Webbee: A Secure Coordination and Communication Application Infrastructure for Handheld Environments in Crisis Scenarios

The Webbee Team, University of Michigan*
{webbee, jamin}@umich.edu

April 28, 2008

Abstract

The Webbee research project's mission is to provide a secure coordination and communication infrastructure to a team of first responders. Our architecture consists of three basic elements: an instant infrastructure, which restores connectivity; the Webbee Coordination Server, comprised of application daemons that provide communication and coordination services on top a secure, mobile handheld-friendly environment; and a Database Server, which serves all data necessary for interactions, and which is enriched with triggers to automatically take action when certain new data is supplied to the database. We have successfully deployed several sample applications using this architecture - such as Gas Prices, Event Reports, and AC^2 - so as to illustrate the architecture's viability, flexibility, and security, especially in disaster scenario environments.

Keywords: Webbee, disaster relief, instant infrastructure, GPS, database triggers, challenge response, quorum

1 Introduction

Ever since the September 11, 2001 terrorist attacks, the United States government has been reevaluating the coordination mechanisms of first responders in disaster scenarios. First responders need to communicate quickly, reliably and securely in times of crisis. However, conventional communication channels such as cell phone networks may not be appropriate during disaster scenarios. First, the disaster may have impaired or destroyed the channels, rendering communication unreliable or impossible. Even if communication via conventional channel were technically feasible, in times of stress people may flood the channel (e.g., to call their loved ones), causing it to become so congested that first responders cannot coordinate as timely or as effectively as they would be able to otherwise. Furthermore, there may be situations in which malicious agents have compromised the network - for example, by stealing a first responder's cell phone - so as to intercept communications and/or transmit deceptive messages. This can seriously undermine first responders' effectiveness - and in the worst case, cause them to unwittingly and severely exacerbate an already bad situation.

*__Primary Investigators:__ Sugih Jamin, Zhuoqing Mao, T. V. Lakshman, Sarit Mukherjee, Jignesh Patel, Limin Wang. __Students, past and present:__ Brendan Blanco, Hyunseok Chang, Yun Jason Chen, Sren Dreijer, Matt England, Joe Flint, Alex Garcia, Dan Harris, Todd Hopfinger, Dan Konson, Neil Panky, Jeff Powers, Bob Sprentall, Patrick Turley, John Umbaugh, Krian Upatkoon, Wenjie Wang, Zhiheng Wang, Byung Suk Yang

Figure 1: A COTS Linksys 802.11 router

We see this problem as being threefold. Firstly, responders need a way to communicate using devices that they likely already have, and to which they are well-accustomed. Secondly, the communications channel must be secure, and this security must take on new dimensions so as to address, for example, the possibility of cell phone theft in a hostile environment, or serious scalability constraints on a handheld device interacting with the security. Finally, while in a time of crisis, the consumer communication infrastructure can sometimes be used, it cannot be relied upon solely. We have chosen to provide secure information exchange over commercial wireless networks to off-the-shelf handheld devices such as smart phones and PDAs, which have achieved massive consumer penetration. Our Webbee Coordination Server, explained below, addresses the special security challenges first responders face, such as a lost or stolen device which might result in network compromise. Finally, when commercial networks are out of commission and connectivity must be restored within the affected zone, we employ an instant infrastructure using common battery-powered commercial off-the-shelf (COTS) 802.11 b/g routers with enhanced software (Figure 1).

This paper consists of five sections. We begin in Section 2 with an overview of the entire Webbee architecture, where we introduce some basic terms and concepts. Section 3 describes the Webbee Coordination Server and its components, including several useful sample applications that we have developed over the Webbee Coordination System. Section 4 quantifies the overhead that each piece of the Webbee architecture creates. The final sections discuss conclusions, related work and future directions.

2 Architecture

There are three major components of the Webbee architecture: the *instant infrastructure*, the *Webbee Coordination Server*, and the *database server*. The instant infrastructure consists of several routers, endowed with custom software designed for mesh networks. Each router is powered by a 12 volt nonspillable battery. The Webbee Coordination Server acts as the communication mechanism amongst Webbee users. The database server is our primary data store, and it has been extended

with triggers to create a highly-reactive environment. Now we further motivate and explore each of these pieces.

2.1 Instant Infrastructure

There are many disaster scenarios such as major fires, tornadoes, and terrorist attacks that can cripple areas in terms of data connectivity. Therefore, the first priority in establishing communication is to restore connectivity. We have designed the instant infrastructure units to fit inside the backpacks of military and first responder personnel, so that it can be easily deployed over wide areas. Each backpack unit consists of a router connected to a battery. The router's power cable is modified to attach to the batteries, and the unit can operate for over 24 hours without battery recharge.

We have tailored and deployed custom router software called SMESH developed at Johns Hopkins University [1] to coordinate the network of routers. Within the SMESH software, one or more of the routers is designated as a network gateway, and will serve out a backhauled internet connection. The remainder (and majority) of the routers connect themselves together in such a way that each has a path to a gateway. The routers interconnect in a time-dynamic fashion, and as infrastructure personnel move about the field, routers constantly associate and dissociate with each other to provide the maximum amount of connectivity possible. The gateway routers do not allow any connections by individual clients - only to other mesh routers. Increasing the area of connectivity is as simple as adding personnel with mesh routers. In principle, the area can be extended indefinitely as long as the mesh routers in the backpack units are each within range of their neighbors.

Because routers and batteries are commercial off-the-shelf products, our instant infrastructure can be deployed quickly and at a very low cost: components are easily accessible and replaceable. The low cost allows for the possibility of outfitting a very large number soldiers and first responders with this technology quickly and cheaply.

Our team has written a mesh network visualization tool to graphically display the state of the mesh network at all times. This light-weight application can allow any user on the network the ability to see the current status of each router and client on the network. The tool can be used to diagnose problems in the network, for example determining which routers have been disconnected, as well as the percentage of bandwidth each client is getting from each router. In Figure 2, we see that routers 10.0.11.22 is associating with both 10.0.11.21 and 10.0.11.23, but that 10.0.11.21 and 10.0.11.23 are not associating directly with each other (hence the thick grayed-out[1] line between the two) - perhaps because they are too far apart. Furthermore, we can determine which clients (e.g., smart phones or laptops) are associating with which routers with this application - and with how much signal strength. For example, clients 10.144.228.213 and 10.99.165.189 are associating with router 10.0.11.22, both with signal strengths of 100%. Client 10.144.228.213 is also associating with 10.0.11.21 with 100% signal strength, but client 10.99.165.189, on the other hand, associates with 10.0.11.21 with only 93% signal strength (perhaps because it is far away from that router).

[1]Within the application itself, dissociative links between routers are actually represented in red. We have converted our application screenshot to grayscale so as to be more black-and-white printer-friendly.

