ReDiCom: Resilient Communication for First Responders in Disaster Management

Jiachen Chen*, Yuxuan Xing†, K.K. Ramakrishnan‡, Mohammad Jahanian‡, Hulya Seferoglu†, and Murat Yuksel§
*WINLAB, Rutgers University †University of Illinois at Chicago
‡University of California, Riverside §University of Central Florida

I. Introduction

Effective communication among first responders during and in the aftermath of a disaster can affect outcomes dramatically. We seek to build a resilient architecture that allows first responders to communicate even with: *1*) damage to infrastructure — civilian and / or specialized communication facilities may be damaged by the disaster, and *2*) dynamically formed groups — first responder teams may be formed dynamically in response to a disaster and team member addresses (*e.g.*, phone numbers, network addresses) may not be known to one another. We propose a resilient network architecture that allows efficient communication among first responders during and after a disaster [1]. We seek to support dynamically formed groups for incident response, allowing first responders to securely and conveniently communicate based on roles (names). The architecture supports communication in disasters by *1*) building resilience into the framework across all the layers, *2*) creating a framework that allows communication by role and identity, rather than addresses, *3*) supporting multiple modalities (data, voice) for communication among dynamically formed first responder teams, and *4*) providing robust and resilient communication and computing even when facilities are error- and disruption-prone.

In our demo, we will show an emulated disaster management situation with a remote command center, and a data mule that shuttles between the command center and a shelter. We will demonstrate the resilient network architecture, including: *1*) propagation of the namespace across fragmented networks, *2*) device-to-device (D2D) communication software that can utilize heterogeneous wireless links (*i.e.*, Bluetooth and WiFi Direct), and *3*) coded cooperative computing mechanisms in heterogeneous and time-varying environments.

II. Demo Architecture

To achieve the above functions, we designed a layered architecture to allow for flexibility in adapting each layer as needed to support different functionality at the lower layers (*e.g.*, different network layers and different link layers). From bottom to top, the architecture has the following layers (see Fig. 1):

The **link layer** exploits D2D communication to complement infrastructure-based communications. It provides a generic "link" abstraction to the upper layer, providing connectivity no matter what the underlying technology is: WiFi, WiFi-Direct, Bluetooth or infrastructure-based cellular or wired network connections. The identity in this layer is `NeighborID`, a variable-length device ID.

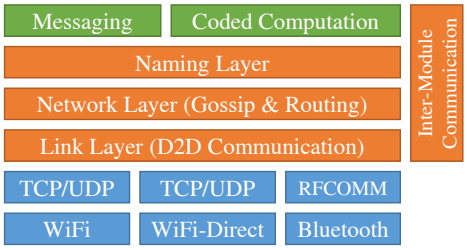


Fig. 1: Architectural view of ReDiCom.

The **network layer** supports data dissemination using flat identifiers over both connected and delay-tolerant links. It provides a "connected network" abstraction to the upper layer. The identity in this layer is `Name` (a fixed-length flat identifier independent of the location, similar to MobilityFirst [2]).

The **naming layer** maintains a graph-based namespace and performs name expansion for handling publications (similar to the protocol in [3]). It provides an abstraction of "connected names" to the applications. The identity in this layer is also a `Name`, but with graph-based relationships among them.

Applications are supported by the **application layer**. A messaging app allows first responders to communicate using the graph-based namespace. A coded computation app allows first responders to perform compute-intensive tasks with the assist from nearby *helpers*.

To assist the communication between layers, we implemented an **inter-module communication** component which mimics the `Context`-based broadcast in Android, but with lower overhead and no limitation on the object size.

III. Demo Topology and Scenario

We assume a disaster happens requiring first responders to assemble teams for search and rescue. We have two places: a coordination center and a shelter which does not have communication infrastructure support (see Fig. 2). At the coordination center, we have an `Incident Commander` that takes control of managing the disaster and a `Dispatcher` that dispatches units upon instructions from the commander. In the shelter, we have several first responders. All the connections in the shelter are based on D2D communication via either WiFi Direct (orange connections) or Bluetooth (blue connections). A `Patrol Car` travels between the coordination center and the shelter to carry messages around, acting as a data mule. We emulate the patrol car movement by manually connecting (and disconnecting) the device with (and from) the `Coordination Center` or `Rescue 1` in the shelter.

A. Messaging

To setup a search and rescue team, the `Incident Commander` can choose to create a set of names based on predefined

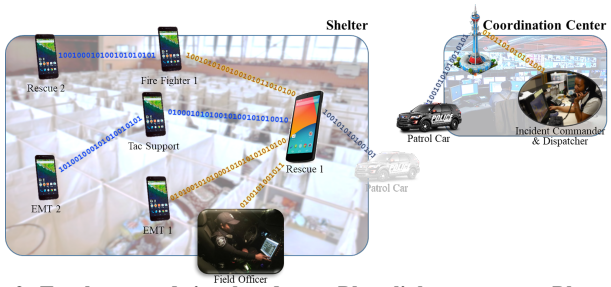


Fig. 2: Topology used in the demo. Blue links represent Bluetooth connections; Orange links represent WiFi connections; Patrol car moves between the Coordination Center and the Shelter.

templates. All the names in the selected template will be added to the namespace with random naming layer `Names`. Corresponding meanings (the human-readable name and type for each name) are maintained in the messaging app.

