Cloud-based computing resources. Finally (3), the system must have a means of data propagation without requiring continuous, end-to-end communication networks.

We solve this problem by presenting a system that enables responders in the disaster area to collect data using their smartphones. Then, the system uses the available computing devices in the area to process the collected data, on-site, to generate the map. Finally, the system uses a Delay-Tolerant Network (DTN) of smartphones and leverages response vehicles as data ferries to communicate without network infrastructure.

2. The second research problem we ask is, **"How can we realize a digital FTR system that can be used in a disaster scenario?"** Our goal here, similar to that in problem 1, is to satisfy the requirements of a digital FTR system. (1) First, the system must be able to handle the storage and computing requirements of the FTR process. It must be able to store and process large amounts of records from many evacuees and missing persons. (2) Then, the system must also be able to disseminate data without the Internet. (3) Finally, the system must also be able to find or match missing persons without requiring text-based personal information.

   We solve the problem by building on the architecture of the digital mapping system. (1) Our system uses the available computing devices in the disaster area to store and process evacuee records on-site. (2) To communicate and disseminate queries for missing people, our system leverages response vehicles as data ferries. (3) Finally, our system uses face recognition to find missing persons when text-based information is unavailable.

1.3 Organization of Dissertation

The rest of this dissertation is organized as follows: we present a review of related literature in Chapter 2. Then, we explain our target disaster scenario in Chapter 3. In Chapter 4, we present our initial designs and evaluations for a digital mapping system that works for small subsections of a disaster area and delivers maps to evacuees. In Chapter 5, we improve our disaster mapping system by adding a load balancing algorithm to enable it to process larger amounts of data from bigger disaster areas. We evaluate the system further using experiments and simulations. In Chapter 6, we present the designs and evaluations of an FTR system that uses concepts from the mapping system in Chapters 3 and 4. Finally, we present our conclusions in Chapter 7.

2 Related Literature

In this chapter, we present a review of studies and discuss the concepts related to our research.

2.1 Digital Pedestrian Map Inference

Many studies have proposed algorithms for converting raw GPS traces to digital maps [44] [28] [61]. They require trace data to be collected for long durations of time by a moving vehicle or pedestrian. However, time is a limited resource in disaster response operations and a faster method of data collection is needed. Blanke et al [8] proposed a crowd-sourced method of collecting trace data during short-term, city-scale events such as festivals or disasters. Their system however, requires that the collected trace data be uploaded to a server for processing. In our study, we consider the scenario where such Cloud-based computing resources are unavailable.

2.2 Family Tracing and Reunification

During disaster evacuation, some families get separated, with members ending up in different evacuation centers. Response teams reunite families via the FTR process outlined in [21]. Response teams first identify separated persons (e.g. found in evacuation centers) who have not been claimed by any family or guardians. When a separated person is identified, responders create a record containing personal information, such as the person's name, guardians, present location, and photo. Records are often created by filling out paper forms. Families looking for separated relatives report their missing relative to response teams, and give a query containing the separated person's information and photos.

Next, the responders proactively search for the separated person's family or guardian. They compare records of separated persons with queries and look for possible matches. Once a match is found and verified, the family is reunited.

To aid FTR, responders commonly use notice boards that show the photos and personal information of separated people. However, paper-based documentation and notice boards cannot efficiently be disseminated nor can they automatically find matches. While digital FTR systems such as RapidFTR [57] exist, they require the Internet and cloud-based resources. In this study, we propose methods that do not rely on cloud resources because they may be unavailable during disasters.

2.3 Face Recognition for FTR

There may be cases where separated persons cannot provide personal information (e.g. very young children or the disabled). Thus, the system we propose in this study uses face recognition to match queries for missing family members with existing records. In our implementation, we use JavaFaces* implementation of the Eigenfaces method [55]. The implementation requires a face image input and compares it with a database of face images (of the same dimensions). The database image with the least distance from the input face (and within a user-defined similarity threshold) is returned as a possible match. Eigenfaces is often used as a baseline method. A characteristic of Eigenfaces is that it reduces computation by representing faces with a small number of coefficients, which is suitable in disaster scenarios where cloud-based computing resources are unavailable.

2.4 Disaster Information Delivery using DTNs

Information delivery is critical during disaster response [5]. However, disasters can destroy network infrastructure, leaving the area without end-to-end communication networks. Because of this, many studies have proposed using Opportunistic and Delay-Tolerant Networks (DTNs) [17]. Pelussi et al [35] surveyed

*https://code.google.com/archive/p/javafaces/

opportunistic networks that were deployed in actual scenarios and discuss different routing techniques. In [30], the performance during disaster situations of opportunistic routing protocols were analyzed. DistressNet [19] [11] [10] is an ad hoc wireless sensor network architecture that provides data collection services in disaster scenarios. Fujihara and Miwa proposed an evacuation guidance service using opportunistic networks [18]. In [1], Kikuchi and Shibata proposed a disaster information system that could handle server group failures using mobile cloud computing. Fajardo et al proposed an aggregation method to minimize delays in data delivery [16]. These studies improve data collection in disaster sites by reducing delivery latency. Our system differs because it considers scenarios where not only fast delivery is required. Our study also considers how the computing requirements of map inference and FTR can be satisfied without Cloud-based computing resources.

2.5 Multi-hop Networks for Disaster Scenarios

In addition to the DTN-based systems in Section 2.4, many studies have leveraged multi-hop ad hoc networks for communication in disaster scenarios. Reina et al conducted surveys on multihop networks that can be used for disaster response [37] [38]. These infrastructureless networks define multi-hop routes between their nodes. They described ad hoc network paradigms, such as Mobile Ad Hoc Networks (MANETs) and Vehicular Ad Hoc Networks (VANETs), and reviewed the existing work under each. In [37], the authors noted that works on MANET-based disaster communication mainly compare different routing protocols. The studies in [40] and [39] evaluated well-known MANET routing methods and found that the Ad hoc On-demand Distance Vector (AODV) protocol [36] was the most suited for disaster scenarios. [13] presented a survey of existing MANETS. The authors presented lessons learned from previous research successful paradigms that evolved from MANETs. [26] and [23] presented the architecture of a MANET-based communication system called P2Pnet that uses personal computers to support large numbers of responders. The study in [50] proposed a VANET-based system called RescueMe that aids in rescue operations and addresses the security and privacy issues of the network. Nishiyama et al presented a

prototype of relay-by-smartphone, a system that uses multihop, device-to-device communication during disasters [32]. While most of existing multi-hop networks for disasters focus on efficient message delivery, our study differs because we propose how to handle computing tasks (i.e. pedestrian map inference and FTR) in a disaster scenario.

2.6 Disaster Scenario Mobility Models

Many studies have proposed mobility models that have been used to evaluate DTN systems [46]. However, most models are random models, such as the Lévy Walk [41] and the Random Waypoint Mobility model [24]. During disaster scenarios, humans move according to their purposes or objectives. Aschenbruck et al proposed a mobility model that represents the behavior of humans in a realistic disaster area [4]. Their study considers factors that influence the performance of communication networks in disaster scenarios such as node movement across different areas and obstacles. Nelson et al proposed a role-based disaster mobility model and showed how it created a different network topology than the random walk mobility model [31]. Similarly, Uddin et al. proposed the Post-Disaster Mobility (PDM) model [56], which emulates the role-based behavior of humans during disaster scenarios (i.e. responders move between targets and coordination centers) and the effects of the disaster on the map topography. They compare the PDM to the Random MapPoint Model and show the effects of the mobility on DTN performance. In this study, we use movement patterns from the PDM model in simulations to model the behavior of responders.

2.7 Distributed Computing in DTNs

The concept of opportunistic computing, where users leverage the available computing resources in the environment has been presented in [14] and [15]. In our study, we propose a system that uses available resources in a disaster area, and uses a load-balancing heuristic to distribute computing tasks to these resources. The study most related to ours is the Serendipity system [48] [47], in which nodes distribute computing tasks to remote mobile-devices. Our system differs because

whereas the Serendipity uses random-walk mobility models (e.g. Lévy Walk and Random Waypoint) and applications such as speech-to-text, our study considers the elements of disaster response scenarios: the mobility patterns of responders and vehicles, the equipment, and map inference and FTR applications for response operations.

3 Disaster Response Scenario

In this chapter, we define the post-disaster scenario, response operations, and available equipment and resources. Figure 3.1 shows a diagram of the scenario.

3.1 Post-Disaster Scenario

Our system targets the first 6 to 72 hours after a disaster strikes city (Figure 3.1a). Response teams have deployed and have started their emergency response operations (e.g. search and rescue and initial assessment). Some paths in the area have been rendered impassable (e.g. due to flooding or structural collapse). Communication infrastructures such as cell sites have been damaged, leaving the area without the Internet. These network infrastructures have not yet been repaired.

Response teams (Figure 3.1b) have deployed to the disaster area. Based on the guidelines in [22], the disaster area has been divided into subsections (Figure 3.1c), each with its assigned response team. Each response team has established a rally point in its subsection (Figure 3.1d) where they setup equipment and rendezvous for meetings. Rally points are structures such as school buildings or local government offices, which are selected based on prior planning. The response commander manages all operations from a rally point.

3.2 Disaster Response Operations

After establishing a rally point, responders start operations in their subsection. They perform tasks such as exploring, searching for and rescuing victims, assessing damages, and looking for resources (Figure 3.1e). They move, often on foot, to perform tasks, and then return to their rally point and wait for new tasks.

This pattern is typical of response teams and has been modeled in the PDM model [56].

Some responders patrol the area in vehicles (Figure 3.1f), following planned routes and passing by rally points to relay information or share equipment between response teams. Routes are assigned by leaders, and are changed upon receiving new orders (Figure 3.1g). We note that the routes shown in Figure 3.1 are an abstraction. In actual operations, passable roads are used.

The response teams create maps of their subsections. After performing a task, responders return to their rally point and update their subsection's map with information collected during their task (Figure 3.1h). Maps show paths and points-of-interest such as victim locations and collapsed structures. Maps are shared with the response commander to aid in decision-making (Figure 3.1i).

3.3 Equipment and Resources

Response teams can access the following inventory:

1. Rally Points - Rally points have electricity, and in case power has been cut, have backup fuel generators or solar panels. Response teams have computer workstations (e.g. laptops) in the rally points. We assume these workstations to have short-range wireless interfaces (e.g. Bluetooth or Wi-Fi), a minimum storage capacity of 250 GB, and CPUs within the 2.0 GHz to 3.0 GHz range.

2. Transportation - For short-range movements within their subsections, response teams move on foot. For long-range movements across subsections, they ride vehicles.

3. Responders - We assume, as part of their inventory, responders have smartphones that have short-range wireless interfaces (e.g. Bluetooth, Wi-Fi, and Wi-Fi Direct) and sufficient storage capacities (i.e. 16 GB of free storage at the minimum). Responders recharge their smartphone batteries while at rally points or in response vehicles.

4. Communication - With the damaged network infrastructure, communication between the devices of response team members are limited to their built-in short-range wireless interfaces (e.g. Wi-Fi or Wi-Fi Direct).

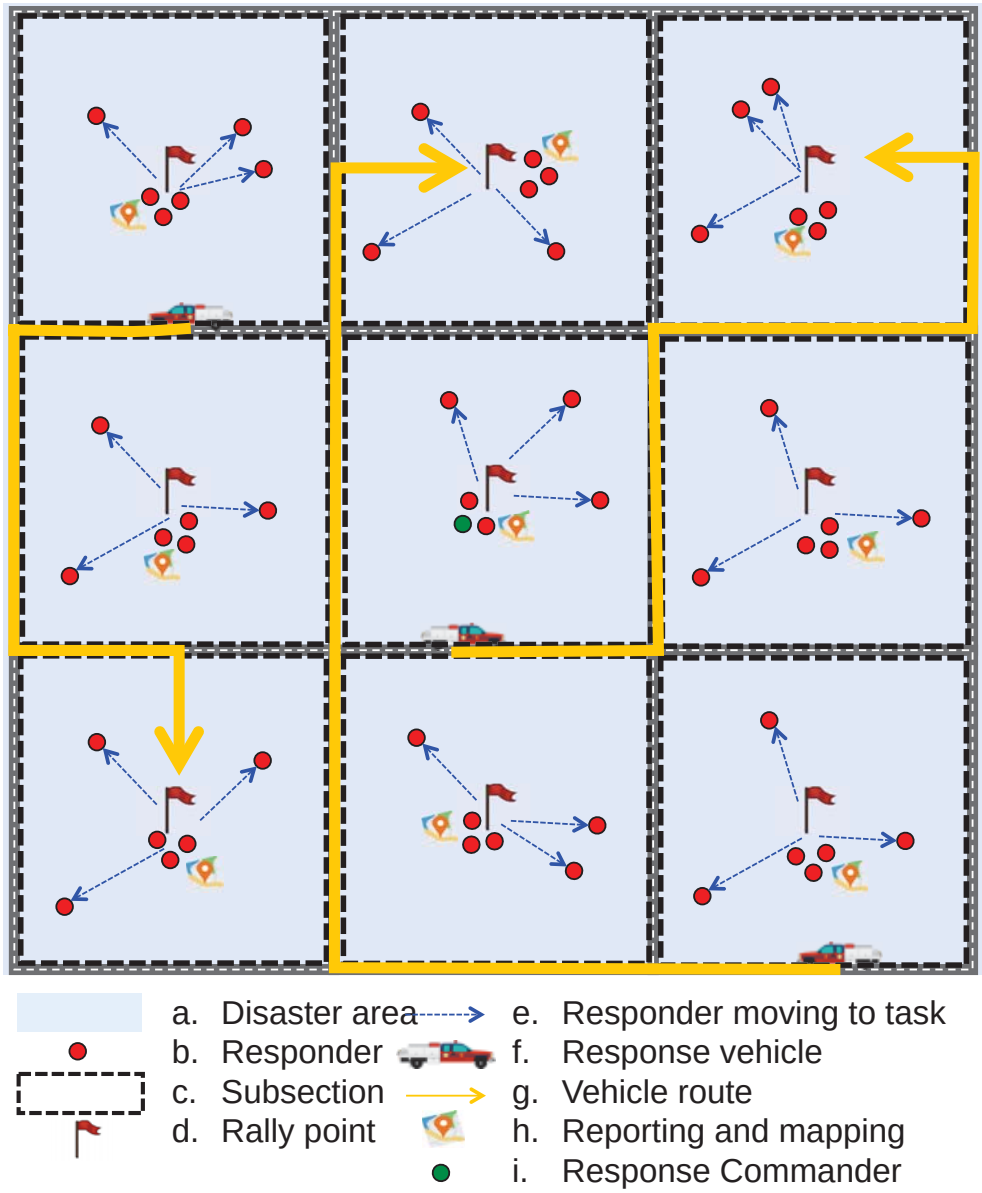


Figure 3.1: The post-disaster scenario. (a) A disaster has struck and (b) responders have deployed. (c) The area has been divided into subsections. (d) Responders have established their rally points. (e) Responders move from their rally point to perform tasks. (f) Response vehicles patrol the area (g) following their routes. (h) Responders return to their rally point after performing tasks. (i) The Response Commander manages all operations.

4 Disaster Area Mapping Using Spatially-Distributed Computing Nodes Across a DTN

4.1 Introduction

In this chapter, we present the initial designs of a system that generates digital disaster area pedestrian maps and delivers them to evacuees looking for a safe refuge.

Current mapping methods include creating paper-based maps that are difficult to replicate and share, and cannot calculate the fastest routes between locations. Also, without the Internet, Cloud-based mapping services are inaccessible.

The main challenges to realizing a system that addresses these shortcomings are (1) how the raw data required to generate the map are collected, (2) how data are transmitted without continuous, end-to-end networks, and (3) how to handle the heavy computing load of map generation without access to Cloud-based computing resources or services.

To solve these challenges, (1) we implemented an Android application called DTN MapEx that responders use to collect the required GPS traces to generate the map. (2) The system establishes a DTN of smartphones that uses epidemic routing to deliver data. (3) The collected raw trace data are sent to Computing Nodes, which are workstations that are deployed in the area,to infer the map.

We evaluate our system using experiments and simulations and compare cases when map inference is performed by smartphones against cases where stronger

computing devices that are available in the disaster area are used. We show that, even though smartphones may be available in the area, using them to generate the map is not efficient, and results to high latencies when large amounts of data have to be processed. Additionally, we show that using stronger computing devices can decrease the processing time of map inference enough to offset communication delays.

In the rest of the chapter, we present the specific assumptions in the scenario we are targeting, the design and workflow of our system, and our evaluations.

4.2 Assumptions

World Assumptions - Our target area is a 3 km x 3 km section of Marikina City, Philippines. We assume that a typhoon disaster has hit the area. Our scenario lasts for $t = 24,000$ seconds. At the start of our scenario, the government agencies in charge of disaster response have started operations. The target area has been divided into $n = 9$ (1 km x 1 km) subsections. A Base of Operations (BoO) has been established in the center subsection where the response team leader is stationed. n Rally Points have been established, one in each subsection (the BoO counts as a Rally Point). Rally Points are sites where response teams rendezvous for coordination meetings and where evacuees take shelter (e.g. school buildings). Figure 4.1 shows the target area.

Agent Assumptions - Agents in the scenario follow the mobility patterns described in the PDM model [56]. At $t = 0$ seconds, the response teams divide into sub-teams. Each sub-team is assigned to a subsection. The sub-teams move to their respective Rally Points and perform tasks (e.g. exploring the subsection, assessing the area and responding to emergencies). After each task, the sub-team members return to their Rally Point and wait for another task. The response team also includes patrols: responders riding vehicles who move around the target area, from subsection to subsection, and Rally Point to Rally Point. At certain times, the team leader radios the sub-teams and patrols to return to the BoO for a coordination meeting. After each coordination meeting, the sub-teams return to their Rally Points and resume tasks.

We assume two types of civilians: volunteers and evacuees. Volunteers are

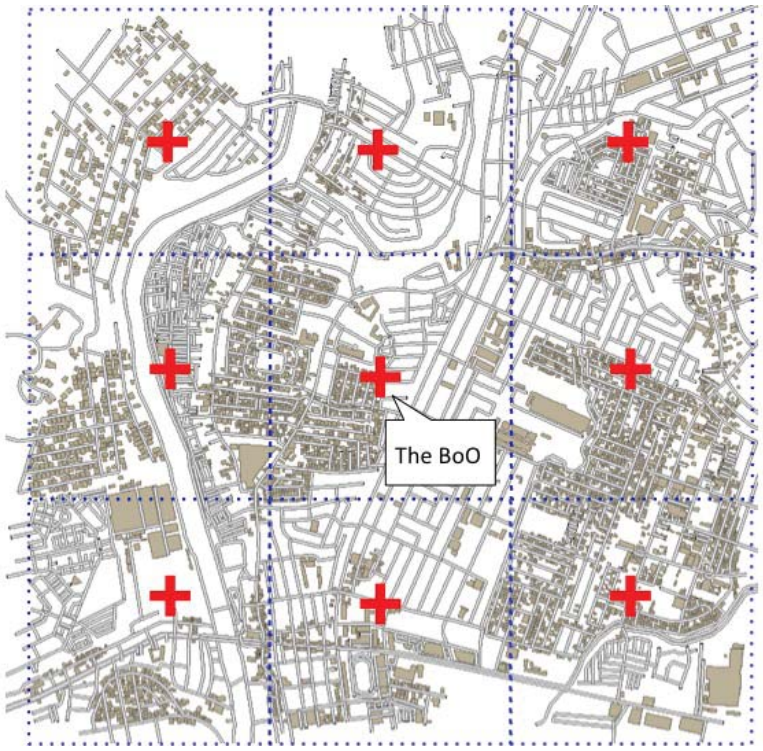


Figure 4.1: The target area. The dashed blue lines mark the subsection borders, and the red crosses represent Rally Points. The BoO is in the center subsection.

people living in each subsection who are able to assist in response operations. Volunteer mobility is similar to that of response sub-teams. Evacuees are people living in the subsection who cannot help in response operations. They need a map of their subsection to know where the Rally Point is so they can seek refuge. Evacuees stay in their starting points (i.e. their homes) until they receive a map.

Equipment Assumptions - We assume that every response team member, volunteer, evacuee, and patrol has a mobile device with storage, short-range wireless connectivity (e.g. Bluetooth or Wi-Fi), and GPS location provider. There is no continuous end-to-end wireless network in the target area. The power has been cut, but each Rally Point has its power source (e.g. fuel-run generators or solar cells). Each Rally Point also has commodity computing equipment that are more powerful than mobile devices (e.g. the laptops of response teams). Finally, every mobile device is assumed to have sufficient storage capacity for the data needed to create the map (e.g. current devices are capable of storing up to 16 GB to 128

GB) and the computing devices in the Rally Point are assumed to have higher storage capacity (e.g. 500 GB).