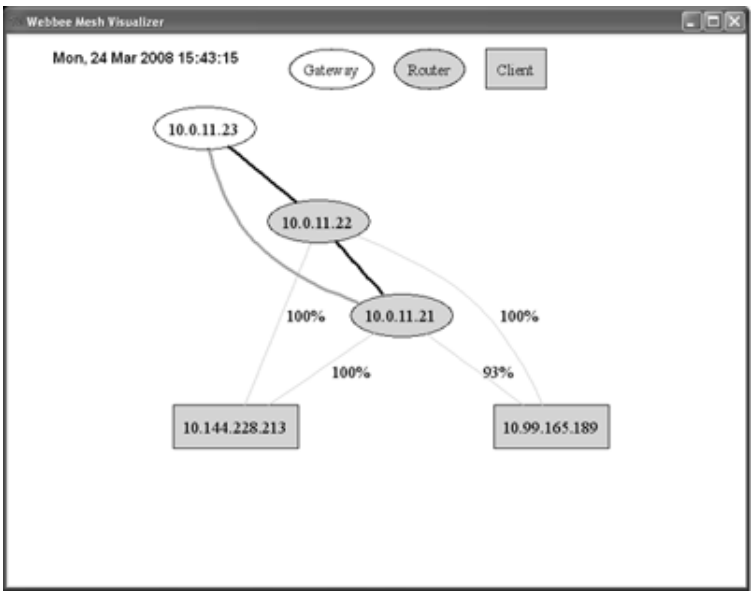


Figure 2: A screenshot from the Webbee Visualization application

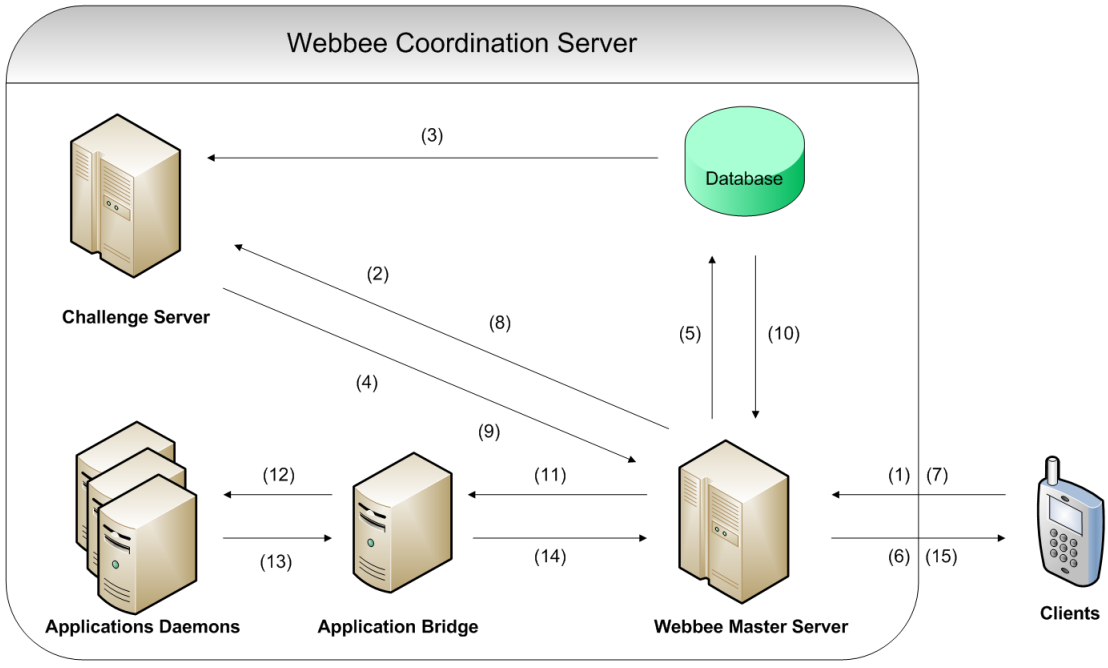


Figure 3: The Webbee Coordination Server

2.2 Webbee Coordination Server

The Webbee project employs a client-server architecture and protocol to implement communications. The clients in this case are smart phones or PDAs, and the server role is occupied by the Webbee Coordination Server. The Webbee Coordination Server is actually an abstraction of many internal components, each of which will be discussed in detail later in this paper. Before that, though, let us provide a high-level illustration of each of the components and their mutual interactions.

The first component is the Webbee Master Server. This server accepts client connections and behaves as an interface between clients and the underlying applications, so that regardless of the application, a user only needs to know the address of the Master Server. Here is a sample use case scenario (see Figure 3): when a client sends a request to the Master Server *(1)*, the Master Server in turn sends the message along to the Challenge Server *(2)*. If the Challenge Server decides that the user needs to be issued a challenge - by examining its policies and looking in the database *(3)* - it notifies the Master Server *(4)*, which in turn stores the request into the database *(5)* and sends a challenge message back to the client *(6)* (if no challenge is needed, the Webbee Master Server forwards the request directly to the application bridge without storing it - skip to #11 in this case). The client then attempts to solve the challenge, sending a candidate solution as a request back to the Webbee Master Server *(7)*, which sends it along to the Challenge Server *(8)*. If the Challenge Server determines that the candidate solution is valid for the challenge *(9)*, the Master Server retrieves the original request (i.e., the client's most recent outstanding request) from the database *(10)* and forwards it to the Application Bridge *(11)*, which reads the packet header so as to dispatch the message to one of the application daemons *(12)*. The individual application daemon consumes the message payload and acts on it accordingly. If the daemon generates a response to the message, it sends it back to the Application Bridge *(13)*, which sends it along to the Webbee Master Server *(14)*, which in turn forwards it along to the appropriate (originating) client *(15)*.

The Challenge Server authenticates users based on a challenge-response system. The Challenge Server uses a *policy* system (GPS variation, temporal invalidation, etc.) to decide whether the user should be challenged. The Challenge Server and its security mechanisms will be explained in detail in section 3.

In order to demonstrate the viability of this kind of system, we needed to develop sample applications. We will look closely at three applications we have developed for the Webbee Coordination System: Gas Prices (web scraping for local gas prices), Event Reports (position based reporting), and AC^2 (voice, text and image messaging). These daemons interact with the database server to complete the task that the client has requested, and all are explained in detail below.

2.3 Trigger-Enhanced Database Server

All communications over the Webbee architecture are logged and tagged with GPS coordinates so that they can be mapped for a more logical view. Our architecture utilizes a Postgres database to serve communication data and facilitate interoperability within the Webbee Coordination Server.

We have specifically exploited the triggering features within the database for our *Event Reports* application, which will be discussed in detail in section 3 of this paper. Voice, text and image messages are stored in this database, as well as IP addresses and port numbers for different application daemons within the Webbee Coordination Server, so that individual daemons can be easily moved to different machines. We also keep a record of the GPS coordinates a user has visited, along with a

timestamp, so as to be able to recreate the path s/he traverses during a mission. The *Event Reports* application takes advantage of this functionality by plotting the path of the user and displaying it on the phone, in real time. Furthermore, when the *Event Reports* server daemon receives GPS coordinates from the user, the coordinates are compared to existing reports' coordinates and radii in the database. If the user's coordinates are within the circle defined by a report's coordinates and radius - that is to say, if the user has gotten close enough to the site where the event was reported - then an SMS message is sent to the user via a database triggering mechanism. The *Event Reports* application will be covered in greater detail in section 3.5.

We plan on capitalizing on the path-inspecting functionality in the development of AC^2's *Command and Control application*, too: this application will allow the paths of all field agents to be inspected for any interval of time supplied by the *Command and Control* user. Furthermore, this path-recording feature could be exploited by the challenge system in defining more sophisticated location-based policies.

3 Webbee Coordination Server Component Detail

In this section, we will take a closer look at the challenge system, the security mechanisms, and some of the applications we have deployed on the Webbee system.

3.1 Webbee Master Server and Challenge Server Interaction

As mentioned above, the Webbee Master Server is the point of interface between the Webbee Coordination Server and a client. The Webbee Master Server's main task is to negotiate incoming traffic from the client between the Challenge Server and the Application Bridge. Specifically, when a request comes in from a client, the Webbee Master Server stores the request in the database, along with meta information identifying the client, the GPS coordinates of the client, the timestamp, etc. The Webbee Master Server then queries the Challenge Server about whether the client needs to be challenged. If the set of policies in effect on the Challenge Server determine that no challenge needs to be issued, it reports back to the Webbee Master server that it is OK to proceed. Otherwise, the Challenge Server creates and issues a challenge back to the Master Server, which in turn is sent back to the client. The client then is prompted to solve the challenge. The client's solution is sent back through the Master Server to the Challenge Server. If it is not a valid solution, the Challenge Server informs the Master Server that no action is to be taken, and that the client is to be informed that his or her request was denied. Currently, the client is prompted to try again. If the solution is valid for the challenge, the Webbee Master Server retrieves the most recent request from the Database and dispatches it to the Application Bridge. Our model, therefore, makes the assumption that clients will only ever need a single request serviced at a time.

3.2 Security

Our security mechanism is broken into three separate subsystems: the Challenge Server, which is the most innovative piece of our security system; upload security, which implements *forward secure signatures*; and download security, which implements a *quorum* system. All three mechanisms are wrapped in an SSL layer. Each is described in detail below.

