Peer Alerting Lifeline: A Study of Backend Infrastructure for a Crowdsourced Emergency Response System

by

Madhav Malhotra
Bachelor of Technology, Maharishi Dayanand University, 2011

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Peer Alerting Lifeline: A Study of Backend Infrastructure for a Crowdsourced Emergency Response System

by

Madhav Malhotra
Bachelor of Technology, Maharishi Dayanand University, 2011

Supervisory Committee

Dr. Yvonne Coady, Department of Computer Science
Supervisor
Dr. Sudhakar Ganti, Department of Computer Science
Departmental Member

Abstract

Supervisory Committee

Dr. Yvonne Coady, Department of Computer Science
Supervisor

Dr. Sudhakar Ganti, Department of Computer Science
Departmental Member

Opioid users are an at-risk community. Risk of opioid overdose among substance users has increased tremendously in the last decade. Many factors, including adulterated drugs and hesitation in calling emergency response services, have led to many individuals not receiving the required harm reduction treatment, during an overdose incident. The problem is further compounded by the fact that many users are using alone in private residences and hence, no support mechanisms are available for them to assist them in case of an overdose situation. To circumvent this scenario, citizen training in Naloxone, an overdose harm reduction drug, has been promoted. However, there lies an essential communication gap between the citizens who have the training and the Naloxone kit and an active overdose event. Many at-risk communities may face the same challenge, especially if they are at risk of social isolation and voluntary/involuntary self-harm.

Through our work, we wish to mobilize change in such at-risk communities, by studying the backend infrastructure of a crowdsourced emergency response system, called as a *Peer Alerting Lifeline*. The system would be responsible, for connecting peer responders, to an actual emergency event. Specifically, in the case of substance overdose, this would allow Naloxone kit holders to be informed of an overdose event in their vicinity and respond to the same. We aim to study the design infrastructure of such a system.

.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank **Dr. Yvonne Coady** for her persistent dedication towards motivating me in completing this work and for providing me with both personal and professional guidance.

I would like to thank **Dr. Derek Jacoby**, whose management of the NALPAL project has allowed me to work on an application that could actually make a difference and save lives.

Finally, I would like to thank **Avneet Kaur,** my partner who has been of the utmost support and whose faith in me has resulted in completing my work.

Dedication

To my parents and my partner Avneet Kaur, without whom this would not have been possible.

Chapter 1

Motivation and Problem

In this Chapter, we discuss the motivation and inception of the idea behind a PAL or a Peer Alerting Lifeline. We examine, via focusing on the context of drug use and overdose, how the system can be used to mobilize harm reduction service during an emergency.

1.1 Drug Overdose Epidemic

North America is facing an increasingly complex challenge of the drug overdose. Opioids are currently the most common cause of death among drug-related overdoses, contributing to 42,249 fatalities in the US [1] and 2,946 apparent opioid-related deaths in Canada in 2016 [2].

Currently, opioids are the leading cause of death in the Americas under 50, surpassing motor vehicle accidents, gun violence or the HIV/AIDS crisis in the early 1990s [3]. The drug usage has increased substantially amongst the age groups between 26-34 and >35 years [6]. The current wave of this crisis is stemming from the stream of adulterated heroin, which is less predictable and is causing various adverse effects on the health of the user, including death [4].

The current opioid epidemic developed from an initial event associated with overdoses related to prescription opioids in the 1990s, followed by a second wave of semi-synthetic opioids (e.g., Heroin) in 2010, culminating in the current situation associated with synthetic opiates (e.g., Fentanyl and its analogs) [4]. The latest wave is likely a result of supply-side shock from the adulteration of heroin with synthetic opioids. Synthetic opioids play essential roles in human and veterinary clinical use, especially for surgical postoperative and terminal cancer care, and have emerged as common ingredients in illicitly manufactured products intended for non-clinical use. Highly potent synthetic opioids have gained prominence as a means to mimic or replace the effects of heroin and other drugs and present concentrated forms that are easier to transport and disguise. Additionally, a multitude of emerging form-factors and routes of administration have facilitated the adoption of these drugs by providing alternatives to comparatively risky and stigmatized intravenous methods of consumption. Techniques such as nasal sprays and pills have contributed to the appeal of opioids and provided more discrete means of administration.

Furthermore, traditional heroin is being adulterated with less predictable synthetic forms, with unknown and unreliable effects [4]. This phenomenon is driven by an international research chemical industry that continues to introduce novel synthetic analogs to the supply chain. These variants have gained popularity for substitution to circumvent drug testing and control regulations, and afford new experiences for users; however, this has rendered opioid usage more dangerous and overdose deaths more likely.

1.2 Overdose Harm Reduction: Naloxone

Naloxone is an opioid receptor antagonist, which can be used to counter the effects of opioids [25]. Naloxone can be administered intravenously, injected into the muscle or via a nasal spray [98]. When administered promptly, the medication can be used to prevent death due to an opioid overdose. Consequently, Naloxone is on the World Health Organization list of essential medicines. Citizens have been trained in Naloxone kit usage, so that they may be able to provide on-ground assistance if they witness an overdose in their vicinity. In British Columbia, pharmacies now provide free Naloxone kits. The 911 emergency responders are equipped with the kits in case of an overdose emergency. In the US, between 1996 to 2014, Center for Disease Control (CDC) estimates that 26,000 lives have been saved by Naloxone [24], via overdose reversals. However, the effectiveness of such programs is dependent on how timely the kit owners are informed of an active overdose event. Figure 1 shows a Naloxone Kit.

Figure 1: Naloxone Kit

1.3 Barriers to accessing emergency services for opioid users

Although we have discussed a potential solution to reverse the effects of a substance overdose, the situation has not improved in the last decade. In the US in 2017, more than 70,000 deaths were caused by substance overdose, which is a 2-fold increase in a decade [46]. More than 30,000 of these deaths were related to Fentanyl and their analogs. In this Section, we discuss three main causes, which have contributed to emergency services not being accessible.

First, **Stigma** or negative attitudes or beliefs, play a crucial role in making it harder for accessing emergency services during an overdose event. There are three kinds of stigma that substance users or families of substance users typically have to deal with [47]:

- *Social stigma*, which is negative beliefs or attitudes towards substance users and their loved ones. Negative labels and prevalent negative images further exaggerate such attitudes. Often, substance use is considered a moral weakness or a deliberate choice. [48]

- *Structural stigma*, which is social stigma from first responders, health care professionals, and government representatives. Such an attitude can lead to ignoring or not paying appropriate respect to people who are facing negative consequences from substance use.

- *Self-stigma*, which individuals often internalize due to the social and structural stigma.

The prevalence of such attitudes is a significant barrier in receiving core services, including emergency services. If someone has faced such a manner, the person is much less likely to reach out for help, even in life-threatening situations [47].

Second, **Inability to Access Emergency Services** during an overdose event. During an active overdose situation, individuals experience a variety of symptoms, such as [49]

- Dizziness, Confusion
- Low Blood Pressure
- Loss of consciousness
- Weak muscles, pinpoint pupils, slowed heartbeat

In situations when the individuals are using the substances alone, they may not be able to call for help. A recent survey in British Columbia for overdose incidents between 2016-2017 [50] found that 72% of males and 63% of female substance users lived in private residences. Another survey found that 50.6% of all overdoses were occurring in private

homes [26]. Such users may themselves be unable to call for help and would not have anyone around them to assist.

Third, **Fear of Legal Action** in case of calling 911. Individuals who either witness an overdose or are overdosing themselves, are afraid of legal action being taken against them, which may lead to prosecution or eviction from housing and losing their job [51,52]. Although, initiatives such as the Good Samaritan Drug Overdose Act [53] in Canada and US have been undertaken and they do provide limited security from arrests, the fear amongst the users persists and hence leads to hesitation in calling of 911 emergency services [51]. Other worries include drug confiscation, family finding out, damaging relationship with the landlord.

In this Section, we have focused on the traditional mechanism of calling 911 emergency responders. In the next Section, we look at a more decentralized model for availing emergency services, via a crowd made up of concerned citizens.

1.4 Crowdsourcing Emergency Response

The idea of crowdsourcing response from citizens during an emergency has been implemented in the past, especially in the case of disaster management and for improving public health and safety [56]. In 2012, during the earthquake in Haiti, a citizen emergency reporting mechanism was set up, to assist the emergency medical services in helping the victims. The system used Short Messaging Service or SMS for allowing the on-ground citizens to inform regarding their state and location [56]. The information was collated and provided to the emergency response teams, which allowed improved planning and execution of relief efforts. Another initiative, known as FireMash was undertaken in Australia, to involve citizens in reporting bushfires in a region [57], with the objective of early management and relief.

Since mobile phones have become ubiquitous, pervasive and users are connected via SMS services, instant messaging apps, and social media, there lies a potential of building a network of Good Samaritans, who would be actively involved in providing relief and assistance during an emergency event. The system can be uniquely tailored for different target audiences, such as drug users, people with disabilities and other at-risk communities. The core goal of the network is to collaborate and provide relief. We delve deeper into this crowdsourcing aspect in Chapter 2.

1.5 PAL: Crowdsourced Emergency Response System

In this thesis, we study the backend infrastructure of a **crowdsourced emergency response system**, known as **PAL or Peer Alerting Lifeline**, which can be used to mitigate the harm caused due to the barriers of access to traditional emergency services. PAL represents a deviation from the conventional centralized emergency response model to a decentralized one, where citizens act as emergency responders.

A PAL solution would allow the two following key functionalities:

• The system would allow connecting Good Samaritans to ongoing emergency events in their vicinity and thus, crowdsource emergency services. The method of relief from the crowd could vary, ranging from providing information to administrating harm reduction services.

• Apart from emergency support, the system would also work towards mitigating social and psychological isolation, by allowing continuous real-time communication between a user and their trusted support group.

In the context of the opioid users, the system is called as **NALPAL** or **Naloxone Peer Alerting Lifeline**.

1.6 Research Goals

Our primary research goal in this thesis is to *study the backend infrastructure of the crowdsourced emergency response component of a Peer Alerting Lifeline Application*. Through this thesis, we wish to:

- Identify the key design decisions to be made, while building the backend infrastructure of such a service.
- Recommend solutions to the identified key design decisions.
- Evaluate our design by building a prototype of the system.

Our study is conducted, in the context of the opioid crisis and hence our solution is called NALPAL or Naloxone Peer Alerting Lifeline. In the subsequent Sections and Chapters, PAL and NALPAL can be used interchangeably. It is also important to note that our primary focus is the application infrastructure to build such a crowd based emergency response system.

1.7 Thesis Outline

Chapter 2 provides background work on the concepts of crowdsourcing and provides an overview of crowd-based emergency response system deployed during the Haiti earthquake. *Chapter 3* discusses the requirements of the PAL system, its fundamental design decisions and provides recommendations for the same. *Chapter 4* provides an evaluation of the design recommendations made in *Chapter 3*, by building a functional prototype of the system. *Chapter 5* summarizes our work, elicits our contributions and provides recommendations for future work.

1.8 Summary

In this Chapter, we provided an overview of the problem and motivation for building a PAL solution. We discussed the context of this thesis's work, i.e., the opioid crisis in North America and provided an introduction to the domain of crowdsourcing in emergency response. In the next Section, we discuss the background and related work on crowdsourcing.

Chapter 2

Background and Related Work

In this Chapter, we introduce the concept of crowdsourcing. We discuss the background work done on defining a crowdsourced initiative and its various components. We also present the overview of an application, which uses crowdsourcing, in the emergency response management domain. Our work in this Chapter forms the basis for our system's core values.

2.1 Crowdsourcing

The word crowdsourcing is formed of two words: crowd, which represents individuals which are connected; sourcing, meaning engaging such individuals in the procurement of goods and services [87]. The first formal reference to such a system was done by Howe et al., where crowdsourcing was described as a mechanism of outsourcing work, which is traditionally done by a company or institution, to a network of people, i.e., the "crowd," via an open call [88]. The crowd may perform this task either individually or in collaboration with each other. In Howe's world, the entire process is governed by an organization/institution, with the idea of producing a good/service for profit [88]. Post Howe's work, many attempts have been made by different authors, to define and categorize such systems.

Brabham [89] described crowdsourcing as a virtual, distributed problem solving and production mechanism. He categorizes it as a movement from an expert individual performing a particular task, to a group of individuals who are assigned with the same job. His work talks about harvesting distributed intellect, through the hypothesis that groups are often more intelligent than the smartest person in the group [89].

For harvesting the wisdom of the crowd, Brabham discusses a crowdsourced system to use the following specifications in its design of the crowd interaction [89]:

1. **Diversity of Opinion:** The system should allow the unique identity and perspectives of the crowd to be expressed through a common platform**.**

2. **Independence:** The participants from the crowd should be able to contribute to the system independently**.**

3. **Decentralization:** The system should be decentralized based on the crowd's geographical and identity diversity**.**

4. **Aggregation:** The system allows collection of the diverse contributions of the crowd.

Brabham further suggests the use of web-based systems for achieving the attributes mentioned above, with its applicability to not only business but also to non-profit domains. Brabham's work focused primarily on the crowds that form a part of the crowdsourcing systems and what features should the networks have to harvest their collective intelligence. A detailed analysis of broader challenges of the crowdsourcing systems on the web was done by Doan et al., whose focus was to categorize the types of crowdsourcing systems [90]. Doan et al. described four key challenges that have to be addressed while designing a crowdsourced system [90]:

1. **How to recruit contributors:** This is important since it is crucial to decide who are the contributors to a particular crowdsourcing system are going to be and how they will be able to access and utilize the system**.**

2. **What the contributors do:** This is essential to design a system which has a clear task to be performed by the crowd**.**

3. **How to combine the contributions:** A crowdsourced system focuses on harnessing the collective work of the group and hence, must have a mechanism to be able to aggregate the contributions**.**

4. **How to manage abuse:** Since the system is large, decentralized and crowds may interact independently with each other, it is essential to be able to handle abuse or exclusion of members of the crowd. Brabham also alluded to this, as to how subgroups in the group may lead to bullying behavior and silencing of other subsets within the crowd [89].

Furthermore, Doan et. al classified the crowdsourced systems, across 9 dimensions, as mentioned below [90]:

1. **Nature of collaboration:** Doan et al. argues that the collaboration with the crowd can be Explicit or Implicit. Explicit collaboration is when the crowd is aware of the problem that they are trying to solve. Implicit collaboration typically occurs, when the crowd's primary task is not to solve a problem at hand, but system designers use the information provided by the crowd for achieving a goal. Video games are a good

example of implicit collaboration, where players implicitly give feedback to the system.

2. **Target problems:** The target problems in the systems may also vary, ranging from solving a technical challenge to creating a new product. The target problem may either be explicitly visible to the crowd or maybe implicitly part of a larger system, such as video games.

3. **Degree of manual effort:** This refers to the degree of manual effort that may be needed to set up the crowdsourcing system. The effort may vary based on the design of the system, ranging from the automatic setup in case of networked software applications to completely manual.

4. **Role of human users:** Doan et al. describes four types of roles that human users can play in a crowdsourced system: *Slaves*, who solve the problems, which have been specified, in a divide and conquer manner. *Perspective providers*, where human users provide perspectives such as reviewing a book, movie. *Content providers*, where human users create content, like for online video streaming sites. *Component providers*, where human users are a component of a target artifact, such as social media networks.

5. **Standalone versus Piggyback:** This is another important consideration for whether the system is a stand-alone system in itself or does it piggyback on existing systems.

6. **How are users recruited:** Doan et al. describes five mechanisms that may be used: *Requirement based*, where the users are mandated to participate via an authority. An example would be a manager and their employees. *Pay based*, where users are paid for participation. Amazon's Mechanical Turk system [91] is an example for the same. *Volunteer-based*, where the users volunteer to become a part of the crowd. *Pay for service based*, in which a user can use a service A only after contributing to a service B. Captcha [92] is an example of such a solution. Finally, it may be via *Piggybacking on existing systems*. An example would be using twitter feeds [93] for sentiment and geographical event analysis.

7. **How are users contributing:** This is another critical dimension of the crowdsourced systems, to understand how users are contributing to the system. For the same, Doan et al. specifies the following four considerations: *Cognitive ability*, which is how cognitively demanding the task is. It is essential to design a system, which can account

for contributions from varying degree of cognitive skills of the crowd. *Impact of contributions*, which is how important a user's input is. Systems have to be designed to account for a spectrum of risk and reward associated with a user's contribution. *Machine contribution*, which means the system should be able to differentiate between human and machine contributions and users should be assigned tasks which are harder for machines. Finally, the *usability of the user interface*. This is important for the users to be able to contribute to the system effectively.

8. **How are user's contributions combined:** This is an essential aspect for allowing crowd's work to come together and actually to assist in a task. Both manual and automated systems have been developed and may be used to combine the contributions of the crowd for a target problem.

9. **How are user's contributions evaluated:** It is essential for crowdsourcing systems to be able to evaluate and hence, identify and remove users, who are not contributing to the system. Furthermore, explicit reward systems may be needed to be set up to motivate the crowd further.

Doan et al. provides us with essential insights into *describing* the different features of a crowdsourced system.

Until now, we have considered crowdsourcing definitions which looks at crowdsourcing solutions in a for-profit domain, such as businesses who wish to use this methodology to solve their problems. Buecheler et al. provides us insight into using crowdsourcing in the non-profit field, vis-a-vis, scientific research [94]. Their work focuses on developing a framework for crowdsource-ability of a scientific research problem. According to Buecheler et al., crowdsourcing in the scientific domain is looked as a problem for harnessing collective intelligence and ability of individuals, for performing tasks in a research process. For leveraging the collective knowledge, they utilize the collective intelligence gene framework by Massachusetts Institute of Technology (MIT) [95] and artificial intelligence's "Three Constituents Principal" [96], for modeling intelligent crowd behavior for a task at hand. The framework is shown in Figure 2 [94]:

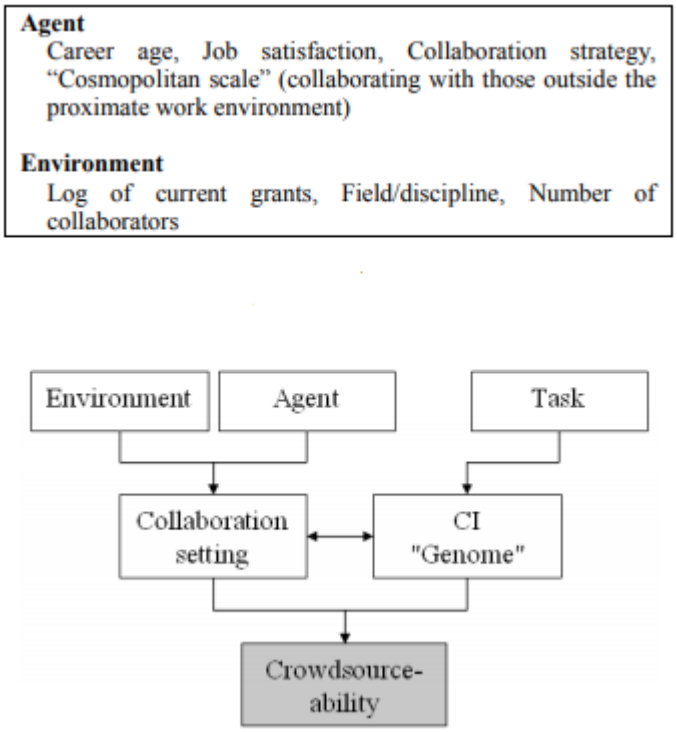

Figure 2: Overview of Schematic Elements of Crowdsource-ability

The framework can be applied to evaluate the crowdsource-ability of a task during a research process. This framework is one of the first examples of using crowdsourcing in a non-profit domain.