Output Requirements - The evacuees need a disaster map of their subsection that shows a pedestrian path to follow to a Rally Point. The map must show the network of roads and paths and the walking speeds along them, which can be used to calculate a least-cost path to safe refuges.

4.3 System for Disaster Area Mapping

4.3.1 Trace Data Collection

The first step in map generation is collecting GPS trace data from the disaster area. Our system achieves this by leveraging the mobility of humans described in 4.2. In the system, all pedestrian response team members and volunteers in a subsection comprise the set SN of i mobile Sensor Nodes. As each Sensor Node walks around its subsection, its mobile device automatically collects GPS and velocity traces. We implemented a prototype application called DTN MapEx [52] for collecting trace data. DTN MapEx has a GPS module that logs the GPS and velocity traces of the Sensor Nodes. We note that all roads in a disaster area are not always covered. Some roads may be impassable and some areas may not be critically damaged, so they are not prioritized for exploration. Disaster response operations often prioritize the critically affected areas for assessment and mapping. Our system can be used to map these priority areas first, and then the other low-priority areas if required.

4.3.2 Communication using DTN

Mobile Devices as DTN Nodes - The system uses mobile devices and their built-in short-range wireless communication interfaces (i.e. Bluetooth or Wi-Fi) to establish a DTN. The DTN is composed of the following nodes: N Sensor Nodes $SN = \{s_0, s_1, ..., s_N\}$; Patrol Nodes $PN = \{p_0, p_1, ...\}$, which function as data mules [45] while they move from one area to another; Evacuee Nodes $EN = \{e_0, e_1, ...\}$, which are the destination nodes of the generated maps; and stationary Computing Nodes $CN = \{c_0, c_1, ...\}$, which will be discussed later.

Post-Disaster Mobility Model - All Sensor and Patrol Nodes follow the mobility patterns in the PDM model. Sensor Nodes follow the Event-Driven pattern, where they move from a Rally Point to a location in the subsection to make an assessment or respond to an emergency. Patrol Nodes follow the Cyclic Route and Center-to-Center patterns where they move from one Rally Point or one subsection to another. They also follow the Converge-Move pattern where they move towards the BoO for coordination meetings. Evacuee Nodes begin at an initial location (i.e. their homes), where they remain until they receive a map. At which point, they follow the map and move towards the Rally Point. Computing Nodes are stationary nodes that are deployed in each Rally Point and remain there until operations cease. Algorithm 1 shows the mobilities of Sensor and Patrol Nodes.

Epidemic Routing - Nodes in the system share DTN bundles using Epidemic Routing, a flooding protocol where bundles are replicated in all encountered nodes [58]. The system uses Epidemic Routing because of its good delivery ratio when node buffers are not congested [30]. In our initial evaluations, we use Epidemic Routing as a baseline with the assumption that nodes have sufficient buffer size to avoid flooding. In real disaster scenarios flooding is possible, thus we will evaluate the performance of other routing protocols in future work.

Implementation with IBR-DTN - As a proof of concept, the DTN MapEx application has a DTN Module, which uses the IBR-DTN implementation of the Bundle Protocol on Android [43]. The DTN module uses the IBR-DTN daemon to manage routing and neighbor discovery. The module shares data using Wi-Fi and Wi-Fi Direct.

4.3.3 Pedestrian Map Generation

The final step in map generation is processing the collected data into a map.

Pedestrian Map Inference - To generate a map from the GPS and velocity traces, we implemented a modified version of the map inference algorithm in [8]. This method was chosen because it was made for short-term, city-scale events like festivals or disasters. We added velocity information by getting the average velocity of the traces belonging to an edge. The output is a weighted, directed graph where the edges represent pedestrian paths, the nodes are intersections and

Algorithm 1 Sensor and Patrol Node Mobility

if $t = 0$ seconds **then**
  Response Team Sensor Nodes initialize in BoO
  Volunteer Sensor Nodes initialize in Random Building
  Sensor Nodes move to assigned Rally Point
  Patrol Nodes initialize in BoO
  Patrol Nodes move to random Rally Point **or** subsection
  Wait $rand(600, 1800)$ seconds for a task
end if
while $t < 10000$ **or** $10000 < t < 20000$ seconds **do**
  Sensor Nodes move to random location in subsection
  Sensor Nodes perform task for $rand(600, 1800)$ seconds
  Sensor Nodes return to Rally Point and wait for next task
  Patrol Nodes move to random Rally Point **or** subsection
end while
if $t = 10000$ seconds **then**
  // Coordination Meeting is Called
  Response Team Sensor Nodes return to BoO
  Volunteer Sensor Nodes return to Rally Point
  Patrol Nodes move to BoO
  Wait $rand(600, 1800)$ seconds to finish meeting
else $\{t = 20000$ seconds$\}$
  // Return to Base
  Response Team Sensor Nodes return to BoO
  Volunteer Sensor Nodes return to assigned Rally Point
  Patrol Nodes return to BoO
  Wait until $t = 24000$ //End of operations
end if

corners, and the weights are the average speeds along the edges. DTN MapEx has a Map Module that runs this algorithm. The edges of the output map are color-coded, where warmer colors represent velocities $v > 5$ km/h and cooler colors represent $v \leqslant 5$ km/h. Figure 4.2 shows a sample output of the Map Module of

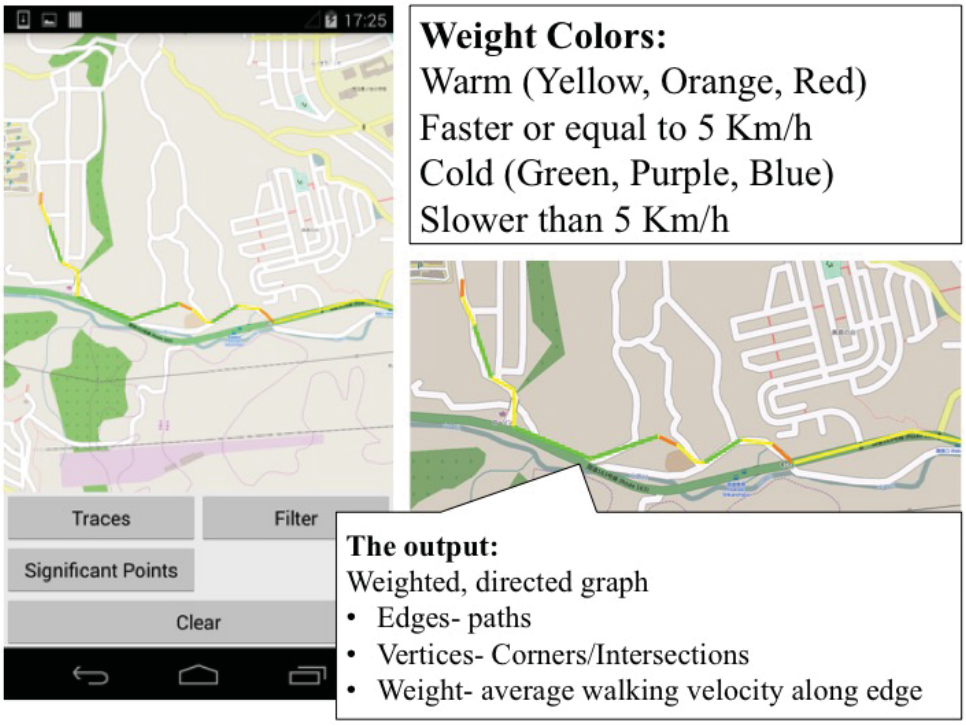


Figure 4.2: The Map Module of DTN MapEx. The output map is displayed as a weighted, directed graph where the edges are the paths, nodes are the corners and intersections, and weights are the average walking speeds along the edges. A sample output map is overlaid on a map tile.

DTN MapEx overlaid on the actual map tile.

Spatial Distribution of Computing Nodes - Although the computing capability of mobile devices has been improving, a single mobile device may still not be enough to efficiently (i.e. time-wise) perform map inference. As such, we introduce Computing Nodes to our system, which are commodity computers with more computing power than mobile devices (e.g. laptops). A Computing Node is deployed in each Rally Point (e.g. the laptops of response teams). Each Computing Node generates the map of its subsection. Sensor Nodes opportunistically route their collected data to the Computing Node of their subsection. When a Computing Node receives data, it executes the pedestrian map inference algorithm, and aggregates it with previous data. The generated map is then routed to Evacuee Nodes. Figure 4.6 shows how the system generates subsection maps.

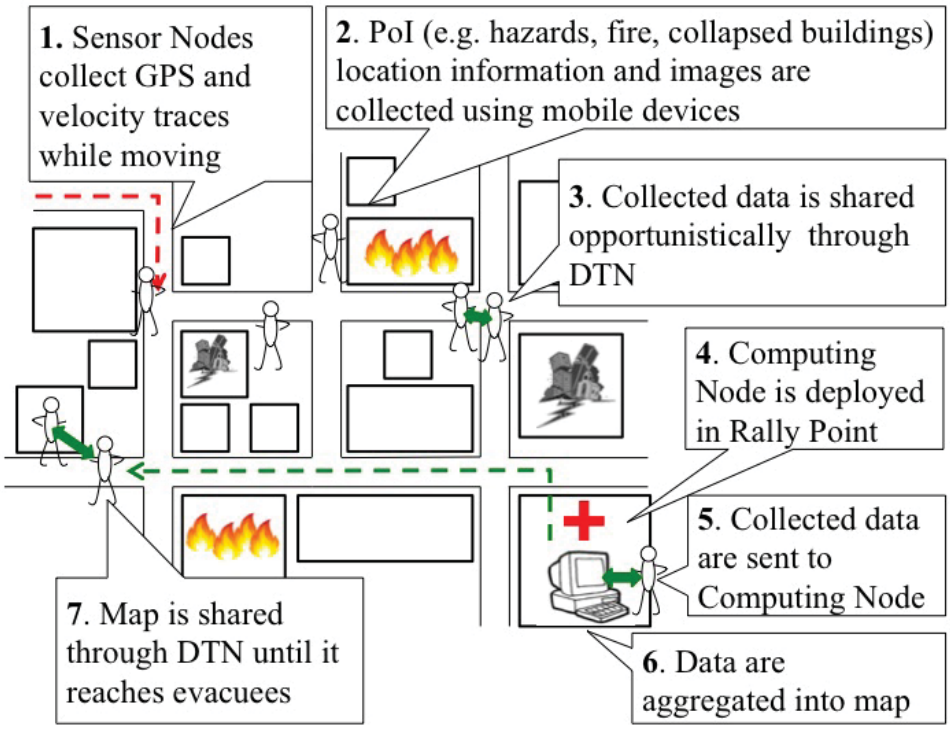


Figure 4.3: Subsection map generation process. Data are collected by Sensor Nodes and are opportunistically routed to the Computing Node in the Rally Point. The Computing Node generates the subsection map from received data and routes it to Evacuee Node. In this work, the system is currently only using GPS and velocity traces. Future work will integrate PoI information collection and photographic functions.

It must be noted that while Evacuee Nodes require the subsection maps, the response teams may need maps with different types of information (e.g. PoIs and assessment information), and team leaders may only need summary information about the entire area. In future work, we will address how PoI information can be added to the subsection maps and how these can be aggregated. We will also study the effects of adding more Computing Nodes to generate maps with varying information granularity (i.e. responders need more detailed maps).

4.4 Evaluations and Discussion

We evaluate our system using the Mapping Time ($\mathbb{T}_{mp}$) metric, which we define as the total time required to collect trace data, generate the subsection map, and deliver it to an Evacuee Node in the subsection. We investigate two cases:

Without System (*wo*) - We measure $\mathbb{T}_{mp}^{wo}$ when the system is not used, and the subsection map is generated locally in an Evacuee Node, $e \in EN$. In *wo*, each Sensor Node $s_i \in SN$ in a subsection generates a single GPS and velocity trace data bundle d_i. We assume that the subsection map can be generated from the set of d_i generated by s_i. We assume that each s_i has received a request from e, asking for d_i. After an offset generation time t_i^{gen} (i.e the time required for s_i to generate data bundle d_i), s_i routes its d_i to the Evacuee Node e. When the d_i arrive at e, they form a queue, and e executes the map inference algorithm on each d_i in a first-in, first-out order. e receives the map when the last d_i has been processed. We define the $\mathbb{T}_{mp}^{wo}$ model as:

$$\mathbb{T}_{mp}^{wo} = \sum_{i=0}^{N} \frac{d_i^{sz}}{e^{sp}} + \sum_{i=0}^{N} f(\Delta t_i^e) \tag{4.1}$$

where

$$\Delta t_i^e = t_i^{arr} - t_{i-1}^{fin},$$

$$t_i^{arr} = t_i^{gen} + t_i^{lat}$$

$$f(\Delta t_i^e) = max\{\Delta t_i^e, 0\},$$

d_i^{sz} is the size of the data bundle (in KB) from s_i, e^{sp} is the speed at which e can execute the inference algorithm in KB/s (i.e. how much data e can process per second), and $f(\Delta t_i^e)$ is the offset time caused by the processing queue of e. d_i arrives in e at time t_i^{arr}, which is t_i^{gen} plus the delivery latency, t_i^{lat}, of d_i from s_i to e. Once in e, d_i waits until e finishes processing any preceding bundle d_{i-1} in the queue (t_{i-1}^{fin}). If d_i arrives once or before d_{i-1} is done (i.e. $\Delta t_i^e \leqslant 0$), e can process d_i as soon as it finishes d_{i-1}, and $f(\Delta t_i^e) = 0$. If d_i arrives after d_{i-1} is done (i.e. $\Delta t_i^e > 0$), e has an offset time $f(\Delta t_i^e) = \Delta t_i^e$ before it can start processing d_i. We assume that i indicates the order of bundle arrival in e (i.e. d_0 from s_0 is first). The time sequence diagram of the *wo* case is shown in Figure 4.4.

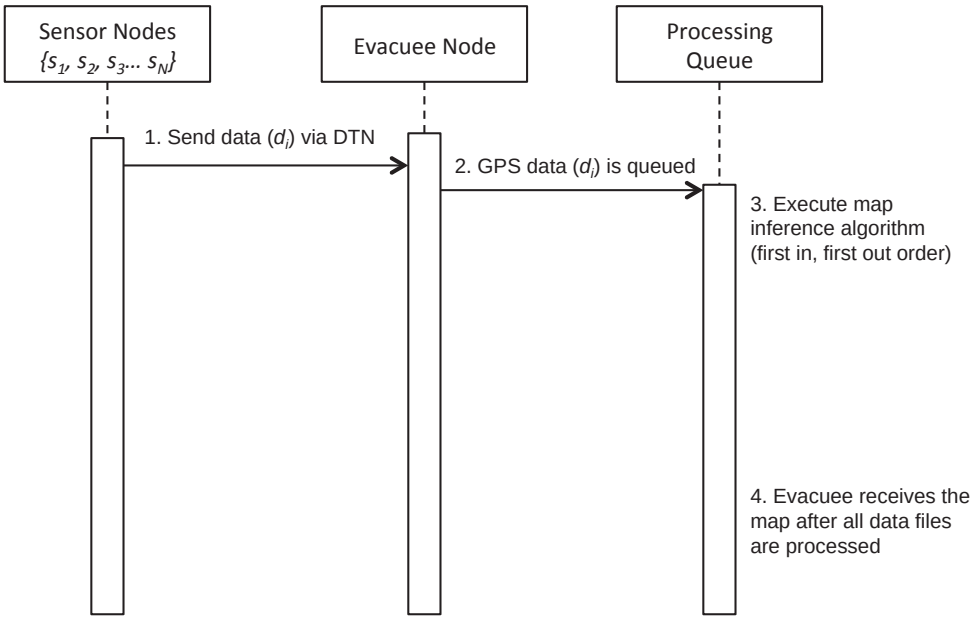


Figure 4.4: The time sequence diagram of the *wo* case. (1) Sensor Nodes send their collected GPS data files to an Evacuee via the DTN. (2) The received data files enter the queue of the Evacuee's smartphone. (3) The smartphone executes the map inference algorithm. (4) The evacuee receives the map after all data files are processed.

With System (ws) - We measure $\mathbb{T}_{mp}^{ws}$ when the system is used, and the subsection map is generated in a Computing Node c. We define the $\mathbb{T}_{mp}^{ws}$ model as:

$$\mathbb{T}_{mp}^{ws} = \sum_{i=0}^{N} \frac{d_i^{sz}}{c^{sp}} + \sum_{i=0}^{N} f(\Delta t_i^c) + t_{mp}^{lat} \tag{4.2}$$

where

$$\Delta t_i^c = t_i^{arr} - t_{i-1}^{fin},$$

$$f(\Delta t_i^c) = max\{\Delta t_i^c, 0\}.$$

In ws, s_i routes its d_i to the c assigned to their subsection. c queues the received d_i and executes the inference algorithm. After all d_i have been processed, the generated map is sent to e. c^{sp} is the speed at which c can execute the inference algorithm (in KB/s), $f(\Delta t_i^c)$ is the offset time caused by the processing queue of

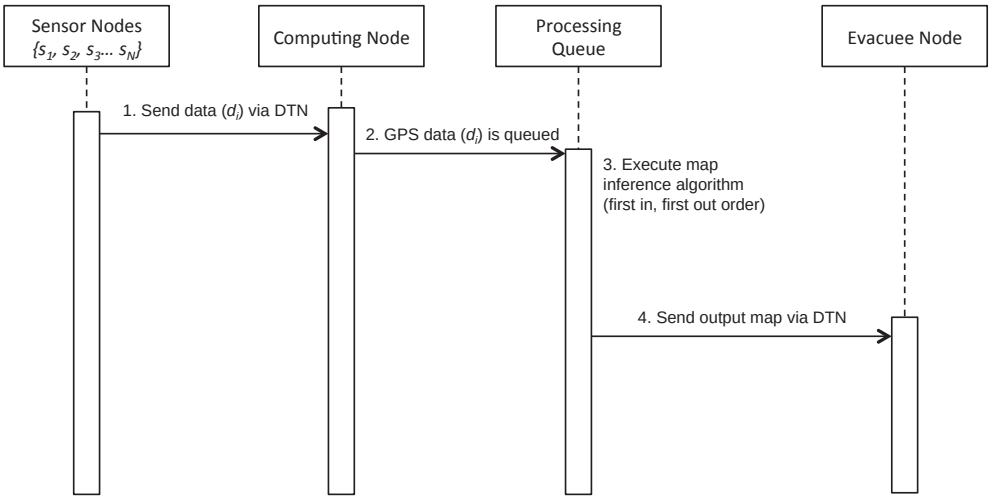


Figure 4.5: The time sequence diagram of the ws case. (1) Sensor Nodes send their collected GPS data files to the Computing Node via the DTN. (2) The received data files enter the Computing Node's queue. (3) The Computing Node executes the map inference algorithm. (4) The generated map is sent to the Evacuee via the DTN.

c. t_i^{arr} is the time d_i arrives in c and t_{i-1}^{fin} is the time c finishes processing d_{i-1}. We assume that i is the order of bundle arrival in c. We assume that c has received a request from e, asking for the map. After c finishes processing all d_i, it routes a map bundle to e. t_{mp}^{lat} is the delivery latency of the map bundle from c to e. We assume that after executing the map inference algorithm, the output is a map that is only a fraction of the total size of all d_i, and that map bundle has the same size as a single d_i. The time sequence diagram of the ws case is shown in Figure 4.5.

We began our evaluations by comparing the performance of mobile devices and a commodity laptop, which represents a Computing Node, in executing the map inference algorithm. We built a GPS and velocity trace data set by asking participants use DTN MapEx. We executed the map inference algorithm on the collected trace data using mobile devices (LG Nexus 4, LG Nexus 5, and Samsung Galaxy 4) and a MacBook Pro, and measured how many KB of data each can process per second. Figure 4.6 shows that the Computing Node was approximately 45 times faster than the mobile devices. From these, we set $c^{sp} =$

45 KB/s and $e^{sp} = 1$ KB/s.

Next, we used the Scenargie network simulator [51] to evaluate the performance of the DTN. We implemented the environment described in 4.2, used the mobility patterns in Algorithm 1, and recorded t_i^{lat} when s_i sent data bundles of different sizes (20, 100, and 500 KB) to a Computing or Evacuee Node. These data bundles represent .csv files containing raw map data, where 100 KB contains approximately 1 Km of GPS trace data. We also recorded t_{mp}^{lat} when Computing Node sent different map bundle sizes d^{sz} (20, 100, 500 KB) to an Evacuee Node.