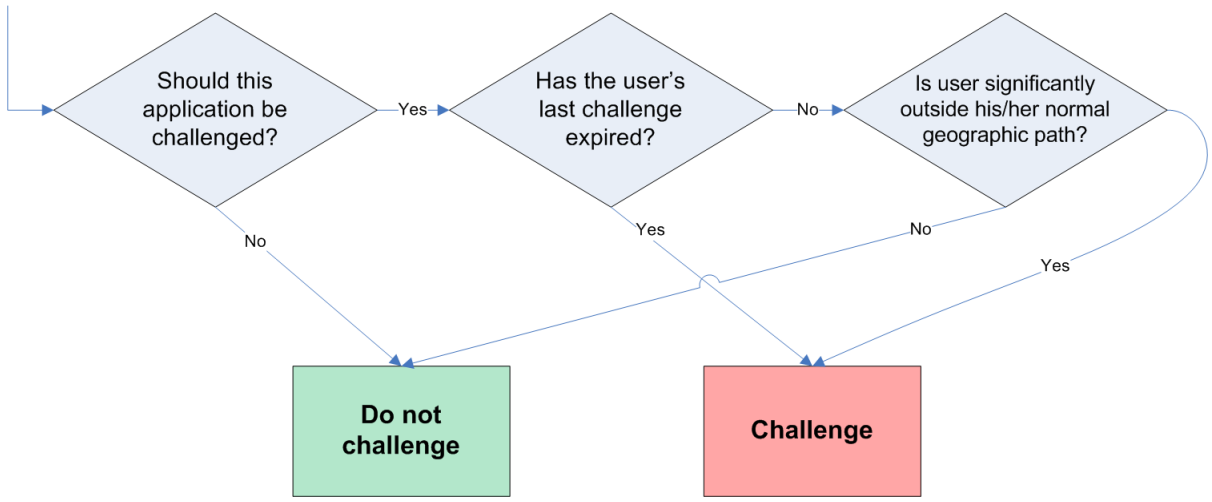


Figure 4: Policy Flowchart for The Webbee Coordination Server

3.2.1 The Challenge Server

The Challenge Server's job is essentially to protect against stolen handheld devices. The Challenge Server is made up of a set of policies and a set of čhallenges. The policies encode specific conditions in which agents need to be challenged. Policies are arranged in a hierarchy, in the sense that, if an agent passes one policy without having to be challenged, there might still be subsequent policies against which s/he must be evaluated. The policy scheme that we have developed for the Webbee Coordination Server is depicted by flowchart in Figure 4.

The first policy to which clients are subjected is an *application-level* test - i.e., *"Should this application be challenged?"* This is a special policy that grants all clients full access to a particular application without having to issue any challenges. Applications that do not need to be secure are particularly suited to traverse the "no" arrow for this policy, and we have included this special policy to demonstrate that, in fact, both secure and non-secure applications can be administered through the Webbee Coordination Server. *Gas Prices*, explained in detail below, is an example of one such application in which security is not critical.

If the application is in fact one that we have specified as requiring challenges (i.e., after following the "yes" arrow from the first policy), a temporal policy is activated, in which the question *"Has this user's last challenge expired?"* is asked. If the current time minus the timestamp of the client's last solved challenge is greater than a specific interval - defined by an administrator - the client is challenged. Otherwise, the system checks the next policy. The policy interval can be defined on a per user basis; for example, an expiration of ten minutes might be appropriate for one client, whereas an expiration of thirty minutes might be appropriate for another, possibly less security-sensitive, client.

The last policy we have implemented in our system is a *geospatial* policy. Simply stated, if the user has strayed far away from the set of last known GPS coordinates, the client is issued a challenge. This helps to safeguard against handheld device theft; if a handheld is stolen and taken away to a malicious infiltrator's base, then even if the temporal policy hasn't been activated, the geospatial policy will come into effect.

Note that only one challenge will ever occur through a traversal through this policy flowchart.

That is to say, for instance, if the temporal policy becomes active, only one challenge will be issued, even if the client's GPS coordinates are out of range for the geospatial policy. Once the client solves the challenge issued by the temporal policy, his or her GPS coordinates are stored as the "most recent" coordinates on the Challenge Server.

It is a fairly simple matter to snap new policies into the flowchart. Though we have only incorporated application-level, temporal and geospatial policies, one might imagine adding a policy that looked at the number of failed solutions to challenges an agent has produced over time, or whether s/he is on a blacklist.

When the policy flowchart determines that a client needs to be challenged, it currently randomly selects one of several possible challenges and issues it to the client. If the client solves the challenge, then the request proceeds as normal, and expirations for the policies are reset so as to ensure the client will not be challenged until s/he deviates significantly from the current state (temporally, geospatially, or otherwise, depending on the policies in effect). If, on the other hand, the client is unable to solve the challenge, then the request is denied. All future requests that require security will also be challenged until the client successfully solves that challenge. Selection from a random set of challenges increases the probability that, even if a device is compromised and the malicious agent knows the solution to one or several of the challenges, eventually s/he will be issued a challenge that s/he cannot easily solve. That is to say, when the client fails a challenge, subsequent requests by the client to the Webbee Coordination Server will cause it to issue the *same* challenge; this mitigates potential damage from malicious clients trying to game the system by exploring the challenge space if they know the correct responses to a subset of the challenges. Were this mechanism not in place, for example, all the malicious agent would need to do is to wait until a favorable challenge - i.e., one to which s/he knew s/he could solve - is issued.

Currently, only text-based challenges have been implemented - for example, the system might challenge the client to respond with a password, or to answer a question that, with high probability, only the legitimate client is likely to know. In principle, and with the right hardware, the challenge system could be extended to issue other kinds of challenges, such as biometric challenges, including fingerprint, voice, and/or retinal scanning.

3.2.2 Upload Security

In our scalable crisis management system, we are operating under the assumption that there are many downloads, but relatively few uploads. For each broadcast message, for example, there will be a single sender but many recipients. We have decomposed our security requirements, therefore, into upload security characteristics and download security characteristics so as to inform our approach. Let us first examine our upload security mechanism.

In terms of upload security, if a handheld is lost, we want to ensure that 1) data that has already been posted cannot be repudiated, and 2) data cannot be post-dated. Our forward secure signatures use a private key that evolves as a function of time; the public key, however, remains the same. This kind of forward secure scheme was proposed by Anderson [2] and implemented by Bellare and Miner [3].

The *forward secure signature* system allows for the repudiation of current and future signatures once a breach is detected, but in such a way that past signatures are still valid. The mechanism by which the forward secure system operates is dependent on the evolution of the signing keys: when a key used to sign a message during time period i is evolved (i.e., evolves into the key for time period $i + 1$), signatures for time period i may no longer be instantiated - neither by legitimate

users nor by attackers. Once an attacker exposes a forward secure mechanism, then only the future signatures become suspect.

We use *server-assisted* signatures to (logically) centralize handheld trust revocation. The server-assisted aspect of the system employs a separate server (the *signature* server) to help create valid signatures. Key material is split into two pieces - one for the device, and one for the server - and requires cooperative computation to create the signatures. The signature mechanism is broken down into two steps: first, the client generates a private key and splits it into two pieces A and B. One piece - A - is kept in the clear. B, however, is encrypted using the server's public key so as to form the *client ticket*. The clear copy of B is discarded after the client ticket has been created. When the client sends data, it signs the message hash beforehand with its clear half (A). When the server decrypts the ticket, it creates its own partial signature and sends it back to the client. The client then combines the two partial signatures into a complete signature. In this way, the signature server is untrusted, in that even if the server becomes compromised, it can glean nothing that it could use to forge user signatures.

In the event that a user's device becomes compromised, there is a means by which the user can disable the key. Each client ticket created contains a *ticket number*. The number is stored in backup storage off of the client. The device's owner can disable the device by retrieving the ticket number from backup storage and adding it to the signature server's blacklist. Subsequently, whenever that client ticket is presented to the signature server, the ticket will be refused since its ticket number is blacklisted. Further details about the upload security can be found in [4].

