Information Rich GIS Dissemination in Disconnected Environments

William Johnston, Nilanjan Banerjee, Jackson Cothren, James P. Parkerson

Abstract Information-rich maps are today rendered from powerful back-end servers. For instance, Google maps or Bing maps are web services that are resident on compute capable and energy hungry server class machines. Unfortunately, access to such Internet-resident web services is infeasible in disconnected environments such as a natural disaster. However, it is important to be able to disseminate map based data to route survivors of natural disasters around dangerous zones and provide directions to nearest relief camps during the aftermath of a natural calamity. To this end, this paper presents a software stack (a map stack) that can serve GIS information from low power embedded nodes. The system combines a spatially enabled SQL database, location-based routing, and multi-scale map rendering to serve information-rich GIS data on common handheld devices at minimal energy consumption. The maps are rendered on browsers on off-the-shelf mobile devices such as smart phones and tablets. The system also provides a crowd sourced capability where end users can annotate maps with up-to-date information on the scene of the natural disaster. We have prototyped a fully functional map stack on a battery powered Gumstix Overo air platform. We show that the map stack is a highly extensible platform that provides low latency, and low energy dissemination of maps during a natural disaster.

Keywords Embedded platform · GIS Systems · Crowdsourcing · Renewable Energy

1 Introduction

Disconnected environments such as natural disasters are characterized by lack of network connectivity and power grid access [1, 2]. A primary challenge in such disconnected environments is access to information for end users. For instance, consider the following: during the aftermath of the Tokyo earthquake [2], the power grid and the Internet were unavailable for months. The available first responder systems after natural disasters provide minuscule amount of information to survivors. For instance, state-of-the-art siren systems provide information on the presence or absence of danger during a tornado strike. Such systems cannot substitute for web services such as Google maps or Bing maps that are often used for dissemination of direction and map data to users.

A plausible substitute to centralized (and inaccessible) web services during natural disasters is a self-sustainable interconnected network of embedded nodes (wireless mesh) that can serve map based information to handheld devices. The mesh acts as a sustainable distributed repository of rich, up-to-date, user-annotated map data that can act as a black-box data source after a natural disaster, a critical information dissemination channel for first responders, and a substrate to locate survivors.

There are several practical challenges that must be addressed before a large scale deployment of such a mesh is feasible. First, for self-sustainability, the mesh
nodes should consume ultra low power. Hence, the software stack on the low power embedded devices should be capable of rendering rich map-based data on a small energy budget. The problem amounts to porting resource hungry software modules to function seamlessly on ultra-low power embedded platforms. While serving map data from resource rich back-end servers is possible, small embedded nodes present unique constraints in memory capacity, energy consumption, and storage that the system should be cognizant of. Second, the mesh should be self adapting and dynamic. During natural disasters, the topology of the environment changes rapidly; a bridge might have fallen or a road might be flooded. Hence, users should have the capability of annotating maps on their hand-held devices with up-to-date information, and push such updates to the mesh. The up-to-date maps can be downloaded by other users in geographically distant locations. Moreover, if the mesh nodes are powered by renewable energy sources such as solar panels, the energy consumption of the platform should adapt as a function of the residual battery capacity.

Driven by the above design challenges, we present a map stack, a software stack that can serve rich GIS data from resource constrained low power embedded nodes. The map stack combines Mapserver [3], Apache [4], Openlayers [5], PostGIS [6], and PgRouting [7] to provide information rich maps on common hand-held devices such as laptops, mobile phones, and tablets from low power embedded nodes. The maps are rendered on standard browsers. The system also provides a crowdsourced plug-in that ensures up-to-date data on the mesh nodes. End users can annotate maps served by the mesh with alerts, possible roadblocks, and infrastructure collapses. We have prototyped a fully functional map stack on the low power Gumstix Overo air platform [8]. We exhaustively evaluate the system’s user interface, energy and memory footprints, and latency performance and show that the map stack is an extensible platform adequate for deployment in disconnected environments.