A holistic survey of existing crowdsourcing systems was done by Estellés-Arolas et al. [87], with the objective of obtaining a unified definition. Their work identified three primary actors in a crowdsourced system, from where the following eight characteristics were determined [87]:

About the Crowd:

- Who forms it
- What it has to do
- What it gets in return

About the Initiator:

- Who it is
- What it gets in return for the work of the crowd

About the Process:

- The type of process it is
- The type of call
- The medium used

Based on the above characteristics, and analysis of existing literature on the same, the following definition was proposed by Estellés-Arolas et. al. [87]:

"Crowdsourcing is a type of participative online activity in which an individual, an institution, a non-profit organization, or company proposes to a group of individuals of varying knowledge, heterogeneity, and number, via a flexible open call, the voluntary undertaking of a task. The undertaking of the task, of variable complexity and modularity, and in which the crowd should participate bringing their work, money, knowledge and/or experience, always entails mutual benefit. The user will receive the satisfaction of a given type of need, be it economic, social recognition, self-esteem, or the development of individual skills, while the crowdsourcer will obtain and utilize to their advantage what the user has brought to the venture, whose form will depend on the type of activity undertaken"

In defining our system in the next Chapter, we have used this definition [87] and have explained our system, by describing its eight characteristics. We believe that this definition allows us to define our system, in an abstract yet sufficiently detailed manner. In the next section, we discuss a project that combined the power of crowdsourcing during an emergency event. .

2.2 Crowdsourcing Emergency Response: Haiti Earthquake

One of the earliest examples of crowd based assistance in emergency response presented itself during the earthquake in Haiti in 2010. On 7th January 2010, a 7.0 magnitude earthquake struck Haiti, in which more than 230,000 people died, and there was mass destruction in various regions of the country [97].

Many local and international organizations attempted to combine their efforts to provide relief to affected individuals and areas. Two of the key challenges that were faced by the relief agencies was a lack of information regarding Haiti's terrain and the status and location of people. The existing system, which primarily focused on information sharing

among the first responders and volunteers from international organizations, failed to aggregate and prioritize data which came from various sources. The system also was initially failing to take into account the information that was being provided by the on-ground Haitian community representatives [97].

Ushahidi [99], which was initially developed in Kenya, as a mechanism for citizens to report incidents of violence post-election, was then conceptualized to be used in this situation. The system was used to aggregate and visualize essential information from individuals on the ground to assist the emergency responders in building effective relief programs [97]. The operation began, by gathering information from various sources, such as Twitter, Facebook, and blogs. The data, which was deemed useful by a group of volunteers and could be geotagged, was then mapped using Google Earth [100] and OpenStreetMap [101] and was available for viewing on Ushahidi's Haiti website.

The incoming information was primarily resulting from digital media such as social networks and email. To directly involve the people on the ground, a collaboration was set up with FrontlineSMS [109], which implemented an SMS based system. The system allowed the people to report their situation along with their location. The system, known as the 4636 project [97], then became an indirect mode of communication, between Haitians and First Responders. The information received from the SMS'es was then mapped to the Ushahidi-Haiti platform and was used by the emergency responders, to organize their relief and rescue efforts. Since the messages that were received were in the local language, and they further required geocoding, a network of more than 1000 volunteers was used, for translating and geocoding the message.

Another important aspect of this effort was to have an accurate map since there was a scarcity of detailed maps of pre-earthquake Haiti. To accomplish this, OpenStreetMap platform [101] was used. Information from various government and non-government organizations was used, by a group of volunteers, for mapping the post-earthquake Haitian terrain.

These efforts resulted in guiding the emergency responders, to plan their efforts to provide relief to the Haitian citizens. Via the 4636 project, the volunteers translated more than 25,000 messages, which was then used to build a crisis map of the on-ground situation in Haiti [97].

Various types of individuals formed the crowd of this crowdsourced solution. The first was the Haitians on the ground, who were the source of information on their state and location. The second was the group of volunteers, which were involved in mapping, translating and geotagging the incoming data and thus were effectively building the crisis

map for post-earthquake Haiti. These sets of individuals provided essential information to the disaster response teams and therefore allowed them to plan their interventions effectively.

Ushahidi-Haiti initiative was a first of its kind project, in which the wisdom and ability of the crowds were brought together, to provide improved emergency response service to Haiti's Citizens. The primary role of the crowd in this scenario was collecting, processing and presenting information in a manner which could inform the relief efforts of the emergency services. Since then, Ushahidi has grown to become a disaster information crowdsourcing and mapping platform and is used for many such efforts across countries.

Ushahidi provides us with an example where the crowds came together during an emergency event, and used the available technology, to assist in relief efforts. We wish to capture the same spirit of collaboration in building a PAL system, where once again the crowds come together to provide relief to individuals who are facing an emergency. Though, unlike Ushahidi, our system would require direct interaction of the crowd and the initiator.

2.3 Crowd based or Peer based?

One of the underlying assumptions for a system to be crowd based, is the size and independence anonymous operation of the individuals who form a part of the crowd. In our system, we envision the system to leverage a relatively small group of individuals, who form the support group of the user. Hence, the word *Peer*, in Peer Alerting Lifeline.

Having said that, the system is similar to a crowdsourced one, since we are trying to harness the wisdom and capability of the crowd to come together and provide relief during an emergency operation. Although the *crowd* is based of a limited set of *peers* of the individual, in principal we believe that it follows the principal of decentralizing emergency response from a single entity to a group of individuals. Hence, we also call it a *crowdsourced emergency response system.* But, we do recognize how it deviates from traditional crowdsourcing definitions.

2.4 Summary

In this Chapter, we looked at the various attempts that have been made to define a crowdsourced system. We discussed the example of Ushahidi project, which was a first in its kind initiative involving the crowd in assisting emergency response services. We delved into detail on the mechanism of crowd interaction in Ushahidi and established the core ethos on which our system is built.

Chapter 3

PAL Design

In this Chapter, we discuss the design of our crowdsourcing based PAL solution. We describe PAL based on our work in Chapter 2. We identify the critical design decisions that are part of developing the system. We evaluate the various available design choices and subsequently, provide recommendations for building the backend infrastructure of the system.

3.1 System Definition

To describe the various elements of the system, we use the characteristics defined by Estellés-Arolas et al. [87]

About the Crowd:

- **Who forms it**: The crowd in our PAL system are the users with two key characteristics. First, the users have the training of using the Naloxone kit and are skilled in providing harm reduction in case of an emergency. Second, the users are willing to become a part of an emergency peer support network. Users with the ability to transport themselves would be especially useful, as they would then be able to reach emergency events which are farther from their locations. Also, we assume that the users have mobile phones available to them. The users may also be friends/family of the substance users.

- **What it has to do:** The primary role of the crowd in the system would be to provide harm reduction service during an ongoing emergency event, in a region that is accessible. The responders would be responsible for reaching the place where the emergency is taking place and administering the harm reduction drug or any other service that is needed. They would also be responsible for communicating with the initiator during an emergency, for obtaining the exact location or any additional information that may be relevant.

- **What it gets in return:** Based on Maslow's needs triangle [102], the system provides the crowd with an opportunity to boost self-esteem and self-fulfillment, since the system offers a chance to save lives, potentially of their loved ones.

About the Initiator:

- **Who it is**: The initiator could either be the person who is facing an emergency situation or a witness to an emergency event.
- **What it gets in return for the work of the crowd:** The system allows access to harm reduction service, during an ongoing emergency, to either the initiator directly or to an opioid user, near the initiator.

About the Process:

- **The type of process it is**: The process of our system is of decentralized problem-solving.
- **The type of call:** The call is limited to the community of peer group that the initiator has made, for providing emergency response services. In the future, we wish to make this an open call, to all the users registered in the system and who can respond to emergency events happening in their vicinity.
- **The medium used:** Two primary mediums are used to make the call: Internet based Messaging and Short Messaging Service. Short Messaging Service is used as a failover mechanism; in case the internet facilities are not available. We discuss this in further detail in the subsequent sections.

3.2 System Goal

The system was envisioned with the following PAL's core goal in mind:

"To provide a reliable mechanism of accessing emergency services, via a crowd made up of a peer support group of the emergency initiator."

To validate our idea and assumptions of our solution, we took the assistance of Aids Vancouver Island, a Non-Profit Organization working towards ensuring well being amongst the people affected by substance use [27]. Upon extensive discussions with AVI, the following features of the system were identified:

- **The system should be accessible via mobile application:** Many alternatives such as website, a fixed kiosk at centers were discussed. The ubiquitous nature of mobile devices and their pervasiveness make them the ideal form factor for building our application interface.

- **The mobile application should be agnostic to the type of user.** This is because, in many situations, substance users are also trained in using Naloxone and have the kit available. Hence, the system should not make a distinction and should have the same uniform interface. The system should allow a user to update if they have a Naloxone kit available.

- **The mobile application should publish dope guides and other informational material.** AVI suggested that their informational material, such as their dope guides which have information about potentially harmful drugs, be also published in the mobile application. This would allow dissemination of useful information to the users, so that they may be able to consume substances safely.

- **There should be a real-time communication, between the crowd responders and initiator in case of an emergency situation.** This is important because information may be needed to be exchanged, such as house number, flat number and estimated time of arrival of the responder.

- **The exchange of messages between the users should not make them vulnerable to prosecution.** The messages between the users should be saved only for a short period within the mobile application and not persisted in our backend servers, to avoid any legal ramifications to the users and the system.

- **The system should be able to work in no internet environments.** AVI identified this as a critical barrier to implementation since the users on the street and in shelter homes often do not have internet and data packs available.

The following high level requirements of the system were identified:

1. ***REQ 1***: A user should be able to set up their dedicated profile and add or remove users from the virtual support group. A user should also be able to mention the availability of Naloxone kit on their profile.

2. ***REQ 2****:* A user should be able to communicate with their virtual support group, both through one-to-one and group conversation. The conversation should be off the record, i.e., no history of the communication should be saved on the server or the mobile device.

3. ***REQ 3****:* A user should be able to communicate an active emergency event with their support group, via the sharing of their location. The support group members should

be able to respond to the emergency, via updating their arrival time. A real-time communication channel should be maintained between the initiator and the responders during the emergency.

4. ***REQ 4**:* The system's emergency response service should be functional with no internet and with no change in the interaction of the user with the mobile application.

5. ***REQ 5**:* A user should be able to get informational updates, such as dope guides published by AVI containing information about harmful and adulterated substances, to assist them in consuming safely and leading healthier lives.

Later, we use the above high level requirements to evaluate the prototype backend build of our system.

3.3 PAL Emergency Life Cycle

Figure 3 shows the various lifecycle stages of an emergency event and their transitions:

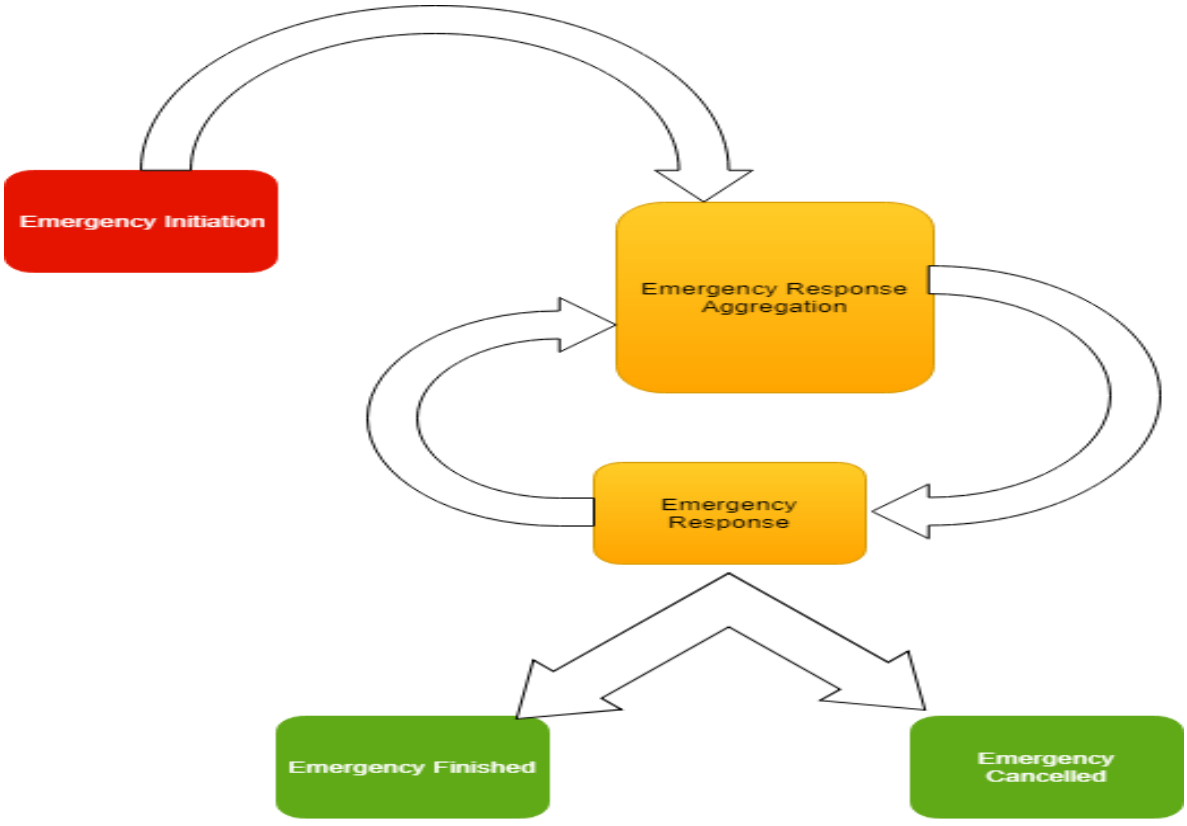

Figure 3: Emergency Lifecycle

- **Emergency Initiation:** Emergency Initiation is done by the person who either themselves are going through or are witnessing an emergency event. The person either uses their mobile phone or the mobile phone of the person at risk, to initiate an emergency event call to the virtual support group. The system notifies the crowd and maintains information about the emergency.

- **Emergency Response Aggregation:** The system moves into this stage, as soon as a user replies to the emergency. The system keeps track of the responses to the crisis. For affirmative responses, the system informs the initiator with the information and distance of the responder.

- **Emergency Response:** The system moves into this phase when the first affirmative response comes to the system. The Emergency Response and Emergency Response Aggregation phases occur in parallel, with system constantly aggregating responses. During the emergency response phase, the responders communicate in real time with the initiator.

- **Emergency Finished:** This phase can only be initiated from the mobile phone of the initiator. This is done when the required assistance has been provided to the emergency event.

- **Emergency Cancelled:** The initiator can only initiate this phase; in case the emergency event was an incorrect/not required one.

3.4 Design Questions

Based on the requirements and the emergency life cycle, we identified the following important design questions for the PAL system:

- ***DQ 1:*** **Mobile Application Development Platform:** Since the system requires the building of a mobile application, this design question focuses on deciding the mechanism for coding and deploying it. The development of a mobile app can be done via two mechanisms: Native and Hybrid [29,33]. In the case of Native application development, the developer(s) has to use the programming language of the target mobile platform, e.g., Java in case of Android [119] and Swift in case of Apple [120]. On the other hand, Hybrid mobile development follows the paradigm of “Write once, run everywhere.” Hybrid development platforms typically use a single programming language (such as Javascript [84]), for writing the application code once. The same code

can then be built and run on different mobile device's operating systems. A famous example would be React Native [28].

- ***DQ 2:*** **Backend Server:** The PAL application would require the setup of a backend application server. This is needed as the users need to set up a dedicated profile for themselves and also, maintain information about friends and active emergencies. This question focused on the choice of the backend server development framework. Many options currently exist, Spring [121] and Node.js [34] being some of the popular ones.

- ***DQ 3:*** **Database store:** The PAL application requires a database store, for storing information about users, their friends and emergencies. Many database choices exist, ranging from traditional Relational Databases to NoSQL databases [36].

- ***DQ 4:*** **Messaging Infrastructure:** This is an important design question since a central part of our application is the communication between an initiator and their peer support group, which needs to be supported by an underlying messaging layer.

- ***DQ 5:*** **Offline System Availability:** This is an important design question since we have identified no internet service as a barrier to accessing our system.

Our thesis focuses on the backend infrastructure for the following capabilities:

1. **The capability for enabling the real-time interaction between the initiator and the crowd**.

2. **The capability for the crowdsourced emergency service to be available in no internet situation**.

Therefore, our thesis focuses on **DQ 4** and **DQ 5** and provides recommendations for their implementation.

3.5 DQ 4: Messaging Infrastructure

One of the essential requirements for the PAL system is the implementation of a communication mechanism between the user. The communication would be needed for the following purposes:

1. For enabling a communication channel between an emergency event initiator and emergency responders, enabling real-time communication between the two during an emergency event.

2. For one-to-one and group instant messaging scenarios, in the application's normal operation.

For the setup of messaging infrastructure, we studied and evaluated the various protocols that currently exist. The following protocols were evaluated, for their appropriateness in building our PAL solution:

1. XMPP: Extensible Messaging and Presence Protocol
2. Matrix
3. Tox
4. MQTT: MQ Telemetry Transport

To test their appropriateness, we have derived the following sets of MUST HAVE and SHOULD HAVE, based on the requirements for the system:

1. ***MH1: The protocol MUST allow off the record messaging.*** On discussions with AVI, we recognized that the exchange of messages among our target users could make them and the system susceptible to legal action. For the same, the protocol must allow the messages not to be stored on the servers and phones.
2. ***MH2: The protocol MUST allow consistent user management across devices.*** This is important as we do not want our users to lose their profile and information about their support group if they log in to the application from different devices.
3. ***MH3: The protocol MUST allow for friend discovery and management in the network.*** Since our solution requires the setup of a virtual emergency and peer support group, the protocol must allow a simple mechanism to search and add friends.
4. ***MH4: The protocol MUST allow provisions for message reliability mechanisms.*** Since we want our system to be reliable in emergencies, it is essential for the chosen protocol to define failover mechanisms in case of message delivery failure.
5. ***MH5: The protocol MUST have end-to-end encryption.*** We enforce this to ensure the privacy of communication between the users.
6. ***MH6: The protocol MUST support both one-to-one and group messaging***. The communication between the users and their virtual support group can be both one-to-one and group.

7. ***SH1: The protocol SHOULD have Software Development Kits (SDK's) for its various components available for development.*** This is necessary so that our development does not need to focus on building the low-level implementation of the protocol.

3.5.1 Matrix

The Matrix protocol provides a real-time communication mechanism, for publishing, persisting and subscribing data, over a global federation of servers, with no single point of control. The protocol can be used for Instant Messaging, Voice over IP, Internet of Things and bridging together existing communication silos, providing the basis of a new open real-time communication mechanism [54].

The Matrix protocol focuses on synchronizing data between a set of users, through their dedicated servers, known as homeservers. The users can be people, devices or services. An exciting aspect of Matrix is that the connected servers can be chosen, built and deployed by the users themselves. This thus then results in a federated infrastructure, in which different implementation of servers, can still provide communication between two entities. This is similar to how users can send email between different email service providers.

3.5.1.1 Architecture

Synchronization of information between the users is achieved, by using a set of JSON [106] over REST [7] Application Programming Interfaces (API). The JSON objects are used to synchronize state and data between the users. The users synchronize their communication state and data with their respective homeservers, which are then responsible for updating the connected homeservers.

Each user is connected to a homeserver. Users who wish to communicate with each other, come together to form a logical entry known as a "room." Each instance of communication inside a room is called an "event." Figure 4 shows the communication mechanism between 2 clients A and B.