We also tested the effects of varying the number of SN, where $N \in \{10, 50, 100\}$. Figure 4.7 shows the resulting Cumulative Distribution Functions (CDFs) of t_i^{lat} and t_{mp}^{lat} when $N \in \{10, 50, 100\}$ and $d^{sz} \in \{20, 100, 500\}$ KB. Two Patrol Nodes were used as data mules. The results show that as N increased, the t_i^{lat} and t_{mp}^{lat} latencies shortened, which is typical DTN behavior.

We then used the c^{sp} and e^{sp} values obtained from the experiment and random values from DTN performance result set as parameters in the $\mathbb{T}_{mp}^{ws}$ and $\mathbb{T}_{mp}^{wo}$ models. We assumed that each s_i generates its data bundle between $t = 1800$ and $t = 3600$ (i.e. after exploring for 30 minutes to an hour, s_i has enough data to send), and set $t_i^{gen} = rand(1800, 3600)$. We implemented the $\mathbb{T}_{mp}^{ws}$ and $\mathbb{T}_{mp}^{wo}$ models in Java, and executed 9 scenarios with combinations of $N \in \{10, 50, 100\}$ nodes and $d_i^{sz} \in \{20, 100, 500\}$ KB. Figure 4.8 shows the mean $\mathbb{T}_{mp}^{ws}$ and $\mathbb{T}_{mp}^{wo}$ from 1000 iterations of each scenario.

The *wo* scenarios show that as N and d_i^{sz} increase, $\mathbb{T}_{mp}^{wo}$ also increases. For the scenario where $N = 10$ and $d_i^{sz} = 20$ KB, the mean mapping times for *ws* and *wo* were almost the same because the amount of data to be processed was low. This is expected because the processing load is low in both cases (i.e. Ten data files with a size of 20 KB). Therefore, the delay was mainly caused by the DTN latency. Because the number of DTN nodes were the same for both scenarios and the data files (i.e. the message bundle sizes) were small, their DTN latencies were similar. With similar DTN performances and small processing latency, the mean mapping times for both cases are similar as expected.

The $N = 100$ nodes and $d_i^{sz} = 500$ KB scenario had the largest $\mathbb{T}_{mp}^{wo}$ (51,888 seconds). This is because as d_i^{sz} or i increases, $\sum_{i=0}^{N} \frac{d_i^{sz}}{e^{sp}}$ (i.e. the time e needs to process the d_i queue) becomes increasingly large due to the slow $e^{sp} = 1$ KB/s. In

a case where both N and d_i^{sz} are large, the value of $\sum_{i=0}^{N} \frac{d_i^{sz}}{c^{sp}}$ accounts for a large fraction of $\mathbb{T}_{mp}^{wo}$. This shows that without the system, the bottleneck is caused by the slow execution speed of the Evacuee Node.

The ws scenarios show that as d_i^{sz} increases, the mean $\mathbb{T}_{mp}^{ws}$ only varies slightly. This is because the faster $c^{sp} = 45$ KB/s keeps $\sum_{i=0}^{N} \frac{d_i^{sz}}{c^{sp}}$ relatively smaller than $\sum_{i=0}^{N} f(\Delta t_i^c) + t_{mp}^{lat}$. This shows that with the system, the delay is caused by the bundle delivery latency in the DTN. Thus, if N is increased, the performance of the DTN should improve (i.e. $\sum_{i=0}^{N} f(\Delta t_i^c) + t_{mp}^{lat}$ will decrease), and reduce the $\mathbb{T}_{mp}^{ws}$. The results support this, and smallest mean $\mathbb{T}_{mp}^{ws}$ values were obtained when $N = 100$ nodes (e.g. when $N = 100$ nodes and $d_i^{sz} = 500$ KB, $\mathbb{T}_{mp}^{ws} = 3,923$ seconds, 13x faster than $\mathbb{T}_{mp}^{wo}$).

We also note that there was a small decrease of approximately 20 sec in the mean mapping times when $d_i^{sz} = 100$ KB and $d_i^{sz} = 500$ KB for $N = 10$ nodes. We attribute this to the fact that the Computing Node can process data relatively quickly. Because of this, most of the delay for these cases are caused by the DTN latencies. Given that both cases had the same number of nodes and the computing load is not as heavy (i.e. processing ten 100-KB data files vs. ten 500-KB data files), such small discrepancies caused by the DTN performance are to be expected. Overall however, $\mathbb{T}_{mp}^{ws} << \mathbb{T}_{mp}^{wo}$ as N and d_i^{sz} increase, which shows the benefit of our system.

4.5 Summary

In this chapter, we presented the initial designs and evaluations of our disaster area mapping system. We showed that our proposed approach of using a DTN with Epidemic Routing to send collected raw data to a Computing Node generates and delivers maps faster to evacuees in the disaster area.

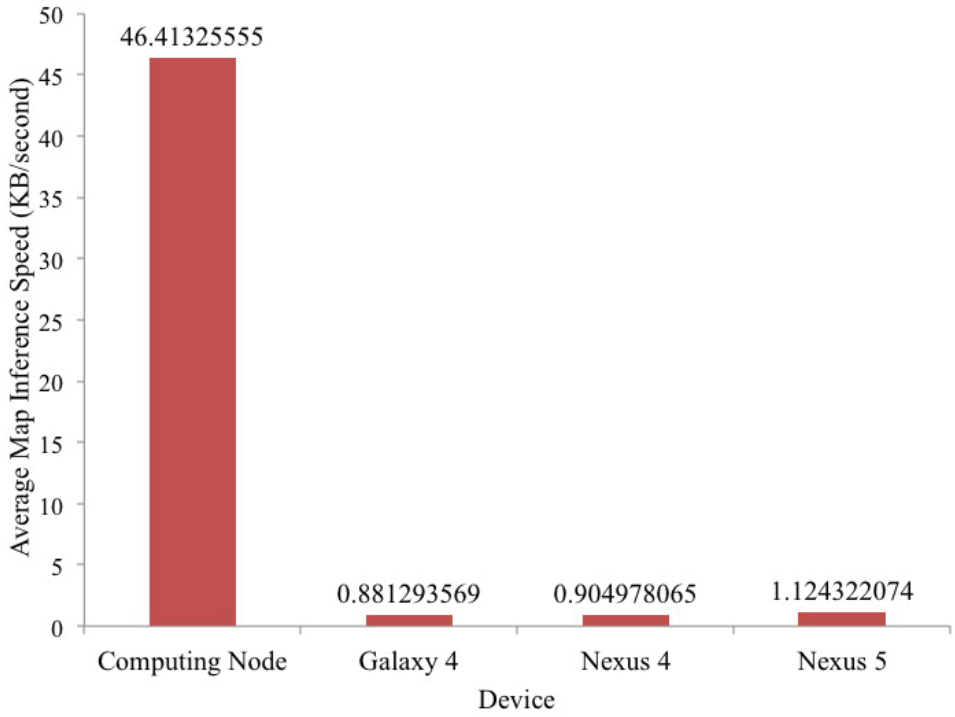


Figure 4.6: Average Map Inference Algorithm execution speeds (in KB/s) of the MacBook Pro Computing Node and the Samsung Galaxy 4, LG Nexus 4, and LG Nexus 5 mobile devices. The Computing Node executed the inference algorithm approximately 45 times faster than the mobile devices.

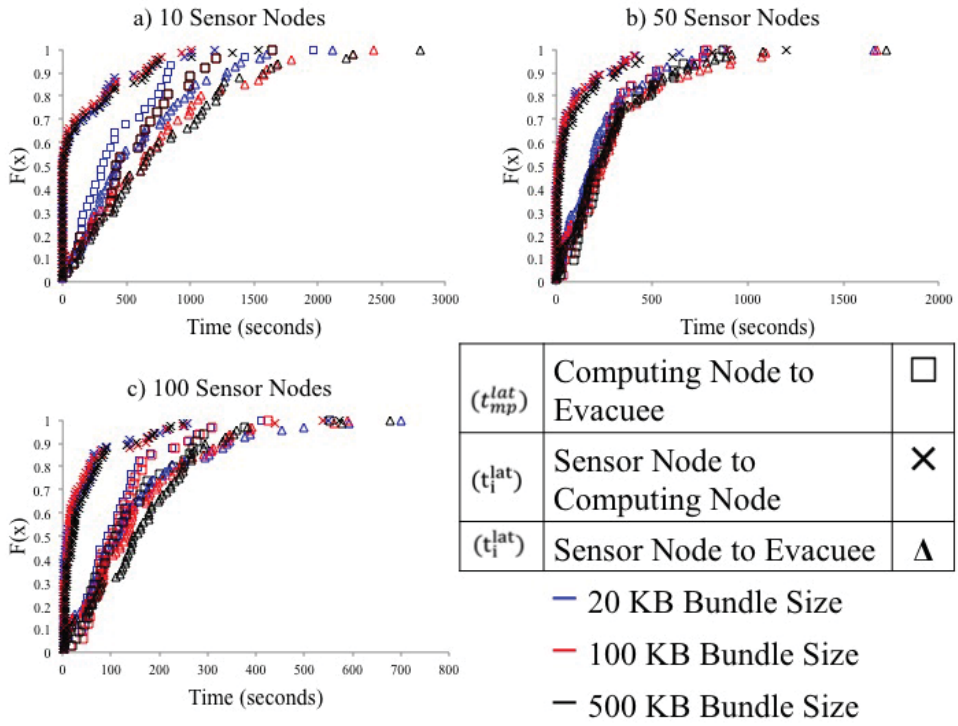


Figure 4.7: DTN performance for $i \in 10, 50, 100$ nodes are shown in CDF a), b), and c) respectively. The delivery latencies for 20, 100, and 500 KB bundle sizes were measured. DTN had $c = 1$, $e = 1$, and $p = 2$ nodes. Other parameters: s and e buffer size = 16 GB, c buffer size = 500 GB; simulation time $t = 24,000$ sec.

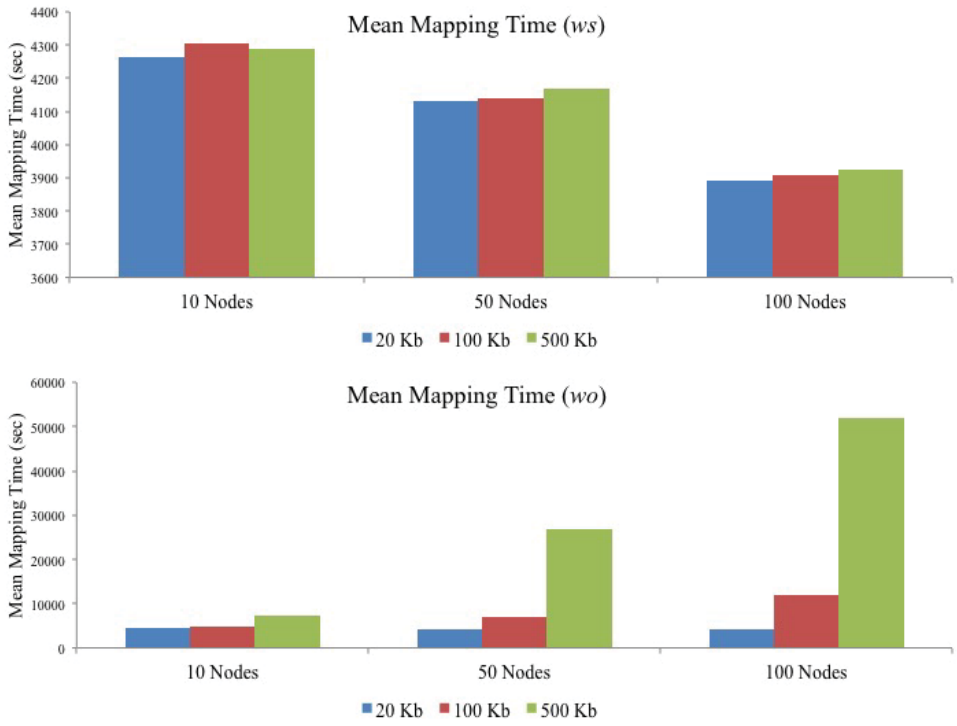


Figure 4.8: The mean Mapping Times $\mathbb{T}_{mp}^{ws}$ (top) and $\mathbb{T}_{mp}^{wo}$ $(bottom)$. $c^{sp} = 45$ KB/s and $e^{sp} = 1$ KB/s. t_i^{lat} and t_{mp}^{lat} values were randomly picked from the DTN performance simulation results, $t_i^{gen} = rand(1800, 3600)$, and 9 scenarios (i.e. combinations of $N \in \{10, 50, 100\}$ nodes and $d_i \in \{20, 100, 500\}$ KB) were executed 1000 times each.

5 Disaster Area Mapping using Computing Nodes with Load Balancing

5.1 Introduction

Here, we build upon our work in Chapter 4. We previously considered a system that generates the pedestrian map of a subsection of the disaster area and delivers the output to evacuees. Here, we consider how the system can generate pedestrian maps of the entire disaster area that consists of many subsections. We also consider how the generated maps can be delivered to the response commander. With the bigger overall area, larger amounts of trace data have to be collected and processed, and messages have to be transmitted across longer ranges.

To improve the system to address the larger scenario, we faced the following challenges: (1) how to collect the raw data and infer maps from them, (2) how to handle short-range communications between subsections and long-range communications across subsections, and (3) how to distribute the computing load of map inference and handle cases where some subsections have larger amounts of raw data to process.

To face the challenges, (1) we use the same approach in Chapter 4 and leverage responders walking in the area to collect the required map traces and use Computing Nodes to infer the map. For (2), we again use a DTN with Epidemic Routing to deliver data across shorter ranges within a subsection. To communicate across longer ranges, we leverage response team vehicles as data ferries to carry messages between subsections. Finally for (3), we add a load-balancing heuristic that uses ferry timetables and statistical information about the load of

Computing Nodes to distribute computing tasks.

In this chapter, we present the design of the system with the added data ferries and load balancing heuristic, and evaluate it using experiments and simulations. We show that load balancing performs better than "naive" cases where computing load is not distributed among computing nodes in extreme cases where one subsection is overloaded with raw trace data.

In the rest of the chapter, we present the specific assumptions in our target scenario, we define the system components, the workflow, and present our implementations and evaluations.

5.2 Assumptions

We assume the post-disaster scenario described in Chapter 3. The response teams are composed of on-foot responders, response vehicles, and the response team commander. The performance of DTN systems are dependent on the target scenario [2]. For a realistic representation, we assume that the mobility of on-foot responders follows the event-driven pattern of the PDM model [56]. On-foot responders start from their rally point and walk to a location in their subsection. At the location, they stay for a time duration to perform tasks. After, they return to their rally point and wait for new tasks.

The response commander (RC) is located at one of the rally points where he manages the actions of all response teams. The RC requires a map of the disaster area.

We assume that the response vehicles patrolling the area follow the PDM model's cyclic movement pattern. Response vehicles follow a route given by team leaders, the waypoints of which are the rally points of subsections. They stop at rally points to do their tasks, after which, they move to the next rally point. At times, response vehicles are given new orders that change their route.

Each responder and response vehicle has a smartphone with wireless interfaces (e.g. Wi-Fi or Wi-Fi Direct) and sufficient available storage (e.g. 16 GB). Also, each rally point has a commodity workstation as described in Section 3.3.

As part of their preparations, response vehicles have a timetable containing average travel times between any two rally points. The timetable is determined

during drills when responders test possible routes through the area. The smartphones of the response vehicles have an application that stores this timetable. Drivers input their routes in the application, which calculates the travel time between waypoints.

5.3 System Components

5.3.1 Sensor Nodes

To generate a digital pedestrian map, GPS trace data have to be collected. To do so, the system follows the crowd-sourcing approach of [8], where many pedestrians collect their GPS traces during short-term, city-scale events like festivals. Each on-foot responder is equipped with a smartphone with a logging application that collects trace data. Responder-smartphone pairs function as the system's Sensor Nodes (SNs).

5.3.2 Computing Nodes

In Chapter 4 and [53], we showed that without Cloud-based computing resources, inferring the map may take a long time, especially if only smartphones are used. To infer the pedestrian map from the collected data, the system uses Computing Nodes (CNs) [53]. CNs are the commodity computing devices that are part of the response team's inventory, which are deployed at each rally point. CNs run a map inference algorithm, using the trace data collected by SNs as input, to generate maps.

5.3.3 Communication Network

Without continuous end-to-end communication networks, the system establishes a DTN to share messages. The smartphones and CNs function as nodes of the DTN. The built-in wireless interfaces (e.g. Wi-Fi or Wi-Fi Direct) of the devices are used to establish contact.

The DTN has two stages of communication: short-range and long-range. Short-range communication occurs within a subsection, between responders belonging

to the same response team. In this stage, SNs have to send their collected trace data to their subsection's CNs for processing. SNs use their smartphone to establish contact with other nodes and forward collected data to the CN via Epidemic Routing [58].

Long-range communication occurs across subsections, between the CNs at different rally points. To forward messages across longer distances, the system leverages response team vehicles as data ferries [59].

5.3.4 Data Ferries

Response vehicles function as data ferries between CNs in different rally points, carrying messages to CNs along their routes. The system uses a forwarding heuristic based on the known route timetables, which is discussed in Section 5.8.2.

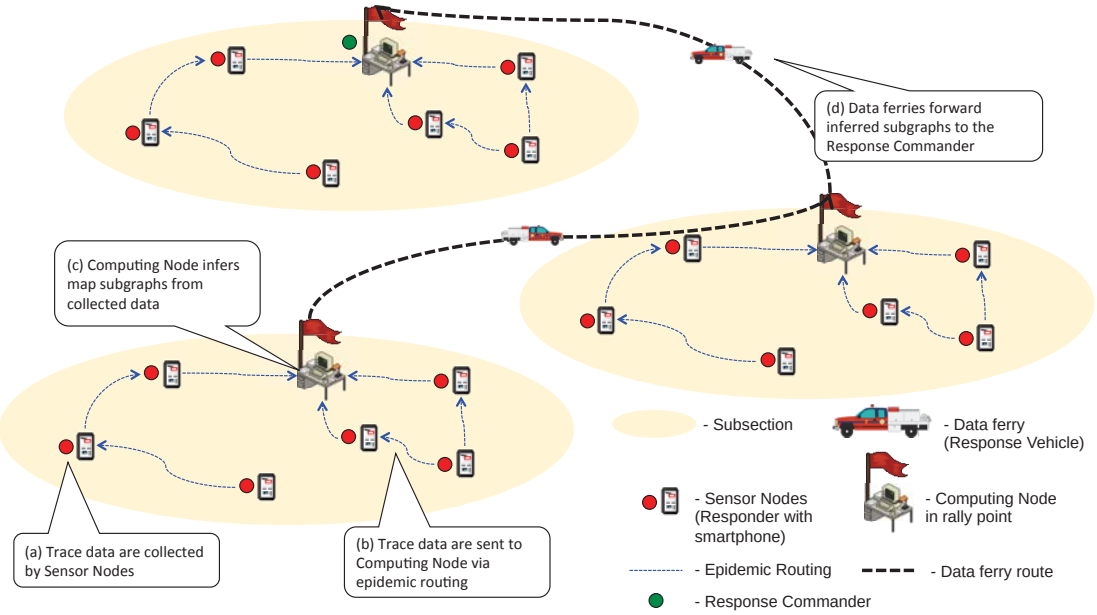


Figure 5.1: The system workflow. In each subsection, responders collect trace data while performing tasks (a). They send their collected trace data to the Computing Node in their rally point via epidemic routing (b). The Computing Nodes infer maps from received data (c), and outputs are forwarded to the RC by data ferries (d).

5.4 Map Inference and the Required Output

Map inference is the extraction of a graph that represents a road network from input raw GPS traces [6]. Our proposed system extracts a graph that shows the pedestrian paths in a disaster area. To do so, it requires trace trajectory data sets D to be collected from people while they walk in the area. D is a set of points p consisting of a sequence of values $\{lat, lon, e, v, t\}$, where lat and lon are the GPS-sourced location coordinates, e is the estimated positioning error, and v is the velocity at time t.

After receiving the input, the system uses an algorithm based in [8] and outputs a digital pedestrian map of the disaster area represented by a graph $\mathbb{M}(V, E)$. The set of edges E represents the pedestrian paths and the set of vertices V represents path intersections, corners, and terminal points. The edges in E have a weight w that represents the average walking speed along the path. $\mathbb{M}$ is composed of subgraphs m, each representing a piece of the map or a part of the disaster area. With such a weighted graph, least-cost path algorithms (e.g. Dijkstra's Algorithm) can be used to find the fastest routes.

5.5 System Workflow

The system's workflow is as follows: trace data collection, data delivery to Computing Nodes, pedestrian map inference, and map delivery. The workflow is shown in Figure 5.1 and all variable definitions are summarized in Table 5.1.