3.2.3 Download Security: The Quorum System

In designing our download security mechanism, it is important that the system scale to a large number of users. To this end, if a member leaves the group, we want to have only a relatively small number of users to have to acquire new keys. The *Quorum* system, described below, implements download security with these kinds of scalability concerns taken into account.

Quorum is a download security mechanism in which agents need to have a minimum number, k, of *keyshares* to securely read a message. At initialization (e.g., the beginning of an agent's shift), each agent receives m key shares, where $m > k$, from a global key-share set consisting of a total of s keyshares. If a user leaves, his or her shares are invalidated for all users. When a user has fewer than k valid shares, s/he must reinitialize - i.e., obtain a new set of valid key shares from the global key share collection. This security mechanism is similar to a threshold cryptosystem, in which k users must cooperate in order to operate.

The system works as follows: when the server is about to broadcast a message, it first encrypts the message under a *message key*. This message key, in turn, is itself encrypted s times - once for each global key - to form a set of s encrypted message keys. The s encrypted message keys and the encrypted message are all sent to each of the agents. The agents decrypt the encrypted message keys using their personal keysets and make sure that exactly k of the decrypted message keys are identical. Otherwise, one or more of their keyshares is invalid. If k of the decrypted message keys are in fact identical, the agent proceeds to decrypt the encrypted message with that decrypted message key.

Figures 6(a) through 6(c) depict a possible usage scenario for which $k = 3$ and $m = 5$. In Figure 6(a), the participants - Amanda, Bob, and Carl - all have a quorum of valid keyshares. In Figure 6(b), when Bob leaves, three of Amanda's keyshares are invalidated, forcing her to obtain new shares. Carl, on the other hand, only has two shares invalidated; he can continue to operate.

Figure 6(c) depicts the scenario in which Amanda has reported a lost or stolen handheld, in which case all of Amanda's keyshares are invalidated. In this instance, Carl must reacquire new keyshares to operate.

3.3 Application Bridge

When a request comes in from the Webbee Master Server, the Application Bridge dispatches it to the appropriate Application Daemon. Specifically, there is an Application Daemon ID embedded in the request header on which the Application Bridge dispatches. If the request to the Application Daemon generates a response, the Application Bridge sends it back through the Webbee Master server, which in turn forwards it to the appropriate client. Many applications in theory can be deployed simultaneously. In fact, the Webbee Coordination Server can dispatch across up to 2^{32} unique application IDs. If the client generates a request containing an application ID for which no application currently exists, the application bridge drops the request and sends an error response back to the client informing him or her of the error.

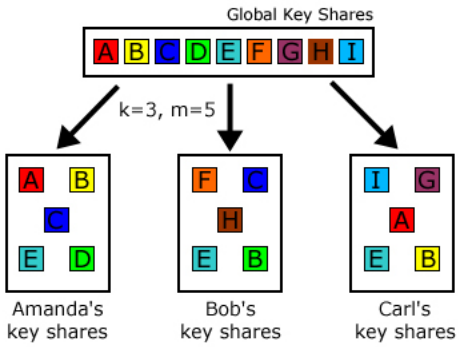
We will now examine three applications we have created and deployed on our Webbee Coordination Server: *Gas Prices*, *Event Reports*, and AC^2.

3.4 Gas Prices

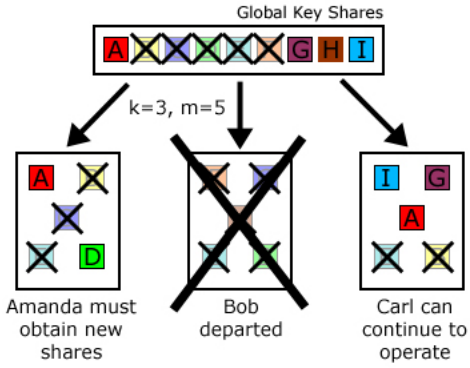
Gas Prices is one of the applications that we have built on top of the Webbee infrastructure. The *Gas Prices* application allows clients to determine which gas stations nearby have the cheapest prices, and how to get there. Client responses essentially consist of the unit's GPS coordinates. First, the *Gas Prices* daemon constructs a map image centered about the client's GPS coordinates through an implementation of the U.S. Census Bureau's TIGER GIS database [5]. The application then queries a public site on the web that publishes up-to-date information about gas prices based on location; it then annotates the map with an overlay of the stations and their prices, and sends it back to the client. The client views the map on the handheld device's display, and may scroll through the map with the device's arrow keys. This will cause a new request with modified GPS coordinates to be issued to the Webbee Coordination Server.

Though there probably are not many instances in which first responders might need to use this kind of application, *Gas Prices* is illustrative in how a daemon can acquire, assemble, and recontextualize data from disparate sources in a way that is immediately useful to the client. We have also developed a web-scraping engine which *Gas Prices* and other applications use. Essentially, an administrator writes a text configuration file called a *scraping script* (Figure 6) that identifies which components of a website the Webbee daemon has an interest. From then on, when the scraping engine executes the scrape, it encapsulates the obtained data into an XML document that is easily parsable and queryable. Client daemons can handily use this mechanism to query websites (including the affixation of POST and GET variables) to dynamically obtain the desired information. The daemon can then recontextualize this data, if necessary (for example, by calculating summary statistics on the acquired data), and send it back to the user. In principle this scraping engine could be used, for instance, to acquire information about weather conditions, maps, stock prices, news stories, book reviews, telephone directories, or any sort of dynamic, data-driven published data - including text, images, and audio.

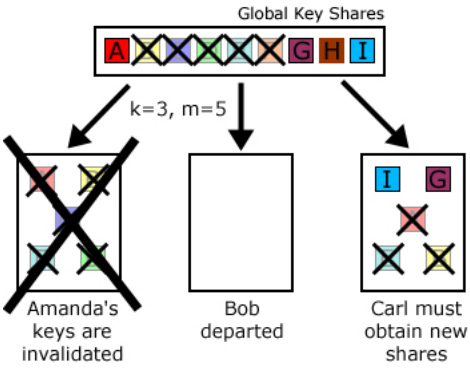
Though the scraping engine is a useful and integral component for applications like Gas Prices, it is brittle in the sense that if the target website's page structure has changed - i.e., the HTML



(a) Amanda, Bob, and Carl initially all have valid keyshares



(b) Bob leaves



(c) Amanda reports lost or stolen handheld

Figure 5: The quorum security system

```
printraw <ver> 0.32</ver>
print
fetch post http://directory.umich.edu/ldapweb-bin/search

if fail
  print Error reading web page.
  print
  goto end
endif
# Found an exact match
searchre fwd end > \ s*1 exact match
if fail
  searchre fwd end > \ s*1 approximate match
  if fail
    goto not_found
  endif
endif
printraw <found>
print
seek start 0
# Name search fwd end <dt><b>Also Known As</b>:</dt>
searchre fwd end <dd> \ s*
select start
searchre fwd start \ s*<br
select end
if pass
  printraw <item name="name">
  select print
  printraw </item>
  print
endif
# Title
seek start 0
search fwd end <dt><b>Title</b>:</dt>
searchre fwd end <dd> \ s*
select start
searchre fwd start \ s*<br
select end
if pass
  printraw <item name="title">
  select cleanprint
  printraw </item>
  print
endif
```

Figure 6: Scraping script fragment that obtains Michigan directory information

tag organization - the scrape breaks. This is because websites are often constructed structurally rather than semantically; if the owner of a website decides to change the layout of his or her page, for example, the script does not know where to look to obtain the desired information. To work properly, the scraping script will have to be reconfigured.

Formerly, the creation of Webbee scraping scripts was, in a sense, unnecessarily unwieldy. Scraping scripts are text files, and for a user to create effective scripts, s/he must have a pretty commanding grasp of the script commands, not to mention at least some passing knowledge of techniques like regular expressions. Thus, for non-engineers, creation of a scraping script can be daunting. Fortunately, there are commercial *graphical* web-scraping front ends - such as *iMacros* [6] - that we have demonstrated can easily be inserted into the Webbee script-creation pipeline. The *iMacros* application is a free plugin for Firefox or Internet Explorer in which the user can record a series of actions representing web page interactions - including clicking on hyperlinks or the submission of form data. Once your browser is pointing to the right place, *iMacros* can then perform the *extraction* of data from web pages, including text, table, audio, and image data.