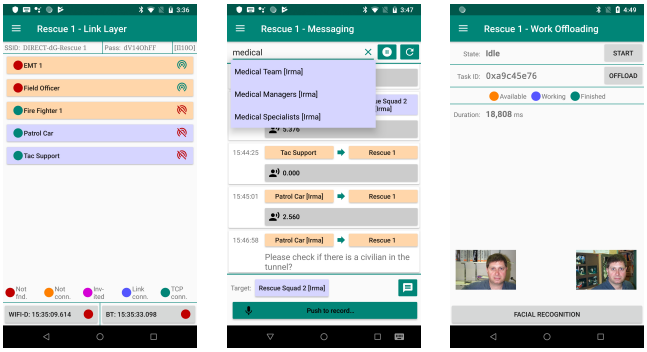
The `Dispatcher` dispatches units by drag-and-drop of names (roles) to particular individuals available in the different organizations. The messaging app keeps track of the new relationships and delivers events to the naming layer (and further to the Gossip module). At the same time, the `Incident Commander` can see the updates since they are in the same connected sub-net. However, none of the first responders in the shelter would see them yet as they are physically disconnected. Once the units are dispatched, the `Incident Commander` can send messages (or push-to-talk) to the roles (or the groups). These messages will be synchronized on all the devices in the Gossip module, including the `Patrol Car` which is also in the coordination center at the beginning.

Once the `Patrol Car` connects to the shelter (`Rescue 1`), the namespace of each device in the shelter will be updated (can be seen on the GUI of `Field Officer` and first-responder smartphones). The role-based messages from the `Incident Commander` will also be delivered to the corresponding first responders. The first responders can talk to each other based on their new roles, and the communication can be delivered over multiple communication technologies (WiFi-Direct and Bluetooth) and multiple hops (in Bluetooth). They can also send buffered messages to the `Incident Commander`. The `Field Officer` can modify the namespace (*e.g.*, add another first responder into the incident namespace). When the `Patrol Car` returns to the coordination center, the updates on the namespace and the messages sent back will be reflected on the GUI of `Incident Commander`.

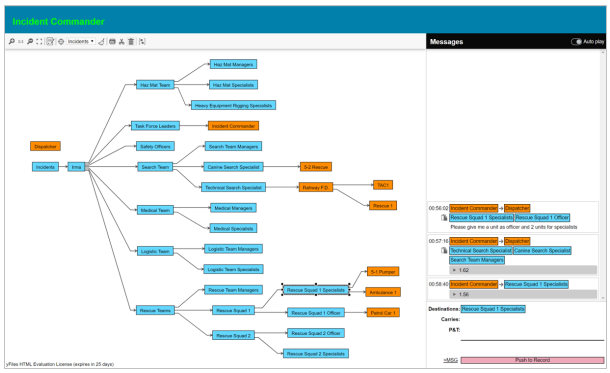
B. Coded Computation

We built two distributed computation applications on top of the naming layer: face recognition and matrix multiplication. The devices in the resilient network form a master/helper cluster. Master devices offload computation tasks to helper devices via D2D connections. The helpers are detected via the pub/sub mechanism.

For face recognition, a master device wants to identify a target person from a set of images taken in the shelter. The master distributes images to helpers and the helpers responds with labels and confidence. After the master gathers all the information, it outputs the closest match of the target person



(a) Link layer (b) Messaging (c) Coded computation



(d) Messaging on PC/Tablet

Fig. 3: Screen shots of ReDiCom.

in the database. Depending on the processing and transmission delay of different helpers, the master device allocates workload dynamically to adapt different helpers.

For matrix multiplication, we assume that the master device wants to compute a matrix multiplication; $\mathbf{y} = A\mathbf{b}$, where A is a matrix, and $\mathbf{b}$ is a vector. The master divides the matrix row-wise equally into two parts A_1 and A_2. The master sends A_1, A_2, and $A_1 + A_2$ (coded matrix) to three helpers. Also, vector $\mathbf{b}$ is sent to all helpers. Helpers calculate $A_1\mathbf{b}$, $A_2\mathbf{b}$, and $(A_1 + A_2)\mathbf{b}$ in parallel. Note that the master device only needs to retrieve the result from the fastest two helpers to calculate $\mathbf{y} = A\mathbf{b}$. Thus, the master can avoid waiting for the slowest helper, which reduces the overall delay.

C. GUI

Fig. 3 shows several screen shots from the demo. The link layer GUI (Fig. 3a) allows the user to control the connected neighbors. The Messaging GUIs (Fig. 3b and 3d) allow first responders send text and push-to-talk messages based on roles. The GUI of coded computation (Fig. 3c) allows user to select target person in the shelter and displays the person found. Figures of the other components are not shown due to the space limitation.

ACKNOWLEDGEMENT

This work was supported by the US NIST (award 70NANB17H188).

REFERENCES

[1] J. Chen *et al.*, "CNS: Content-Oriented Notification Service for Managing Disasters," in *ICN*, 2016.

[2] A. Venkataramani *et al.*, "MobilityFirst: A Mobility-Centric and Trustworthy Internet Architecture," *SIGCOMM*, 2014.

[3] M. Jahanian *et al.*, "Graph-based Namespaces and Load Sharing for Efficient Information Dissemination in Disasters," in *ICNP*, 2019.