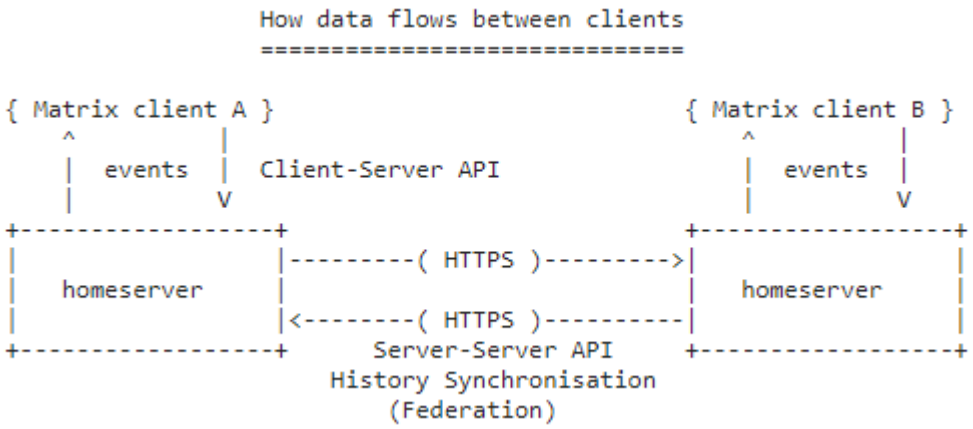

Figure 4: Matrix interactions [54]

A room can be made of a single homeserver, with multiple clients communicating with each other or can be made up of many homeservers, with each homeserver having a certain number of clients. The events in a room are synchronized between the homeservers and their respective clients, by using the "Server-Server API" and "Client Server API" respectively [54]. Users have the flexibility to set up their custom homeservers or can use the available Matrix homeserver implementations [55].

A client action, such as sending a message, corresponds to a *message* event in the Matrix system. Each homeserver maintains the event history for a particular client, through directed acyclic graphs, known as the event graphs [54]. The event graphs, when needed, are then updated across other homeservers in a particular room, using the "Server-Server API."

A flow diagram of API calls in sending information between client A and client B is shown in Figure 5:

Figure 5: Matrix Room Interaction [54]

An important aspect to notice here is the receiver of the message makes the HTTP GET request. The receiving of messages in the Matrix protocol is done via polling a long-lived GET connection with the homeserver of the receiver.

3.5.1.2 Evaluation

In this section, we evaluate the protocol based on the criteria defined:

MH1: The protocol MUST allow off the record messaging

The Matrix protocol relies on synchronization of information between homeservers. This mechanism ensures that if a new homeserver/ client connects to a room at a later stage, the message history is not lost and can be synchronized with the new device. The new device uses the client-server API for synchronization of messages from the homeserver to its local system. Messages, which are coined as “events,” are stored within each homeserver in a directed acyclic graph data structure, to preserve the chronological ordering of the messages. The history is synchronized to all the homeservers that have joined or will join a particular room in the future, using the server-server API. Thus, the message history is preserved in the homeservers.

As a potential solution, some server implementation(s) [110] do have a purge API, that can be used to remove event (message) history. However, this control would only be available to the admin, who will then have to monitor, on a real-time basis, all the message events, capture their event id, wait for delivery of the message and then, call the purge API, to keep the database clean of the events. The process is tedious and would increase the bandwidth on the server. Additionally, not all Matrix homeserver implementations support this mechanism.

MH2: The protocol MUST allow consistent user management across devices

In the Matrix system, each user is connected to the network via their homeservers. Each user's profile information is stored in the homeserver and has a unique Matrix ID associated with it. The Client-Server API provides many authentication mechanisms, such as password based, token, OAuth2 (login via social network). By default, the user can register themselves on the homeserver using their username, email, and password. The user information is then connected to the Matrix ID and thus, even if the user logs in from different devices, the user identity and data is maintained and can be synchronized from the homeserver.

MH3: The protocol MUST allow for friend discovery and management in the network

In the Matrix system, users are connected via the presence of virtual rooms. Each room can be used either for one-to-one or for group communication. A user's Matrix ID is namespaced in the context of their homeserver and thus, can be used by a client to search for and find another user in the system and to set up a virtual room for enabling communication between them. The information about the friends, which are the different rooms and their participants, is maintained by a user's homeserver. The data can be maintained across devices. On a login from the new device, the information can be synced from the homeserver.

For adding friends, since each user has a consistent Matrix ID associated with them, the same can be used for searching and adding friends to a room. A list of such rooms forms the friend list for a user. Some homeserver implementations also allow using linked information, such as email and mobile, to search and add friends to a room.

MH4: The protocol MUST specify provisions for message reliability mechanisms

In the Matrix system, message delivery occurs via synchronization through the users/devices that are connected to a particular homeserver. The messages are stored in the homeservers, and hence, the messages are not lost in case the that a recipient is not

available. When the recipient comes online, the history of messages can be synced from the homeserver and hence ensures reliability in delivery. Furthermore, since the user management is consistent across devices, this also ensures that messages are delivered irrespective of where the user is logged in from.

MH5: The protocol MUST have End to End encryption

The Matrix protocol libraries currently have the end-to-end encryption feature available, but it is now in their late Beta phase. The rollout is expected by the end of this year.

MH6: The protocol MUST support both one-to-one and group messaging

The Matrix protocol uses a virtual room as the context for users to communicate with each other, via their respective homeservers. A room can have two users connected or have multiple users/homeservers as part of the room. This allows for both one-to-one and group messaging.

SH1: The protocol SHOULD have SDK's for its various components openly available for development.

The Matrix protocol has a client and homeserver component. Client-side SDK's are currently available for many programming languages, such as Javascript [84], Python [103]. The SDK's have an active developer community with regular improvements being pushed. Matrix has many homeserver implementations, with the most famous being Synapse [110], which is available for deployment to both on-premise and cloud platforms.

3.5.2 Tox

Tox protocol started as a response to Edward Snowden's leaks regarding the NSA spying activity [58]. With the aim to ensure authenticated and private communication mechanism between users, Tox was developed to be distributed, peer-to-peer and end-to-end encrypted. The protocol aims to be independent of any server infrastructure and provides mechanisms for connecting to a peer network via bootstrapping [59].

Tox relies on using the network address of a user, as the mechanism for searching, identifying and connecting to a peer machine. Tox maintains a distributed hash table data structure for managing information about the connected peers in the network. Once the connection is established, Tox uses dedicated secure channels between the users for communicating text, audio, video and file transfers.

3.5.2.1 Architecture

Each node within the network has associated with it a set of public and private keys. The public and private keys are generated, as part of the initial setup of the node and rely on the Networking and Cryptography library (NACL) for their generation [60]. The set of keys play an essential role in the communication since the packets of connection are encrypted with the public key of the receiver and the secret key of the sender. This is possible by generating a combined key, which is built from the two key pairs of the communicating nodes (SK1, PK1) and (SK2, PK2) [58].

Tox protocol relies heavily on using distributed hash tables (DHT) for searching, identifying and connecting the nodes within a network. The protocol uses a simplified version of the Bit Torrent's "distributed sloppy hash table," which is an implementation of Kademila [61] and uses User Datagram Protocol (UDP) for its functioning [59].

It is crucial in the context, to understand how the DHT is self-organized in a Tox network. Each Tox node has an ephemeral key pair, known as the DHT public and private keys. The DHT public keys are obtained from the network address of the node and hence, change if the node restarts. The DHT public key acts as the unique address for the node. Each node has a client list, which is a list of nodes which are nearby the current node. The size of the list is fixed, and the distance between the nodes is calculated by calculating the XOR of their DHT public keys [59].

The look-up mechanism for searching for a Tox node starts from an initial Tox node, knows as the bootstrap node. If the node has information of the peer being searched, the connection information is returned. Otherwise, it forwards the query to the *k* closest DHT nodes. This process continues until the connection information is retrieved. The connection information can then be used by the initiating Tox node to connect to a peer [58]. Figure 6 shows the initial Tox Network Setup using a bootstrap node.

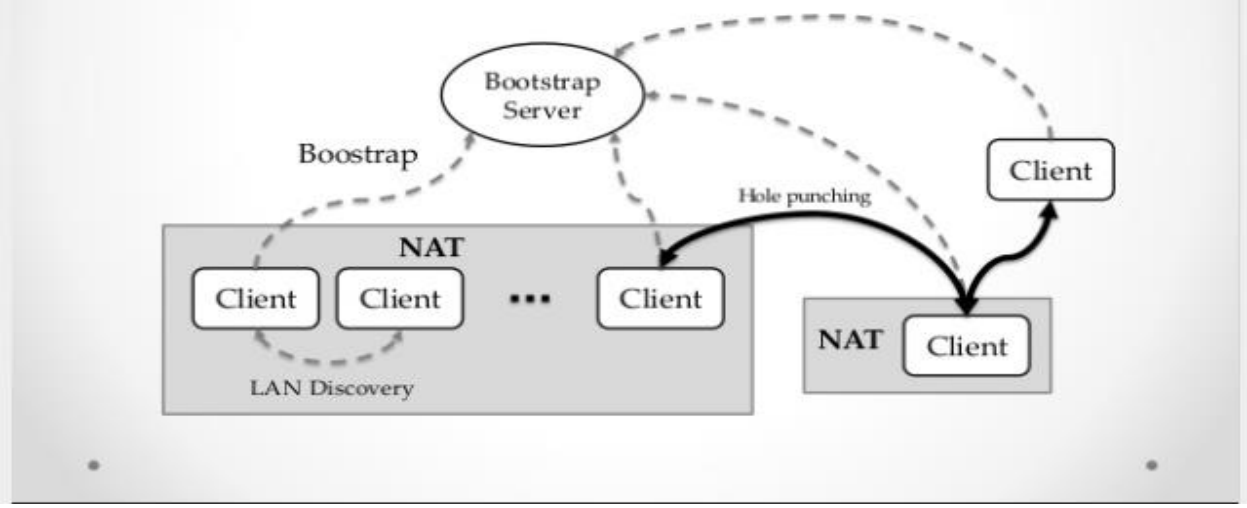

Figure 6: Tox Network Setup [59]

3.5.2.2 Evaluation

In this section, we evaluate the protocol based on the criteria defined:

MH1: The protocol MUST allow off the record messaging

The Tox protocol is a distributed peer-to-peer protocol. The messages are sent between the clients that are directly connected and hence, are not stored at a central server. The user devices can implement deletion of messages from storage after a certain amount of time.

MH2: The protocol MUST allow consistent user management across devices

Tox is a peer to peer system, where a user is identified and added as a friend, by the use of a TOX ID. The format of a TOX ID is shown in Figure 7:

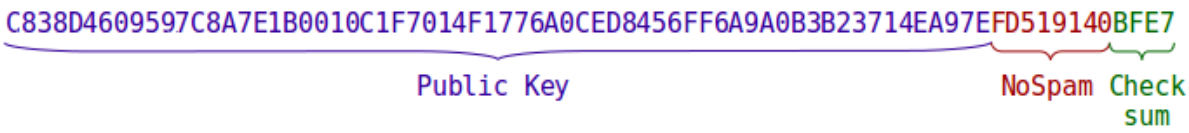

Figure 7: TOX ID Format [58]

The public key here refers to the long-term public key of the user being contacted. Apart from TOX ID, there is no provision of maintaining any other profile information and would require deployment of custom server and database for doing the same.

MH3: The protocol MUST allow for friend discovery and management in the network

In the case of the Tox system, there is no central server for managing the friend list of a user. Thus, a user may lose the ability to connect to a friend, if the user connects from a new device. Custom server and database would once again be needed to manage friend lists.

Friend discovery in Tox can be done by using the TOX ID and searching the user using the Tox's DHT mechanism.

MH4: **The protocol MUST specify provisions for message reliability mechanisms**

The Tox protocol communicates messages between peers by developing a direct communication channel between the sender and the receiver. In case the receiver is not available, the client software has to build their mechanisms to ensure delivery. Furthermore, message delivery may fail because a user's Tox ID may have changed due to a change in their network address. Thus, there might be no way in that scenario to ensure the delivery of the message to the user. Custom server and database would be needed to maintain friend lists and to update the new TOX ID's.

MH5: The protocol MUST have End to End encryption

The Tox protocol uses Combined Key, generated from the NACL library, to end-to-end encrypt messages. The combined key enables a symmetric encryption mechanism for the communicating peers to encrypt/decrypt information.

MH6: The protocol MUST support both one-to-one and group messaging

Apart from peer-to-peer communication, the Toxcore library allows group chat, by setting up a network of interconnected peers.

SH1: The protocol SHOULD have SDK's for its various components openly available for development.

The Toxcore implementations are available in many programming languages such as Java [111], Python [112] and Javascript [113] and are in under active development.

3.5.3 XMPP: Extensible Messaging and Presence Protocol

The inception of the XMPP protocol dates back to 1999 when Jabber [114] was created, which was an open technology for instant messaging and presence. The next two decades saw widespread adoption among the Internet Engineering and Task Force(IETF) [122], with the formalization of its core and instant messaging specifications [64,65]. Many popular messaging services, such as Firebase Cloud Messaging [38] and Apple Push Notifications [83], are now using this or a customized version of this protocol, to build their messaging infrastructure. One of the currently popular instant messaging client, Whatsapp [66], uses a variation of this protocol, called as FunXMPP [67], for providing their instant messaging services.

XMPP is a server-centric protocol. The server can either be a single central server or a network of federated servers, thus making it interoperable with various service providers, similar to electronic mail services. XMPP relies on maintaining an open Transmission Control Protocol (TCP) [112] connection between the clients and servers and between two servers (if required), to provide a continuous stream of messages between two connected clients. XMPP also provides mechanisms for channel and data level security, with support for end-to-end encryption.

3.5.3.1 Architecture

The standard XMPP protocol specification is broadly divided into two parts: The XMPP Core specification [64], which explains the mechanisms for connection and communication between clients and servers. The XMPP Instant Message and Presence Specification [65], which, builds upon the XMPP Core specification, to provide a more detailed protocol for instant messaging.

XMPP works on a distributed client-server architecture, where a client needs to connect to a server to communicate with other clients within the network. In a typical scenario, a client connects to a server via opening a TCP connection with the server. The server then authenticates the client, sets up a secure channel of communication with the client and is ready to accept/transmit messages. It is the responsibility of the server to route the incoming messages to the correct recipient, which is directly connected to the same server as well. If the recipient is not directly connected but is connected to another server, server-server communication channels are used to deliver the messages to the intended recipient. The communication between the various entities is done through data streams [64].

Each client in the XMPP protocol has a unique identification, known as JID or Jabber ID, which is used to identify them by the servers. A first-time client first initiates a connection with the XMPP server, using a long-lived TCP connection. The client needs to know the network information for the server, to be able to establish the initial contact. Once the connection is established, the server performs authentication and authorization on the client to ensure the identity. Upon completion of the security check, the client and server are now able to exchange messages, in the form of Extensible Markup Language (XML) [118] stanzas, using a dedicated XML stream [64].

An XML stream is a container, for the exchange of XML Messages between any two entities of the network. The start of an XML stream occurs by an opening “stream header” `<stream>`, while the end of the stream is denoted by a `</stream>` tag. At the initiation of a client in the XMPP network, an initial unidirectional XML stream is set up, which is

used to perform the initial security check via TLS or SASL negotiation [69]. After the negotiation is completed, the server sets up another XML stream from the server to the client, called as the "response stream," to begin the exchange of continuous messages between the two entities.

The communication on an XML stream occurs, by small units of information, called as the XML stanzas. An XML stanza has a root tag, which is typically `<message/>`, `<presence/>` or `</iq>` (Info/Query). The `<message/>` element is used for pushing information to the network, the `<presence/>` element is used to update state information about the client to the server, and the `<iq/>` element is used for querying/updating information in the server. A typical interaction of XML stanzas over a request and response stream is shown in Figure 8:

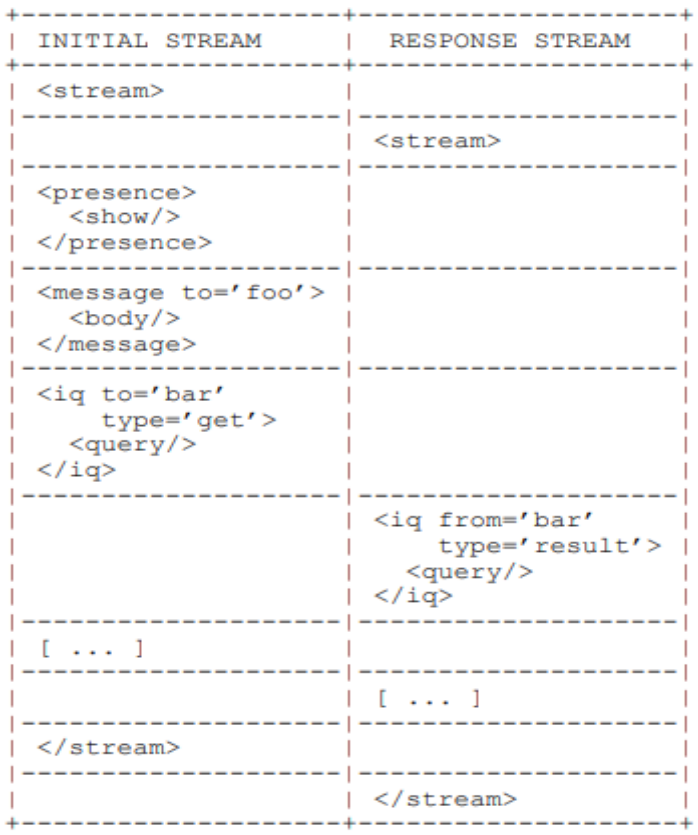

Figure 8: XMPP Streams

The XML stanza for sending a message between two clients, Juliet and IM, connected to the same server example.com, is shown in Figure 9:

```
<stream:stream
    from='juliet@im.example.com'
    to='im.example.com'
    version='1.0'
    xml:lang='en'
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'>
  <message>
    <body>foo</body>
  </message>
</stream:stream>
```

Figure 9: XMPP Stanza

It is the responsibility of the server to identify the recipient of the message request and to route the message, to the recipient. If the recipient is not found, or if some other error occurs, the server can respond with a special XML stanza, known as `<error />` with the error information

3.5.3.2 Evaluation

In this section, we evaluate the protocol based on the criteria defined:

MH1: The protocol MUST allow off the record messaging

The XMPP protocol supports, in a dedicated specification, the concept of Off the Record Messaging. The specification [70] relies on using message processing hints [71], to inform the server, via XML tags within the `<message />` element, to not persist any messages on the server. An example such message is shown in Figure 10:

```
<message
    from='romeo@montague.lit/laptop'
    to='juliet@capulet.lit/laptop'>
  <body>V unir avtug'f pybnx gb uvqr zr sebz gurve fvtug</body>
  <no-copy xmlns="urn:xmpp:hints"/>
  <no-store xmlns="urn:xmpp:hints"/>
</message>
```

Figure 10: XMPP OTR Messaging [71]

The `<no-store />` ensures that the message is not stored in the server, even if the recipient is offline. This directive can then be used in all the communications that are needed to be off the record.

MH2: The protocol MUST allow consistent user management across devices

XMPP has a separate User Profile Protocol Specification [72]. The protocol specification can be used for maintaining user identity on the XMPP servers. Since XMPP is a central

server-based architecture, the information for the user is saved by the server and can be retrieved through multiple devices, via a process of authentication.

For setting the user profile information when a client first time connects to the server, under the </iq> tag. This is shown in Figure 11:

```
<iq type='set' id='setfull'>
  <profile xmlns='urn:xmpp:tmp:profile'>
    <x xmlns='jabber:x:data' type='submit'>
      <field var='FORM_TYPE' type='hidden'>
        <value>urn:xmpp:tmp:profile</value>
      </field>
      <field var='nickname'>
        <value>Hamlet</value>
      </field>
      <field var='country'>
        <value>DK</value>
      </field>
      <field var='locality'>
        <value>Elsinore</value>
      </field>
      <field var='email'>
        <value>hamlet@denmark.lit</value>
      </field>
    </x>
  </profile>
</iq>
```

Figure 11: XMPP User Profile [72]

The profile information can then be retrieved from any other device, using the <iq />, with type as 'get.' This can only be done, once the user has passed the security check.

MH3: The protocol MUST allow for friend discovery and management in the network

In the XMPP system, the users are identified through their user profiles that are set up on the central servers and which are linked to their unique Jabber ID. This gives the flexibility in using various fields such as emails, mobile for searching and identifying friends.