Luckily, the macros generated by *iMacros* are themselves simple text files. We have demonstrated that it is fairly easy to translate an *iMacros* macro file into a scraping script consumable by Webbee. In principle, one could write a fairly universal *iMacros*-to-Webbee script translator without very much trouble.

3.5 Event Reports

Another applicaiton we have built on top of the Webbee infrastructure is the *Event Reports* application. The *Event Reports* application allows clients to log useful and mission-critical incidents they observe while out in the field. Other clients are notified about these incidents only once they become geospatially relevant. Clients can specify details about an incident by typing out a short message on the handheld device. In addition, they can specify a range of applicability, represented as a radius in meters, for the event. The event explanation text, the range, and the GPS coordinates of the client at the time the event was reported are all stored in the database.

As other clients move into the range of a reported event, their handheld devices are notified. For example, if a client reports the message "Live IEDs (improvised explosive devices) detected" at GPS coordinates <N42°18.606, W083°42.0> for a radius of 100 meters, then the moment any other client enters the defined circle (i.e., his handset transmits GPS coordinates within the defined range), his or her handset is sent the notification via SMS, "Live IEDs detected." This allows the client to receive updates about his or her environment that are immediately important to the client. A client is thus relieved of the burden of manually sifting through all reports to determine which parts are immediately important and actionable, and which are not, thus enabling him or her to react faster and more effectively.

A more comprehensive example is given below (Figure 7). A report about a fire is submitted at the Chicago Mercantile Exchange, at GPS coordinates 41°52'53"N, 87°38'13"W *(A)*. One Chicago Fire Department unit *(B)* and two Chicago Police Department units *(C)* and *(F)* receive the alert about the fire. Another report about an unrelated incident is submitted by an informant across the city *(D)*. Here, one Fire Department unit is alerted *(E)*, as is one Police Department unit *(F)*. Notice that *(F)* receives alerts about both incidents, since it is in range of both. Also notice that the Police Department unit *(G)* receives *no* alerts, since it is not in range of either of the incidents.

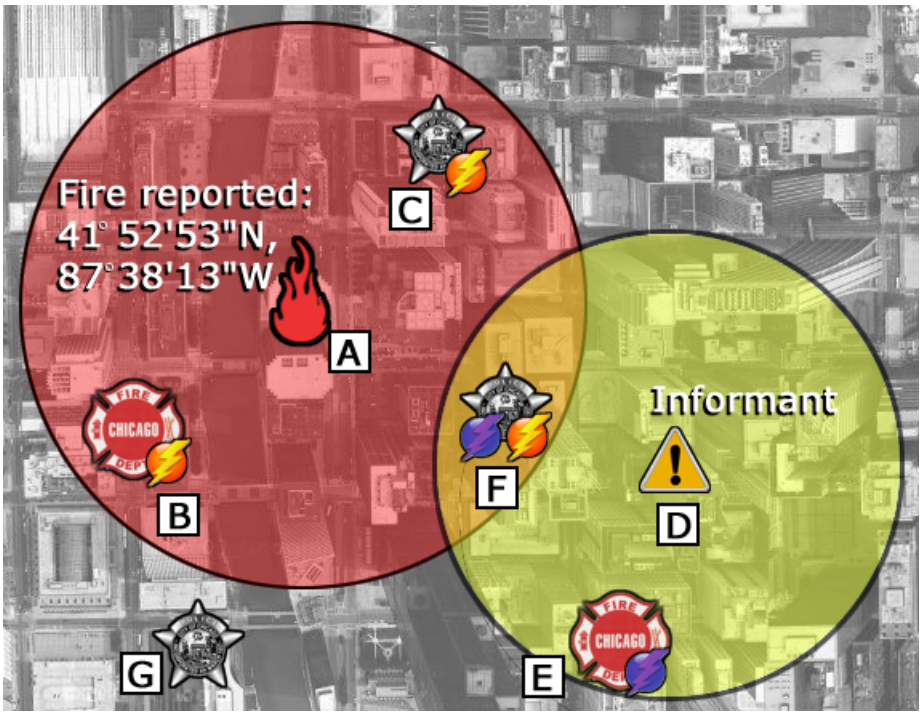


Figure 7: An example Event Reports scenario

3.5.1 Database optimizations

One of the key measures we have adopted so as to improve performance in the Webbee database - not just for Event Reports, but for all Webbee applications - is to streamline indexing into the table. For alpha-numeric data, for example, we have employed B-trees for range search queries, hashes for exact match queries, and full-text indexing for street names, event descriptions, etc. For GIS data, we employ a *Generalized Search Tree (GiST)* [7]. For historical trajectories, we use the *Scalable and Efficient Trajectory Index (SETI)* indexing mechanism [8]. For projected trajectories, we use the *STRIPES* [9] mechanism. These measures have markedly improved the retrieval performance of queries into the database.

3.5.2 Event Reports - exploiting database triggers for better performance

Event Reports clients update their coordinates through a small component of the application called the *position feeder*, which periodically and frequently transmits the client's GPS coordinates to the Webbee Coordination Server (this is in contrast to the other Application Daemons we have implemented, which only receive GPS coordinates on the receipt of a *request*). Notification of important events for a client is implemented through database triggers. The Webbee server contains an Information Server, which is a Postgres database with PostGIS [10] extension, that is integrated with an instance of a Visualization Server. The Visualization Server is a UMN MapServer that renders map data for visualization [11], which in turn interacts with an instance of the U.S. Census Bureau's Topologically Integrated Geographic Encoding and Referencing (TIGER) database [5] (Figure 8). When a client enters a designated waypoint such as an event report region - that is,

GPS coordinates and a radius - the database automatically triggers the insertion of a new record, containing the waypoint information and the entering user's information, into a special table in the Information Server Postgres database. Meanwhile, the event reports daemon (in a separate process) periodically checks this table to see if any new records were created. If so, the daemon creates a new SMS and sends it to the target user, notifying him or her about the waypoint.

The majority of the heavy lifting for this mechanism is done through an extension of Postgres triggers (Figure 9). Fewer queries are required, resulting in faster performance and fewer burdens on the system.

There are different flavors of triggers that are heavily employed within the Webbee system. There are *system-wide triggers*, which are deployed and dropped by the database administrator and are visible to all authorized users who choose to listen to related notifications. There are also *user-defined triggers*, which are deployed and dropped by individual users and which are visible only to their owners.

There are several varieties of supported trigger events in the Webbee system. There are *location-specific* notifications (e.g., "NOTIFY me WHEN I come WITHIN 2 miles of a gas station"); there are *alpha-numeric* triggers (e.g., "NOTIFY EVERYBODY WHEN the gas incident worsens"); and there are *mixed* notifications (e.g., "NOTIFY me WHEN I come WITHIN 2 miles of a gas station WITH gas price LOWER THAN $2.50").

Unfortunately, trigger support in Postgres is comparatively primitive: triggers are defined on tables, and if there are n triggers on a table, then whenever that table gets updated, n triggers occur. This can lead to performance problems if updates are frequent. Also, Postgres does not provide out-of-the-box support for multi-table triggers. This becomes a problem, for example, with mixed notifications: "NOTIFY me WHEN I come WITHIN 2 miles of a gas station WITH gas price LOWER THAN $2.50" requires one trigger on the *gas_station* table, and another trigger on the *user_location* table.

To address this problem, we have implemented a *trigger meta table*, which encodes relationships between trigger class identifiers and ownership, and is referenced before trigger evaluations. For example, consider again the mixed notification example above: "NOTIFY me WHEN I come WITHIN 2 miles of a gas station WITH gas price LOWER THAN $2.50." When the user's location is updated, for example, the trigger meta table is examined on the user id trigger class identifier. When the gas prices are updated, for example, entries in the meta table are examined on the gas station id and the trigger class identifier. The upshot of the implementation of the trigger meta table is that performance is up to 8X faster than without the meta table for alpha-numeric triggers, and up to 10X faster for spatial range triggers. The gain in performance increases as the total number of triggers increases (Figures 11(a), 11(b)).