2 Research Contributions

The design, implementation, and evaluation of our map stack presents the following core research contributions.

– To the best of our knowledge, this map stack is the first fully functional software system that can serve rich map data on common hand-held devices from resource constrained low power embedded nodes. Our map stack can disseminate maps with varying resolution on end user devices, proportional to the residual battery capacity. The software system uses the present location of the user, data on dangerous zones, and a routing algorithm to route users around areas of danger.

– The map stack provides users with the ability to annotate maps with a user-defined layer. The user-defined layer may contain information such as fallen bridges, or collapsed infrastructure. The layer is pushed back to the embedded node. The system seamlessly uses this information to route survivors around user-annotated dangerous zones.

– We have implemented and evaluated a fully functional map stack on a low power embedded platform on metrics such as energy consumption, dissemination latency, and user interface on popular hand-held devices.

3 Related Work

Our system builds on previous work on GIS and map dissemination, crowd-sourcing enhanced mobile systems, and low power embedded and sensor systems. Here we compare and contrast our map stack system with the most related literature.

3.1 GIS and Map Dissemination

Traditionally, map dissemination on mobile devices and desktops occurs using web services resident on powerful back-end servers. Recently, there has been a paradigm shift towards implementing GIS servers on embedded systems. For instance, research on architectural elements of GIS on embedded mobile platforms has been performed [12]. Special focus has been paid on GPS and location based services [34, 41]. These systems, however, still rely on back-end services running on powerful servers [23]. A recent set of natural emergencies have also triggered the use of map servers during emergency response management [37]. Unfortunately, there is little to no research on a comprehensive, and practical software stack that can function efficiently on low power embedded devices in a disconnected environment.

3.2 Crowdsourcing

Crowdsourcing refers to using a human network to solve a computationally expensive problem. With the growing popularity of smartphone devices, several crowdsourcing based applications have been designed. For instance, common applications include image tagging and search [25, 40], and vehicle tracking and mapping [11,
Crowdsourcing approaches have also been adopted to extract information during the aftermath of a natural disaster [15, 24, 38, 43]. While most crowdsourcing approaches are used to populate backend servers, our system uses crowdsourcing in a disconnected environment.

3.3 Low Power Embedded Systems

A primary design constraint of embedded, sensor, and mobile systems is energy management. Approaches such as CPU DVFS [16, 39, 42], disk spindown [14, 18], turning off banks of RAM [19], microsleep [19], Wi-Fi power saving modes [28, 29], and multiple radios [27, 31] are applied for conserving energy on laptops and mobile phones. Unconventional approaches, such as Hierarchical Power Management (HPM) [9, 10, 32] combines multiple platforms of varied power draw and capabilities to explore trade-offs between energy consumption, responsiveness, and availability. Hardware researchers have also proposed energy-efficient platforms such as the mPlatform [17], PASTA [30], and LEAP [13]. Additionally, there is considerable research in the sensor community on green systems [20–22, 33]. The driving application of these systems is environmental sensing. Our work is complementary to research performed in low power embedded systems. Our goal in this paper is to design a low power, adaptable software stack that can function on the designed hardware platforms to provide map-based data to survivors of natural disasters.

4 Design Goals

Nodes deployed during emergency conditions such as a natural disaster should be portable, untethered (battery powered) and self-sustainable (preferably solar powered). However, portable embedded devices that can function on limited battery power are constrained in memory, storage, and compute power. For instance, the Gumstix platform [8] used in our prototype has 128 MB RAM and a 600 MHz micro-processor. Traditional GIS data dissemination systems reside on powerful backend servers with computational and storage capabilities and hence, are difficult to directly port to embedded devices. Therefore, a clean slate design is required that adheres to the following unique design goals.