Furthermore, the XMPP Instant Messaging and Presence Protocol Specification [70] identifies the concept of maintaining a contact or buddy list, called as a roster, on the server. A user can add contacts to their buddy list and can query/retrieve the list from the XMPP servers. Each roster, has a list of items, with each item corresponding to a unique Jabber ID. The <iq/> stanza can then be used, to add items (users) or retrieve users from the server. An example stanza for extracting the roster is shown in Figure 12:

```
C: <iq from='juliet@example.com/balcony'
       id='bv1bs71f'
       type='get'>
     <query xmlns='jabber:iq:roster'/>
   </iq>
```

Figure 12: XMPP roster query stanza [70]

A sample response is shown in Figure 13:

```
S: <iq id='bv1bs71f'
       to='juliet@example.com/chamber'
       type='result'>
     <query xmlns='jabber:iq:roster' ver='ver7'>
       <item jid='nurse@example.com'/>
       <item jid='romeo@example.net'/>
     </query>
   </iq>
```

Figure 13: XMPP roster response stanza

As we can see, XMPP allows lookup and management of friends in the protocol.

MH4: The protocol MUST specify provisions for message reliability mechanisms

The XMPP protocol uses a central server for routing information between the users. There exists a separate protocol specification, on top of the XMPP Core, known as Advanced Message Processing [104] which specifies mechanisms to be performed by the server, for ensuring the delivery of messages. These include storing messages offline for a short period and then deleting them, once the delivery of the message has been done. It also ensures mechanisms for forwarding the message to the secondary recipient, if the message sending failed to the primary recipient. Furthermore, if a client connects to the service at a later stage (after a sender has already sent a message), the Flexible Offline Message Retrieval Protocol [73] can be used for retrieving any offline messages. This would be especially beneficial, in case a user logs in at a later stage and retrieves an active emergency notification.

MH5: The protocol MUST have End to End encryption

The XMPP core protocol has defined an additional standard, on top of XMPP core protocol, for end-to-end signing and encryption of information between two users [74].

MH6: The protocol MUST support both one-to-one and group messaging

Apart from one-to-one communication, XMPP has an additional specification, on top of XMPP Core, for setting up of multi-user chat [117]. The specification lays down the mechanisms of setting up of rooms/groups on the server and the mechanism for ensuring communication between the members of the room.

SH1: The protocol SHOULD have SDK's for its various components openly available for development.

The XMPP protocol has a client and server component. Both client and server-side SDK's are highly mature and have evolved over the last two decades. SDK's are available for all programming languages [123,124].

3.5.4 MQTT

MQTT stands for Message Queuing Telemetry Transport [75]. MQ Telemetry Transport was built at IBM in 1999, as a lightweight and straightforward publish-subscribe messaging protocol, designed for constrained devices and low bandwidth, high-latency or unreliable networks. MQTT has seen rapid development since its inception and in 2013, was adopted by OASIS [76] for standardization. MQTT has been used in a variety of projects [77], with a popular one being the Facebook Messenger [78].

MQTT protocol uses the underlying network layer, to build a client-server publish-subscribe messaging mechanism. The protocol's core ideology is to allow clients to subscribe to what is known as "topics." Messages sent by a user which correspond to the specified topic, are received by the clients who have subscribed to that topic. MQTT protocol also defined message reliability mechanisms to ensure the delivery of the messages [79].

3.5.4.1 Architecture

The MQTT protocol is a client-server publish-subscribe messaging system. MQTT has a central server, more commonly called as the "broker." Many clients connect to the system, via a TCP connection to the broker. Each broker has a set of "topics" that the users can "subscribe" to. Clients can also publish messages on the "topics," and they would be received by clients who have subscribed to the same topic [79].

The communication between a client and a server occurs by a series of data packets, called the "control packets" [80]. Each MQTT control packet has a control packet type and serves a particular purpose in the secure connection with the server. Control packet type typically defines the purpose of the specific communication. For example, on initiating a connection, an initial control packet of the type "CONNECT" [80] is sent from the client to the server. This packet typically contains the unique identification of the client and the authentication information (username/password). The interactions that occur between the client and the server occur through a series of such control packets. Figure 14 shows the various control packet types.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request

Figure 14: MQTT Packet Types [80]

As we can see, with these minimum number of control packets, the communication between the clients can be accomplished. For example, if a client wants to publish for a particular topic, the control packet PUBLISH is sent, with the message on the payload. The messages are then routed to the receivers who have subscribed to that particular topic.

3.5.4.2 Evaluation

In this section, we evaluate the protocol based on the criteria defined:

MH1: The protocol MUST allow off the record messaging

In the MQTT protocol, a client communicates with the server by using control packets. The control packets have a fixed and a variable header. Under the variable header, MQTT provides the retained flag, which can be used to direct the server to whether to retain the messages or not. Since MQTT is essentially a publish/subscribe service, the flag is by default true, so that new subscribers do not lose the messages. We can use this flag to ensure that the server does not retain the messages.

MH2: The protocol MUST allow consistent user management across devices

The MQTT protocol identifies users and their information, through the concepts of client_id and session flag. Each user, connected to the MQTT broker, must specify a unique client_id, along with a set of username and password. If authenticated, the user gains access to the server, and the user can now send/receive messages. The session flag saves the

information about the session, which means that a connected user's preferences are preserved if the session flag is set to true [80]. In case a user logs in from a different device, using the same client id and credentials, the session is established. For maintaining advanced user profile information, such as username, kit information, a separate server and database needs to be deployed.

MH3: The protocol MUST allow a mechanism of friend discovery and management in the network

MQTT protocol is built to be a lightweight communication mechanism. Due to this, the protocol does not inherently specify a mechanism of maintaining friend list and for searching for friends. It does allow flexibility in deploying our server, managing the friend information there and thus, use MQTT only as a communication protocol. There exists the essential overhead of building a custom server for this, but is possible.

MH4: The protocol MUST specify provisions for message reliability mechanisms

The MQTT protocol has a set of quality of service levels, as defined in its standard [80]. It has three provisions for ensuring message delivery. QoS 0, which has no particular retry mechanisms and thus can lead to loss of messages. QoS 1, which gives the ownership of delivery to the server and therefore, the server is responsible for retrying the delivery of the message to the clients that are disconnected. QoS 2, which gives the ownership to the sender of the message and allows resending of messages, if an acknowledgment of message publishing from the recipient of the message is not received.

MH5: The protocol MUST have End-to-End Encryption

MQTT protocol does not at a protocol level specify any provisions or mechanisms for end-to-end encryption. It does, in a nonnormative way, in its protocol specification, suggests the use of application message encryption. Many MQTT solutions, such as HiveMQ [118] provide mechanisms for the same in their MQTT infrastructure. Thus, this could potentially be done using such third party solutions.

MH6: The protocol MUST support both one-to-one and group messaging

The MQTT protocol is ideal for one-to-one and one-to-many messaging of devices. The protocol does not inherently have any mechanisms for maintaining group and group members. It does, although, have a mechanism for building subscription lists, which can be used to perform one-to-many messaging, but does not inherently have a mechanism to

ensure groups communication together. Solutions, such as implemented by Facebook messenger, can be done by deploying custom servers for managing group information.

SH1: The protocol SHOULD have SDK's for its various components openly available for development.

The MQTT protocol has a client and server component. Both client and server-side SDK's are highly mature and thus, have evolved over the last two decades [125].

Table 1: Comparison of IM Protocol Evaluation

Requirement	Matrix	Tox	XMPP	MQTT
MH1: The protocol MUST allow off the record messaging	✗	✓	✓	✓
MH2: The protocol MUST allow consistent user management across devices	✓	✓-	✓	✓
MH3: The protocol MUST allow a mechanism of friend discovery and management in the network	✓	✓-	✓	✓
MH4: The protocol MUST specify provisions for message reliability mechanisms	✓	✓-	✓	✓
MH5: The protocol MUST have End to End Encryption	✓	✓	✓	✓-
MH6: The protocol MUST support both one-to-one and group messaging	✓	✓	✓	✓-
SH1: The protocol SHOULD have SDK's for its various components openly available for development	✓	✓	✓	✓

3.5.5 Recommendation

XMPP protocol best matches our requirements since it meets our list of Must Haves completely. Additionally, XMPP has formally drafted and evaluated specifications and mature SDK's available. Hence, **XMPP is our recommended protocol for building the PAL messaging infrastructure**.

3.5.5 Limitations to Recommendation

Our current recommendation for XMPP protocol has the following limitations:

1. The current recommendation is based on the assumption of an online environment. The protocol may be needed to be further customized, for enabling offline availability (as discussed in Section 3.6).
2. Our evaluation is based on technical literature review of the protocols. We have not actually implemented them individually and compared their real-time operation. Hence, there is a possibility that certain requirements that seem to be complete may have additional issues while implementing them.

3.6 DQ 5: Offline System Availability

Upon discussion with AVI regarding the implementation of the system, a critical challenge that was discovered in making the system unusable is lack of availability of internet/data pack among our targeted users. Since the system is built with an idea of providing support during emergencies, it is imperative that the lack of internet access not hamper the system's accessibility of such services.

To ensure the availability of services, we defined the following requirements for the system to be available in offline conditions:

1. ***MH1: The solution MUST allow access to PAL's emergency support services****:* We need to ensure that our system works the same, for both offline and online environments and hence, all the services for the online environment should also be available offline. This means enabling communication between a user during an emergency event (initiator) and their support group (crowd).

2. ***MH2: The solution MUST be agnostic to the user:*** *We want our solution to stay agnostic to the user, and the interface on the mobile devices should remain the same*.

With the above mentioned features in mind, related work on the same issue was explored.

3.6.1 Caching

Caching of REST Web Services [30], as a solution to the lack of network availability, has been thoroughly researched. Ma et al. [7] proposed a dynamic network availability aware mechanism of prefetching and caching of web service content, to be available to the user, when the network is not available. This approach aims to use the availability of network to

prefetch and cache the resources. Elbashir and Deters [8] presented a mechanism of building proxy caches, in what they called as nomadic applications (applications which connect to the internet intermittently).

Various modifications and improvements to caching methodologies have been made for web services in recent years. Fernandez et al. [9] designed a caching system for SOAP-based web services, which uses the "expire" information in response to check if the cached data can be used. Deters et al. [10], based on Fernandez's work, built a client and server side caching model for checking the freshness of data. [11-16] further elaborated on these efforts and built various mechanism, which largely focused on prefetching and "smart" fetching of information, to be used in offline conditions.

3.6.1.1 Evaluation

In this section, we evaluate caching in the context of our system, based on our predefined requirements.

MH1: The solution MUST allow access to PAL's emergency support services

The caching mechanism does not provide any means of communication during an offline situation. There is no means to the setup of instant messaging or emergency messaging in case of caching since it focuses on prefetching of data and is useful to view local copies of information, and not for real-time communication.

MH2: The solution MUST be agnostic to the user

The caching mechanism can be handled within the mobile application and hence, remains agnostic to the user.

3.6.2 Offline synchronization based applications

A solution that is employed in regions with limited internet/network connectivity is applications store data offline and then synchronize the information when the internet service becomes available. Such applications typically depend on storage mechanisms within the phone. The data is synced to the server on either a periodic or internet availability basis. All mobile development frameworks such as for Android phones and iPhones provide mechanisms and tools in their software development kits, for building this functionality.

Ren et al. [21] developed a protocol and communication mechanism, called SyncML, for performing data sync between mobile devices and servers/other mobile devices. Go et al. [22,23] developed a mobile data synchronization platform, known as Simba, to facilitate

the developers, in the process of building data synchronization based services, by exposing a platform agnostic data model and API.

3.6.2.1 Evaluation

In this section, we evaluate offline synchronization in the context of our system.

MH1: The solution MUST allow access to PAL's emergency support services

Similar to the issue for caching, offline synchronization does not provide mechanisms for real-time communication, during offline environments. This would mean that the instant messaging and emergency messaging services would not be available.

MH2: The solution MUST be agnostic to the user

The offline synchronization mechanism can be handled within the mobile application and hence, does remain agnostic to the user.

3.6.3 SMS Based Services

Another alternative for mitigating the lack of internet availability could be to build services that are pure based on SMS or Short Messaging Service. Several such systems have been explored in the past. Olasina [17] evaluated the impact of SMS based m-Banking services in University of Ilorin, Nigeria and found a positive correlation to the customer's perceived ease of use. Montes et al. [18] evaluated the impact of an SMS based strategy to improve adherence to schizophrenia medication protocol and found a positive correlation between the patient's adherence to protocol and subscription to the message based nudge service. Kannisto et al. [19] performed an extensive overview of SMS based reminders in health care services. The survey concluded that 77% of the existing solutions showed adaption and improvement of results [19]. The services that are implemented using the SMS based services can typically be divided into two categories:

1. **Nudge based services**: These typically include one-way communication from SMS producer to the consumer.

2. **Interactive services**: These include services in which a consumer can interact with the SMS service provider and get specific assistance, similar to what was done in m-Banking [17].

The design of the PAL system comes under the category of an interactive system. Susanto et al. [20] found some adaption constraints in SMS based interactive e-governance service. Some of the pertinent ones to our context are:

1. **Perceived risk of user privacy**: Susanto et al. found that the users have privacy concerns, regarding the SMS provider monitoring and exploiting their personal information, which may be shared over a message.

2. **Self efficacy of using SMS**: Susanto et al. also found that some people did not use the SMS based service since they did not know how to use SMS messaging interface. This inability could be a potential bottleneck in our context since our solution would be used in situations of emergency, where a person's ability to use the service via typing characters, may be compromised.

3.6.3.1 Evaluation

In this section, we evaluate SMS based services in the context of our system.

MH1: The solution MUST allow access to PAL's emergency support services

The SMS based services can be potentially used, for enabling communication between the users of our system during an ongoing emergency. However, Services such as adding friends, maintaining a friend list, managing emergency, etc. would not be conveniently possible and the user would have to be able to actually "type" control SMS into their SMS app. This necessarily would make the system inherently more complicated. Also, a user going through an emergency event may not have the cognitive ability to be able to use this service effectively.

MH2: The solution MUST be agnostic to the user

The SMS based solution is not agnostic to the user, and the user would need to shift between mobile app based interface and an SMS based interface, in case of an offline environment. Hence, the mechanism of service access for the user would shift in case of an offline environment.

Table 2 summarizes our efforts of studying various mechanisms for building offline applications and how appropriate they are to our two key requirements:

Table 2: Comparison of Offline Failover Evaluation

Requirement	Caching	Offline Synchronization	SMS Based System
MH1: The solution MUST allow access to PAL's full range of instant messaging and emergency services	✗	✗	✗
MH2: The solution MUST be agnostic to the user	✓	✓	✗

As we can see, our primary requirement MH1 is not completed by any of the mechanisms that we have investigated. Consequently, we decided to build a novel approach to accessing offline real-time services. In the next section, we are going to discuss in detail our recommendation and how does the system work.

3.6.4 Final Recommendation: SMS Based Failover

Since we could not find a satisfactory existing mechanism for offline conditions that suits our context, we had to think of a novel approach to build a failover mechanism in case of no internet situation. Assuming that SMS services are going to be available even if there is no internet and taking a cue from how SMS services were used in Haiti during their natural disaster, as described in Chapter 2, we focused our attention to using the SMS layer as the mechanism for the system's failover operation.

The SMS service in mobile phones provided with exciting features that further motivated us to pursue this architecture:

1. *SMS Services are independent of the availability of internet packs*. The SMS services are dependent on the cellular network providers and hence, could still be used in phones with no internet packs.

2. *SMS infrastructure is already setup, and the users are already a part of it*. All the cellular network providers provide the SMS infrastructure and hence, does not require the configuration of a messaging infrastructure, unlike setting up an infrastructure for internet based messaging.

3. *Addressing users is simple in SMS since their mobile phone numbers can be used*. This gives a natural mechanism for sending messages to a particular user since the actual routing and delivery of the message is done by the underlying cellular infrastructure and the sender needs to only know the mobile number of the recipient.

4. *SMS is a free-flowing message format*. We can customize the payload according to our needs and based on our emergency flows.

The challenge at hand was, how do we convert the flow that would happen in case of an online scenario to an offline SMS based scenario? In the next Section, we discuss the mechanism on which we have built our failover infrastructure.

3.6.5 Programmable SMS

Programmable SMS are services that allow building a custom SMS gateway (which in this context means the SMS number and the underlying service in a programmable SMS provider), which can be used to program the interaction with a user, by providing the infrastructure of building a custom web server which processes the messages. The custom web server can then be used to send SMS messages to other recipients. The interaction between the various components of a programmable SMS system is shown in Figure 15:

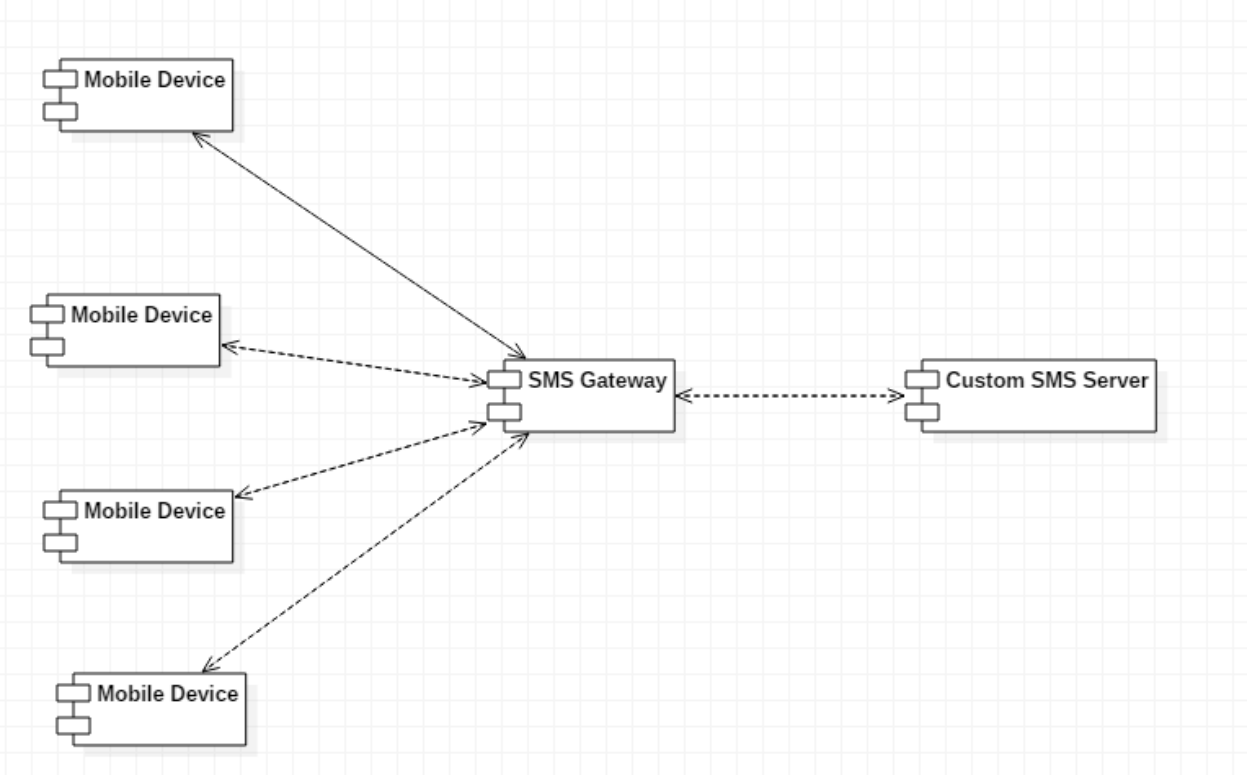

Figure 15: Programmable SMS Infrastructure

As we can see, the users are interacting in this setting, by using the SMS gateway and the web server. The SMS gateway routes the incoming SMS Messages to the SMS Server, which then instructs the gateway to perform a response operation. The SMS Server is the component where the custom logic and flows for the interaction of users with the system

can be written. In the next section, we detail out the logic and flows that were implemented in case of emergency lifecycle implementation.

3.6.6 Proposed SMS Based Emergency Lifecycle Implementation

In this section, we delve deeper into emergency lifecycle implementation during an offline phase of an ongoing emergency. We make the assumption that a user has already registered themselves on the system and had created a roster of their friends. The information is available with the mobile device. We utilize the programmable SMS framework, as we had explained earlier, in our implementation design.