3.6 AC^2

Another application that we have built using the Webbee infrastructure is the *Agent Contingency and Action Coordinator* application. The *Agent Contingency and Action Coordinator* (AC^2) application facilitates client communication in the field by providing a full messaging system that incorporates features such as text messaging, voice messaging, and picture messaging. In addition to sending messages by username, AC^2 allows users to send messages by *radius*. The mechanism is as follows: The sender includes his or her GPS coordinates, as well as a sender-defined radius in meters, within the header of his or her message. When the message is sent to the server, all agents' last known GPS coordinates are examined. The message is then sent to every agent who is

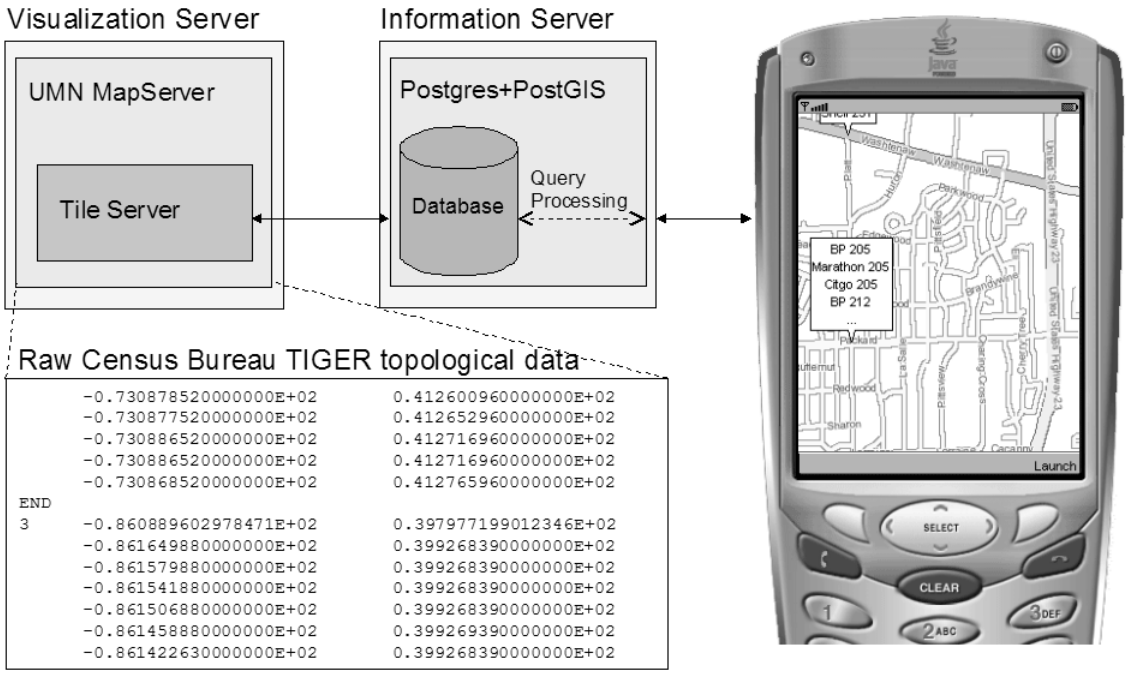


Figure 8: Interactions between client, information server, visualization server, and the TIGER GIS database

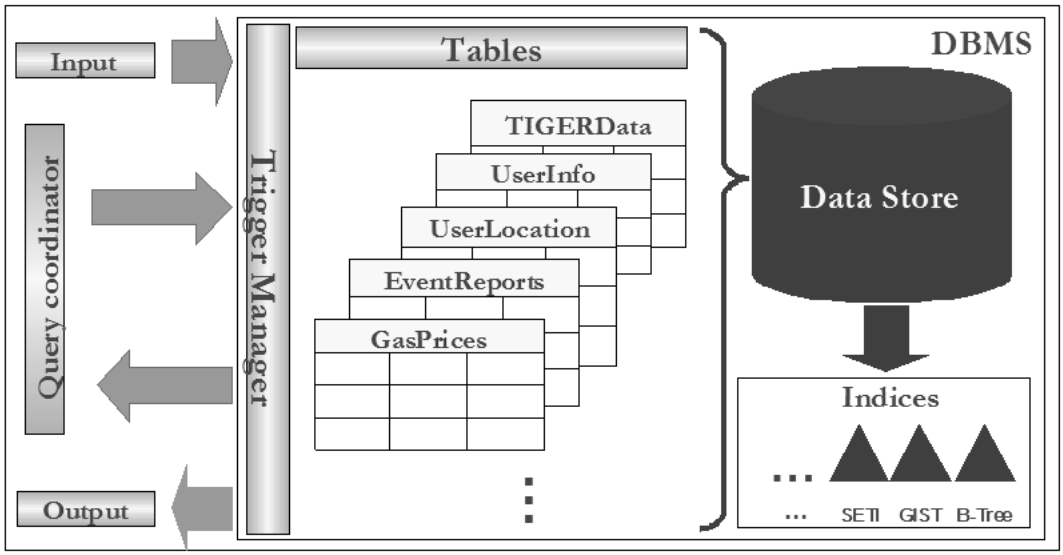
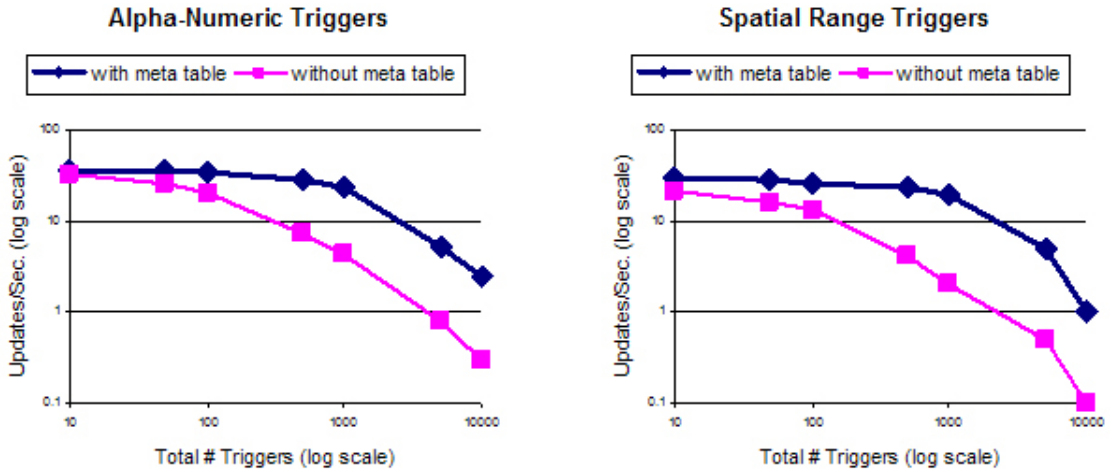