5.5.1 Trace Data Collection

The first step in the workflow is collecting trace data (Figure 5.1a). SNs (i.e. responders) use a smartphone application called DTN MapEx [52] while walking to record their GPS traces, estimated positioning error, movement speed, and the current timestamp. These correspond to the required values $\{lat, lon, e, v, t\}$ of p as described in Chapter 5.4. The collected trace data are stored in the smartphone's local database and exported as .CSV files. Trace data can be split to multiple .CSV files of uniform size. From basic testing of DTN MapEx, we found that 100 KB of GPS data corresponds to a 1 Km path. To improve the

Variable	Definition
D	Trace Data File
$\mathbb{M}(V, E)$	Disaster area pedestrian map
$m(V, E)$	Subgraph (i.e. a piece of the pedestrian map)
SN	Sensor Node
CN	Computing Node
RC	Response Commander
$CN_{\texttt{RC}}$	RC's Computing Node
i	Number of trace data files in CN queue
$p(D)$	Subgraph generation time function
D_{sz}	Size of D (KB)
CN_{sp}	CN map inference execution speed (KB/s)
$T_{\texttt{gen}}$	Generation time of D
$\Delta T_{\texttt{DTN}}$	DTN delivery latency of D to CN
$\Delta T_{\texttt{queue}}$	Time spent at CN queue
$\Delta T_{\texttt{ferry}}$	Ferry delivery latency of $m(V, E)$ to $CN_{\texttt{RC}}$
$\Delta T_{\texttt{total}}$	Total delivery time of subgraph $m(V, E)$
$\Delta T_{\texttt{ferryAlt}}$	Ferry travel time to the alternate CN
$\Delta T_{\texttt{queueAlt}}$	Time spent at alternate CN queue

Table 5.1: Variable definitions

quality of the output map, responders can continuously collect GPS data (i.e. do multiple passes along the same roads over the duration of operations). Having more recent data can show how the state of a path changes over time (e.g. how congestion decreases or increases during operations).

5.5.2 Trace Data Delivery to Computing Nodes

In the next step of the workflow, SNs send their collected trace data files to the CN in their subsection's rally point for processing (Figure 5.1b). The delivery latency of D, starting from the time it was generated ($T_{\texttt{gen}}$), from the SN to the CN is denoted as $\Delta T_{\texttt{DTN}}$.

To perform short-range communication within a subsection (i.e. to send trace

data from SNs to CNs), SNs use their smartphones as DTN nodes. Because of the movement of SNs, the topology of the network is constantly changing. To send trace data from SNs to CNs, the system uses Epidemic Routing [58]. While other DTN routing techniques such as PRoPHET [27] and MaxProp [9] exist, we chose Epidemic Routing because it has a high delivery ratio given that nodes in the network can handle the power consumption of high transmission rates and buffer congestion [49].

We assume that the system can handle the power consumption overhead. Recent smartphones can reach up to 13 hours battery life when their Wi-Fi interface is in use [12], which is enough for a half-day's response operations. Any smartphones that are running out of power can be recharged at rally points. However in future works, a power-saving mechanism can be added to a system such as a wake-sleep duty cycle.

We also assume that system can handle buffer congestion. Recent smartphones have built-in storage ranging from 16 GB to 128 GB. Even if a SN has only 5 GB of buffer space left, it can still store approximately 50,000 Km of trace data (e.g. a 100 KB .CSV contains approximately 1 Km of traces, sampled at 1 Hz). From these, we assume that the SNs' smartphones will have sufficient buffer size to store their generated messages and any replicas created by Epidemic Routing. However as a practical contingency, a purging mechanism (e.g. Time to Live) can be used to remove old messages.

We also note that, based on the evaluations in [30], smaller message sizes improves DTN performance. Thus, compressing the collected trace data into smaller file sizes can be a practical solution to reducing the transmission overhead and congestion in the DTN. As such, a trace file compression function can be added to the SNs' DTN MapEx application. The trace files can then be decompressed when they reach a CN. However, such functions imply additional computing load. Furthermore after decompression, the trace data still have to be processed for map inference. Because the focus of our system is distributing the computing load of map inference in a DTN paradigm, we leave compression as future work.

Node mobility significantly impacts a DTN's performance [46]. As mentioned, we use the PDM model's event-driven movement pattern as the mobility of on-foot responders. SN functions and mobility are described in Algorithm 2.

Algorithm 2 Sensor node functions

1: *sensor.spawnAtRallyPoint*
2: *sensor.moveToTaskLocation*
3: *sensor.PerformTask*
4: *sensor.returnToRallyPoint*
5: *sensor.waitForNewTask*
6: *sensor.connectToNetwork*
7: *sensor.forwardMessages*
8: *Return to 2*

5.5.3 Pedestrian Map Inference

The next step in the workflow is processing the collected trace data to infer a digital map. CNs execute an inference algorithm, such as the one in [8], using the collected trace data as input. Each CN has a processing queue, and it process trace data file D in a first-in, first-out order. When a new trace data file is received by a CN, it enters the tail-end of the CN's processing queue. The trace data file waits until the CN finishes all other files in the queue before getting processed.

When a trace data set D is processed, the output corresponds to a subgraph m of $\mathbb{M}(V, E)$. In a real scenario, the subgraphs represent the parts of the disaster area that were explored by the SN. The entire pedestrian map $\mathbb{M}$ is gradually updated as more trace data sets are collected and processed into subgraphs.

After a trace data set D is processed, the CN outputs the subgraph $m(V, E)$. The total time spent by D in the queue (i.e. its queuing time plus processing time), $\Delta T_{\texttt{queue}}$, is obtained by:

$$\Delta T_{\texttt{queue}} = \sum_{i=1}^{i} \Delta p(D_i) \tag{5.1}$$

where i is the number of trace data files in the queue (with D_1 being first), and $\Delta p(D)$ is the function that outputs the estimated time required for a CN, which has a map inference speed CN_{sp} (i.e. the kilobytes of trace data the CN can process per second), to process a trace data file D with size D_{sz} kilobytes:

$$\Delta p(D) = D_{sz}/CN_{sp} \tag{5.2}$$

D_{sz} can be set as a system parameter. For example, the data collecting application of SNs will automatically generate a trace data .CSV file after it collects D_{sz} kilobytes of data. CN_{sp} values may vary depending on the CN specifications. However, an estimate value can be obtained through repeated executions of the map inference algorithm on test trace data files of varying sizes, as we obtained in Chapter 4 and [53]. We also note that $\Delta p(D_1)$ may be shorter than $\Delta p(D_2)$ onward because the first data file in the queue may already be in the middle of processing when the latest data file arrives.

5.5.4 Subgraph Delivery

In Chapter 4 and in [53], we evaluated the delivery latency to evacuees in a subsection. In this study, as the next step in the workflow, the subgraphs are delivered to the response commander, RC, who needs them for situation awareness. The RC requires a map of the disaster area, which is composed of subgraphs from all subsections.

The RC is stationed at one rally point. The Computing Node at this rally point, $CN_{\texttt{RC}}$, is the destination of all generated subgraphs. Each subgraph $m(V, E)$ waits in its source CN until it is picked-up and delivered to $CN_{\texttt{RC}}$ by a data ferry. The total latency of $m(V, E)$ delivery to $CN_{\texttt{RC}}$ (i.e. its waiting time plus ferry delivery time) is denoted as $\Delta T_{\texttt{ferry}}$.

When the subgraph $m(V, E)$, reaches $CN_{\texttt{RC}}$, it is marked as delivered. The total delivery time of each subgraph (i.e. the Subgraph Delivery Time), from the time its trace data file D was generated by its source SN and is processed by a CN, until it is delivered to $CN_{\texttt{RC}}$ is denoted as $\Delta T_{\texttt{total}}$, and is given by:

$$\Delta T_{\texttt{total}} = \Delta T_{\texttt{DTN}} + \Delta T_{\texttt{queue}} + \Delta T_{\texttt{ferry}} \tag{5.3}$$

5.6 The Subgraph Inference and Delivery Problem

Because speed is critical in response operations, the pedestrian map has to be created in minimal time. The entire map is gradually created as subgraphs (i.e. the pieces of the map) are inferred and delivered to their destination. In this study, the system aims to deliver subgraphs to the RC. Thus, the goal is to minimize the time at which subgraphs are inferred and delivered to the RC, which we define as the Subgraph Inference and Delivery (SID) problem below.

Given a set of trace data files $\mathbb{D} = \{D_1, D_2, ..., D_n\}$ collected from a subsection and their corresponding set of generated subgraphs $\mathbb{M} = \{m_1, m_2, ..., m_n\}$, we aim to:

$$\text{Minimize} \quad \sum_{m \in \mathbb{M}} \Delta T_{\texttt{total}}(m) \tag{5.4}$$

where $\Delta T_{\texttt{total}}(m)$ of subgraph m, which includes the DTN latency, the time in the processing queue, and the ferry latency to the $CN_{\texttt{RC}}$, as defined in Equation 5.3.

Many studies have proposed methods for reducing DTN and data ferry message delivery latency [49] [7] [60] [59]. We use Epidemic Routing as a baseline method to obtain $\Delta T_{\texttt{DTN}}$. Ferry delivery time $\Delta T_{\texttt{ferry}}$ is dependent on the number of ferries available in the system.

Because there are many existing methods to reduce the DTN latency and ferry latency (i.e. $\Delta T_{\texttt{DTN}}$ and $\Delta T_{\texttt{ferry}}$), our system focuses on minimizing $\Delta T_{\texttt{total}}$ by reducing the delay in the processing queue, $\Delta T_{\texttt{queue}}$. As defined in Equation 5.1, bottlenecks can occur if the amount of trace data to be processed is high. In Chapter 4 and in [53], we focus on inferring maps of individual subsections. In this study, we consider the requirements of mapping the entire disaster area. Thus, instead of relying on a single computing device, the system spatially distributes processing tasks to the CNs in each subsection. Furthermore in cases when the amount of trace data generated within a single subsection is large, the system uses load balancing to offload tasks from a CN with a long queue to other CNs.

5.7 Challenges

To realize a system that solves the SID problem, we addressed the following challenges:

Given that the system relies on commodity workstations for map inference, the first challenge is how to generate maps from large amounts of collected trace data (e.g. when the disaster area and the distances covered by the SNs are large). The system must leverage the available CNs in the area to process the trace data.

The second challenge is that data have to be shared throughout the system. While short-range communication within a subsection can easily be solved using Epidemic Routing, long-range communication across subsections is a different challenge. As mentioned, the system uses data ferries for long-range communication. However, the data ferries follow routes that are not optimized for decreasing message delivery latency. Response vehicles (i.e. the ferries) follow routes based on orders given by response team leaders. While these routes further the goals of the response team, they are not always the fastest for data delivery. The system must have a forwarding method that works around this characteristic of the ferries.

Finally, in some scenarios, a large quantity of trace data may be generated from a single subsection. This may be caused by the subsection's response team having significantly more tasks to perform than other response teams and results in larger amounts of trace data collected in one subsection. With more trace data, bottlenecks that may occur in the CNs of these subsections, which the system has to handle.

5.8 Solutions

5.8.1 Distributed Map Inference

Without Cloud-based computing resources, the intuitive method of inferring the pedestrian map would be to use a simple client-server architecture. SNs (i.e. the clients) forward all collected trace data sets to a single, centralized workstation (i.e. the server) in one rally point (e.g. the location of the RC) for processing. This approach requires the least overhead in terms of equipment deployment.

However, it may not be able to process large amounts of trace data collected by SNs.

To handle such cases, our system distributes map inference tasks among multiple CNs. The system does this by leveraging the spatial distribution of response teams that is consequently created by response operation protocols. During operations, each response team is assigned to cover a subsection of the disaster area, and each response team establishes a rally point with a CN. Given this, the system uses the following heuristic to distribute map inference tasks:

All trace data files collected by SNs belonging to the same response team are forwarded to and processed by the CN in that response team's rally point.

Typical distributed architectures require the current states of nodes to control task distribution. In a disaster scenario where end-to-end communication networks are not available, fast status updates cannot be obtained. Using this heuristic divides processing without communication overhead.

5.8.2 Long-Range Communication

Algorithm 3 Data ferry behavior and functions

```
1: ferry.spawnAtRallyPoint
2: ferry.receiveNewRoute
3: ferry.startMoving
4: if ferry.arrived then
5:     ferry.stopMoving
6: end if
7: if ferry.hasNewOrders then
8:     ferry.receiveNewRoute
9: end if
10: if rallyPoint.isLastStop then
11:     ferry.receiveNewRoute
12: end if
13: ferry.startContact
14: ferry.stopContact
15: Return to 2
```

The system delivers data across long distances, between subsections (e.g. send subgraphs to the $CN_{\texttt{RC}}$) by leveraging response vehicles as data ferries. However, unlike typical data ferries that have routes designed to reduce delivery latency [7] [60], response vehicles routes are controlled by response team leaders. The system requires a method that works around routes that are not optimized for message delivery. To achieve this, the system uses a forwarding heuristic based on information about the routes of response vehicles:

Given known ferry routes and timetables, the system forwards messages to the fastest ferry to the destination.

The details of the forwarding heuristic are described in Algorithm 4 and are discussed below:

1. When a ferry arrives at a rally point, it stops for a `SERVICE_TIME` duration. `SERVICE_TIME` is an abstraction during which the ferry exchanges messages with the rally point's CN. In a real scenario, the response vehicle operators execute their leader's orders during `SERVICE_TIME`.

2. During `SERVICE_TIME`, if the ferry receives a new route, it updates the route in its smartphone.

3. After the ferry updates its route, the ferry switches to the `CONTACT` state, during which it purposefully establishes a connection with the rally point's CN.

4. When a new ferry joins the network, all nodes (i.e. other ferries and the CN) in the network loop through their messages. Messages that are addressed to the current CN are forwarded to it and marked as delivered.

5. For messages that are not addressed to the current CN, ferries compare their route timetables. The undelivered messages are forwarded to the ferry with the least travel time, $\Delta T_{\texttt{travel}}$, to the destination. $T_{\texttt{travel}}$ is calculated by:

$$\Delta T_{\texttt{travel}} = \texttt{SERVICE\_TIME\_LEFT}$$
$$+(j-1)(\texttt{SERVICE\_TIME}) + \sum_{j=1}^{j} \Delta t(j)$$

where j is the number of rally points the ferry passes until the destination, $\Delta t(j)$ is the travel time from the current rally point to the next in its route, and `SERVICE_TIME_LEFT` is the remaining service time at the current CN. A `SERVICE_TIME` is added at each stop prior to the destination, hence $j-1$.

6. After the `SERVICE_TIME_LEFT`, the ferry ends the `CONTACT` state and switches to the `MOVING` state, and drives to the next rally point in its route.

Algorithm 4 Message forwarding during contact

```
for ∀ node : network.getNodes do
  for ∀ message : node.getMessages do
    if message.isAtDestinationCN then
      CN ← message
    else
      fastestferry=fastestFerryTo(message.destination)
      fastestferry ← message
    end if
  end for
  for ∀ message : node.getMessages do
    if !node.pass(message.destination) then
      CN ← message
    end if
  end for
end for
```

5.8.3 Load balancing

While the solution in Section 5.8.1 divides map inference tasks spatially, it is possible to encounter cases where SNs in one subsection generates significantly larger amounts of trace data than those from other subsections. To handle such cases, the system uses a load balancing heuristic based on the round-robin method. The round-robin load balancing method was chosen because of its low complexity and it does not require continuous communication between nodes [33] [25] [34]. In a

system that relies on DTN, node communication latency is significant, and load balancing methods that require real-time node information are not feasible. Load balancing is done as follows:

1. The CNs in the system are first divided into fixed clusters. As part of its preparations prior to deployment, the response team gives the cluster assignments (i.e. which CNs belong to the same cluster). The clusters can be statically assigned because there is not a large quantity of CNs and all of them are stationary.

2. Each CN maintains a table with three columns: (1) the `ID`s of all CNs, (2) their corresponding average queue length, `AVG_QUEUE`, and (3) the `TIMESTAMP` of the latest known value of (2). The values for the first column are input prior to deployment (i.e. static clusters). The initial values of `AVG_QUEUE` are set to 0. All `AVG_QUEUE` are rounded up, to the nearest integer.

   Each CN constantly obtains and updates its own `AVG_QUEUE` value by counting the number of trace data files it has queued at the current timestamp and getting the mean of all counts it has done. After calculating the mean, the CN updates its own `AVG_QUEUE` value and its corresponding `TIMESTAMP`.

   Each data ferry also maintains the same three-column table. When a data ferry enters the `CONTACT` state with a CN, they exchange `AVG_QUEUE` values with more recent `TIMESTAMP`s.

3. Each CN also stores a round-robin table that contains all other CNs in its cluster. This table is created prior to deployment, based on the fixed cluster assignments.

4. When a trace data file D from an SN arrives at a CN, it enters the processing queue of the CN but is also assigned an alternate CN. The alternate CN is selected in round-robin order from the table in 3.

5. When a ferry enters $CONTACT$ with a CN, it loops through the trace data files in the CN's queue. If the ferry will pass a message's alternate CN, the system decides whether to forward the trace data file to the ferry or not.

The decision is made by comparing the trace data file's queuing time, $\Delta T_{\texttt{queue}}$ at the current CN with ferry's travel time $\Delta T_{\texttt{ferryAlt}}$ to the alternate CN and the estimated queuing time at the alternate, $\Delta T_{\texttt{queueAlt}}$. $\Delta T_{\texttt{queueAlt}}$ is estimated by using the `AVG_QUEUE` value of the alternate CN as the value of i in Equation 5.1.

$$\Delta T_{\texttt{queue}} > \Delta T_{\texttt{ferryAlt}} + \Delta T_{\texttt{queueAlt}} \tag{5.5}$$

If Equation 5.5 is true, then the trace data file is forwarded to the ferry, otherwise it remains in the current CN's queue.

Whenever the `AVG_QUEUE` values are updated (i.e. when a new ferry with more recent values enters contact), the system again checks Equation 5.5 for each message. If the result becomes false, the message is left at the CN.

6. When a trace data file arrives at its alternate CN, it enters the processing queue and is given a new alternate, based on the CN's round-robin table.

5.9 Implementation

5.9.1 Trace Data Collection with DTN MapEx

We implemented a trace logging application for Android devices called DTN MapEx [52]. Responders run DTN MapEx to record their GPS traces, estimated positioning error (i.e. the location accuracy provided by the Location API [3]), movement speed, and the current timestamp. DTN MapeEx writes the collected trace data to .CSV files. Each trace data file contains a piece of the pedestrian map (i.e. when processed, it outputs a subgraph. Figure 5.2 shows screenshots of DTN MapEx.

Optimizing GPS trace logging (e.g. sampling frequency, compensating for inaccuracies) has been addressed by many, including [28] and [61], and is out of the scope of this study. The default sampling frequency of DTN MapEx is 1 Hz, which is enough for 1.5 m/s walking speeds. For other map inference applications that require high density trace data (e.g. vehicular road map inference with

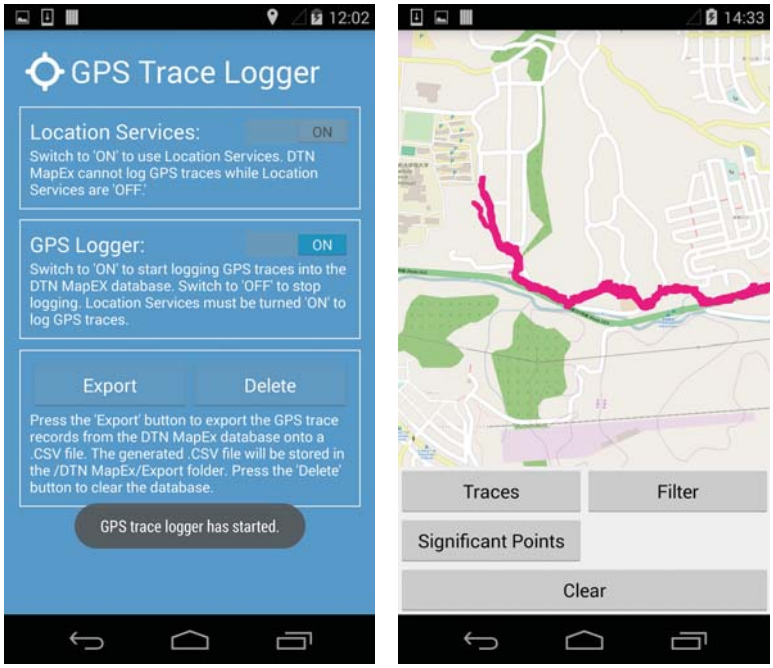


Figure 5.2: The DTN MapEx application's GPS Trace Logger (*left*) and sample trace data overlaid on precached map tiles (*right*).

fast-moving cars, high-accuracy mapping), smartphones with hardware capable of high GPS sampling frequencies or interpolation can be used [44]. These however, can result in larger trace data sizes. We also note that using GPS services quickens battery drain. In such cases, smartphones with low battery can be charged at a rally point. However, power is an important resource during disasters, and in future works we will consider power-efficient GPS sampling.