- **Emergency Initiation:** During this phase, the user presses the emergency button on their device. On pressing the button, an emergency SMS would be sent to the SMS gateway, which will route the message content to the PAL SMS Server. The SMS contains the pre-set location of the user. The PAL SMS Server then looks up the friend list of that particular user. The PAL SMS Server then sends the Emergency SMS Message to the friend list. The SMS Server makes a record of an active emergency and the user who has initiated it.

- **Emergence Response Aggregation:** The user who has received an emergency alert responds with an SMS to the SMS gateway, which routes it to the SMS Server. The user, if answered affirmatively, can either send their distance to the SMS Server or can send their pre-set location to the SMS Server. The SMS server calculates the distance of the responder from the initiator. For positive user responses, the SMS Server sends the information to the initiator, via an SMS, with the user identification of those users and the distance from the initiator. The SMS Server maintains the information of the responders in a database.

- **Emergency Response:** During this phase, initiators and responders communicate to each other, via SMS which is routed via the SMS gateway and the SMS Server. It is the responsibility for the SMS gateway to route the messages between the initiators and the responders.

- **Emergency Finished/Cancellation:** This would be similar to the case of online situation.

3.6.7 SMS Types

In the previous section, we discussed the flow of emergency lifecycle with an SMS failover implementation. In this section, we discuss the various message types and the direction of the messages between our three entities: *Initiator, PAL SMS Server, and Responder*.

1. **Initiator -> PAL SMS Server**: There are primarily two types of SMS, that would go from the Initiator of an emergency event to the PAL SMS Server, via the SMS gateway:

   Emergency Initiation SMS: This SMS is sent from the initiator and contains the location information about the initiator. This is the message that informs PAL SMS Server for the start of an ongoing emergency.

   Emergency Chat SMS: This SMS is sent from the Initiator to the PAL SMS Server, with the purpose of routing the message to the responders and thus communicating with them. It is the responsibility of PAL SMS Server to do the said routing.

2. **PAL SMS Server -> Initiator:** There are primarily three types of SMS that would go from PAL SMS Server to Initiator:

   Emergency Responder SMS: This is sent to the Initiator, with the information about the identity and distance of the user who has responded affirmatively to the emergency. One SMS is sent for each affirmative response and it is the responsibility of the mobile device to collate all the responses.

   Emergency Responder Chat SMS: This is sent to the Initiator when a responder sends a chat communication to be sent to the initiator.

   Emergency Confirmation SMS: This is sent to the initiator, to confirm that the emergency has been received and a unique emergency identifier is assigned to the initiator.

3. **Responder->PAL SMS Server:** There are primarily two types of SMS that would go from Responder to Initiator

   Emergency Response SMS: This SMS is sent from the responder and informs the PAL SMS Server about their response to the emergency. If affirmative, the SMS contains either the pre-set location or the distance of the user from the Initiator.

   Emergency Chat SMS: This is a general chat message, similar to the one from the initiator, for sending a message to all the responders and the initiator.

4. **PAL SMS Server->Responder:** There are primarily two types of SMS that would go from PAL SMS Server to Responder:

   Emergency Request SMS: These are sent from the PAL SMS Server to the potential responder(s) to inform them about an ongoing emergency and the location of the initiator.

   Emergency Routed Chat SMS: These are SMS which are either sent from the initiator or the responders and are routed by the PAL SMS server to the active responders of the emergency. These messages can either be initiated by the initiator or other active responders.

3.7 PAL Integrated Architecture

Based on our recommendations for the messaging infrastructure and offline functionality, in this Section, we propose an integrated architecture for PAL. Figure 16 shows the high-level diagram of the integrated architecture.

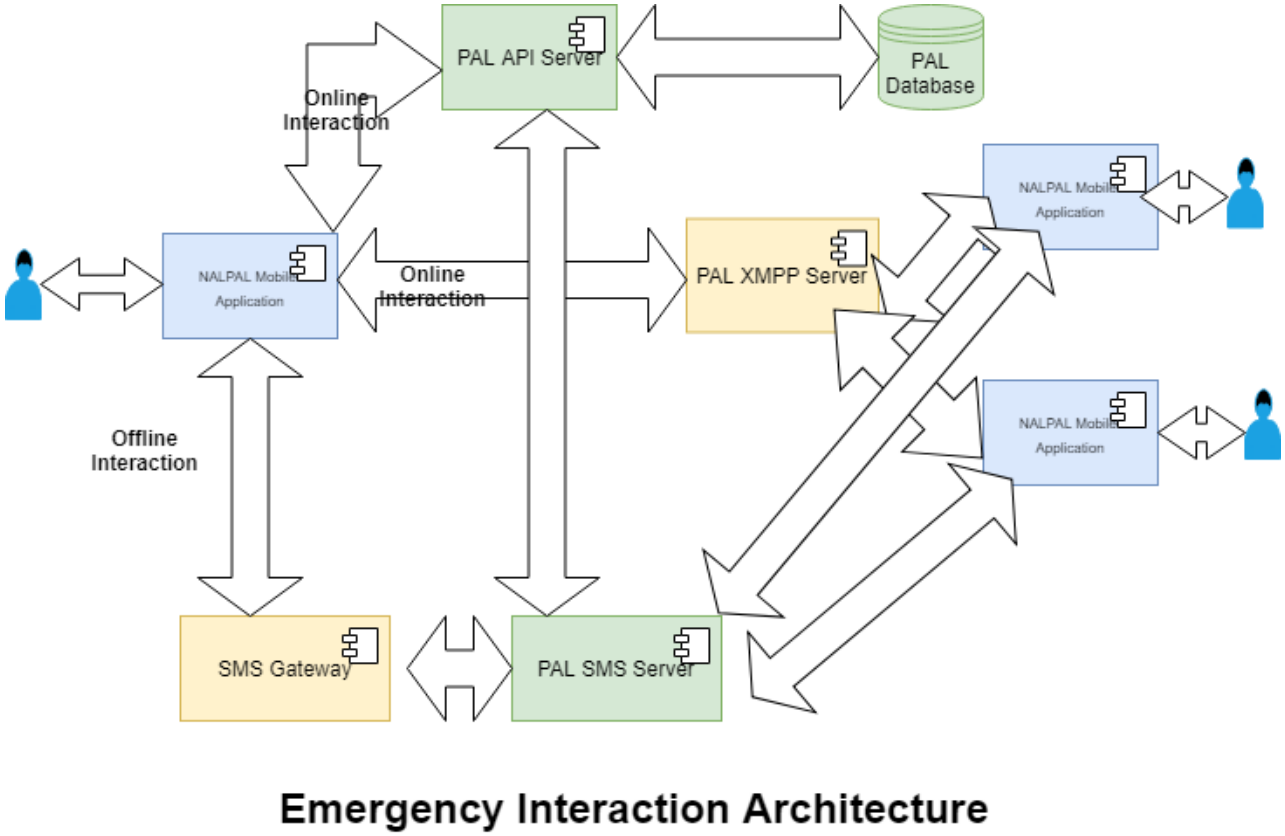

Figure 16: PAL Integrated Architecture

The above architecture represents the various components and their interactions during an ongoing emergency. We describe the various parts and the roles that they perform in this system:

- **PAL Database:** The PAL Database Store will consist of the following data objects:

  User Profile: The database store will maintain information about the user profile. Although this could be offloaded to the XMPP Server implementation as well, we would keep this information in our custom server. The reason for that is to enable offline operations. For offline transactions to work, the PAL Server would need access to user(s) profiles. Hence, it could be convenient if we could maintain the same in our database store.

  User Friend Information: The PAL Database Store will consist of information about a user's friend list as well. For reasons similar to for user profiles, we choose to have a custom implementation for maintaining this information.

  Emergency Details: The Database store will contain information about current and past emergencies. The database will also provide information about the respondents of the active emergency.

- **PAL API Server:** The PAL API Server would provide the Application Programming Interfaces for the following functionalities:

  Managing User Profile: The API Server would connect to the database store to provide the functionality to the mobile application.

  Maintain User Friend(s): Similar to above, the server would provide the API for maintaining user friend information

  Maintain Emergency Information: The Server would be responsible for ensuring the API for managing emergency lifecycle.

- **PAL XMPP Server:** The PAL XMPP Server would have the following responsibilities:

  One-to-One and Group Communication: The XMPP Server would be responsible for one-to-one communication and group communication between users.

Emergency Group Communication: The XMPP Server would be responsible for the group conversation during an emergency. This can also be extended to normal group conversation during non-emergency times.

Emergency Notifications: The XMPP Server would be responsible for providing emergency notifications to the users.

- **PAL SMS Server:** The PAL XMPP Gateway would be responsible for managing the entire lifecycle of an emergency event, including notifying responders and enabling communication between the initiator and the responder. The SMS Server would rely on the underlying PAL Server for obtaining friend list and maintaining emergency information.
- **PAL SMS Gateway:** PAL SMS Gateway would be responsible for providing a unique number for SMS operation during an emergency and for routing the SMS content to PAL SMS Server.

3.8 Summary

In this Chapter, we focused on the critical design decisions of the system. We discussed our evaluations of the various choices that are available to us. We provided a recommendation for building the messaging infrastructure and the offline failover mechanism as well. We also gave an overall integrated architecture on the system. In the next Chapter, we evaluate our design by assessing our prototype and discussing its feasibility and offline availability aspects.

Chapter 4

Design Evaluation

In this section, we evaluate the design and architecture recommendations that were made in the previous Chapter. We focus on *feasibility* and *availability* in our evaluation. We show the degree to which we were able to meet our requirements and present the limitations of our work.

4.1 Feasibility Evaluation

Our goal in feasibility evaluation focuses on the following question:

"*Is it feasible to build a crowdsourced emergency response system, from the integrated architecture recommendation?*"

To the same end, we built a prototype backend system, using the design recommendation that we have made in Chapter 3. We then evaluate the extent to which the prototype is complete for our five primary requirements. In subsequent Sections, we detail out the implementation detail for the various components of our architecture.

4.1.1 PAL Database Store

In the previous Chapter, we defined the various data objects that the PAL database store would have to persist. To the same end, we needed to decide on a database store for our application. Since this was a proof of concept, we wanted to use a database store that was easy to set up and was flexible in case of changes to our database design.

With these concepts in mind, we decided to use IBM Cloudant Database [37], which is a high-performance JSON data store [106]. JSON or Javascript Object Notation is a key value based lightweight data representation and transport format, used commonly for exchanging information over the internet [106]. The flexibility of storing JSON objects as data was the primary motivator for us to use the database. Being a JSON data store, Cloudant does not require setting up of a pre-defined schema. The various JSON data formats that are used are explained below:

- **User Profile:** The user profile is stored in a ***users*** database. Figure 17 shows the schema:


```
{
  "_id": "08b4c658c503d037cfaf41308d670624",
  "_rev": "1-240511718887b8f1891233fda0a0bd75",
  "username": "hnddycfvl",
  "email": "zbnzicgmail.com",
  "hasNalaxone": false,
  "password": "$2b$10$wecGQhBx0cRSDvR7Mov5XOHdqKic/b.5xAEKW4UNc7bDJfPUrjmrq",
  "mobile":"7789224530"
}
```


Figure 17: User Profile Object

- **User Friend Information:** The user's friend information is stored on two separate databases. One for friend requests, called as the ***friendrequests.*** Figure 18 shows the schema:


```
{
  "_id": "1b065ea4053a1d339aac5b004fa2278e",
  "_rev": "1-3894c592ad21fc94e16a81e45c815538",
  "requestorID": "08b4c658c503d037cfaf41308d670624",
  "responderID": "08b4c658c503d037cfaf41308d674fae",
  "isAccepted": false
}
```


Figure 18: User Friend Request Schema

Once the friend request is accepted in the database, the *isAccepted* flag is set to true an entry is made in the ***friends*** database. The schema is similar to Figure 18, with additional information regarding the nicknames that the users can set for each other.:


```
{
  "_id": "1b065ea4053a1d339aac5b004fa2278e",
  "_rev": "2-24a140812d0566e5cb81a2ce3af8f224",
  "requestorID": "08b4c658c503d037cfaf41308d670624",
  "responderID": "08b4c658c503d037cfaf41308d674fae",
  "isAccepted": true,
  "requestorNickName": "annie",
  "responderNickName": "Madhav"
}
```


Figure 19: User Friend Schema

- **Emergency Information:** For storing the emergency information inside the database, two databases are used. One, for registering an emergency event, called as the ***emergency*** database. The schema for the database is shown in Figure 20:


```
{
  "_id": "1b065ea4053a1d339aac5b004fa2278e",
  "_rev": "1-d53dd7e344b7e545f9b48f12fe25e63e",
  "initiatorID": "08b4c658c503d037cfaf41308d670624",
  "isActive": true,
  "latitude":"53.726669",
  "longitude":"-127.647621"
}
```


Figure 20: Emergency Schema

For each emergency, the responders to the emergency are stored in a ***emergencyresponses*** database. The schema for the same is shown in Figure 21:


```
{
  "_id": "1b065ea4053a1d339aac5b004fa2278e",
  "_rev": "1-ef4e56a7a6c6abbd2721cfaca6c59e32",
  "emergencyId": "19bf72f9e8b29950a38e5f29d619b1fe",
  "responderID":"08b4c658c503d037cfaf41308d6a7347",
  "distance":"5"
}
```


Figure 21: Emergency Response Schema

4.1.2 PAL API Server

The next component that we discuss here is the PAL API Server that has been deployed, and which is connected to the IBM Cloudant Database. Node.JS [31] was used as the development platform for building the API Server. Node.JS has the advantage of providing a non-blocking server, allowing faster processing and better resource utilization [32]. The API was made using JSON [106] over Representational State Transfer (REST) [7] standard, which has become the de-facto standard for building web services and also, works well with our backend database store. REST standard uses the HTTP verbs [107] and Uniform Resource Identifiers (URI) [108] for addressing and consumption of web services. JSON is the most common format of data exchange when the REST API is built.

Our application server is deployed on IBM Cloud (Bluemix) Node.JS Servers [35], which provided fast service and integration with IBM Cloudant JSON datastore [37]. The building

of the application was done using the Node Package Manager [43], which allows management of dependencies and builds the development and production deployments. The unit testing of the API was performed using Postman scripts [44].

In Chapter 3, we identified 3 primary types of workflows that have to supported by our PAL API Server. The API that have been developed are shown in tables 3,4 and 5:

Table 3: User Management API

API Operation	Description
registerUser	The API would be used by the users to register themselves on the system.
updateUserInformation	The API would be used by the users to update their information on the system.
login	The API would be used by the users to login into the system.
logout	The API would be used by the users to logout of the system.

Table 4: Friend Management API

API Operation	Description
sendFriendRequest	The API would be used by the users to add another user to their friend/support list.
verifyFriend	The API would be used by the users to verify addition of friends to their friend/support list.
listFriends	The API would be used to retrieve a list of active friends.
listPendingRequests	The API would be used to retrieve a list of pending friend requests.
rejectFriend	This API would be used by a users to reject a friend request.

Table 5: Emergency Management API

API Operation	Description
notifyEmergency	The API would be used by an initiator to notify an ongoing emergency to their friend group. The location of the emergency requestor would be sent as a message to the friend group, using the XMPP Server.
respondEmergency	This API would be used by a user to respond to an active emergency.
cancelEmergency	This API would be used by the initiator to cancel the emergency.
sendMessage	The API would be used by users to send messages to each other, both during an emergency situation and during normal chat. The API is built on top of our XMPP Server, which is explained in Section 4.1.3.
retrieveEmergencyInformation	This API would be used to retrieve information about an emergency, based on emergency-id
retrieveEmergencyReponders	This API would be used to retrieve the list of responders to an actual emergency

4.1.3 XMPP Server

In Chapter 3, we elaborated upon the functions that have to be performed by the XMPP Server, which included enabling one-to-one communication, group communication, and emergency notification. For the same, we can set up our custom XMPP server, such as those provided by ejabbered [81]. The benefit of such an approach is that we get the full range of functionality and flexibility of the XMPP protocol. The way that we have defined the architecture of our application, we do not need the server to provide us with all the functionalities. The management of the user profile and friend list is done by our own custom PAL API Server and Database Store. Hence, we only need a mechanism that would allow addressing of users/devices for one-to-one, one-to-many and many-to-many communications.

Since our form factor is mobile based applications, we looked into the native messaging platforms that are built on XMPP specifications, for both Android and iOS. Both Apple Push Notifications [83] for iOS and Firebase Cloud Messaging [38] for Android phones use XMPP based server implementations to enable messaging with their devices. We decided to build our messaging infrastructure based on them. To further speed up the process of development, we decided to use a hybrid cloud wrapper, that encapsulates both these services and allows us to write messaging code without keeping the device operating system in consideration. IBM Push Notification Service [40] was decided to be used as the mechanism for building the messaging layer of our application. Similar tools are also provided by Azure Service Bus [41] and Amazon's SNS [42].

IBM Push Notification Service [40] is offered as part of IBM's Bluemix Cloud Suite [35]. The Push Notification Service allows setting up of underlying Firebase Cloud Messaging [38] and Apple Push Notification [39] configuration, as shown in Figure 22:

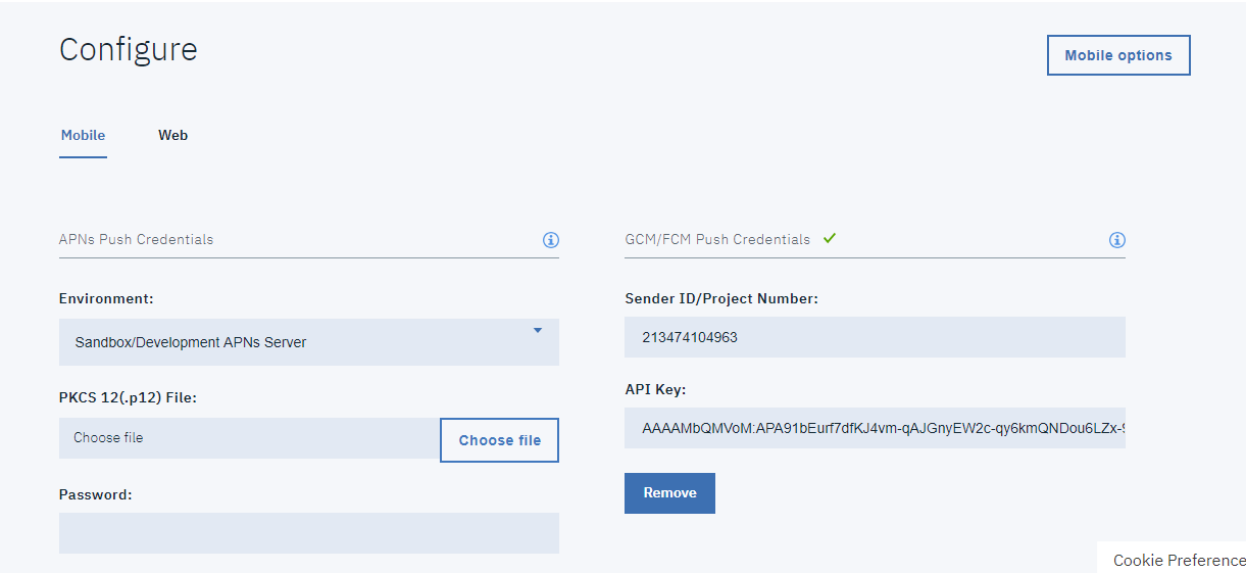

Figure 22: IBM Push Notification Setup

The IBM Push Notification Service provides us with the following capabilities:

- ***One-to-One Communication****:* For enabling one-to-one communication, each device must first be registered on the IBM Push Notification Platform via a unique subscription ID which can be generated by the device. For enabling setting up of this connection, Push Notification Service provides libraries for Android [85] and iOS [86]. In our case, the user-id that is unique to each user, is used as the addressing mechanism to register a device on the push notification service. Once a device is registered, the device is ready for receiving messages, based on the underlying vendor's messaging layers. Sending of messages in Push Notification Service is done

via RESTful API. The API can be called directly or by various Server Side SDK's that are available. To facilitate this function, a special API operation *sendMessage* was built in the PAL API Server. The operation would utilize IBM Push Notification Javascript Server Side SDK (Since our PAL API Server is built on top of Node.JS) for sending a message to either a single user or a group of users. The receiving of the messages was handled by the device side SDK's, which would listen for any incoming messages.