- **Low power and adaptive systems:** A primary design goal of untethered embedded nodes is long lifetime. The nodes should last for months on medium sized batteries or near-perpetually on moderate sized solar panels. Additionally, the system should adapt its energy consumption as a function of the residual battery capacity. For instance, in a solar panel powered node, it is important to adapt the energy consumption of the system as a function of the energy harvested from the panels. To meet this important design goal, our map stack renders map layers adaptively in a multiresolution fashion (§5).

- **User-friendly interface:** After a natural disaster, it is important that the map data is rendered to a survivor’s handheld using a friendly user interface. Hard to use UIs can cause user irritation, making the system useless. Moreover, it is important that the system does not force the user to download a specialized application for every personal handheld device he owns. Since rendering high resolution maps can involve transfer of a substantial amount of data from low power embedded nodes, for optimal user experience, it is important that the speed of download is high. To meet the above goal, our map stack provides an information rich, easy to use user interface on standard web browsers where users can view and annotate maps. Additionally, the system uses a set of optimizations to render maps quickly (§5).

- **Accurately rendered maps:** During the aftermath of a natural disaster, the topology and scene of the affected area changes quickly. For instance, a bridge might have fallen or a building might have collapsed. It is important that the map rendered to the user is up-to-date. One of the goals of the map is to provide directions to safe locations. It is useless and probably dangerous for the user if the system does not route him around such dangerous zones. Since network connectivity is unavailable during disconnected scenarios, it is important that alternative methods are used to populate maps with topology change information. Our system solves the problem by using a crowdsourcing-based approach where survivors can annotate maps with information about the scene, that is propagated back the embedded node (§5).

5 System Architecture and Implementation

Our system is a highly extensible software stack that is fine tuned as a map rendering engine for low power embedded platforms. It strictly adheres to the design goals outlined in §4. The overall architecture for the system is illustrated in Figure 1. The architecture uses open source tools such as pgRouting [7], Mapnik [36], Apache 2 [4], Mapserver [3], and custom software writ-
Our system employs **multi-scale dissemination** for low latency and adaptive map tile dissemination. The data transferred to a client device is divided into two types of layers: base tiles and a set of augmented layers. The base layer consists of image tiles of the area of interest. These tiles are generated offline using Mapnik [36] and a custom Python script and then stored in a PostgreSQL database resident on the low power embedded node. The image tiles are compressed jpeg files and an associated shape file that maintains a mapping of the tile boundaries and the jpeg images. While in-situ generation of tiles is possible on powerful backend servers, they are compute intensive for battery powered and compute constrained devices such as the Gumstix platform. The augmented tiles layers consists of four layers: (1) *safety zone layer*: annotations of areas that are safe such as relief camps and food stores. These zones are map annotations that are already available from public databases or are added by the users during the aftermath of a natural disaster. (2) *danger zone layer*: this layer corresponds to area annotations that must be avoided by survivors. These include fallen bridges, collapsed building, or a flooded street. Again, these annotations can be fed into the system offline or added by survivors using a web browser. (3) *crowd points and polygons*: these are point or area annotations that users explicitly add to the system after the disaster. Such annotations could be safe or danger zones. These user-annotated areas help maintain up-to-date scene information after a disaster. The points and polygons are generated using a crowdsourcing approach described in §5.5. (4) *route layer*: this layer contains information on the route from the present location of the user to the nearest safe location. The route calculation algorithm is presented in §5.3. The augmented layers are rendered from vector data while the base tiles are rendered from jpeg images.

The system can adaptively render map layers to the user’s device based on residual battery capacity or the amount of energy harvested from the panels (if the device is solar panel powered). For instance, the route to the nearest safety location may not be calculated if the residual battery capacity is low or the amount of energy that is likely to be harvested in the future is small. Transferring a subset of layers to a client device reduces the amount of data transferred over the radio and the number of queries made to the backend database, saving energy. Another optimization our system performs is intelligently choosing the point of aggregation for the map tiles and the vector layers. In our system, this aggregation occurs at the client. While Mapserver can be used to combine the vector data from the database with the map tiles to produce new tiles, our experiments

---

1 The Gumstix image, the associated bitbake, and the tools would be made open source at the time of publication.
Fig. 2 The figure illustrates the routing logic implemented in the system. It assigns two types of edge weights to road segments: (1) high weights are assigned to edges that overlap with “danger zones”, and (2) low weights are assigned to other edges. The system uses pgRouting (which uses the A* search algorithm) to find the shortest route to the destination. The starting point of the route is chosen as the nearest intersection point from the actual user location. The other option (Point A) is avoided since it requires additional computation to calculate. 