5.9.2 Computing Node Map Inference Application

We developed a Java program that executes a map inference algorithm based on the work of Blanke et al [8]. We modified the algorithm to add weights to the edges of the output graph, which correspond to the average of the walking speeds of all underlying points of the edge (i.e. the points used to infer the edge). The program loads trace data .CSV files then runs the map inference algorithm and generates a subgraph. The edges and vertices of the subgraph can then be exported to .CSV files for sharing. The Java program is shown in Figure 5.3

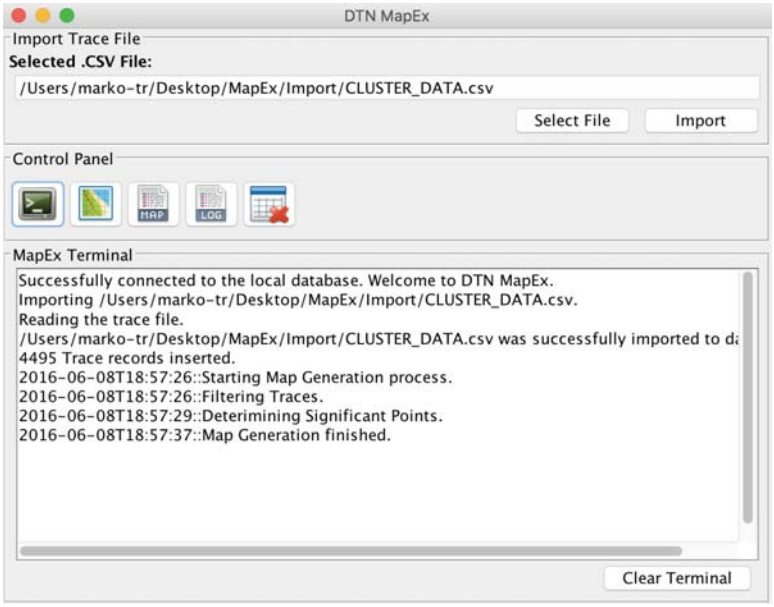


Figure 5.3: The Computing Node Java application accepts .CSV files containing trace data sets and executes a map inference algorithm to generate subgraphs.

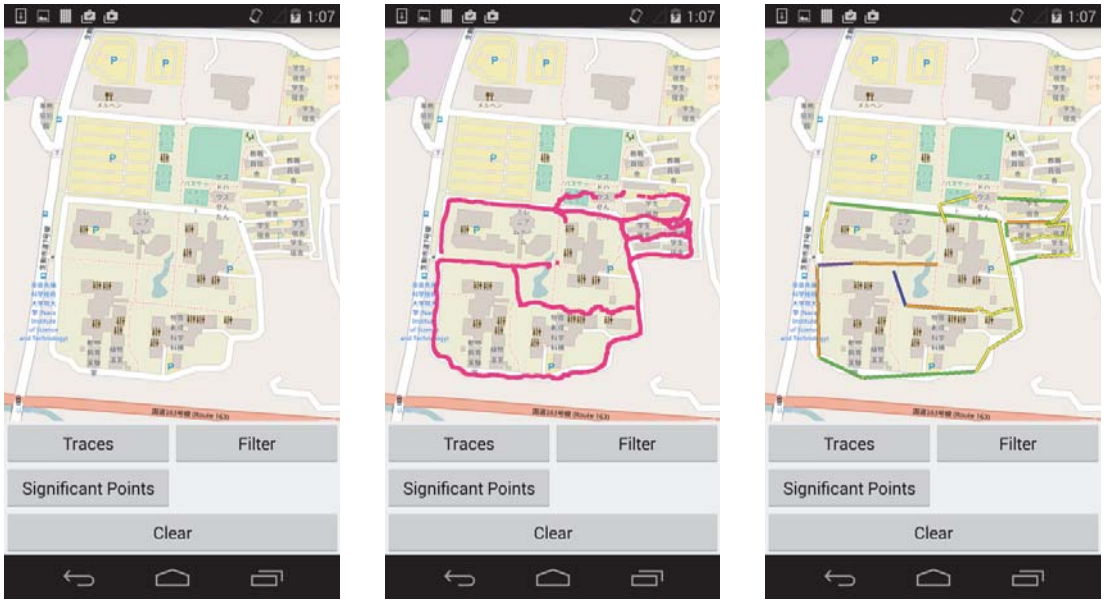
and an example output map composed of subgraphs is shown in Figure 5.4. The subgraphs are overlaid on offline map tiles and show the color-coded walking speeds along the explored paths.

5.9.3 DTN Module

To establish a DTN and share trace data and generated map files between nodes, we added a DTN module (shown in Figure 5.5) to DTN MapEx that uses the IBR-DTN for Android [43] and a modified version of the ShareBox application [20]. The DTN Module uses either Wi-Fi Direct or Wi-Fi to establish contact.

5.10 Evaluating the Load-Balancing Heuristic

The goal of these evaluations is to compare the performance of the system's load balancing heuristic to the intuitive case of using only a single CN to handle all subrgraph inference tasks and to a case where multiple CNs infer the subgraphs, but do not use any load-balancing. We use the total subgraph delivery time



(a) (b) (c)

Figure 5.4: (a) Map tiles/background images. (b) Raw GPS trace data. (c) Output map composed of subgraphs. The output subgraphs are overlaid on the map tiles. Edges of the subgraphs are color-coded based on their average walking speeds (i.e. cooler colors mean slower speeds, warmer colors mean faster speeds).

$\Delta T_{\texttt{total}}$ as the performance metric to show how fast subgraphs are inferred and delivered for each case.

5.10.1 Disaster Area

The target disaster area, shown in Figure 5.6, is a 3 Km x 3 Km area in Marikina City in the Philippines, which is often hit by natural disasters. As prescribed by the guidelines in [22], the area is divided into nine, 1 Km x 1 Km subsections. A response team, consisting of 100 responders, is deployed at each subsection. Each responder has a smartphone with 16 GB of storage space and has the DTN MapEx application. The number of responders depends on the available manpower in the disaster area. The Marikina City Disaster Coordination Council has a total manpower of approximately 5,000 people. Approximately 2,700 of whom are public order and safety personnel and volunteer responders [29]. The city has

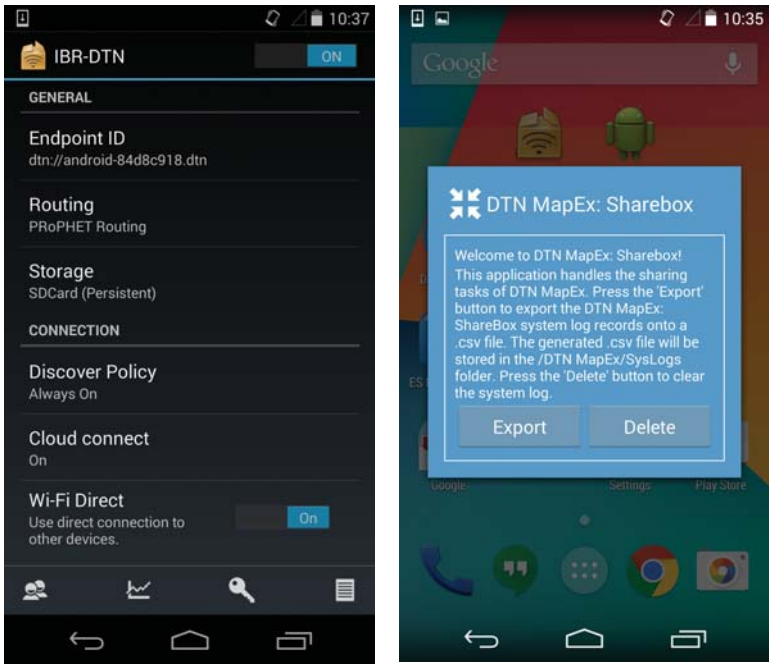


Figure 5.5: The IBR-DTN daemon (*left*) and the DTN Module (*right*).

a total area of 22 Km^2, making 100 moving responders per square-kilometer a reasonable assumption. Each response team has a rally point in a school, medical facility, or parish building in their subarea. Each rally point has a laptop that functions as a CN. The CN has a processor speed of 2.6 GHz (with two cores), 8 GB of memory, and 256 GB of storage space. The rally points (and their corresponding CNs) were given ID numbers from 1 to 9.

5.10.2 Simulation Model and Parameters

We modeled the disaster area as a complete graph $A(V, E)$ with a set of vertices V that represents the locations of rally points, and a set of edges E that represents the response vehicle path between rally points. Each edge has a weight corresponding to the travel time $\Delta T_{\texttt{ferry}}$ between rally points.

The model requires the following: the DTN delivery latency of trace data files $\Delta T_{\texttt{DTN}}$ from SNs to CNs, which will model how fast trace data files enter the queue of CNs; the edge weights $\Delta T_{\texttt{ferry}}$, which will model how fast ferries move between rally points; the map inference speed CN_{sp} of CNs; and the

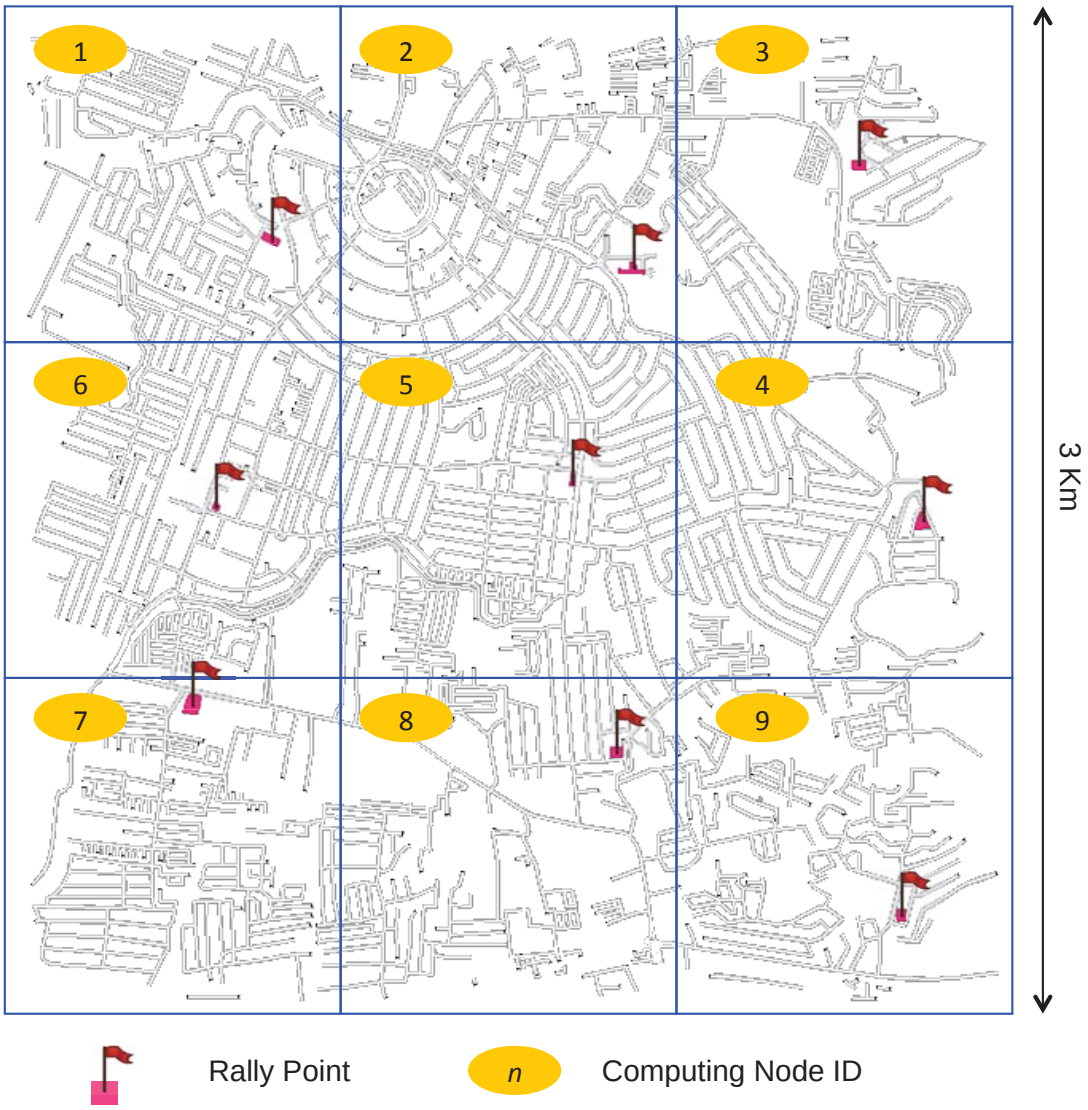


Figure 5.6: The disaster area in Marikina City, Philippines. The locations of Rally Points and the IDs of their corresponding Computing Nodes are shown.

`SERVICE_TIME`, which models the duration of ferries' `CONTACT` state at rally points.

We used the Scenargie Simulator [51] and the subsections in Figure 5.6 to generate end-to-end delivery latency traces of a DTN using Epidemic Routing for short-range communication in a subsection. We assume a sufficiently large storage buffer size of 16 GB for the SNs, thus the system can support Epidemic

Figure 5.7: The CN clustering for Decentralized + Load Balancing.

Routing without performance decrease due to congestion. The nodes of the DTN represent SNs and they follow the mobility pattern described in Algorithm 2. We set the wait duration of SNs in their rally points and in their task locations to random values from 20 to 30 minutes. In a 1 Km x 1 Km subsection, 100 SNs move at 1.5 m/s walking speed and send messages, representing .CSV data files, to a CN. The output traces were used in the model as $\Delta T_{\texttt{DTN}}$.

We again used the Scenargie Simulator and the map in Figure 5.6 to obtain the travel time $\Delta T_{\texttt{ferry}}$ (in seconds) of a vehicle moving at 60 Kph between every rally point pair. The values represent the time a response vehicle needs to travel between rally points. The $\Delta T_{\texttt{ferry}}$ values are shown in Table 5.2.

Getting an exact value of CN_{sp} is difficult because of the uniqueness of comput-

ID	1	2	3	4	5	6	7	8	9
1	0	178	314	346	193	122	236	310	492
2	178	0	145	226	165	275	411	281	483
3	314	145	0	365	342	431	605	444	631
4	346	226	365	0	179	316	446	258	438
5	193	165	342	179	0	171	326	127	317
6	122	275	431	316	171	0	144	250	430
7	236	411	605	446	326	144	0	238	369
8	310	281	444	258	127	250	238	0	209
9	492	483	631	438	317	430	369	209	0

Table 5.2: Travel time ($\Delta T_{\texttt{ferry}}$) in seconds between two rally points with the corresponding IDs.

ing devices (e.g. specifications, resource usage of OS and background programs). In Chapter 4 and [53], we approximated the map inference performance of a commodity laptop CN (2.6 GHz processor with two cores, 8 GB memory, and 256 GB of storage) through multiple executions of the inference algorithm using input trace data files of different sizes. We measured $CN_{sp} \approx 45$ KB/s, which we use in this study.

In the model, contact between a CN and ferries occurs at the vertices (i.e. rally points) and lasts for a `SERVICE_TIME` duration, during which ferries purposefully joins the CN's network. In [52], we experimentally evaluated how long digital files of different sizes can be forwarded between different smartphones using the DTN MapEx application and the Wi-Fi Direct interface. We found that it requires approximately 3 seconds to forward a 1-MB trace data file. We use this experimental result in our model, and set `SERVICE_TIME` to 5 minutes (300 seconds), which allows forwarding of 300 MB (i.e. approximately 1000 Km-worth of trace data, sampled at 1 Hz). This is ample for ferries to exchange messages.

To represent ferry mobility described in Section 5.8.2 in our simulation model, the waypoints of ferry routes are generated randomly, and are non-repeating (i.e. a ferry visits a rally point only once for every new route). Finding the actual probability of a ferry receiving new routes at rally points is a difficult problem and is out of the scope of this study. Thus, we assumed that ferries have a 2%

chance of receiving new routes at rally points.

We note that there are cases when a ferry ends `CONTACT` seconds before a new ferry joins the network. In such cases, we assume that the leaving ferry can extend its `SERVICE_TIME` for possible message exchange with the new ferry. The extension only lasts a few seconds to a few minutes (i.e. < 5 minutes), thus we consider it negligible.

5.10.3 Cases

We used $\Delta T_{\texttt{total}}$ as a metric to evaluate performance. As defined in the SID problem in Section 5.6, we aim to minimize the Subgraph Delivery Time ($\Delta T_{\texttt{total}}$). Lower $\Delta T_{\texttt{total}}$ values are better because they indicate that subgraphs are generated and delivered to the RC faster. We executed simulations to evaluate $\Delta T_{\texttt{total}}$ of the following cases:

1. Centralized: This models a case without the proposed system. We use this case as a point of comparison because it represents the intuitive solution of approach a single computing machine as a server that executes map inference by itself. Instead of having a CN at each rally point, only a central CN at the RC's rally point executes the map inference algorithm and all other rally points have a relay node. SNs forward their collected trace data files D to the relay node in the rally point. Ferries pick up the trace data files from relay nodes and deliver them to the CN_{RC} for processing. We evaluate this case to show the effects of not having a system that considers the processing requirements of pedestrian map inference.

2. Decentralized: This models the case described in Section 5.8.1 where collected trace data are spatially distributed. The rally point of each subsection has a CN that processes the trace data D collected by the SNs. Then, subgraphs are forwarded by data ferries to the CN_{RC}.

3. Decentralized + Load Balancing: This models a case with the system and the load balancing heuristic in Section 5.8.3. Here, the response team of one subsection collected significantly larger amounts of data than those in other subsections.

5.10.4 Simulation Sets

(a) Common Parameters

Parameter	Value(s)
Simulation Duration	10 hours
`SERVICE_TIME`	300 seconds
CN_{sp}	$\approx$ 45 KB/s
$\Delta T_{\texttt{DTN}}$	Scenargie Simulator trace
$\Delta T_{\texttt{ferry}}$	See Table 5.2

(b) Scenargie Simulator Parameters

Parameter	Value(s)
Subsection Area	1 Km x 1 Km
SN Density/Subsection	100
SN Walking Speed	1.5 m/s
SN Wireless Com.	802.11ac
SN Buffer Size	16 GB
Data Ferry Speed	60 Km/h

(c) Simulation Set I

Parameter	Value(s)
Ferry density	5, 10, 15, 20, 25
Trace data file size	500 KB
Files generated per Subsection	300 to 400
CN_{RC}	CN 5
Simulation Runs	10 Runs

(d) Simulation Set II

Parameter	Value(s)
Ferry density	10
Trace data file size	125, 250, 500, 750, 1000 KB
Files generated per Subsection	300 to 400
CN_{RC}	CN 5
Simulation Runs	10 Runs

(e) Simulation Set III

Parameter	Value(s)
Ferry density	10
Trace data file size	500 KB
Files generated in Subsection 1	1,500, 3,000, 6,000
Files generated in other Subsections	300
CN Clusters	{1,2,3}; {4,5,6}; {7,8,9}
CN_{RC}	CN 5
Simulation Runs	10 Runs
CN Avg. Queue Length Update	1 Hz

Table 5.3: The simulation parameters used for evaluations.

In our first set of simulations (Set I), we compared the performances of Centralized and Decentralized to show how the system decreases Subgraph Delivery Time. In this set, we varied the ferry density. Because our system relies on ferries to deliver messages across rally points, we simulated scenarios with 5, 10, 15, 20, and 25 ferries. In an actual disaster, these simulations show how our system will perform if the response team has access to only a few or to many response vehicles. We maintained the trace data file size at 500 KB, which approximately contains 5 Km of traces. Each SN generates 3 to 4 files within the 10-hour simulation time, representing them walking approximately 15 to 20 Km throughout the simulation duration of 10 hours. The data files arrive at the rally point after a DTN delivery latency duration based on the traces generated using the Scenargie simulator. 10 runs were executed for each ferry density value for the Centralized and Decentralized cases.

In our second set of simulations (Set II), we compared the performances of Centralized and Decentralized to show how the system handles increasing size of trace data files. In this set, we vary the trace data size D_{sz} to evaluate how the system performs when response teams have to cover more ground in the disaster area or are collecting trace data at higher sampling rates. We simulated scenarios with trace data file sizes of 125, 250, 500, 750, and 1000 KB. The smaller data sizes represent scenarios where SNs either walk shorter distances or are using devices with lower sampling frequencies, while the larger data sizes represent scenarios where longer distances are walked or higher GPS sampling rates were used. We maintained the number of data ferries to 10 (i.e. each of the 9 response teams has one vehicle, plus one additional vehicle). Again, each SN generates 3 to 4 files within the 10-hour simulation time. The data files arrive at the rally point after a DTN delivery latency duration based on the traces generated using the Scenargie simulator. 10 simulation runs were executed for each D_{sz} value and both Centralized and Decentralized cases.