- ***Emergency Notification and Group Communication****:* Both the cases are handled together by the IBM Push Notification Service. We use the concept of "subscription topics" to achieve this operation. Since the service is essentially push-based, it provides us with the feature of topics that user(s) can subscribe to. When the message is sent from a user for that particular topic, all the subscribed users receive the message. The topics, known as "tags" in IBM push notification terminology, can be created dynamically. During the emergency initiation process, a unique tag with the emergency-id is created. The users who respond to the emergency affirmatively, are subscribed to the tag when they use the *respondEmergency* API. Post that, all communication between the initiator and responders is done with the messages addressed to the unique emergency tag. The *sendMessage* API can be used with an emergency-id, to send messages to the responders. This ensures that all the users who respond to the emergency receive all the subsequent notifications and messages. In our current implementation, the group functionality only works during emergency and has not been implemented for normal operation.

4.1.4 PAL API workflows

This section describes the sequence diagrams of various PAL workflows.

4.1.4.1 User Registration and Login Workflow

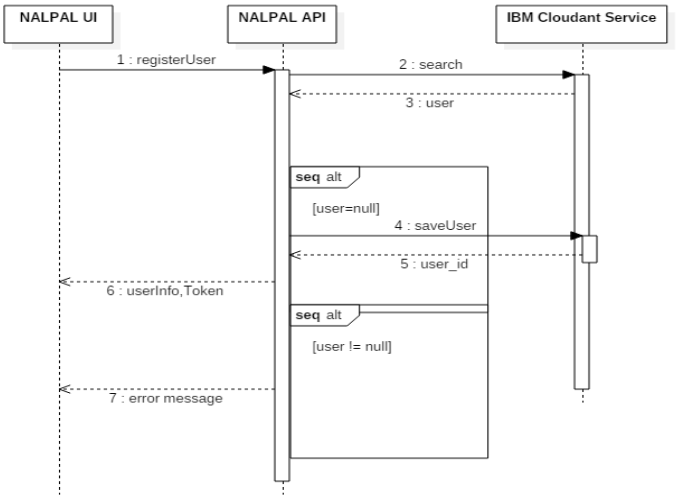

Figure 23: User Registration Sequence Diagram

Figure 23 shows the workflow of user registration. The API checks if the user with the same username/email/mobile is all ready registered. If that is the case, an error response is returned. Otherwise, the user is registered. Upon registration, the system also returns an authorization token, which has to be used to authenticate and authorize the users in the subsequent API calls. Figure 24 shows the login sequence diagram.

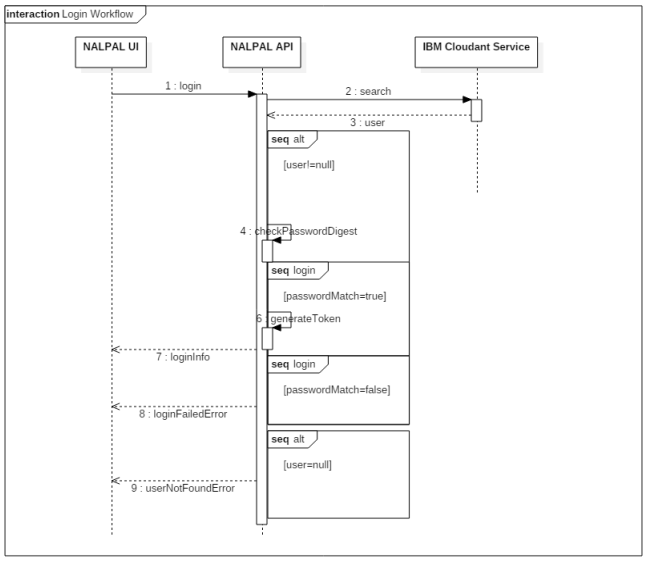

Figure 24: User Login Sequence Diagram

4.1.4.2 User Friend Management Workflow

The friend management workflow assists the user in building a virtual peer group of friends. The flow for adding a friend is shown in Figure 25 and Figure 26.

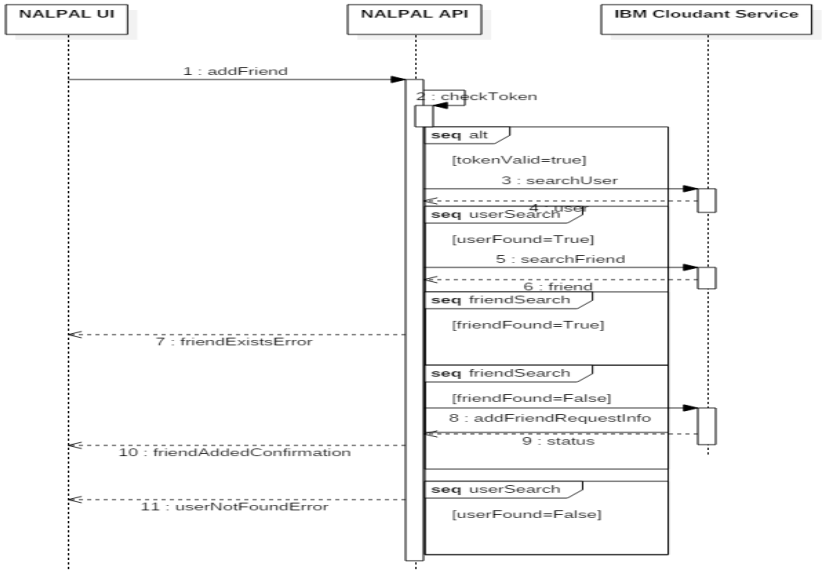

Figure 25: Add Friend Sequence Diagram

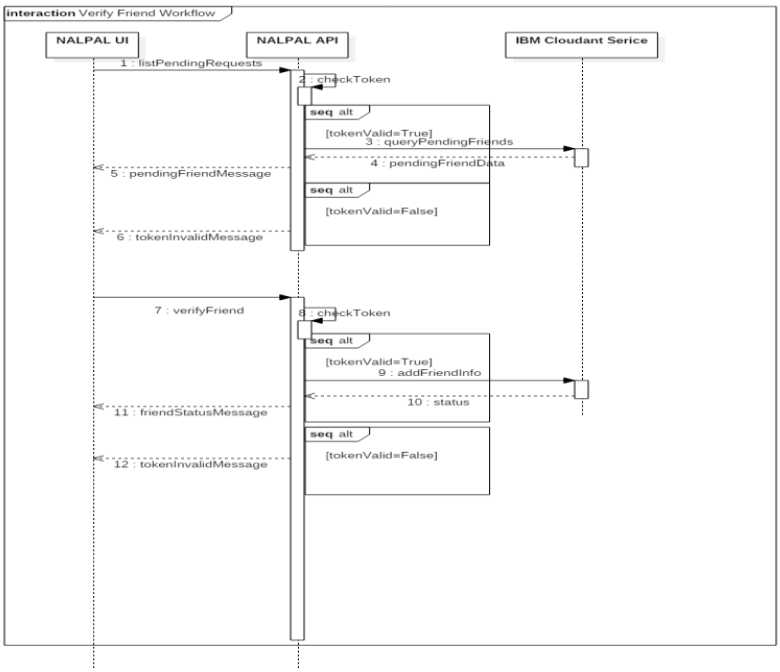

Figure 26: Verify Friend Sequence Diagram

4.1.4.3 User Messaging Workflow

This workflow corresponds to communication between users. Users which are added in each other's friend list are the only ones that can send messages to each other. The messages are not saved in the backend server and are temporary stored in the user's mobile device, from where they are deleted after a fixed amount of time. The flow for the same is shown in Figure 27:

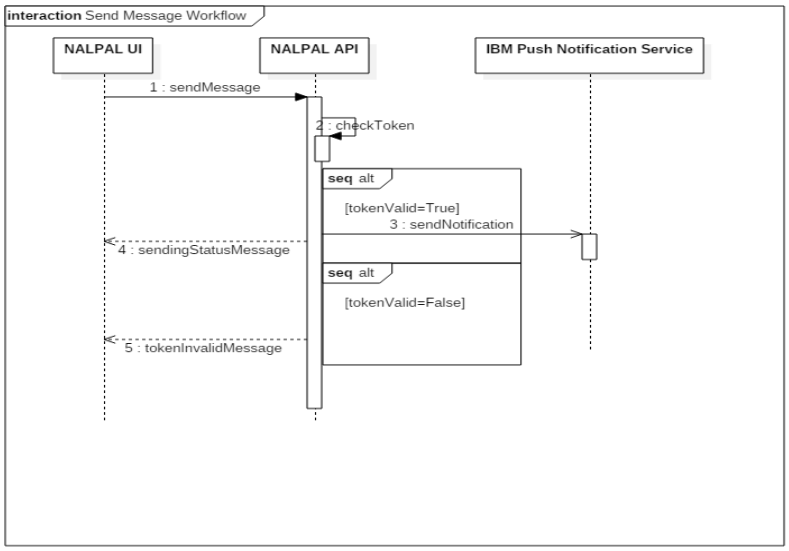

Figure 27: Send Message Sequence Diagram

The push notification service then sends a message to the receiver's device, where the message is processed by the IBM Push Notification SDK and displayed to the user.

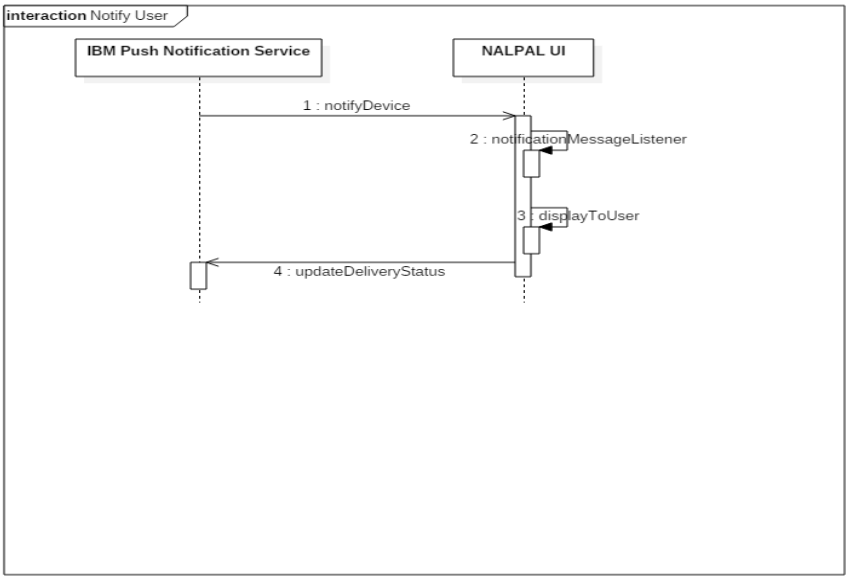

Figure 28: Confirm Delivery Sequence Diagram

4.1.4.4 User Emergency Workflow

This is one of the primary workflows of the NALPAL application. The workflow has two components to it: Emergency Requestor and Emergency Responder. Initially, the emergency requestor, broadcasts its position to its peer group. Figure 29 depicts this:

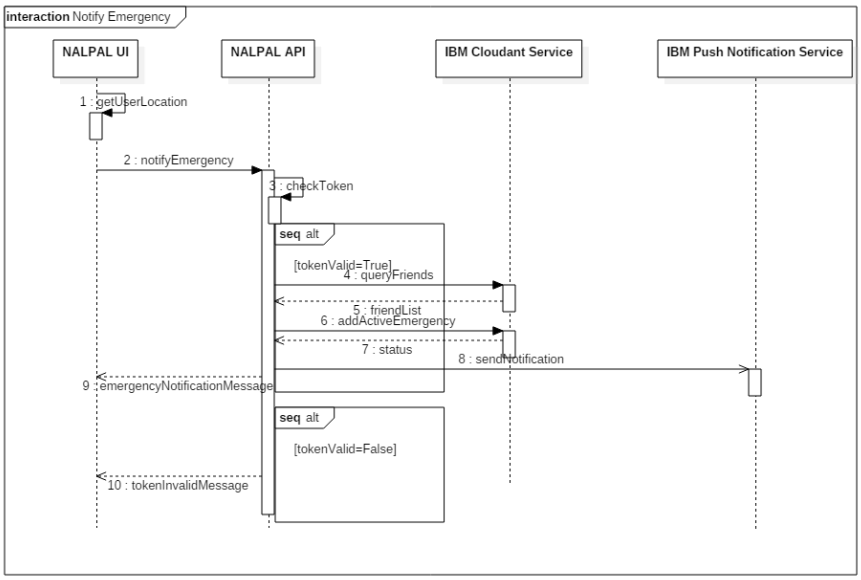

Figure 29: Notify Emergency Sequence Diagram

The friends for the user are retrieved from the database and an emergency message is sent to them, via the IBM Push Notification Service. The users, who are then in the vicinity of the requestor can respond to the emergency. Once the message has been sent, the user can decide to respond to emergency, if they have the naloxone kit and are in the vicinity of the emergency requestor. The workflow for emergency response is shown Figure 30: `

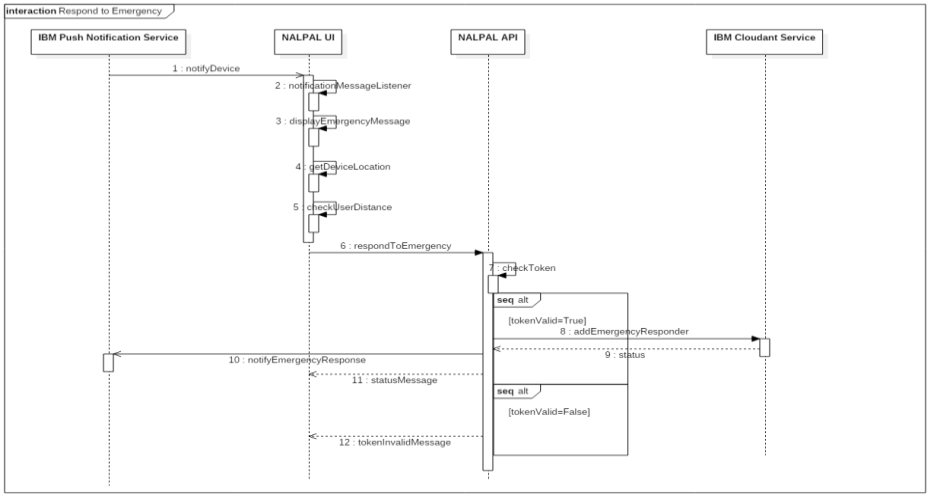

Figure 30: Respond to Emergency Sequence Diagram

4.1.5 PAL SMS Gateway

In the previous Sections, we have primarily discussed the online version of our application. We have discussed set up of the PAL API Server, the XMPP Server, and the PAL Database Store. In this section, we go into the detail of the offline failover aspect of our application.

The first step towards setting up the offline SMS failover part is to set up a programmable SMS Gateway, that allows us to develop and connect to a PAL SMS Server. For our development purpose, we chose Twilio for setting up the Programmable SMS Gateway. Twilio [105] is a popular programmable voice and SMS system, which has a comprehensive set of Restful API and Programming Language SDK's available, for allowing programmatically handling of SMS and Voice communication [105].

One of the major requirements that we need to fulfill is that the gateway should be able to communicate the contents of the SMS to a custom server. Twilio performs this, with the assistance of Webhooks [45]. A Webhook is an API call that is made by Twilio when it receives an SMS on one of its numbers. The Webhook URL can be programmed to be a custom server, and the contents of the message are sent, as part of the URL Request. The contents of the message can then be parsed by the custom server, and the message can thus be processed.

A view of Twilio's webhook is shown in Figure 31:

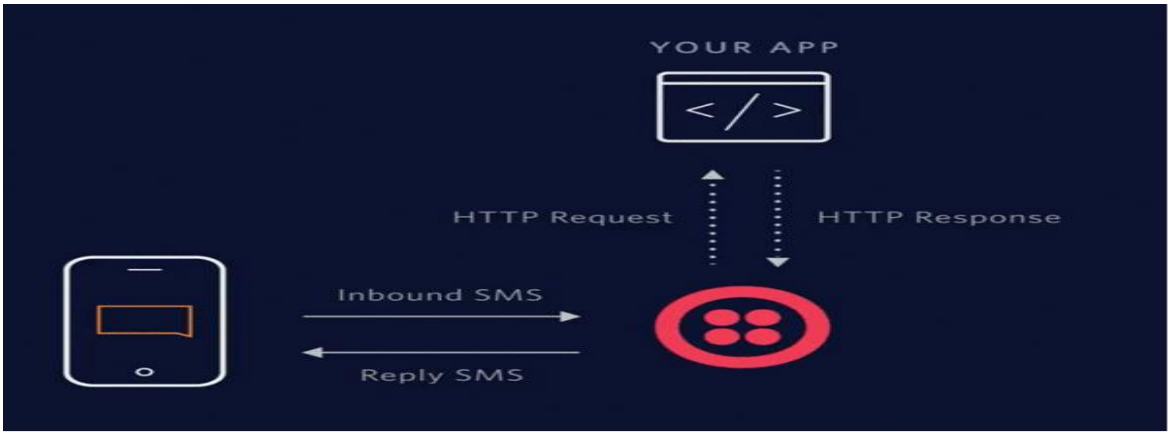

Figure 31: Twilio Webhook [105]

As we can see, Twilio does allow us to process our incoming message and route it to a custom server, which in our case would be the PAL SMS Server. In the next Section, we go into the details of the PAL SMS Server and its various flows that were implemented.

4.1.6 PAL SMS Server

In the last Section, we discussed the setup of PAL SMS Gateway, which would parse and send the SMS Messages to a PAL SMS Server. The PAL SMS Server would be responsible for carrying out all the functionalities of the emergency operation, during an offline environment.

Before we delve deeper into the flow(s) that are supported by our PAL SMS Server, we first discuss the various SMS Message Types that we described in Chapter 3. These primarily represent the different types of transactions that can take place within the system, between the initiator, PAL SMS Server, and Responders. Our PAL SMS Server implementation was also done, using the Node.JS environment and deployed on IBM Cloud. For the SMS Server to perform its operation, we identified the SMS Payloads that would be sent to and from the server.

4.1.6.1 PAL SMS Payload

Each SMS of the various SMS types consists of some common elements, which would be necessary for processing of the messages and state within the system. The common elements of each SMS would include of the following parts:

- **Transaction ID**: A unique transaction ID for the particular transaction. This is primarily needed for the mobile device. Since the process of accessing service is asynchronous, to match a received SMS to an earlier SMS request, a transaction ID would be needed. Hence each message within the same workflow has the same transaction ID.

- **Transaction Type**: This represents the type of this SMS Message. This is needed by the PAL SMS Server, to process the message correctly and perform the required operation. This is analogous to how API URL change for different kind of operations. Each transaction type was given a code between 0-7.

- **Auth Token**: We make an important assumption that the user is logged into the system when using the SMS failover. To validate that, auth token needs to be sent. This auth token is generated by a shared key between PAL API and SMS Server and

hence decoding it would allow the SMS Server to confirm a particular user and extract their identity.

Each SMS Type has its specific set of payload.

- **Emergency Initiation SMS:** The Emergency Initiation SMS consists of the location of the initiator. The mobile application would have the feature of using a pre-set location during offline environments.

- **Emergency Chat SMS:** This chat message consists of the unique *emergency-id* of the active emergency and the message *payload*. This SMS type is used by the initiator for sending a message to the responders.

- **Emergency Confirmation SMS:** This message consists of the unique *emergency-id* and is sent to the initiator, after an emergency initiation.

- **Emergency Responder SMS:** This message consists of the *user-id*, *emergency-id* and *distance* of the responder from the initiator.

- **Emergency Responder Chat SMS:** This message consists of *user-id* of the sender, the *emergency-id* and the *payload*, which will be routed to the initiator and all the responders to the emergency

- **Emergency Response SMS:** This message consists of the *user-id* of the responder, their *status* of response and their *distance*, in case the responder has responded in affirmative

- **Emergency Request SMS:** This message consists of the *emergency-id* and *location* of the initiator and sent to the friend group of the initiator.