5.2 Backend Databases

The host uses PostgreSQL which is spatially enabled using PostGIS to store vector data. The PostgreSQL database allows for easy storage and manipulation of data. Basic data types can be indexed, and efficiently searched and extracted. With the addition of PostGIS, geographically referenced spatial data types, such as nodes, vectors, and polygons, are added to the mix. These spatial data types can be spatial indexed for efficient search and retrieval. Furthermore, libraries such as pgRouting extend PostGIS data types to include graphs (or networks) and provide simple but powerful graph traversal functions. PostGIS includes many operations that work on these data types. The enabling of the use of spatial data allows questions to be asked such as, “what is the node in the road network that is closest to this location?” Scripts can populate the database with its initial data, and programs written in PHP extract and update the data during the operation of the system.

5.3 Routing logic

The routing logic is illustrated in Figure 2. The location of the user is extracted using HTML 5 (supported by most handheld browsers). If the device is a smartphone with a GPS unit, the web browser will extract the latitude and longitude using the onboard GPS unit. The goal of the routing engine is to find a “safe route” from a user’s present location to a safety zone, such as a relief or food camp. A safe route is defined as a set of directions that avoids any dangerous zone such as a fallen tree, bridges, buildings, and flooded streets. The routing is done against a table that is a list of weighted segments that make up the road network. The segments are weighted by their length. If a segment is in a danger zone it’s weight is multiplied by a very large number (as illustrated in Figure 2). The large number will assure that routes will never pass through the danger zone and that routes that begin in the danger zone will start by taking the shortest route out of the danger zone. (It should be noted that a route may end up going through a danger zone if that is the only route possible.) The routing table is accessed by the routing program, pgRouting. The pgRouting engine uses a network developed from the public Arkansas Centerline File (ACF) [26]. It is a generic program that does not understand PostGIS vector data. As such, the data in the routing table is not spatially enabled. Aside from the weight, the part of the table that pgRouting uses simply is a list of starting and ending node identifiers. So before the route can be queried, the starting and end nodes need to be identified. For this purpose we create a table with all the nodes in the road network and their identifiers. The nodes in this table are spatially enabled and this table can be queried, with postGIS commands, for the road network nodes that are closest to the starting and ending locations. Once those nodes are known, a route can be found using pgRouting’s A* path finding algorithm. This list of nodes is converted into a list of vectors with another query to the database and then returned to the client in GeoJSON format. Routes begin at the nodes closest to the user’s present location. For instance in Figure 2, while Point A (which is the closest perpendicular point near the user’s location) could also be selected, it incurs an additional overhead to calculate the intersection between two lines. For computational ease, we select Point B as the starting point, since the interaction coordinates are stored a priori in
William Johnston, Nilanjan Banerjee, Jackson Cothren, James P. Parkerson

Fig. 3 The figure shows an example rendered map. The red layer shows the dangerous zone and the green line shows the route from the user’s present location to a safe location. The blue polygon and markers shows crowd points inserted by users. Clicking on these points show the user annotated text.

the database. The ACF data used to create the routing graph for this current implementation does not model traffic pattern restrictions such as one-way streets or turn limitations. While pgRouting can model such restrictions in it’s node and edge data types, we chose not to include them in project for two reasons. First, the ACF is not designed for routing and does not natively contain the necessary information. Second, and more importantly, these types of restrictions may not be appropriate in scenarios which arise in a disconnected environment and would cause the system to provide overly conservative routing.