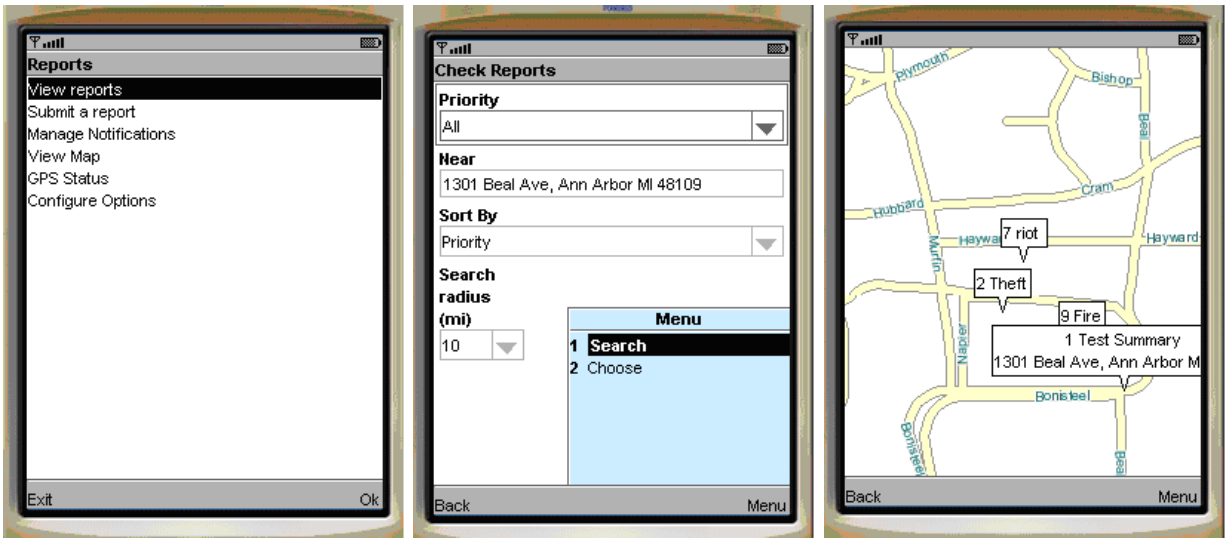
Figure 9: Details on database and triggers



(a) Meta table performance comparison for alpha-numeric triggers

(b) Meta table performance comparison for spatial range triggers

Figure 10: Trigger evaluations



(a) Initial menu

(b) Searching for reports

(c) Mapped reports

Figure 11: Mobile client screenshots for the Event Reports system

within the defined circle. This type of mechanism was designed for field situations where location-specific messaging might be useful, such as the transmission of orders for all agents within a specific location.

Another innovative feature of AC^2 is the capability of withdrawing messages. In essence, if a user has sent a message, and then at some later time decides that, due to a change in circumstances, s/he no longer wants the message to be consumed by other agents, s/he can direct the message to be withdrawn. That is to say, the message will be removed from the inboxes of all agents to whom the user sent it. This kind of functionality is useful in situations in which agents have decided that a reported incident has stopped being of interest. For example, if an agent initially reports seeing a suspicious package (e.g., a potential IED) at a given location, but on further inspection, determines that the package is not a threat, s/he has the ability to retract the message so as to prevent confusion and unnecessary action among his or her colleagues. All messages - including withdrawn messages - persist in the Webbee server log so as to provide a traceable audit trail.

3.6.1 AC^2 - Push-to-talk

We have recently completed work that enables AC^2 to support push-to-talk functionality. Push-to-talk is a half-duplex communication mechanism in which one person transmits and all others receive - similar to a walkie-talkie mechanism. Clients that want to participate in push-to-talk must first either join or create a conversation group within the application. Anything that is said from the time the button is pushed to when it is released is streamed to all other members registered in that conversation group. Once a user leaves a group, s/he does not receive any more messages (until s/he joins another group). Of course, all push-to-talk messages are logged on the Webbee server. The push-to-talk functionality is implemented in a component called the *broadcast server*, and push-to-talk was mainly developed to showcase *quorum*, discussed in section 3.2.3. At the time of this writing, we have almost fully integrated the broadcast server and the quorum server.

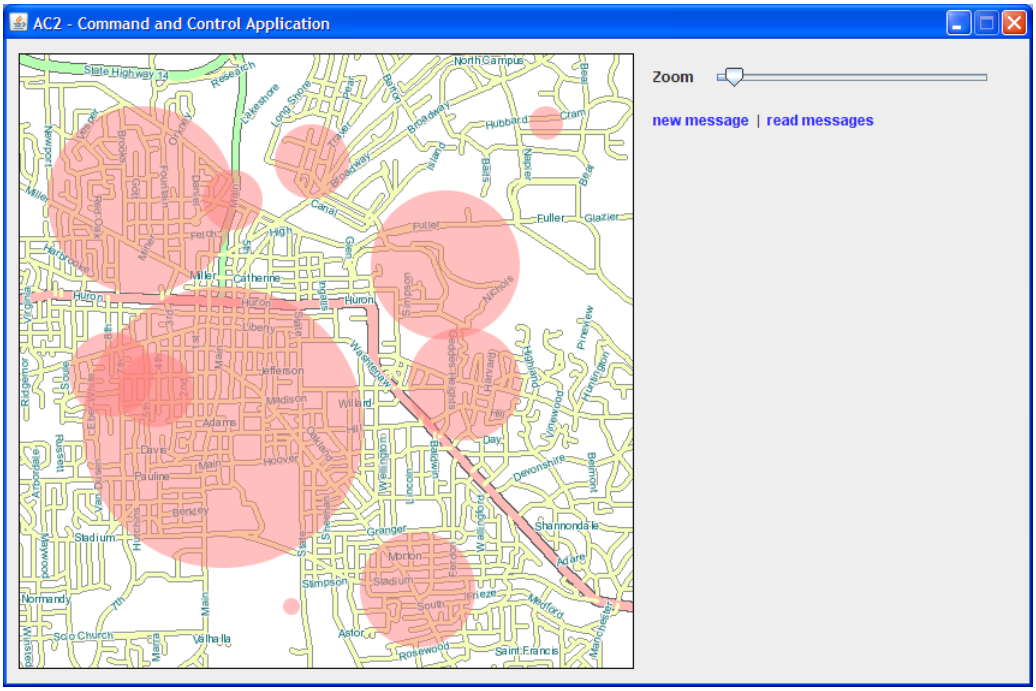
3.6.2 AC^2 - Command and Control

The *Command & Control* component is a desktop application in which commanding clients (e.g., generals) can visually identify all field clients' positions and operate as if they were in the field. As in the Gas Prices application, we are using the U.S. Census Bureau's TIGER map database, which generates a map image given GPS coordinates and a zoom level. We populate the map with an overlay that depicts the position of all field agents. We plan to be able to use the tool to track agents' movement over time. Furthermore, we have implemented a facility in which clients can choose a position on the map, and drag-and-drop a circle centering on that position. The desktop client will then be able to send a message to all field agents who are within that circle.

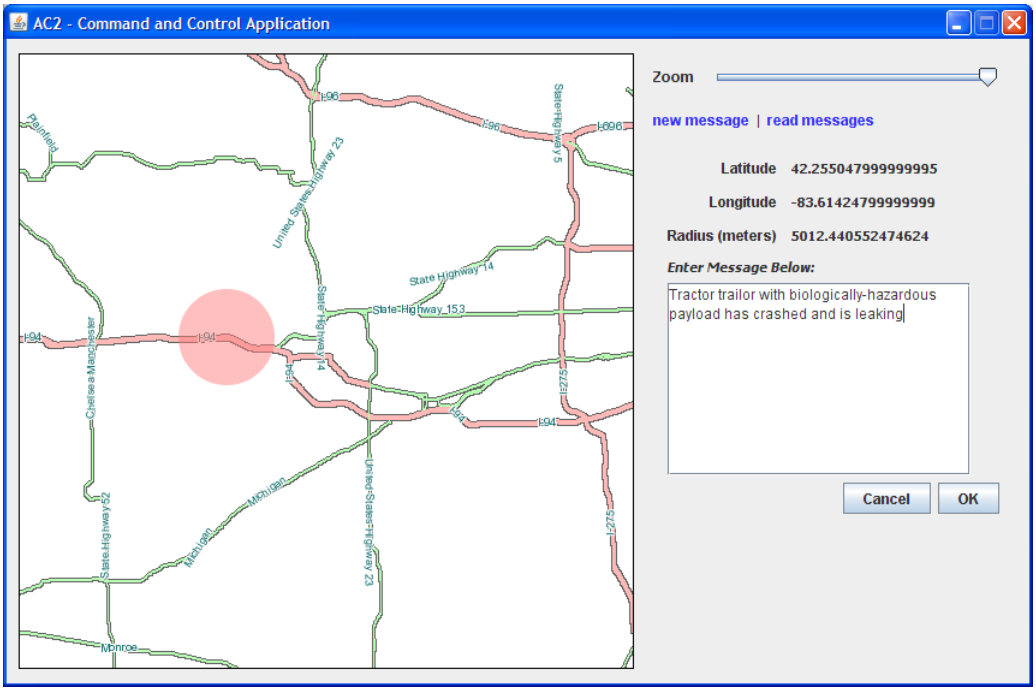
3.6.3 AC^2 - Future Work

Currently, we are integrating AC^2 and *Event Reports* into a single master application. We are midway through this process. This merger will relieve the smart phones of the burden of having two separate applications installed. Furthermore, we will be able to integrate some exciting functionality between the two applications. For example, we may want to be able to send audio and images - a feature of AC^2 - as components within reports - functionality implemented in *Event Reports*.