- **Emergency Routed Chat SMS:** This message consists of the *emergency-id*, *payload* and *user-id* of the sender of the message.

4.1.6.2 PAL SMS JSON Structures

For each of the above-mentioned message types and structures, JSON was used as the format and payload of the SMS. Table 6 shows the payload for three different transaction types. It is the responsibility of the SMS Server to parse the transaction type of these SMS and to perform the requisite action.

Table 6: JSON SMS Structures

Message Type	JSON Structured SMS

Emergency Initiation	{
	"transaction_id":"08b4c658c503d037cfaf41308d670624",
	"transaction_type":"0",
	"auth_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
	eyJpZCI6IjdjODEyN2YyNmNiNGQzNmY4YzViNzRkMGFhN2VkNzFkIiwiaWF0IjoxNTQxODE5MjQ0LCJleHAiOjE1NDE5MDU2NDR9.
	HRVnTTI1kKIeGuG77QdBrsLEHiEd9AJx6UMXcOUtjSI",
	"latitude":"20.593683",
	"longitude":"78.962883"
	}

Emergency Chat	{
	"transaction_id":"08b4c658c503d037cfaf41308d670624",
	"transaction_type":"1",
	"auth_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
	eyJpZCI6IjdjODEyN2YyNmNiNGQzNmY4YzViNzRkMGFhN2VkNzFkIiwiaWF0IjoxNTQxODE5MjQ0LCJleHAiOjE1NDE5MDU2NDR9.
	HRVnTTI1kKIeGuG77QdBrsLEHiEd9AJx6UMXcOUtjSI",
	"message":"I am in room 203",
	"emergency_id":"08b4c658c503d037cfaf41308d682e17"
	}

Emergency Confirmation	{
	"transaction_id":"08b4c658c503d037cfaf41308d670624",
	"transaction_type":"2",
	"emergency_id":"08b4c658c503d037cfaf41308d682e17"
	}

4.1.7 Results

In this Section, we evaluate if our current system has met the requirements which were identified in Chapter 3.

4.1.7.1 REQ 1

This requirement relates to a user being able to setup their own personal profile on the system and then, being able to setup their virtual support group. For this functionality, we have the following sets of API that we have built within the system:

1. ***registerUser***: This API can be used by a user, for setting up their profile in the system. The profile is identified uniquely, by a combination of username/mobile/email. Each user is associated with a unique token, that is received as a response from the API and acts as an authentication token for subsequent API requests.

2. ***login***: The login API uses username and password for logging into the system. Once the user logs into the system, they are provided with a unique token which can be used for subsequent API calls with the backend.

3. ***updateUserInformation***: The API can be used by a user, for updating their information in the system. This primarily includes the status of their current Naloxone kit.

4. ***sendFriendRequest***: The API can be used, for sending a friend request to a user. A user can be added based on their email/mobile and can be assigned a unique nickname for ease of search.

5. ***verfyFriend***: The API can be used by the user, for verifying a friend request. Each friend request is given a unique requestor ID and each positive response is mapped to an association with the friend responder in the database. Each user has the facility, to give each other a nickname to uniquely identify them.

6. ***rejectFriend***: The API can be used for rejecting a friend request. The request structure is similar to verifyFriend.

7. ***listPendingRequests***: This API can be used for listing of pending friend requests, with their requestor ID's. The user can then use the requestor Id to confirm or reject a friend.

8. ***listFriends***: This API can be used to retrieve a list of confirmed friends.

For this particular requirement, as we can see, we have built in a set of 8 API's for enabling a user to perform their function. The API workflows complete our user management and friend management. Hence, the infrastructure is **complete for REQ1.**

4.1.7.2 REQ 2

This requirement is related to providing mechanisms for off the record communication between a user and their virtual support group. Apart from just enabling the communication, the system also needs to follow the Must Haves that were decided for the choice of our instant messaging protocols, i.e., XMPP.

For this requirement, we have implemented a *sendMessage* API within the system for sending a message to another user. The API can be used for transmitting the message to either a single user or a group of users. The API supports two messaging modes: chat and

emergency. A chat is a simple messaging which is like a regular chat. Emergency, when the messaging is happening during an ongoing emergency. For receiving messaging, IBM native push libraries are used, which map a device to their corresponding token. We use IBM Push API for sending the message to the device. Group messaging is supported, but only in emergency mode, where the emergency-id forms a subscription topic to which the responders are subscribed. The same functionality can be extended to managing groups in normal operations. Hence, our current workflows are **complete for REQ2**.

4.1.7.3 REQ 3

This requirement is related to managing an emergency within the system. For this, we have created a set of API's:

1. ***notifyEmergency***: This API can be used by a user to notify their friends about an active emergency. Each user can share their latitude and longitude, for their support group to be notified of. The API would send an emergency message to their friend/group.

2. ***respondEmergency***: This API can use used by a user, to respond to an emergency. Each emergency is given a unique emergency-id, to which a person can respond to. The responder shares their time from the emergency event and any message that they wish to send to the initiator.

3. ***cancelEmergency***: This API can be used by an initiator for cancelling an emergency.

4. ***retrieveEmergencyInformation***: This API can be used for retrieving the information regarding the status and initiator of an emergency.

5. ***retrieveEmergencyResponders***: This API can be used for retrieving the information about responders to an emergency.

The current set of API provide the required communication flow between an initiators and the responders. Thus our architecture is **Complete for REQ 3.**

4.1.7.4 REQ 4

This requirement pertains to setting up an offline failover mechanism for the system to work in no network conditions. In Chapter 3, we described our implementation for the SMS server, that can be used to access PAL backend API. We have used JSON [105] based SMS structures, to communicate with the SMS Server and to enable communication during offline environments.

The system was tested, with sending of the defined message structures and we were able to complete the following steps during an active emergency:

1. An initiator, by sending SMS, was able to inform the system for an active emergency.
2. The responder(s) received the emergency alerts via SMS and were able to respond with their status for the particular emergency.
3. The responders were informed to the initiator and the initiator and responder could then successfully communicate with each other.

For testing this aspect of the application, we used an android phones SMS interface and tested regarding all the requisite steps of this setup.

The system has been setup to provide the required offline functionality and thus, our current architecture is **Complete for REQ 4.**

4.1.7.5 REQ 5

This requirement refers to setting up fetching and displaying informational content on the mobile app. Our current implementation does not have that system in scope, since we have not yet developed the front end interface for the application. Hence, **REQ 5 is out of scope for our feasibility analysis.**

Table 7: Prototype Requirement Completeness Results

Requirement	Status
REQ 1	Complete
REQ 2	Complete
REQ 3	Complete
REQ 4	Complete
REQ 5	Out of Scope

As we can from Table 7, except for out of scope requirement REQ 5, we have indeed been successfully able to setup the backend infrastructure for all of remaining requirements. The backend infrastructure has been setup, in accordance with the recommended architecture and thus, does depict the feasibility of the same.

4.2 Emergency Service No-Internet Availability Evaluation

One of the critical requirements that the PAL system is hoping to achieve is to provide availability of crowdsourced emergency services, during no internet conditions. The challenge is to ensure that in no-internet situation, a user is stable to ask for their peer support group for help.

For the same, we have proposed a solution based on using Programmable SMS as a failover mechanism in case of no internet services. The system can use the SMS layer to send and receive the information between the users during an active emergency. Since the actual adoption of this system critically depends on fulfilling this requirement, our evaluation efforts are focused on the following two questions

"Does the SMS based failover system drop any SMS messages under load conditions"

and

"How much does the message delivery times between two users vary when using the SMS failover as the load on the system increases"

These evaluations are important as we are relying on an underlying system, Twilio, to provide the SMS processing infrastructure. It is, thus, important to check if the infrastructure is reliable and ensures that the emergency services remain available to the users.

With these two questions in mind, on load testing of the system was performed. The setup of the experiments and the results have been explained in the subsequent sections.

4.2.1 Experiment Design

Based on the key evaluation questions that we aimed answer, the experiment captured message delivery times and any message delivery losses, when using the SMS based failover mechanism. Since we are yet to roll out the application to actual users, we tested the system under load. We simulated SMS based emergency lifecycle events using the SMS sending API provided by Twilio.

The system was tested with continuous running for 40 minutes. The load started from 1 SMS/second for the first 10 minutes and increases by 1 SMS after every 10 minutes. The top load that the system reached was 4 SMS / second in the last 10 minutes of our test run.

The first thing that we captured is the time that it took for messages to be delivered between two users. We used the *Emergency Chat* message as our type of transaction and

our aim is to measure the time that it takes for a message to be delivered between two users.

The second thing that we captured is if there is any drop in messages at any of the different layers. This is important, since we must ensure that the failover system does not drop any messages / SMS during its running under a no internet condition.

For carrying out the simulation, we used only a workflow of SMS based chat communication between two users. This would typically happen after a crowd has affirmatively responded to an emergency and initiator and responder are communicating with each other. The direction of our messages was from the initiator to the responder. The responder in this case was also another Twilio Programmable SMS number, which allowed us to easily retrieve delivery timestamps of the messages.

4.2.2 Results

The average message delivery times from the situation of 1 SMS per second to 4 SMS per second are shown in Table 8:

Table 8: One-to-One Message Delivery Times

Experiment Scenario	**Average Delivery Time (in seconds)**	**PAL Server Configuration**	**Messages Lost**
One SMS Per Second for 10 minutes	2.59	1 instance, 256 MB each (Figure 32)	0
Two SMS Per Second for 10 minutes	6.17	1 instance, 256 MB each (Figure 33)	0
Three SMS Per Second for 10 minutes	11.21	1 instance, 256 MB each (Figure 34)	0
Four SMS Per Second for 10 minutes	6.77	2 instances, 512 MB each (Figure 35)	0

As we can see, as the number of SMS per second increases, the delivery time of the system increases. Having said that, there was no loss of messages in the current operation

of the system. The histogram for the delivery times for the on load situations are shown Figure 32, Figure 33, Figure 34 and Figure 35.

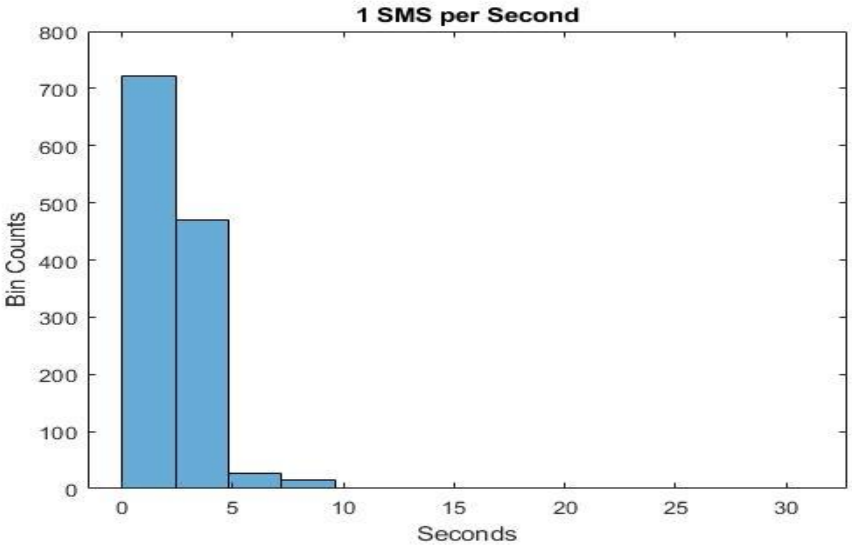

Figure 32: 1 SMS per second histogram

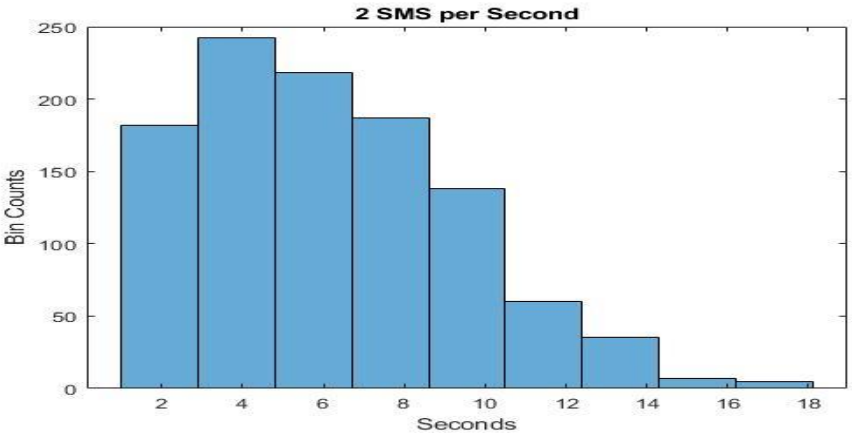

Figure 33: 2 SMS per second histogram

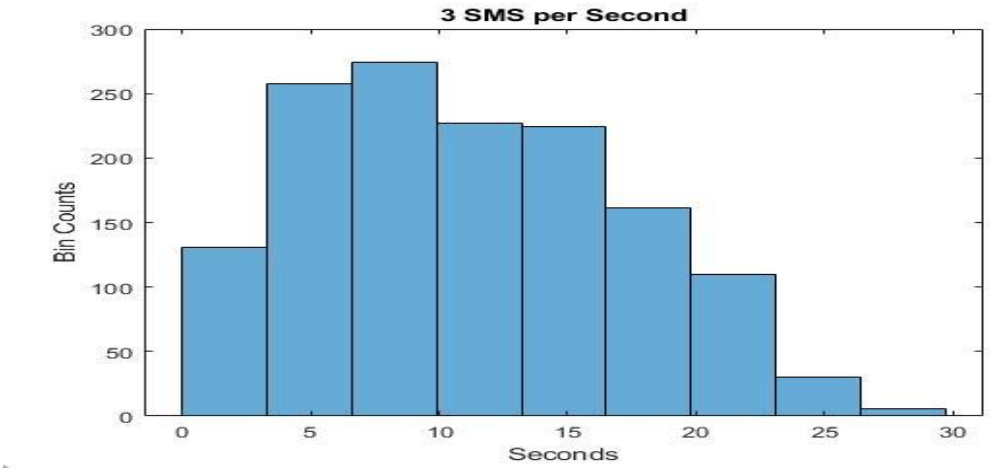

Figure 34: 2 SMS per second histogram

An interesting situation occurred when the load reached 4 SMS per second. There was a sudden drop in messages when the system reached that particular load. On further investigation, we found that our current configuration of backend PAL Server, with a single server instance allocated with 256 MB, was not able to handle such loads.

Due to this issue, testing with done with increasing the number of instances and the memory size of each instance. Upon 4 test rounds, a configuration of 512 MB with 2 parallel running instances sufficed and removed any drop of messages. For that particular configuration, the histogram is shown in Figure 35:

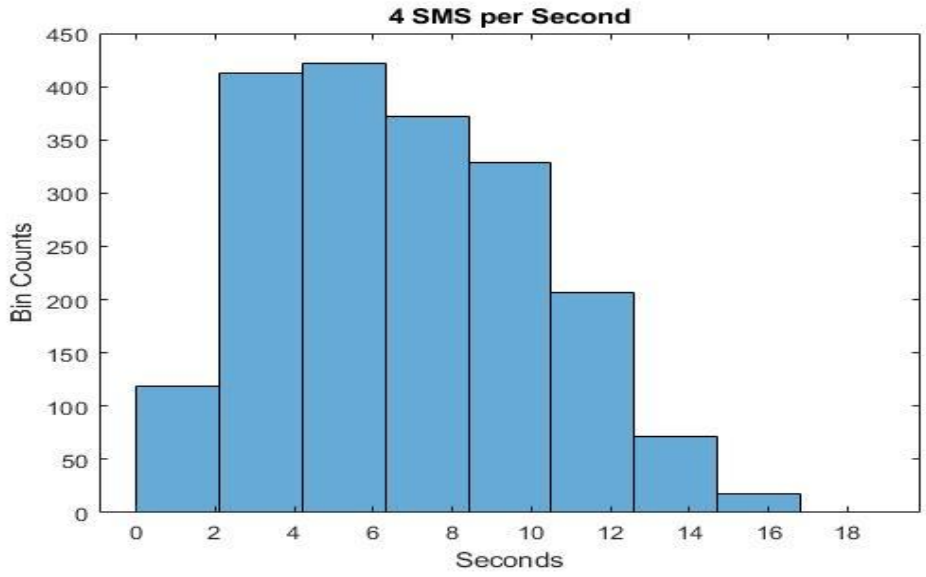

Figure 35: 4 SMS per second histogram

4.3 Conclusion and Summary

In this Chapter, our focus was on evaluation of our design recommendations. We chose *feasibility* and *no-internet emergency service availability* as the two measures for evaluating the design recommendations that were made in the system. For performing our feasibility analysis, we successfully implemented a working backend infrastructure, that supported all but one requirement of the PAL System. Through this, we proved that it is indeed possible to build a PAL type application infrastructure. For our availability analysis, we tested message delivery times on load and found that message delivery time almost doubles, as the load on the server increases by a unit SMS per second. Having said that, the SMS Gateway and Server were reliable and there was no drop in SMS Messages, which showed that the emergency service was accessible during no-internet situation.

Chapter 5

Contribution and Future Work

In this Chapter, we conclude and summarize the work that has been done in the thesis. We discuss the trade-offs that we have made and future work that may be done in this field.

5.1. Contributions

Through our study of the PAL system, we have been able to provide the following contributions to the design and implementation of such systems:

- We performed a technical review and comparison of many current open source messaging protocols. Through this, we discovered that for a crowdsourced system such as ours, which relies heavily on availability, security, and confidentiality, XMPP protocol seems to be the most well-defined mechanism to be used. We recommend using an XMPP server implementation, such as eJabber, for building the messaging infrastructure of such applications.

- We proposed a novel approach to solving the problem of system availability, during no internet conditions in a mobile application. We believe that such a system can be used in the future, for many applications that work in low/no internet conditions. Our recommendation proposes a novel approach of using programmable SMS, for building the failover mechanism of the system.

- By developing a working prototype, we confirmed the feasibility of such a solution. The prototype components are no means ready to be used for an actual system but are generic enough to verify that such a system, with SMS failover, can be built.

- We also evaluated the no-internet emergency service availability aspect of the system, by testing for message drops and delivery times, for SMS messages during the failover operation. This testing was done on load to check if any messages are dropped and how does the delivery times vary when the load increases. We observed that the message delivery times almost double as the load increases by a unit value.

Although we have made significant contributions towards the design aspect of such PAL solutions, we have a long way to go. Currently, we have only built an initial prototype of the backend infrastructure, and we have not developed a final product, with the mobile

application. Additionally, we have not done any user study for the acceptance of the system, and thus, that needs to be further explored. In the next section, we give an overview of future work that can be done on our work.

5.2 Future Work

The current PAL system is a study of design and a prototype implementation of a PAL based application. There are many aspects of the system that warrant further research:

- Although our current implementation has focused on the widespread drug overdose epidemic in North America, we believe that the solution is extensible to other at-risk communities, who are prone to social isolation and self-harm.

- The current backend system has only been tested in a simulated environment. Although extensive work was done to understand the on-ground requirements, through our collaboration with AVI, the real test of the system among its target users is yet to be done. It remains to be seen whether the users adopt such a system on the ground and if NALPAL can help in preventing deaths from overdose.

- Since our current system is a prototype, it has many moving parts which warrant further analysis and validation, before the final product is launched. These include the backend server configurations, messaging services, and the SMS failover service. Although we have shown that such a system is feasible, it is by no means a final product and needs polishing for the same.

- In our current system, we have only looked at the privacy issues of our user community, form the perspective of making the system immune from prosecution of user messages. That may not be sufficient, as the user's personal information and their support group, which may consist users with various illicit activities such as drug dealers, may also make the system vulnerable and authorities interested in accessing the said data. That needs to be further investigated and architecture redesigned to circumvent such risks.