5.4 Networking

The node serving the map needs to act as an access point. There are two challenges that must be addressed for deployed embedded nodes to be useful. First, the node should have sufficient range. Second, the network configuration should be set up such that whenever a user associates with the node, any request to a server on the Internet renders the map. To address the first problem, we have experimented with different external antenna gains. We present a evaluation of the range in §6. To answer the second question, the gumstix computer is setup with an ad-hoc network to allow Wi-Fi enabled computers such as smartphones to connect and access the services on the Gumstix. A DHCP server is set up on the Gumstix to provide connecting computers with an address in the same subnet space as the Gumstix. Currently, the ad-hoc network is set up as 192.169.91.X with the ad-hoc server at 192.168.91.1. When deployed with many other units, each unit will have its own unique set of subnet addresses to serve. A domain name service is also set up so that every request for any computer name always returns the address of the single Gumstix on which the name servers is resident. The reason for this is that the user of the system needs to see the emergency information irrespective of the Internet-resident server the user tries to contact.

5.5 Crowd sourcing and Client-end Software Modules

Data resident at the backend database on the embedded node can have stale data. Since the nodes are disconnected from the Internet and there is no backhaul connection to a GIS server, it is important that some alternate way is used to keep the information on the node database up to date. To this end, we use a crowd approach. Our system allows the user to annotate maps rendered on his handheld device. The users begin by selecting the kind of location information they want to provide and annotate. He has two choices, a point location, or an area location. After making the choice, the user can designate a point by touching the location on the map, or designate an area by drawing a polygon on the map by touching the location of the corners. After the location is designated a panel appears that collects text information about the location. After the text is entered, the client program takes the location information and bundles it with the text information and sends it to a cgi-bin app on the host which in turn inserts the information into the spatially enabled database. For simplicity, the database has one table for point information and another table for area information. With the crowd source data now loaded into the database, any other user of the system will see the new information on their maps.

5.6 An Example Rendered Map

We have implemented the system on the Gumstix Overo Air. The Air is an embedded platform with a 600 MHz processor, 128 MB RAM, an inbuilt 802.11b/g Wi-Fi radio, and an external 8GB SD card. The prototype is powered by a 1000 mAh Li-ion battery and is illustrated in Figure 4. The software modules for Mapstack such as Apache 2, Mapserver, the DHCP server, Wi-Fi module in ad-hoc mode, and the backend databases, are invoked during the bootup process of embedded Linux. A client such as a smartphone or laptop associates with the Wi-Fi module on the Gumstix Overo and obtains
an IP address. The maps can be rendered on the client device by trying to download any webpage on the web browser. Figure 3 shows an example map rendered on a desktop. The red area represents a danger zone. The green circle represents the user’s current location. The black and white shelter icon represents a safety location. The green line from the user’s current location to the safety location represents the shortest route that avoids the danger zone. Blue areas and blue markers are crowd sourced information.

6 Evaluation

We evaluate the system while focusing on the following questions.

- How much energy does rendering maps consume?
- What is the latency of disseminating map layers?
- How usable are the maps rendered on mobile devices?

While answering these questions, we present several performance micro-benchmarks of the system. Additionally, we present trade-offs between resolution of maps rendered and energy consumption of the device.

6.1 Experimental Setup

The experiments were conducted in a laboratory setting using a 70MHz 1 GS/s Tektronix oscilloscope. The setup is illustrated in Figure 5. A 1.008Ω high tolerance sense resistor was placed in series with the power supply and the Gumstix module. The voltage drop across the sense resistor is used to measure the current drawn by the system. We set up several experiments where different components (such as map layers) were incrementally rendered. For each experiment, power traces were collected using the Oscilloscope. For accurate timing and power measurements, the software script was used to control an onboard general purpose Input/Output pin on the Gumstix. The pin was set high at times when our experiment started. The voltage level of the pin acts as an external trigger for the oscilloscope to take a measurement. All measurements presented in the section are an average of 128 independent trials. To automate the data collection, a script on a client device (a desktop) requested maps from the server (Gumstix) once every 30 seconds. The latency measurements were derived from the power traces.
6.2 Energy Consumption