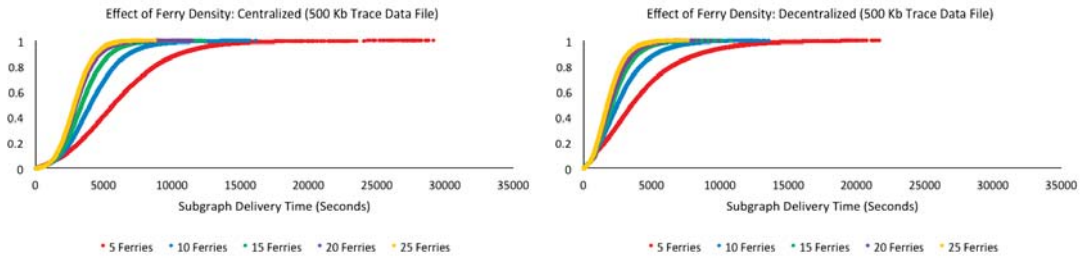
In our third set of simulations (Set III), we compared the performances of Centralized, Decentralized, and Decentralized + Load Balancing to show how the system, using the load balancing heuristic, handles the case where one subsection is generating significantly more trace data files than others. This represents one response team moving more frequently than others because there are more tasks

to be done in their subsection and/or are using high trace sampling frequencies. In this set, we increase the amount of trace data files generated in a single subsection. Subsection 1 generates 1,500, 3,000, and 6,000 trace data files, while other subsections were controlled to generate 300 trace data files each. We set the number of data ferries to 10 and the trace data file size to 500 KB. 10 simulation runs were executed for each trace data file density value and the Centralized, Decentralized, and Decentralized + Load Balancing cases.

The parameters for simulations are listed in Table 5.3. In all simulations, we assign CN number 5 (i.e. the center) as the CN_{RC}. The CN clusters for the Decentralized + Load Balancing case are shown in Figure 5.7 and listed in Table 5.3e.

5.11 Results and Discussion

5.11.1 Simulation Set I: Effect of Ferry Density



(a) Centralized Case

(b) Decentralized Case

Figure 5.8: CDF curves showing the effect of ferry density on the subgraph delivery times of the (a) Centralized and (b) Decentralized cases.

Figures 5.8a and 5.8b show the subgraph delivery time ($\Delta T_{\texttt{total}}$) cumulative distribution function (CDF) curves of the Centralized and Decentralized cases for various data ferry densities, for $D_{sz} = 500$ KB. For both cases, as data ferry density increases from 5 to 25, the $\Delta T_{\texttt{total}}$ decreases. The CDFs also show expected diminishing returns in terms of $\Delta T_{\texttt{total}}$ as ferry density increases (i.e. the travel time between source and destination is the limit). The improvement from

5 ferries to 10 ferries is larger than that from 15 ferries to 25 ferries.

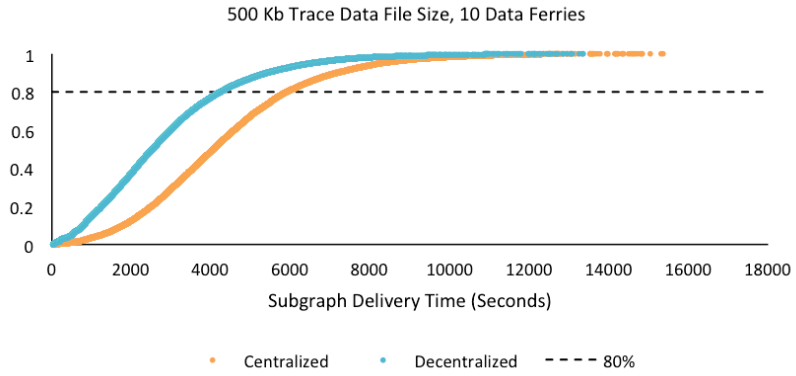


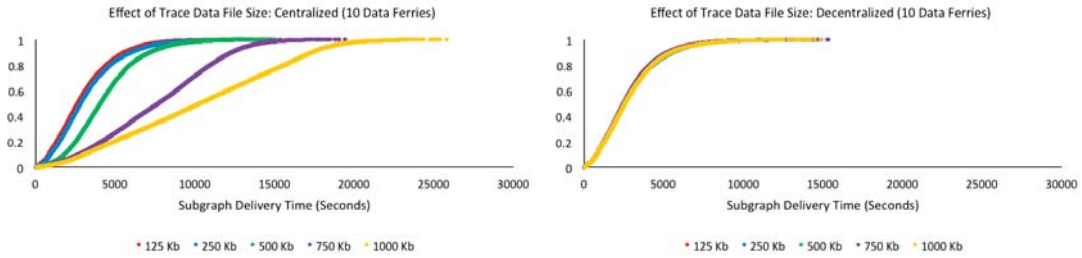
Figure 5.9: CDF Comparison of the Centralized and Decentralized cases when D_{sz} = 500 KB and Ferry Density = 10.

Figure 5.9 shows the $\Delta T_{\texttt{total}}$ CDF curves of the Centralized and the Decentralized cases, while D_{sz} = 500 KB. In the Centralized case, 5,895 seconds ($\approx$ 100 minutes) had elapsed before 80% of the subgraphs were delivered to the CN_{RC}. In the decentralized case, only 4,230 seconds ($\approx$ 70 minutes) had elapsed before 80% of the subgraphs were delivered. Given a constant D_{sz} and the same ferry density, the 30-minute improvement can be attributed to a reduced $\Delta T_{\texttt{queue}}$ in the Decentralized case.

5.11.2 Simulation Set II: Effect of Trace Data File Size

The results in Section 5.11.1 are supported by the results of Simulation Set II. Figures 5.10a and 5.10b show the $\Delta T_{\texttt{total}}$ CDF curves of the Centralized and Decentralized cases for various D_{sz} values, while data ferry density was maintained at 10 ferries. Figure 5.10a shows that as D_{sz} increases, the $\Delta T_{\texttt{total}}$ values of the Centralized case increases. Given the constant ferry density, this indicates a longer $\Delta T_{\texttt{queue}}$ duration at the CN_{RC}.

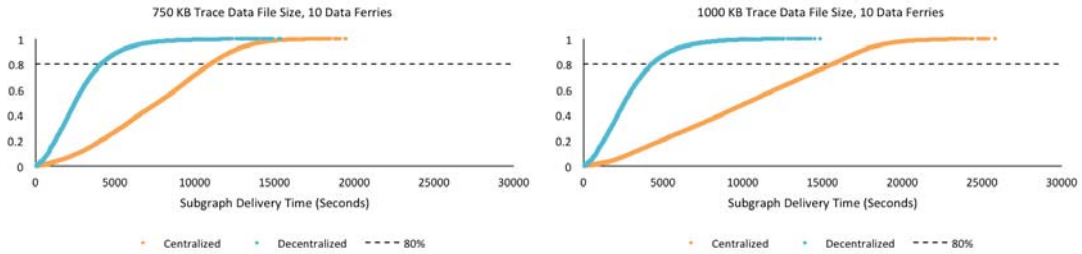
In contrast, Figure 5.10b shows that the increase in D_{sz} had negligible effect on the $\Delta T_{\texttt{total}}$ values of the Decentralized case. This indicates that dividing the processing tasks among the CNs reduces the effect of increasing D_{sz}. Because



(a) Centralized Case (b) Decentralized Case

Figure 5.10: CDF curves showing the effect of trace data file size on the subgraph delivery times of the (a) Centralized and (b) Decentralized cases.

more CNs were processing the trace data files, the queuing time for each file was shorter, even if its size was larger.



(a) $D_{sz} = 750$ KB (b) $D_{sz} = 1000$ KB

Figure 5.11: CDF Comparison of the Centralized and Decentralized cases when (a) $D_{sz} = 750$ KB and (b) $D_{sz} = 1000$ KB, while Ferry Density = 10.

Figures 5.11a and 5.11b show the differences between the Centralized and Decentralized cases for D_{sz} values of 750 KB and 1000 KB, respectively. These simulations model a scenario in which SNs collect larger amounts of trace data either by walking more or using higher GPS sampling rates. For $D_{sz} = 750$ KB, 10,956 seconds ($\approx$ 180 minutes) had elapsed before 80% of the subgraphs were delivered to the CN_{RC} in the Centralized case, while only 4,103 seconds ($\approx$ 70 minutes) had passed before 80% of the subgraphs were delivered in the Decentralized case. For $D_{sz} = 1000$ KB, 15,591 seconds ($\approx$ 260 minutes) had elapsed before 80% of the subgraphs were delivered to the CN_{RC} in the Centralized case,

while only 4,267 seconds (still ≈ 70 minutes) had passed before 80% of the subgraphs were delivered in the Decentralized case. The results show that when the system is used (i.e. the Decentralized case), the reduction in $\Delta T_{\texttt{total}}$ increases as D_{sz} increases.

5.11.3 Simulation Set III: Effect of Trace Data File Density

Here, we model a scenario where one response team generates more trace data files than others. Figures 5.12a, 5.12b, and 5.12c show the $\Delta T_{\texttt{total}}$ CDF curves of the Centralized, Decentralized, and Decentralized + Load Balancing cases for increasing trace data file densities in Subsection I, while data ferry density was maintained at 10 ferries, D_{sz} was maintained at 500 KB, and the trace data file density of other subsections was maintained at 300 files. Results show that as the trace data file density increases, the $\Delta T_{\texttt{total}}$ values of all three cases also increase. This increase is caused by the $\Delta T_{\texttt{queue}}$ of the files in Subsection 1 (i.e. where file density was increased).

Figure 5.13 shows that the Decentralized + Load Balancing case outperforms the Decentralized and Centralized cases. In an extreme case when 6,000 files were generated at Subsection 1, 20,513 seconds (≈ 340 minutes) had already elapsed before 80% of the subgraphs were delivered to the CN_{RC} in the Centralized case and it took 14,481 seconds (≈ 240 minutes) to deliver 80% of the subgraphs in the Decentralized case. In contrast, only 6,744 seconds (≈ 110 minutes) had elapsed before 80% of the subgraphs were delivered to the CN_{RC} in the Decentralized + Load Balancing case. This is due to the trace data files being distributed from the CN in Subsection 1 to the other CNs belonging to its cluster (i.e. in Subsections 2 and 3).

We note that our system uses a heuristic to balance computing load. While results show that our system reduces the time required to generate and deliver the output subgraphs, the load balancing is not perfectly "fair" (i.e. the queues in all Computing Nodes are not always equal).

To achieve the optimal situation of realizing a perfectly fair system, two main components are required: (1) a master node that fairly allocates tasks to machines

(i.e. Computing Nodes) in the system and (2) the real-time state of the whole system (i.e. based on constant updates from the network nodes, Computing Nodes) so that the master can decide where to allocate tasks.

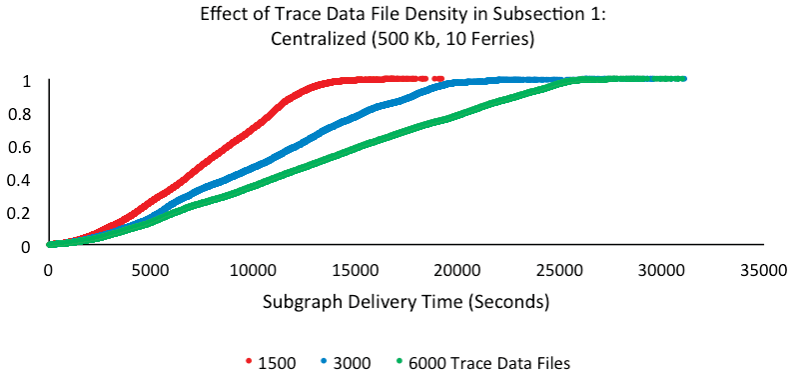
However, realizing such an optimally fair system given our assumed disaster scenario is very difficult because of the following reasons. First, even if a master is available, it cannot have the real-time state of the system because there is a large network latency (i.e. DTN delay). Second, the topology and state of the DTN is highly variable. Ideally, the master should know when all contacts of all DTN nodes will occur so that tasks can find the optimal route to the the optimal Computing Node. The constantly-changing topology of DTNs (i.e. nodes are mobile) make such accurate "prediction" impossible to realize. Finally, even if the real-time state of the system is known, the delay in task delivery is also a problem. Even if the master can decide allocation based on what it knows about the system's current state, the delay in task delivery means that the state has likely changed by the time the task reaches its destination.

Additionally, the trade-offs entailed by trying to achieve the optimal system should also be considered. In our evaluations, we show how our proposed system can improve map generation by approximately two hours. Even if the optimal system can further reduce the required time, it will most likely require additional resources (e.g. more computing devices or a better network), which increase implementation and power-consumption overhead. If the optimal system can further reduce the required time by 20%, but requires 100% more resources, then using our heuristic method may be a sufficient alternative.

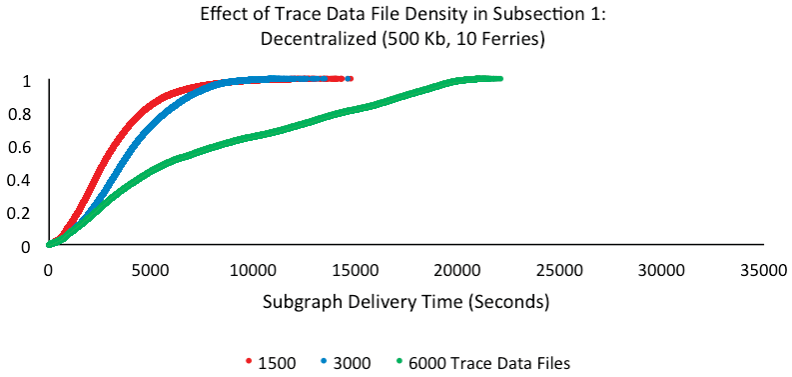
It is possible to simulate the ideal scenario by first generating a trace of the trajectories and contacts of the system's DTN nodes. These traces will can then be used by the master to "predict" the current and future states of the network. Assuming that the real-time system states of all Computing Nodes are always known by the master, whenever a new task enters the system, the master can use a "brute force" approach to determine the optimal allocation of the task. However, such a simulation is out of the scope of this study and is left for future work.

5.12 Summary

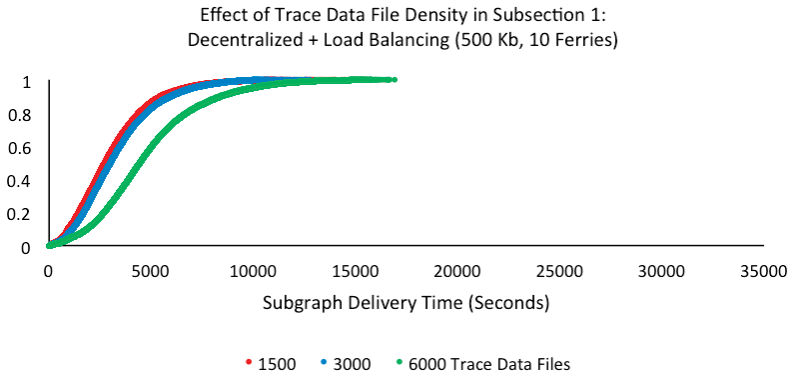
In summary, our results in this chapter show that spatially distributing map inference tasks to CNs significantly decreases $\Delta T_{\texttt{total}}$ compared to a scenario where all trace data are processed by a centralized workstation (e.g. approximately a three-hour improvement in the case of Figure 5.11b). Also, our load balancing algorithm improves performance when individual CNs have a long queue of trace data files to process (Figure 5.13). With the overall reduction of $\Delta T_{\texttt{total}}$, results show that the system is capable of solving the SID problem.



(a) Centralized Case



(b) Decentralized Case



(c) Decentralized + Load Balancing Case

Figure 5.12: CDF curves showing the effect of trace data file density on the subgraph delivery times of the (a) Centralized, (b) Decentralized, and (c) Decentralized + Load Balancing cases.

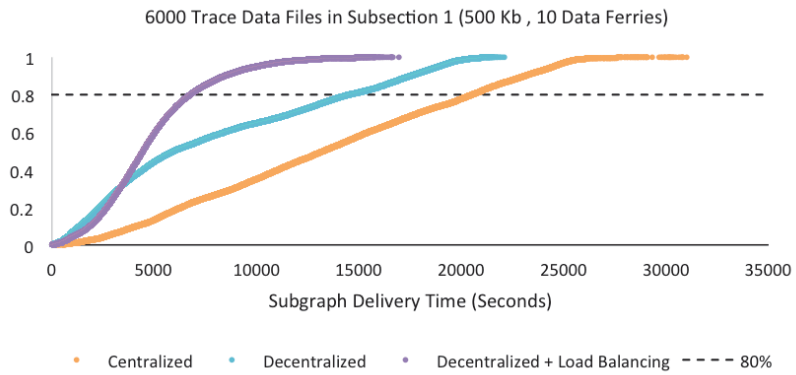


Figure 5.13: CDF Comparison of Centralized, Decentralized, and Decentralized + Load Balancing for an extreme case when trace data file density in Subsection 1 = 6,000, D_{sz} = 500 KB, and Ferry Density = 10.

6 The Milk Carton FTR System

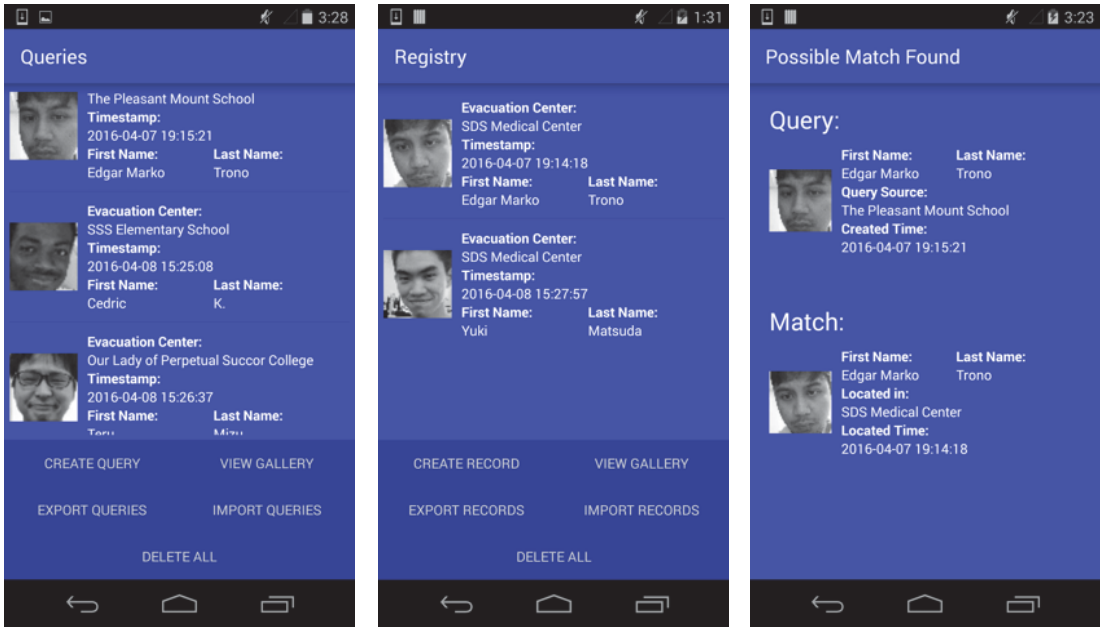
6.1 Introduction

In this chapter, we extend our work by considering FTR as another disaster response application that can be implemented using the system architectures presented in Chapters 4 and 5. Here, we face the problem of realizing a digital FTR system that can be used in disaster scenarios.

While current FTR methods try to make the process more efficient, they still have shortcomings. First, traditional methods use paper-based registries and notice boards, which are difficult to replicate and share, and cannot automatically find matches. Second, existing digital FTR systems require Cloud-servers for data storage. Such resources may not be available in a disaster area without Internet access. Finally, existing systems mainly rely on text-based personal information to find missing persons. Some people (e.g. children, the elderly, and persons with disabilities) however, may be unable to provide such information. The overcome these shortcomings, we present Milk Carton: a digital FTR system that uses face recognition.

To realize Milk Carton, we faced the following challenges: (1) how to query for and find missing persons without requiring text-based information, (2) how to the storage and computing requirements of the system without Cloud-based resources and services, and (3) how to transmit data without continuous, end-to-end networks.