- In our analysis of the messaging protocols, TOX seemed to be a front runner when it came to security, as it does not have any central server available and is built with the idea of privacy. Having said that, our bottleneck in maintaining user and friend list information across devices. This can specifically be solved, by using the Tox's underlying protocol, Kademlia [126] directly, for building the messaging network. Kademlia allows the network to contain key-value paired information, in a distributed

manner across nodes of the network. This architecture can thus be leveraged, to resolve the issues of TOX and hence build a secure and private messaging network.

- Another variation of the proposed SMS based failover system, could be to potentially utilize the cellular network connections, for sending and receiving of small packages of data directly. This could be done, in conjugation with cellular network providers, for setting up a failover service in case internet services are not available.

- Lastly, we have looked at system availability only from the aspect of no internet conditions. Another important issue to look at is the availability of messaging service and how we can build a multi-service/multi-cloud failover to circumvent that situation in case the backend servers go down.

Our primary hope through this work is that PAL applications would be further explored in various user domains. We have established that there is great promise in leveraging the wisdom and will of the crowd to provide emergency services.

Bibliography

[1] P. Seth, R. A. Rudd, R. K. Noonan, and T. M. Haegerich, “Quantifying the epidemic of prescription opioid overdose deaths,” Am. J. Public Health, vol. 108, no. 4, pp. 500–502, Apr. 2018

[2] Special Advisory Committee on the Epidemic of Opioid Overdoses, “National report: Apparent opioid-related deaths in Canada (January 2016 to December 2017) web-based report,” Ottawa: Public Health Agency of Canada, Tech. Rep., Jun. 2018

[3] J. Katz, “Drug deaths in America are rising faster than ever,” The New York Times, Jun. 2017. Accessed on 04-January-2018.

[4] D. Ciccarone, “Fentanyl in the US heroin supply: A rapidly changing risk environment,” Int. J. Drug Policy, vol. 46, pp. 107–111, Aug. 2017

[5] T. J. Cicero, M. S. Ellis, H. L. Surratt, and S. P. Kurtz, “The changing face of heroin use in the united states: a retrospective analysis of the past 50 years,” JAMA Psychiatry, vol. 71, no. 7, pp. 821–826, Jul. 2014

[6] K. A. Mack, C. M. Jones, and M. F. Ballesteros, “Illicit drug use, illicit drug use disorders, and drug overdose deaths in metropolitan and nonmetropolitan areas - united states,” MMWR Surveill. Summ., vol. 66, no. 19, pp. 1–12, Oct. 2017

[7] Ma, Shang-Pin, et al. "Framework for enhancing mobile availability of RESTful services." Mobile Networks and Applications 21.2 (2016): pp. 337-351.

[8] Elbashir K, Deters R (2005) Transparent caching for nomadic ws clients. In: Proceedings of the IEEE international conference on web services. ICWS ’05. IEEE Computer Society, Washington, DC, pp. 177–184.

[9] Fernandez J, Fernandez A, Pazos J (2005) Optimizing web services performance using caching. In: Proceedings of the inter- national conference on next generation web services practices. NWESP ’05. IEEE Computer Society, Washington, DC, pp. 157– 162

[10] Liu X, Deters R (2007) An efficient dual caching strategy for web service-enabled pdas. In: Proceedings of the 2007 ACM symposium on applied computing. SAC ’07. ACM, New York, pp. 788–794

[11] Papageorgiou A, Schatke M, Schulte S, Steinmetz R (2011) Enhancing the caching of web service responses on wireless clients. In: Proceedings of the 2011 IEEE international conference on web services. ICWS ’11. IEEE Computer Society, Washington, DC, pp. 9–16

[12] Chang C, Ling S, Krishnaswamy S (2011) Promws: proac- tive mobile web service provision using context-awareness. In: 2011 IEEE international conference on pervasive comput- ing and communications workshops (PERCOM Workshops), pp. 69–74

[13] Liyanaarachchi A, Weerawarana S (2012) An end-to-end caching protocol for web services. In: 2012 international conference on advances in ICT for emerging regions (ICTer), pp. 96–102

[14] Katsaros G, Kubert R, Gallizo G (2011) Building a service- oriented monitoring framework with REST and Nagios. In: Pro- ceedings of the 2011 IEEE international conference on services computing. SCC '11. IEEE Computer Society, Washington, DC, pp. 426–431

[15] Zeginis C, Konsolaki K, Kritikos K, Plexousakis D (2012) ECMAF: an event-based cross-layer service monitoring and adaptation framework. In: Proceedings of the 2011 international conference on service-oriented computing. ICSOC'11. Springer, Berlin, pp. 147–161

[16] Spillner J, Utlik A, Springer T, Schill A (2013) RAFT-REST—a client-side framework for reliable, adaptive and fault-tolerant rest- ful service consumption. In: Lau K-K, LamersdorfW, Pimentel E (eds) Service-oriented and cloud computing, vol 8135 of lecture notes in computer science. Springer, Berlin, pp. 104–118

[17] Olasina, G. (2015). Factors influencing the use of m-Banking by academics: Case study SMS-based m-Banking. The African Journal of Information Systems, 7(4), 4.

[18] Montes, J. M., Medina, E., Gomez-Beneyto, M., & Maurino, J. (2012). A short message service (SMS)-based strategy for enhancing adherence to antipsychotic medication in schizophrenia. *Psychiatry research*, *200*(2-3), pp. 89-95.

[19] Kannisto, K. A., Koivunen, M. H., & Välimäki, M. A. (2014). Use of mobile phone text message reminders in health care services: a narrative literature review. *Journal of medical Internet research*, *16*(10).

[20] Susanto, T. D., & Goodwin, R. (2010). Factors influencing citizen adoption of SMS-Based e-Government Services. Electronic journal of e-government, 8(1).

[21] Ren, L., & Song, J. (2002). Data synchronization in the mobile Internet. In Computer Supported Cooperative Work in Design, 2002. The 7th International Conference, pp. 95-98. IEEE.

[22] Go, Y., Agrawal, N., Aranya, A., & Ungureanu, C. (2015, February). Reliable, Consistent, and Efficient Data Sync for Mobile Apps. In FAST (Vol. 15), pp. 359-372.

[23] Perkins, D., Agrawal, N., Aranya, A., Yu, C., Go, Y., Madhyastha, H. V., & Ungureanu, C. (2015, April). Simba: Tunable end-to-end data consistency for mobile apps. In Proceedings of the Tenth European Conference on Computer Systems (p. 7). ACM.

[24] Wheeler, E., Jones, T. S., Gilbert, M. K., & Davidson, P. J. (2015). Opioid overdose prevention programs providing naloxone to laypersons-United States, 2014. MMWR. Morbidity and mortality weekly report, 64(23), pp. 631-635.

[25] Micah Robbins "How naloxone saves lives—and why it's so controversial". Retrieved from iodine.com. Accessed on 05-April-2018.

[26] Ambrose, G., Amlani, A., & Buxton, J. A. (2016). Predictors of seeking emergency medical help during overdose events in a provincial naloxone distribution programme: a retrospective analysis. BMJ open, 6(6), e011224.

[27] AVI What We Do. Retrieved from http://avi.org/what-we-do. Accessed on 20-March-2018.

[28] React Native. Retrieved from https://facebook.github.io/react-native/. Accessed on 10-September-2018.

[29] Robinson, David. Exploring the State of Mobile Development with Stack Overflow Trends. Retrieved from https://stackoverflow.blog. Accessed on 10-September-2018.

[30] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine

[31] About Node.js. Retrieved from https://nodejs.org/en/about/. Accessed on 30-May-2018.

[32] Overview of Blocking vs Non-Blocking. Retrieved from https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/. Accessed on 30-May-2018.

[33] Dua, Kinjal. A Guide to Mobile App. Development: Web vs. Native vs. Hybrid. Retrieved from https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/. Accessed on 25-October-2018.

[34] Node.js. Retrieved from https://nodejs.org/. Accessed on 30-May-2018

[35] What is IBM Cloud?. Retrieved from https://www.ibm.com/cloud/. Accessed on 30-May-2018.

[36] Foote, Keith D. A Review of Different Database Types: Relational versus Non-Relational. Retrieved from

http://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/. Accessed on 01-May-2018

[37] IBM Cloudant. Retrieved from https://www.ibm.com/ca-en/marketplace/database-management. Accessed on 05-May-2018

[38] Firebase Cloud Messaging. Retrieved from https://firebase.google.com/docs/cloud-messaging/. Accessed on 12-June-2018.

[39] Apple Push Notification Service. Retrieved from https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW1. Accessed on 12-June-2018

[40] Push Notifications. Retrieved from https://console.bluemix.net/catalog/services/push-notifications. Accessed on 15-June-2018

[41] Azure Service Bus. Retrieved from https://azure.microsoft.com/en-ca/services/service-bus/. Accessed on 15-June-2018

[42] Amazon SNS. Retrieved from https://aws.amazon.com/sns. Accessed on 15-June-2018

[43] NPM. Retrieved from https://www.npmjs.com/. Accessed on 10-May-2018

[44] Postman. Retrieved from https://www.getpostman.com/. Accessed on 15-July-2018

[45] Webhook. Retrieved from https://en.wikipedia.org/wiki/Webhook. Accessed on 17-September-2018

[46] Overdose Statistics. Retrieved from https://www.drugabuse.gov/related-topics/trends-statistics/overdose-death-rates. Accessed on 01-September-2018

[47] Stigma. Retrieved from https://www.canada.ca/en/health-canada/services/substance-use/problematic-prescription-drug-use/opioids/stigma.html. Accessed on 01-September-2018

[48] Olsen, Y., & Sharfstein, J. M. (2014). Confronting the stigma of opioid use disorder—and its treatment. Jama, 311(14), pp. 1393-1394.

[49] Fentanyl Overdose. Retrieved from https://drugabuse.com/library/fentanyl-overdose/#signs-and-symptoms-of-fentanyl-overdose. Accessed on 02-September-2018

[50] Illicit drug overdose deaths in BC. Retrieved from https://www2.gov.bc.ca/assets/gov/birth-adoption-death-marriage-and-

divorce/deaths/coroners-service/statistical/illicitdrugoverdosedeathsinbc-findingsofcoronersinvestigations-final.pdf. Accessed on 02-September-2018

[51] Despite 'Good Samaritan' law, many drug users too scared of arrest to report overdoses. Retrieved from https://www.cbc.ca/news/politics/good-samaritan-drug-overdose-fentanyl-politics-parliament-1.4786094. Accessed on 02-September-2018

[52] Barriers to Calling 9-1-1 during Overdose Emergencies in a Canadian Context. Retrieved from http://www1.uwindsor.ca/criticalsocialwork/barriers_calling_911. Accessed on 02-September-2018.

[53] About the Good Samaritan Drug Overdose Act. Retrieved from https://www.canada.ca/en/health-canada/services/substance-use/problematic-prescription-drug-use/opioids/about-good-samaritan-drug-overdose-act.html. Accessed on 02-September-2018.

[54] Matrix Specification. Retrieved from https://matrix.org/docs/spec/. Accessed on 01-May-2018.

[55] Matrix. Retrieved from https://matrix.to/. Accessed on 01-May-2018.

[56] Besaleva, L. I., & Weaver, A. C. (2016, January). Crowdsourcing for Emergency Response. In Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS) (p. 248). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[57] Vivacqua, A. S., & Borges, M. R. (2012). Taking advantage of collective knowledge in emergency response systems. Journal of Network and Computer Applications, 35(1), pp. 189-198.

[58] Tok Specification. Retrieved from https://toktok.ltd/spec.html. Accessed on 10-May-2018.

[59] Distributed hash table. Retrieved from https://en.wikipedia.org/wiki/Distributed_hash_table. Accessed on 10-May-2018.

[60] NACL. Retrieved from https://nacl.cr.yp.to/box.html. Accessed on 10-May-2018.

[61] Kademila. Retrieved from http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf. Accessed on 14-May-2018.

[62] Toxcore design proposal. Retrieved from Thttps://docs.google.com/document/d/1op6zGR0KYdF7tTWSSX79KQieJu30vLZ6XG327kIBhxQ/edit#. Accessed on 10-May-2018.

[63] Moving Tox across devices. Retrieved from https://wiki.Tox.chat/users/faq#can_i_move_profile_across_devices_manually. Accessed on 22-May-2018.

[64] Extensible Messaging and Presence Protocol (XMPP): Core. Retrieved from https://datatracker.ietf.org/doc/rfc6120/. Accessed on 05-May-2018.

[65] Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Retrieved from https://datatracker.ietf.org/doc/rfc6121/. Accessed on 01-May-2018.

[66] WhatsApp. Retrieved from https://www.whatsapp.com/. Accessed on 01-May-2018.

[67] FunXMPP protocol. Retrieved from https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol. Accessed on 01-May-2018.

[68] SASL. Retrieved from https://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer. Accessed on 15-May-2018.

[69] TLS Retrieved from https://en.wikipedia.org/wiki/TLS. Accessed on 20-May-2018.

[70] Current Off-the-Record Messaging Usage. Retrieved from https://xmpp.org/extensions/xep-0364.html. Accessed on 20-May-2018.

[71] XMPP Message Processing Hints. Retrieved from https://xmpp.org/extensions/xep-0334.html. Accessed on 15-May-2018.

[72] XEP-0154: User Profile. Retrieved from https://xmpp.org/extensions/xep-0154.html. Accessed on 01-May-2018.

[73] XEP-0013: Flexible Offline Message Retrieval. Retrieved from https://xmpp.org/extensions/xep-0013.html#remove-all. Accessed on 15-May-2018.

[74] End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). Retrieved from https://xmpp.org/rfcs/rfc3923.html. Accessed on 15-May-2018.

[75] MQTT FAQ. Retrieved from http://mqtt.org/faq. Accessed on 20-May-2018.

[76] OASIS. Retrieved from https://www.oasis-open.org/. Accessed on 20-May-2018.

[77] MQTT projects. Retrieved from http://mqtt.org/projects. Accessed on 20-May-2018.

[78] Building Facebook Messenger. Retrieved from https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920. Accessed on 15- May-2018.

[79] MQTT Man Page. Retrieved from http://mosquitto.org/man/mqtt-7.html. Accessed on 15- May-2018.

[80] MQTT Version 3.1.1. Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. Accessed on 15- May-2018.

[81] ejabbered. Retrieved from https://www.process-one.net/en/ejabberd. Accessed on 10-May-2018.

[82] Firebase Cloud Messaging XMPP Server. Retrieved from https://developers.google.com/cloud-messaging/ccs. Accessed on 10-May-2018.

[83] Apple Push XMPP query. Retrieved from https://www.quora.com/What-technology-does-the-iOS-Apple-Push-Notification-Service-APNS-use-to-maintain-a-persistent-connection-with-each-device-to-receive-such-fast-push-notifications. Accessed on 22-May-2018.

[84] Javascript. Retrieved from https://www.javascript.com/. Accessed on 10-April-2018.

[85] IBM Push Android Client SDK, https://github.com/ibm-bluemix-mobile-services/bms-clientsdk-android-push. Accessed on 22-May-2018.

[86] IBM Push Swift Client SDK. https://github.com/ibm-bluemix-mobile-services/bms-clientsdk-swift-push. Accessed on 22-May-2018.

[87] Estellés-Arolas, E., & González-Ladrón-De-Guevara, F. (2012). Towards an integrated crowdsourcing definition. Journal of Information science, 38(2), pp. 189-200.

[88] Howe, J. (2006). The rise of crowdsourcing. Wired magazine, 14(6), pp. 1-4.

[89] Brabham, D. C. (2008). Crowdsourcing as a model for problem solving: An introduction and cases. Convergence, 14(1), pp. 75-90.

[90] Doan, A., Ramakrishnan, R., & Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. Communications of the ACM, 54(4), pp. 86-96.

[91] Amazon Mechanical Turk. Retrieved from https://www.mturk.com/. Accessed on 01-November-2018

[92] CAPTCHA. Retrieved from https://en.wikipedia.org/wiki/CAPTCHA. Accessed on 01-November-2018.

[93] Twitter. Retrieved from https://twitter.com/?lang=en. Accessed on 12-June-2018

[94] Buecheler, T., Sieg, J. H., Füchslin, R. M., & Pfeifer, R. (2010). Crowdsourcing, open innovation and collective intelligence in the scientific method: a research agenda and operational framework. In The 12th International Conference on the Synthesis and Simulation of Living Systems, Odense, Denmark, 19–23 August 2010 (pp. 679-686). MIT Press.

[95] Malone, T. W., Laubacher, R., & Dellarocas, C. (2009). Harnessing crowds: Mapping the genome of collective intelligence.

[96] Pfeifer, R., & Scheier, C. (2001). Understanding intelligence. MIT press.

[97] Heinzelman, J., & Waters, C. (2010). Crowdsourcing crisis information in disaster-affected Haiti. Washington, DC: US Institute of Peace.

[98] Naloxone. Retrieved from https://en.wikipedia.org/wiki/Naloxone. Accessed on 05-March-2018

[99] Ushahidi. Retrieved from https://www.ushahidi.com/. Accessed on 10-August-2018.

[100] Google Earth. Retrieved from https://www.google.com/earth/. Accessed on 02-November-2018.

[101] Open Street Map. Retrieved from https://www.openstreetmap.org. Accessed on 02-November-2018.

[102] Maslow's hierarchy of needs. Retrieved from https://en.wikipedia.org/wiki/Maslow%27s_hierarchy_of_needs. Accessed on 20-October-2018

[103] Python. Retrieved from https://www.python.org/. Accessed on 01-April-2018.

[104] XMPP Advanced Message Processing. Retrieved from https://xmpp.org/extensions/xep-0079.html. Accessed on 10-May-2018.

[105] Twilio. Retrieved from https://www.twilio.com/. Accessed on 01-November-2018.

[106] JSON. Retrieved from https://www.json.org/. Accessed on 01-November-2018

[107] HTTP Verbs. Retrieved from https://restfulapi.net/http-methods/. Accessed on 01-November-2018

[108] Uniform Resource Identifiers. Retrieved from https://en.wikipedia.org/wiki/Uniform_Resource_Identifier. Accessed on 01-November-2018

[109] FrontlineSMS. Retrieved from https://www.frontlinesms.com/. Accessed on 01-November-2018.

[110] Synapse. Retrieved from https://github.com/matrix-org/synapse. Accessed on 01-November-2018.

[111] JVM Toxcore. Retrieved from https://github.com/TokTok/jvm-toxcore-c. Accessed on 01-November-2018

[112] Python Toxcore. Retrieved from https://github.com/TokTok/py-toxcore-c. Accessed on 01-November-2018

[113] Javascript Toxcore. Retrieved from https://github.com/TokTok/js-toxcore-c. Accessed on 01-November-2018.

[114] Jabber. Retrieved from https://www.jabber.org/faq.html#jabber. Accessed on 01-November-2018

[115] Transmission Control Protocol. Retrieved from https://en.wikipedia.org/wiki/Transmission_Control_Protocol. Accessed on 01-November-2018.

[116] Extensible Markup Language. Retrieved from https://en.wikipedia.org/wiki/XML. Accessed on 01-November-2018.

[117] XMPP Multi User Chat. Retrieved from https://xmpp.org/extensions/xep-0045.html. Accessed on 20-May-2018.

[118] HiveMQ. Retrieved from https://www.hivemq.com/. Accessed on 20-May-2018.

[119] Android Developer Website. Retrieved from https://developer.android.com/. Accessed on 10-November-2018.

[120] Apple Developer Website. Retrieved from https://developer.apple.com/. Accessed on 10-November-2018.

[121] Spring. Retrieved from https://spring.io/. Accessed on 10-November-2018.

[122] Internet Engineering Task Force. Retrieved from https://www.ietf.org/. Accessed on 01-November-2018.

[123] XMPP Clients. Retrieved from https://xmpp.org/software/libraries.html. Accessed on 10-May-2018.

[124] XMPP Servers. Retrieved from https://xmpp.org/software/servers.html. Accessed on 10-May-2018.

[125] MQTT Libraries. Retrieved from https://github.com/mqtt/mqtt.github.io/wiki/libraries. Accessed on 01-November-2018.

[126] Kademlia. Retrieved from https://en.wikipedia.org/wiki/Kademlia. Accessed on 21-12-2018.