Our first set of experiments focussed on measuring the energy consumption of the Gumstix while rendering map layers to a client device. Figure 6 illustrates a collected power trace when the Gumstix was actively rendering maps to a client device. The figure shows that the total time to successfully disseminate a high resolution map on a desktop browser is 14.5 seconds. This includes the time to fetch the base layers, the augmented layers, routing layers, and transfer the data over a Wi-Fi connection. The time does not include the overhead of associating with the Gumstix node. Rendering maps incurs an additional average power consumption of 200 mW over the power consumption of an idle Gumstix system with its Wi-Fi radio switched on and the CPU operating at 600 MHz.

We next diagnose the energy bottleneck during map dissemination. To this end, we performed a suite of experiments where map layers were incrementally rendered to the client. This includes the base layer, the danger zones, the crowdsourced points, crowdsourced polygons (areas), extracting routes, and safety zones. Figure 7 shows the total energy consumed by the system when selective layers were rendered. Disseminating the routing layer consumes the maximum amount of energy (84% of the total energy) followed by the base layer (10% of the total energy). These measurements illustrate the efficacy of the multi-scale map dissemination. For instance, if the residual battery capacity is low, the system can adaptively choose to not render the base layer or the routing layer. The trade-off lies in the usability of the system. Without the routing layer, users would have to route themselves around dangerous locations manually. Such a system is still useful since the maps show the present location of the user with the base layer, dangerous areas, and safety locations. Removing the base layer, however, would render a map without the base image tiles, but with annotations from other layers.

6.3 Dissemination Latency

The goal of our next set of experiments was to measure the dissemination latency of rendering maps. Through controlled experiments, we broke down the latency of disseminating a high resolution map (14.5 seconds). Figure 8 shows the time taken to render different layers selectively. The routing layer takes 12.5 seconds and accounts for 86% of the total time, followed by the base layer. Again adaptively choosing to render map layers as a function of the residual battery capacity and user mobility can speed up the process considerably. In addition to rendering maps, another potential source of overhead is the latency of associating with the Gumstix module over Wi-Fi. It takes 2.7 seconds to associate with a handheld device, such as a smartphone, and download the Javascript app, which is a small fraction of the overall dissemination latency.

We have performed additional experiments at a lower CPU frequency. While the time to disseminate the tiles increases, and the power consumption decreases proportionally, the overall energy consumption is similar. We plan to evaluate the utility of dynamic frequency
and voltage scaling as part of the multi-resolution map rendering as future work.

Result Summary: If used continuously, the map stack increases the energy consumption of the Gumstix (over a idle mode) by 11% (a equivalent reduction in lifetime). With all features rendered, the average latency of dissemination is 14.5 seconds, which can support static users but would not be useful for highly mobile users. However, by intelligently deciding not to disseminate the routing layer, the latency can be reduced by 86% (to approximately 2 seconds). Such a latency can support highly mobile users (e.g. users in vehicles).

6.4 Usability

We next evaluate the user interface for the maps rendered by Javascript app on popular handheld devices (the iPAD, and the iPhone). Since the Javascript app does not require custom software to be installed on handhelds, it will be functional on any device supporting a HTML 5.0 compatible web browser. While we have not performed a usability study on the UI design, we have taken care to ensure that the map features are built akin to popular map engines such as Google maps or Bing maps. Hence, users would be already trained to use the map features on their handhelds after a natural disaster. Figure 9 shows the maps on the iPAD and the iPhone. The maps provide zooming and panning features. The zoom in and zoom out features use multi-touch on the phone’s user interface. When the user zooms into a map, additional map tiles are adaptively rendered from the Gumstix server. The pan feature uses buttons on the side and top of the maps rendered. The UI also provides controls to annotate the maps with crowd sourced data points and polygons.