For (1), for cases where text-based information are unavailable, our system executes a face recognition algorithm to find possible matches to missing person queries; for (2) we use a similar approach to those in Chapters 4 and 5 and use spatially-distributed Computing Nodes to handle the storage and computing requirements of the system; and for (3) we use response team vehicles to ferry



(a) (b) (c)

Figure 6.1: Milk Carton Application: (a) Query Creation (b) Record Creation (c) Possible Matches

data across the system.

In this chapter, we present the designs of the system and perform initial experimental evaluations to see how fast a Computing Node can perform the face recognition algorithm given different database sizes and photo resolutions. We also evaluated our system's face recognition accuracy.

The rest of this chapter shows the our assumptions about the target scenario, the components and workflow of the system, and our evaluations.

6.2 Assumptions

6.2.1 Target Scenario

The target scenario is a city within the first 72 hours after a disaster has struck. The disaster has caused people to evacuate. There are many evacuation centers in the area, which are managed by response teams. During evacuations, some

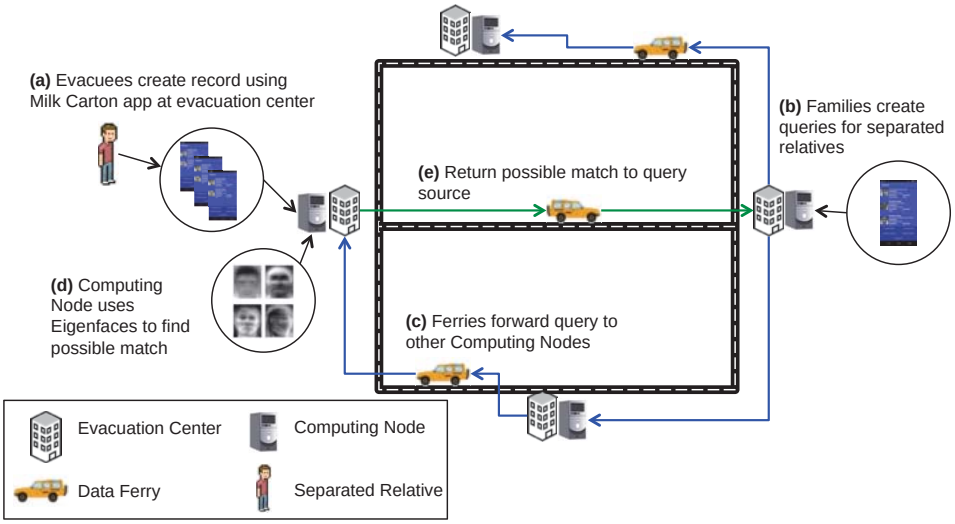


Figure 6.2: Milk Carton Workflow Diagram

families are separated and end up at different evacuation centers. We assume that the area's network infrastructure was damaged and is not yet operational.

6.2.2 Equipment

At each evacuation center, the response team has set up its equipment, which includes a commodity computing device such as a laptop or mini PC (e.g. with a 2.5 to 3 GHz processor and 4 GB RAM). The device has a wireless network interface (e.g. Wi-Fi) that is used to create a wireless local network, and sufficient storage space (e.g. 256 to 512 GB available space). The team also has smartphones with cameras, wireless network interfaces, and the Milk Carton application [54]. Smartphones and computing devices in the system have an implementation of the bundle protocol (e.g. IBR-DTN [43]), and they use their wireless network interfaces to establish local connections.

6.2.3 Response Team Vehicles

Response teams have vehicles patrolling the disaster area. Based on the cyclic pattern described in the Post-Disaster Mobility (PDM) model [56], the vehicles follow a route and visit evacuation centers along the route. Response vehicle

drivers have smartphones with wireless interfaces and sufficient storage space (e.g. 16 to 32 GB).

Based on prior preparations and drills, the travel times of response vehicles between any two evacuation centers have already been estimated. The smartphones of vehicle drivers have a route management application. Drivers input their routes in the application, which automatically calculates the travel times between evacuation centers.

6.3 Milk Carton Components

The Milk Carton system has the following components:

1. Milk Carton Application - the response teams that manage evacuation centers have smartphones with the Milk Carton application, which they use to create evacuee records and queries for separated family members. Figure 6.1 shows screenshots of the application.

2. Computing Node - The computing device at each evacuation center functions as a Computing Node. They are used to store records and queries. Computing Nodes also execute the Eigenfaces algorithm to find records that possibly match queries. To handle

3. Data Ferries - Without the Internet, the system leverages response team vehicles as data ferries. Response vehicles deliver data across the system using their smartphones.

6.4 System Workflow

Figure 6.2 shows the workflow of the Milk Carton system.

6.4.1 Record Creation

The first step in the workflow is record creation (Figure 6.2a). Responders use the Milk Carton application to create records of people entering evacuation centers (i.e. including separated people). Using the application, text-based personal

information of evacuees are recorded. To handle cases where personal information is unobtainable, the application also captures photos of registrants. Photos are added to the records along with a timestamp and the location or name of the evacuation center. Records are transferred (e.g. via local wireless network) from the smartphone to the evacuation center's Computing Node.

The number of evacuees can be very large (e.g. 100,000 were evacuated during the 2016 Kumamoto Earthquakes*). A single computing device may not be able to store and process that much data. To handle storage and the consequent computing requirements of a large number of records, Computing Nodes only store records created at the evacuation center where they are deployed. This naturally divides the load to manageable amounts (e.g. an evacuation center in Kumamoto can host approximately 1000 evacuees†).

6.4.2 Query Creation

The next step in the workflow is query creation (Figure 6.2b). In another evacuation center, families report separated relatives to the response team in that evacuation center. The response team uses the Milk Carton application to create a query. Responders input a photo (e.g. recent pictures provided by the family) and personal information of the separated relative. Queries also include a timestamp and the location or name of the evacuation center. Queries are transferred to the evacuation center's Computing Node.

6.4.3 Query Forwarding

Next, queries are forwarded to other Computing Nodes in the area (Figure 6.2c). Computing Nodes replicate their stored queries, and each replica's destination is a Computing Node in a different evacuation center. To forward queries without the Internet, the system uses response team vehicles as data ferries. Whenever a ferry visits an evacuation center, its smartphone establishes contact with the evacuation center's Computing Node. During contact, queries are forwarded to the ferry that will reach the destination Computing Node the fastest, based on

*http://www.asahi.com/ajw/articles/AJ201604190001.html
†http://mainichi.jp/english/articles/20160420/p2a/00m/0na/014000c

the known ferry routes (see Section 6.2.3).

6.4.4 Finding and Returning Matches

The final steps in the workflow are finding and returning possible matches to queries (Figure 6.2d and 6.2e). When a Computing Node receives a query it first searches for records with matching personal information (e.g. name, birthdate). If no matches are found, it executes the Eigenfaces algorithm and compares the query's photo with the photos of the records in its database. If a photo with the least distance and within a user-defined similarity threshold is found, the photo's corresponding record is returned as a match. The match is addressed to the query's source and is forwarded by data ferries.

6.5 Evaluations

In this study, we present initial evaluations of the system. Our goal is to (1) see how a Computing Node performs, in terms of execution speed of the Eigenfaces face recognition algorithm in an evacuation center setting, and (2) evaluate the accuracy of the face recognition implementation.

The evaluation scenario is as follows: The Computing Node is deployed in an evacuation center. It has a database of records containing the faces of the evacuees in the center. It has received a query for a separated person and will search its database for a possible match by executing the Eigenfaces face recognition algorithm.

We evaluated how much time the Computing Node requires to execute the matching task, given different database sizes of 250, 500, and 1000 evacuee records (i.e. the center has a max capacity of 1000 evacuees) and different face image dimensions of 20 x 20, 50 x 50, and 100 x 100 pixels.

To represent the Computing Node, we used an Intel NUC mini PC with a 2.7 GHz processor, 4 GB of memory. We used a database of faces from the Extended Yale Face Database B[‡]. We used one image as the query photo and executed the JavaFaces implementation of the Eigenfaces face recognition algorithm and

[‡]http://vision.ucsd.edu/ leekc/ExtYaleDatabase/ExtYaleB.html

measured the time from start to finish. We executed 100 query runs for each database size and each face dimension. Figures 6.5, 6.6, and 6.7 show the mean execution times of the query runs.

To evaluate the face recognition accuracy, we executed queries on the system using different photo resolutions (i.e. 20 x 20, 50 x 50, and 100 x 100 pixels), number of photos per unique person (5, 10, 20, and 50 photos per unique person), and the number of eigenfaces used by the algorithm (i.e. 10, 20, 30, 40, 50, 60, 70, 80, and 90 eigenfaces). We used two photo databases: (1) the Extended Yale Face Database B and (2) we took photos if 19 participants from different angles with varying expressions and lighting conditions (the UBI database). We used 10 photos for each unique person as queries. We measured the accuracy as the number of correctly recognized queries divided by the total number of queries. The parameters for the experiment are listed in Table 6.1. Figure 6.3 shows samples from the Yale Extended database. To preserve the privacy of of the participants, samples from the UBI database are not shown in this dissertation.

Parameter	**Value**
Photo Resolution	20 x 20, 50 x 50, 100 x 100 pixels
Photo Database	Yale Extended B, UBI Database
Unique Persons (classes) in Yale DB	20 persons
Unique Persons (classes) in UBI DB	19 persons
Photos (samples) per Person	5, 10, 20, 50
Queries per Person	10
Eigenfaces	10, 20, 30, 40, 50, 60, 70, 80, 90

Table 6.1: Accuracy Evaluation Parameters

The results in Figures 6.5, 6.6, and 6.7 show an expected trend that the mean execution time of the query runs increase as the face image dimensions and database size increase. The results also show that the Intel NUC Computing Node can execute the Eigenfaces algorithm in minimal time. Even for the case with the largest image dimensions (i.e. 100 x 100 pixels) and a full evacuation center (i.e. 1000 records), it only took approximately 7 seconds to complete a query.

Figure 6.3: Samples from the Extended Yale Database B

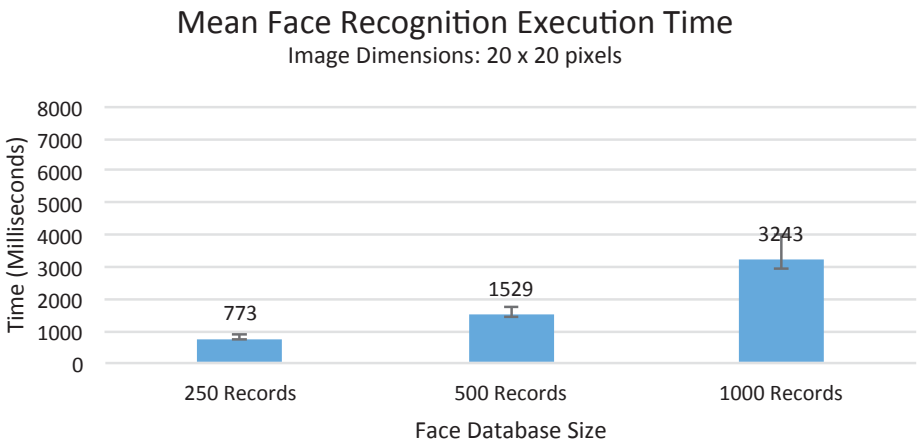


Figure 6.4: Mean execution time of the Eigenfaces face recognition algorithm for 20 x 20 pixel image dimensions and different face database size

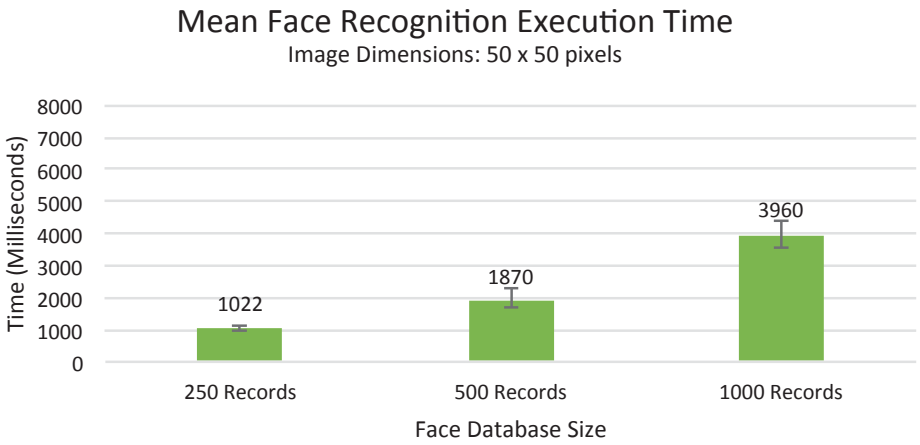


Figure 6.5: Mean execution time of the Eigenfaces face recognition algorithm for 50 x 50 pixel image dimensions and different face database size

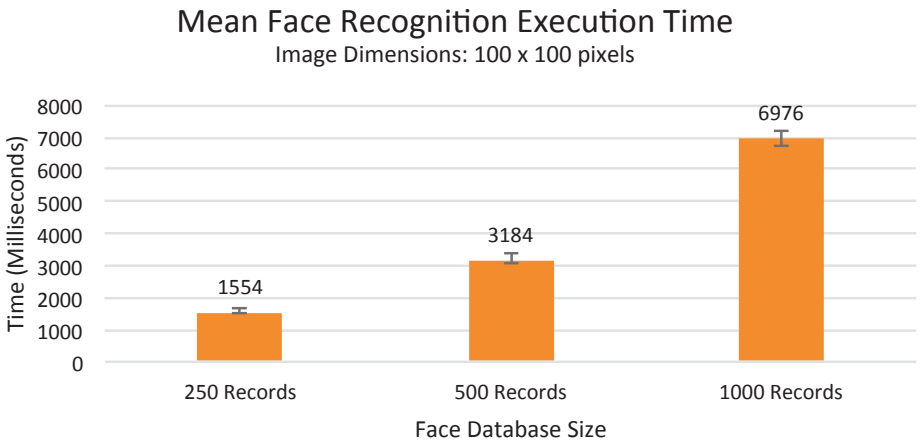
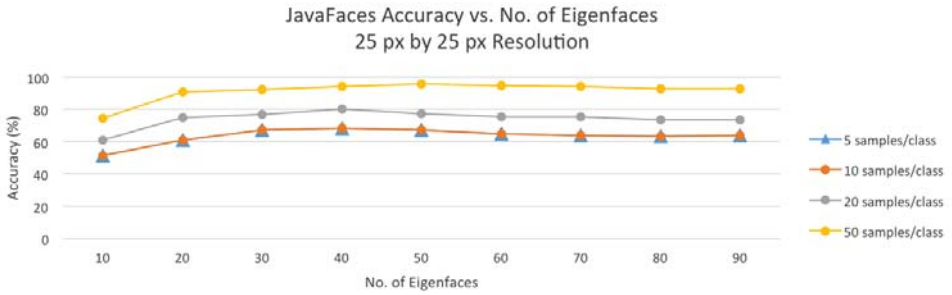
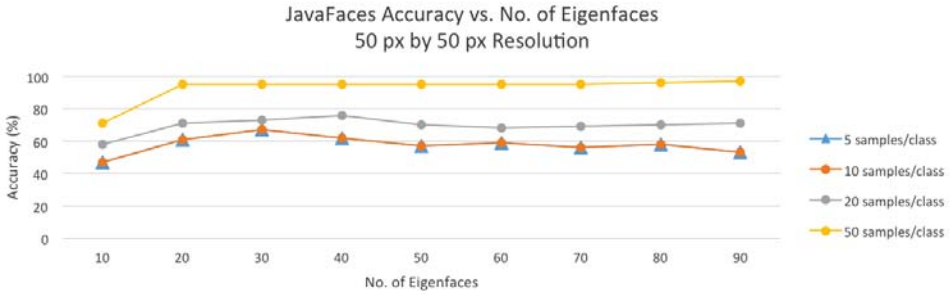


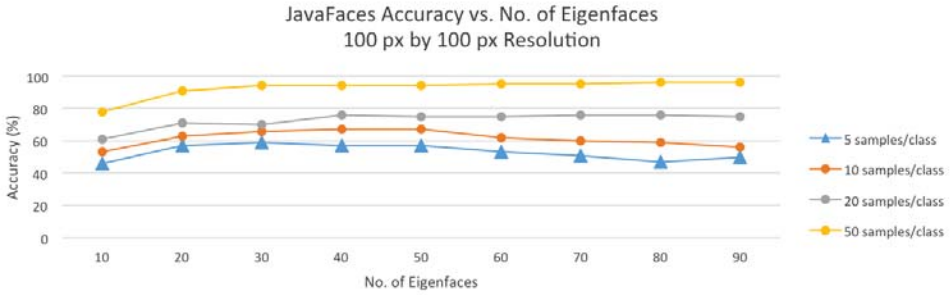
Figure 6.6: Mean execution time of the Eigenfaces face recognition algorithm for 100 x 100 pixel image dimensions and different face database size



(a) 25 x 25 pixels



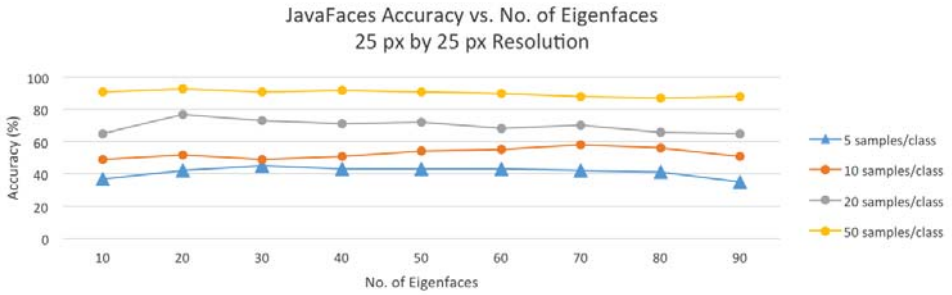
(b) 50 x 50 pixels



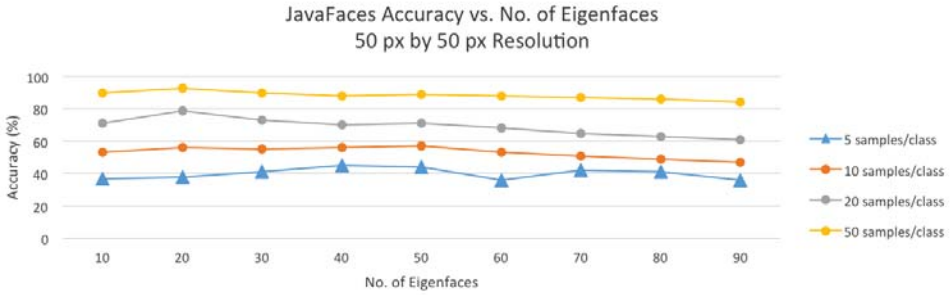
(c) 100 x 100 pixels

Figure 6.7: Milk Carton's face recognition accuracy for the Extended Yale Database B at photo resolutions of (a) 25 x 25, (b) 50 x 50, and (c) 100 x 100 pixels. The number of eigenfaces used and the number of unique photos (i.e. samples) per person were varied.

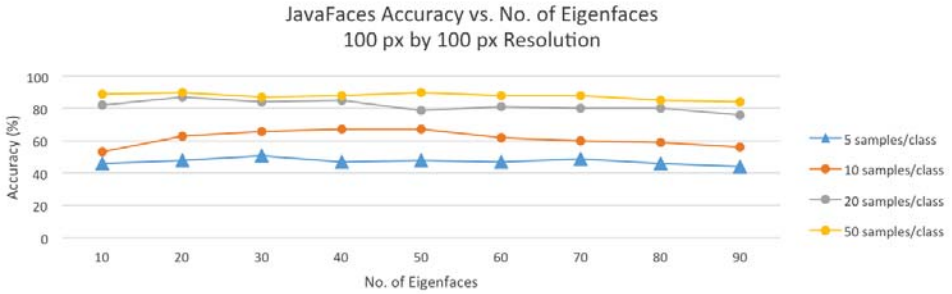
Figures 6.8 (a), (b), and (c) show the accuracy of Milk Carton's face recognition for the Yale Extended Database B. The results are based on different numbers



(a) 25 x 25 pixels



(b) 50 x 50 pixels



(c) 100 x 100 pixels

Figure 6.8: Milk Carton's face recognition accuracy for the UBI Database B at photo resolutions of (a) 25 x 25, (b) 50 x 50, and (c) 100 x 100 pixels. The number of eigenfaces used and the number of unique photos (i.e. samples) per person were varied.

of eigenvalues used and different unique photos (i.e. samples) per person for resolutions of 25 x 25, 50 x 50, and 100 x 100 pixels, respectively. From these

results we make the following observations. First, the accuracy of the system increases if a higher number of unique photos or samples are available per person. This implies that for implementation purposes, a mechanism or function that can take multiple photos of an evacuee, from different angles is needed. This can be done quickly by using the burst-capture mode of smartphone cameras, for example.