Other functionality we would like to implement is an Action Coordination facility within AC^2. This functionality would allow commanding agents to assign tasks across subordinate field agents.



(a) Viewing recently sent messages in the AC^2 Command and Control application



(b) Composing a message at arbitrary coordinates in the Command and Control application

Figure 12: Command and Control application screenshots

Conventional action coordination requires significant communication; the AC^2 application, however, reduces this communication burden significantly. Tasks might be of the form of a single task or a to-do list. Crisis situations often require the sharing of tasks across teams. A subordinate field agent will have the option to accept or reject the tasks, based on his or her situation. If the agent accepts, when it is done s/he notifies the server that the task is complete. This Action Coordination facility will provide the ability to have multiple agents complete one specific task. Commanding agents might, for example, assign a task that needs to be completed by twenty agents. We also plan to have the Command & Control Console fully integrate with the Action Coordination functionality described above.

Other Command & Control features we that are on our wishlist are the ability to track field agents' trajectories and predict future trajectories (so as to efficiently allocate supplies on actual supply routes). In addition, in scenarios where many reports or messages were submitted at around the same time near the same location, the digestion of the massive influx of data can be daunting. With natural language processing techniques like summarization, an application could condense a large volume of messages and/or reports into a summary that a commander could quickly internalize and act upon. We feel that this functionality would be of particular import in the coordination of field agents in times of crisis and confusion.

4 Evaluation

In the table below, only the overhead of the Webbee server and the Challenge System are measured. Currently, all challenges and solutions are only plaintext. If other challenge methods are used (e.g., biometrics), the overhead of the system may increase. Since the Challenge System needs to select a challenge from the challenge pool, send it to the client, receive the answer from the client, and then verify it, it has relatively large overhead. However, due to the fact that challenges do not occur very often, and that when challenges are not issued, the overhead is not significant (12.484 ms), we believe the performance of our Challenge System to be reasonable.

Table 1: Performance of the Challenge System

	Webbee Server Only	Challenge System on but no challenge assigned	Challenge System without any policy	Challenge System with policies
Average time (ms)	30.920	39.502	150.302	206.145
Standard Deviation (ms)	0.986	2.610	11.446	10.957
Overhead (ms)		8.581	88.462	144.305
Fraction (%)		21.724	58.856	70.002

Table 2: Round-trip time as a function of the number of challenges

Number of Challenges	Round-trip Time (ms)
1	150.302
100	154.6089649
500	173.2781172
1000	200.3992605
5000	440.559392

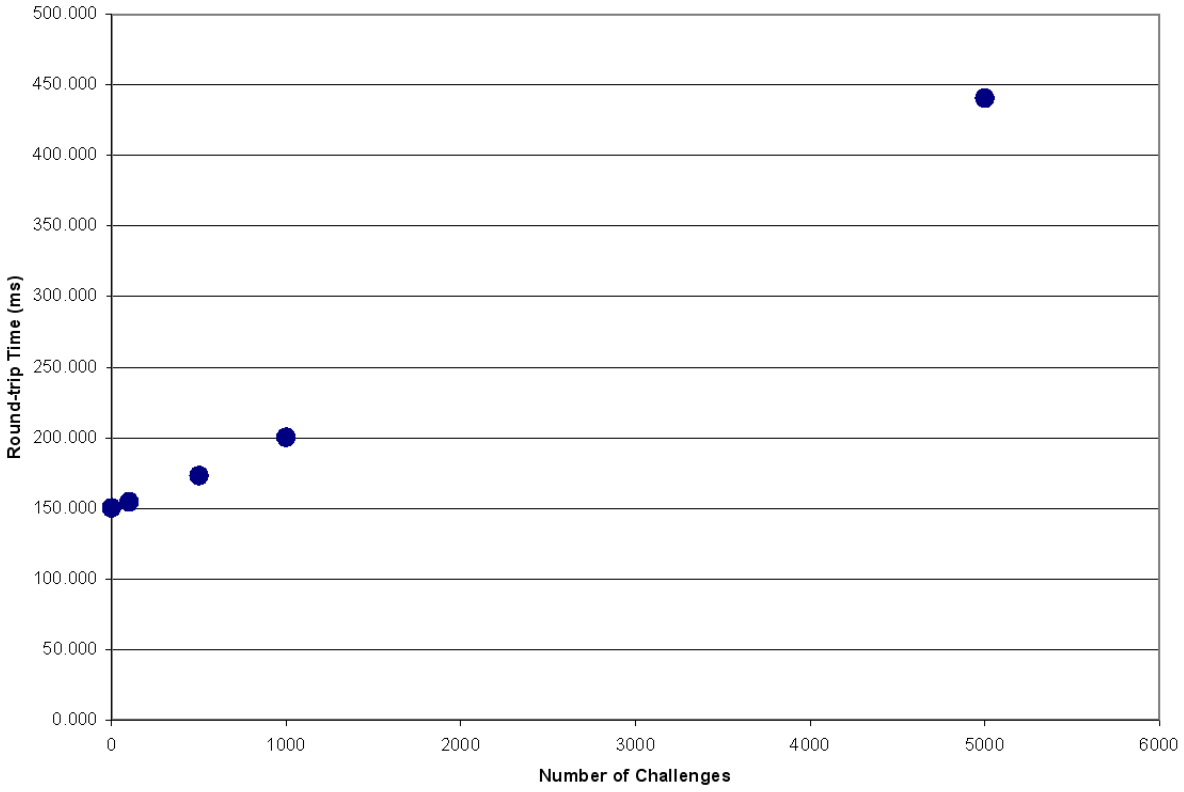


Figure 13: Round-trip time as a function of the number of challenges

5 Conclusion

As we have seen, the Webbee system is a robust, mobile, scalable communications and coordination framework that can handle various kinds of applications at various levels of security. Webbee is particularly suitable for first-responders in disaster scenarios. The Challenge-response system, an important component of Webbee's security, is a scalable security paradigm that is appropriate for situations in which communication and coordination is conducted through mobile handheld devices. The implementation of a policy hierarchy in this system strikes a nice balance between clients' situation-dependent security and future extensibility. The Quorum system, which we have used for download security, is another important mechanism that scales well when there are comparatively many downloads and few uploads. Finally, we have realized significant performance gains by implementing database optimizations like trigger meta tables and streamlined indexing.

References

[1] http://www.smesh.org

[2] R. Anderson. Invited Lecture. *Fourth Annual Conference on Computer and Communications Security*, 1997.

[3] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. *Lecture Notes in Computer Science*, 1666:431-448, 1999.

[4] Matt England, Axel Garcia-Pea, and Sugih Jamin. *Practical Strong Security for Mobile Handheld Devices*.

[5] http://www.census.gov/geo/www/tiger/index.html

[6] http://wiki.imacros.net/Main_Page

[7] http://gist.cs.berkeley.edu/

[8] V. Prasad Chakka, Adam C. Everspaugh, Jignesh M. Patel. *Indexing Large Trajectory Data Sets with SETI*. http://www-db.cs.wisc.edu/cidr/cidr2003/program/p15.pdf

[9] Jignesh M. Patel, Yun Chen, V. Prasad Chakka. *STRIPES: An Efficient Index for Predicted Trajectories*. Proceedings of the 2004 ACM SIGMOD international conference on Management of data, Paris, France. ISBN:1-58113-859-8

[10] http://postgis.refractions.net/

[11] http://mapserver.gis.umn.edu/