6.5 Microbenchmarks

For completeness, we performed a set of experiments to determine two micro-benchmarks: (1) node range: the range of the system is crucial for the usefulness of the node. Since these nodes will be deployed in the wild, it is important that the system covers a large geographic area. To this end, we performed experiments with different antenna gains. Table 1 shows the range of the node as a function of the antenna gain. The system does not consume any additional energy at higher antenna gains rather the energy used is more efficiently converted into a signal. With a gain of 9dbi, the gumstix node can cover approximately 200m, which is comparable to several outdoor Wi-Fi mesh deployments. (2) storage overhead: the storage footprint of the backend database should be small for the system to be practical on low power embedded nodes. The map tile database consumes 66MB while the PostGreSQL database consumes only 314MB to cover one county. Clearly, the storage footprint for the system is low, and hence is applicable across a range of embedded devices. Additionally, we use a pluggable SD card to load the root image with the database and the Linux kernel making the system plug and play and highly portable.

<table>
<thead>
<tr>
<th>Antenna</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>2dbi</td>
<td>110 m</td>
</tr>
<tr>
<td>5dbi</td>
<td>148 m</td>
</tr>
<tr>
<td>7dbi</td>
<td>165 m</td>
</tr>
<tr>
<td>9dbi</td>
<td>199 m</td>
</tr>
</tbody>
</table>

Table 1 The table shows the range of the device as a function of the antenna gain.

7 Future Work

There are several avenues of future work that we are pursuing to make the system practical and deployable in a natural disaster scenario. Here, we summarize some of our future directions.

- Node Design: We are designing a high customized node based on Hierarchical Power Management [32] to act as a server for the map stack. It consists of a low power micro-controller device, a high power Gumstix platform, and a custom designed power supply board. The system can be powered by portable solar panels and would be weather proof for deployment in an emergency scenario. The use of the micro-controller provides high availability at low energy consumption, while the Gumstix can be switched
on on-demand to render maps. The system will mitigate the high idle power consumption of the platform, as illustrated in Figure 6. The system uses customized profiling, hardware systems, and wakeup controller algorithms for energy efficiency.

- **Mesh network:** We are designing a network of embedded nodes that are meshed together over a low power radio, such as Zigbee. The network of nodes can be used to disseminate map data to users in a large geographic area. Moreover, crowdsourced data on one embedded node can be disseminated to other nodes, that survivors can access. Additionally, the mesh provides fault tolerance; if a node dies due to hardware failures or energy constraints, the system can synchronize data across multiple mesh resident devices.

- **Map stack:** We are working on optimizations for our software stack. For instance, the current map network is made up of *segments only* rather than polylines. That is, there are many nodes with only two edges attached. To optimize our design, the segments can be merged into one edge that is composed of a polyline. This will reduce the number of edges in the network and speed up routing.

8 Conclusion

In this paper, we present an extensible software stack for rendering maps (a map stack) on popular handheld devices from low power embedded nodes. The map stack can be used for disseminating map data to users during the aftermath of a natural disaster. It is tuned to have a low energy footprint on the device. It uses multi-scale map rendering, efficient routing, and delegation of computation to client devices to reduce energy consumption. Additionally, it implements a crowdsourced approach that allows users to annotate maps with useful information. In a disconnected environment such an approach can help keep the maps on the embedded nodes up-to-date. We have implemented a fully functional map stack on the Gumstix Overo platform and exhaustively evaluated its energy consumption, latency of map dissemination, and user interface. Our evaluation shows that map stack is an extensible low overhead plug-n-play system for map rendering from low power embedded devices.

9 Acknowledgement

This work is supported in part by NSF awards CNS-1018112, CNS-1055061 CNS-1115728, and CNS-1158759. Any opinions, ndings, and conclusions expressed in this work are those of the authors and do not necessarily reect the views of the NSF.

References