Second, the resolution of the photos do not have a significant effect on the accuracy. The accuracy values when the resolution was 25 x 25 pixels already exceeded 90% when the number of unique samples per person was 50. Even when the resolution increased to 100 x 100 pixels, the accuracy values were in the same range. From this we learned that to reduce computation time and storage requirements, low-resolution photos can be used, as long as the number of samples is high.

Third, we observed that the accuracy increased the most when the number of eigenfaces used changed from 10 to 20. The accuracy did not change significantly as the number of eigenfaces increased from 30 to 90. The number of eigenfaces used are often determined either heuristically or arbitrarily (i.e. a changeable parameter). These results show that using more eigenfaces is not always the most efficient option. If more eigenfaces are used, computation speed will be slower (i.e. more calculations are needed). For practical implementation, it is best to have a function that controls the eigenfaces, which allows users to start by using less eigenfaces and increase it as needed.

Figures 6.9 (a), (b), and (c) show the accuracy of Milk Carton's face recognition for the UBI database. The results are based on different numbers of eigenvalues used and different unique photos (i.e. samples) per person for resolutions of 25 x 25, 50 x 50, and 100 x 100 pixels, respectively. Accuracy results from using the UBI database were very similar to the results from using the Yale Extended Database B. We again observed that the resolution does not significantly affect the accuracy, the greater the number of samples used yielded more accurate results, and using higher numbers of eigenfaces does not always result in higher accuracy.

The main difference from using the two databases is that Milk Carton was more accurate when using the Yale Extended Database B than when using the UBI database. Milk Carton was able to exceed 90% accuracy for the Yale Extended

Database B. For the UBI database, Milk Carton only reached between 80% and 90%. This lower accuracy can be attributed to the higher variance in the photos of the UBI database where the participants were asked to make different expressions and poses. Even with the lower accuracy however, 80% is still a reasonable performance in actual implementation. If more accurate results are required, other face recognition libraries can be used, and can be evaluated in future work.

From these initial results, we conjecture that Milk Carton's approach of using multiple Computing Nodes in a disaster area can quickly and accurately run face recognition because with this architecture, evacuee records are distributed into manageable amounts (i.e. compared to another scenario where a only single machine in the disaster area stores all records and processes all queries).

6.6 Summary

In this chapter, we present the designs and the results of the initial evaluations of the Milk Carton FTR system. In our evaluations, we show how our approach of using spartially-distributed computing nodes reduces the overall computing load of face recognition because the size of each face database is limited to the capacity of evacuation centers. We also show that our current implementation can achieve reasonable face recognition accuracy values.

7 Conclusions and Future Work

7.1 Summary

In this dissertation, we presented disaster response applications that use distributed computing over a DTN. The motivation for our work was the need for digital systems that aid disaster response operations such as disaster area mapping and FTR. These systems must function in disaster areas without continuous, end-to-end communication networks and must be capable of handling computing requirements without access to Cloud-based resources and services.

We presented two systems: DTN MapEx and Milk Carton. DTN MapEx generates disaster area pedestrian maps and delivers them to evacuees and response commanders. The system uses a DTN of smartphones and leverages response team vehicles as data ferries for communication. It handles the computing requirements of pedestrian map inference by using Computing Nodes: commodity workstations that are deployed in the disaster area. The Computing Nodes receive the collected raw data and infer the map. The system also uses a load balancing heuristic to distribute computing tasks.

Milk Carton is a system that aids in FTR. Similar to DTN MapEx, Milk Carton uses data ferries to deliver messages and Computing Nodes to handle the computing requirements of face recognition that is needed to find missing people during the FTR process.

We evaluated both systems using experiments and simulations and found that by distributing computing load to Computing Nodes, the savings in processing latency were enough to offset the communication latency caused by the DTN and data ferries. Using our proposed load-balancing algorithm, the system was able to reduce the time required to process and generate and deliver pedestrian maps by approximately 2 hours, compared to an approach without load balancing.

7.2 Future Work

In future works, we aim to further evaluate our system via experiments during response operation drills or during actual operations. Next, in cases where response vehicles are unavailable, we aim to design and evaluate architectures that use alternate data ferries, for instance drones. We also intend to improve the DTN MapEx application by adding compression functionality to reduce message size, and by including power and device storage management schemes.

Additionally for the Milk Carton System, we aim to further evaluate the system using simulations and actual deployment. We also intend to evaluate the accuracy of the Eigenfaces face recognition algorithm in terms of its accuracy in finding matches.

Acknowledgements

This work would not have been possible without the support of many people. First, I would like to thank my supervisor, Prof. Keiichi Yasumoto, who provided valuable input and guided me, with extreme patience, throughout the duration of my stay. I would also like to thank the present and past professors of the Ubiquitous Computing Laboratory: Associate Prof. Yutaka Arakawa, Assistant Prof. Hirohiko Suwa, Assistant Prof. Manato Fujimoto, and Dr. Morihiko Tamai, all of whom gave valuable comments and contributed ideas for my research. I also thank Prof. Shoji Kasahara for his valuable input and comments that helped clarify and improve the study. I would also like to thank the laboratory's secretary, Mrs. Megumi Kanaoka, who handled the administrative matters during conferences and who was always very gracious despite the language barrier. I am also grateful to all the members of the Ubiquitous Computing Laboratory for all the support and assistance they provided and for all the fun social gatherings. I would like to thank my own family for their support. In addition, I am forever grateful to the Wakazono family for their kindness and treating me as their own. This work is partly supported by the Japanese Government Monbukagakusho: MEXT Scholarship and JSPS KAKENHI Grant Numbers 26220001 and 16H0291410.

References

[1] Mobile Cloud Computing for Distributed Disaster Information System in Challenged Communication Environment, author=Kikuchi, Yosuke and Shibata, Yoshitaka. In *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on* (2015), IEEE, pp. 512–517.

[2] Agussalim, and Tsuru, M. Comparison of DTN Routing Protocols in Realistic Scenario. In *Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on* (2014), IEEE, pp. 400–405.

[3] Android Developers. Location-Android Developers. `https://developer.android.com/reference/android/location/Location.html`, 2012. Last Accessed: 2016-06-18.

[4] Aschenbruck, N., Gerhards-Padilla, E., and Martini, P. Modeling mobility in disaster area scenarios. *Performance Evaluation 66*, 12 (2009), 773–790.

[5] Bharosa, N., Lee, J., and Janssen, M. Challenges and obstacles in sharing and coordinating information during multi-agency disaster response: Propositions from field exercises. *Information Systems Frontiers 12*, 1 (2010), 49–65.

[6] Biagioni, J., and Eriksson, J. Inferring Road Maps from Global Positioning System Traces. *Transportation Research Record: Journal of the Transportation Research Board 2291*, 1 (2012), 61–71.

[7] Bin Tariq, M. M., Ammar, M., and Zegura, E. Message Ferry Route Design for Sparse Ad Hoc Networks with Mobile Nodes. In *Proceedings*

of the 7th ACM international symposium on Mobile ad hoc networking and computing (2006), ACM, pp. 37–48.

[8] Blanke, U., Guldener, R., Feese, S., and Tröster, G. Crowd-sourced Pedestrian Map Construction for Short-term City-scale Events. In *Proceedings of the First International Conference on IoT in Urban Space* (2014), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 25–31.

[9] Burgess, J., Gallagher, B., Jensen, D., and Levine, B. N. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *INFOCOM* (2006), vol. 6, pp. 1–11.

[10] Chenji, H., Zhang, W., Stoleru, R., and Arnett, C. DistressNet: A disaster response system providing constant availability cloud-like services. *Ad Hoc Networks 11*, 8 (2013), 2440–2460.

[11] Chenji, H., Zhang, W., Won, M., Stoleru, R., and Arnett, C. A wireless system for reducing response time in Urban Search & Rescue. In *Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International* (2012), IEEE, pp. 215–224.

[12] Chester, B. Battery Life and Charge Time - The Moto E (2015) Review. `http://www.anandtech.com/show/9129/the-moto-e-2015-review/7`, 2015. Last Accessed: 2016-06-17.

[13] Conti, M., and Giordano, S. Mobile ad hoc networking: milestones, challenges, and new research directions. *IEEE Communications Magazine 52*, 1 (2014), 85–96.

[14] Conti, M., Giordano, S., May, M., and Passarella, A. From opportunistic networks to opportunistic computing. *IEEE Communications Magazine 48*, 9 (2010), 126–139.

[15] Conti, M., Kumar, M., et al. Opportunities in opportunistic computing. *Computer 43*, 1 (2010), 42–50.

[16] Fajardo, J. T. B., Yasumoto, K., Shibata, N., Sun, W., and Ito, M. Disaster Information Collection with Opportunistic Communication and Message Aggregation. *Journal of information processing 22*, 2 (2014), 106–117.

[17] Fall, K. A Delay-tolerant Network Architecture for Challenged Internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2003), ACM, pp. 27–34.

[18] Fujihara, A., and Miwa, H. Disaster Evacuation Guidance Using Opportunistic Communication: The Potential for Opportunity-Based Service. In *Big Data and Internet of Things: A Roadmap for Smart Environments*. Springer, 2014, pp. 425–446.

[19] George, S. M., Zhou, W., Chenji, H., Won, M., Lee, Y. O., Pazarloglou, A., Stoleru, R., and Barooah, P. DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response. *Communications Magazine, IEEE 48*, 3 (2010), 128–136.

[20] IBR TU Braunschweig. GitHub-ibrdtn/android-sharebox: ShareBox - Delay-Tolerant File Sharing. `https://github.com/ibrdtn/android-sharebox`, 2015. Last Accessed: 2016-06-15.

[21] International Committee of the Red Cross. Interagency Guiding Principles on Unaccompanied and Separated Children. `https://www.icrc.org/eng/assets/files/other/icrc_002_1011.pdf`, 2004. Last Accessed: 2016-08-24.

[22] International Search and Rescue Advisory Group. International search and rescue advisory group guidelines and methodology. `http://www.ifrc.org/docs/idrl/I927EN.pdf`, 2012. Last Accessed: 2016-06-17.

[23] Jang, H.-C., Lien, Y.-N., and Tsai, T.-C. Rescue Information System for Earthquake Disasters Based on MANET Emergency Communication Platform. In *Proceedings of the 2009 International Conference on Wireless*

Communications and Mobile Computing: Connecting the World Wirelessly (2009), ACM, pp. 623–627.

[24] Johnson, D. B., and Maltz, D. A. *Dynamic Source Routing in Ad Hoc Wireless Networks*. Springer US, Boston, MA, 1996, pp. 153–181.

[25] Kaur, J., and Kinger, S. A Survey on Load Balancing Techniques in Cloud Computing. *International Journal of Engineering Research and Technology 2*, 8 (2013), 800–804.

[26] Lien, Y.-N., Jang, H.-C., and Tsai, T.-C. A MANET Based Emergency Communication and Information System for Catastrophic Natural Disasters. In *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on* (2009), IEEE, pp. 412–417.

[27] Lindgren, A., Doria, A., and Schelén, O. Probabilistic Routing in Intermittently Connected Networks. *SIGMOBILE Mob. Comput. Commun. Rev. 7*, 3 (2003), 19–20.

[28] Liu, X., Biagioni, J., Eriksson, J., Wang, Y., Forman, G., and Zhu, Y. Mining Large-scale, Sparse GPS Traces for Map Inference: Comparison of Approaches. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), ACM, pp. 669–677.

[29] Marikina City Disaster Coordinating Council. Marikina City Disaster Coordinating Council Preparedness Program. `http://alliance-healthycities.com/PDF/WHO20Awards/Marikina_WHOAward2004_Emergency2020Preparedness.pdf`, 2004. Last Accessed: 2016-07-28.

[30] Martín-Campillo, A., Crowcroft, J., Yoneki, E., and Martí, R. Evaluating opportunistic networks in disaster scenarios. *Journal of Network and Computer Applications 36*, 2 (2013), 870–880.

[31] Nelson, S. C., Harris, III, A. F., and Kravets, R. Event-driven, Role-based Mobility in Disaster Recovery Networks. In *Proceedings of the Second ACM Workshop on Challenged Networks* (New York, NY, USA, 2007), CHANTS '07, ACM, pp. 27–34.

[32] Nishiyama, H., Ito, M., and Kato, N. Relay-by-smartphone: realizing multihop device-to-device communications. *IEEE Communications Magazine 52*, 4 (2014), 56–65.

[33] Nuaimi, K. A., Mohamed, N., Nuaimi, M. A., and Al-Jaroodi, J. A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on* (2012), IEEE, pp. 137–142.

[34] Patel, N., and Chauhan, S. A survey on load balancing and scheduling in cloud computing. *International Journal for Innovative Research in Science and Technology 1*, 7 (2015), 185–189.

[35] Pelusi, L., Passarella, A., and Conti, M. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *IEEE Communications Magazine 44*, 11 (2006), 134–141.

[36] Perkins, C., Belding-Royer, E., and Das, S. Ad hoc on-demand distance vector (aodv) routing. Tech. Rep. IETF RFC 3561, 2003.

[37] Reina, D., Askalani, M., Toral, S., Barrero, F., Asimakopoulou, E., and Bessis, N. A Survey on Multihop Ad Hoc Networks for Disaster Response Scenarios. *International Journal of Distributed Sensor Networks 2015* (2015), 3.

[38] Reina, D., Coca, J. M. L., Askalani, M., Toral, S., Barrero, F., Asimakopoulou, E., Sotiriadis, S., and Bessis, N. A Survey on Ad Hoc Networks for Disaster Scenarios. In *Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on* (2014), IEEE, pp. 433–438.

[39] Reina, D., Toral, S., Barrero, F., Bessis, N., and Asimakopoulou, E. Modelling and assessing ad hoc networks in disaster scenarios. *Journal of Ambient Intelligence and Humanized Computing 4*, 5 (2013), 571–579.

[40] Reina, D., Toral, S. L., Barrero, F., Bessis, N., and Asimakopoulou, E. Evaluation of Ad Hoc Networks in Disaster Scenarios. In *Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on* (2011), IEEE, pp. 759–764.

[41] Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S. J., and Chong, S. On the Levy-walk Nature of Human Mobility. *IEEE/ACM transactions on networking (TON) 19*, 3 (2011), 630–643.

[42] Richards, C. When Communications Infrastructure Fails During a Disaster. `http://www.drj.com/articles/online-exclusive/when-communications-infrastructure-fails-during-a-disaster.html`, 2015. Last Accessed: 2016-06-18.

[43] Schildt, S., Morgenroth, J., Pöttner, W.-B., and Wolf, L. IBR-DTN: A lightweight, modular and highly portable bundle protocol implementation. *Electronic Communications of the EASST 37* (2011), 1–11.

[44] Schroedl, S., Wagstaff, K., Rogers, S., Langley, P., and Wilson, C. Mining GPS traces for map refinement. *Data mining and knowledge Discovery 9*, 1 (2004), 59–87.

[45] Shah, R. C., Roy, S., Jain, S., and Brunette, W. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks 1*, 2 (2003), 215–233.

[46] Shahzamal, M., Parvez, M., Zaman, M., and Hossain, M. Mobility models for delay tolerant network: A survey. *International Journal of Wireless & Mobile Networks 6*, 4 (2014), 121.

[47] Shi, C., Ammar, M. H., Zegura, E. W., and Naik, M. Computing in Cirrus Clouds: The Challenge of Intermittent Connectivity. In *Proceedings*

of the first edition of the MCC workshop on Mobile cloud computing (2012), ACM, pp. 23–28.

[48] Shi, C., Lakafosis, V., Ammar, M. H., and Zegura, E. W. Serendipity: Enabling Remote Computing Among Intermittently Connected Mobile Devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing* (2012), ACM, pp. 145–154.

[49] Song, L., and Kotz, D. F. Evaluating Opportunistic Routing Protocols with Large Realistic Contact Traces. In *Proceedings of the second ACM workshop on Challenged networks* (2007), ACM, pp. 35–42.

[50] Sun, J., Zhu, X., Zhang, C., and Fang, Y. RescueMe: Location-Based Secure and Dependable VANETs for Disaster Rescue. *IEEE Journal on Selected Areas in Communications 29*, 3 (2011), 659–669.

[51] Takai, M., Owada, Y., and Seki, K. A Comparative Study on Network Simulators for ITS Simulation IEEE802. 11 Medium Access Control (MAC) Models. In *16th ITS World Congress and Exhibition on Intelligent Transport Systems and Services* (2009).

[52] Trono, E. M., Arakawa, Y., Tamai, M., and Yasumoto, K. DTN MapEx: Disaster area mapping through distributed computing over a Delay Tolerant Network. In *Mobile Computing and Ubiquitous Networking (ICMU), 2015 Eighth International Conference on* (2015), IEEE, pp. 179–184.

[53] Trono, E. M., Fujimoto, M., Suwa, H., Arakawa, Y., Takai, M., and Yasumoto, K. Disaster area mapping using spatially-distributed computing nodes across a DTN. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)* (2016), IEEE, pp. 1–6.

[54] Trono, E. M., Fujimoto, M., Suwa, H., Arakawa, Y., and Yasumoto, K. Milk carton: A face recognition-based ftr system using opportunistic clustered computing. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)* (2016), IEEE, pp. 759–760.

[55] Turk, M. A., and Pentland, A. P. Face Recognition using Eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on* (1991), IEEE, pp. 586–591.

[56] Uddin, M. Y. S., Nicol, D. M., Abdelzaher, T. F., and Kravets, R. H. A post-disaster mobility model for delay tolerant networking. In *Winter Simulation Conference* (2009), WSC '09, Winter Simulation Conference, pp. 2785–2796.

[57] UNICEF. RapidFTR Homepage. `http://http://www.rapidftr.com/`, 2010. Last Accessed: 2016-08-24.

[58] Vahdat, A., Becker, D., et al. Epidemic routing for partially connected ad hoc networks. Tech. Rep. CS-200006, 2000.

[59] Zhao, W., Ammar, M., and Zegura, E. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (2004), MobiHoc '04, ACM, pp. 187–198.

[60] Zhao, W., Ammar, M., and Zegura, E. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies* (2005), vol. 2, IEEE, pp. 1407–1418.

[61] Zheng, K., Zheng, Y., Xie, X., and Zhou, X. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *2012 IEEE 28th International Conference on Data Engineering* (2012), IEEE, pp. 1144–1155.

Publication List

Major Publications

1. Edgar Marko Trono, Morihiko Tamai, Yutaka Arakawa, and Keiichi Yasumoto. "DTN MapEx: Disaster area mapping through distributed computing over a Delay Tolerant Network." Mobile Computing and Ubiquitous Networking (ICMU), 2015 Eighth International Conference on. IEEE, 2015.

2. Edgar Marko Trono, Manato Fujimoto, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. "Disaster area mapping using spatially-distributed computing nodes across a DTN." 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops). IEEE, 2016.

3. Edgar Marko Trono, Manato Fujimoto, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. "Milk Carton: A Face Recognition-Based FTR System Using Opportunistic Clustered Computing." Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on. IEEE, 2016.

4. Edgar Marko Trono, Manato Fujimoto, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. "Milk Carton: Family Tracing and Reunification System using Face Recognition over a DTN with Deployed Computing Nodes." International Workshop on Information Flow of Things (MOBIQUITOUS Workshops). ACM, 2016.

5. *Accepted for Publication*: Edgar Marko Trono, Manato Fujimoto, Hirohiko Suwa, Yutaka Arakawa, and Keiichi Yasumoto. "Generating pedestrian maps of disaster areas through ad-hoc deployment of computing resources across a DTN." Computer Communications Journal. Elsevier.

Other Publications

1. Edgar Marko Trono, Yuka Kume, Yutaka Arakawa, Keiichi Yasumoto, and Masayuki Ariyoshi. "Implementation of a mountainside sensor network with an ambient air pressure-based routing scheme." Technical report of the Institute of Electronics, Information and Communication. MoNA, Mobile Networks and Applications 114.417 (2015): 81-83.

2. Yuka Kume, Edgar Marko Trono, Yutaka Arakawa, Keiichi Yasumoto, and Masayuki Ariyoshi. "A Mountain Slope Monitoring System by Using Used Smartphones and Pressure-Based Routing Method." Technical report of the Institute of Electronics, Information and Communication. MoNA, Mobile Networks and Applications 114.417 (2015): 25-30.

3. Seigi Matsumoto, Edgar Marko Trono, Yutaka Arakawa, and Keiichi Yasumoto. "RecureShare—Internet-less application distribution mechanism for internet-less emergency communication systems." Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on. IEEE, 2015.