WebBee: A Platform for Secure Mobile Coordination and Communication in Crisis Scenarios

Sugih Jamin
University of Michigan[1]

Recently, disaster scenarios and terrorist attacks have made apparent some fundamental shortcomings in first responders' conventional coordination infrastructures. For example, unsatisfactory device connectivity and security vulnerabilities made evident by devices' inherently mobile nature have the potential to seriously compromise first responders' effectiveness. To address these shortcomings, our team designed and built WebBee, a secure coordination and communication infrastructure. This article will take a high-level look at WebBee's architecture, and examine some interesting, non-trivial sample applications we have deployed on top of it.

Ever since the September 11, 2001 terrorist attacks, the United States has been re-evaluating coordination for first responders in disaster scenarios. First responders must communicate reliably and securely in times of crisis. However, communication channels such as cell phone networks may be impaired or destroyed during disaster scenarios. Even if communication was *technically* feasible through these channels, extreme congestion might render them useless for first responders. Another problem is that these channels are more vulnerable to compromise: A malicious agent could steal a first responder's cell phone and intercept communications. This can seriously undermine a first responder's effectiveness in crisis situations.

The first responders have three primary needs. They must be able to communicate using devices they likely already have and are well-accustomed with. Secondly, the communication channel must be secure in mobile environments. Finally, while in a time of crisis, the consumer communication infrastructure can sometimes be used, it cannot be relied upon solely. WebBee addresses each of these concerns.

Architecture

There are three major components of the WebBee architecture (as shown in Figure 1 on the following page): the *instant infrastructure*, the *WebBee coordination server*, and the *database server*. The system has been designed so that components can be distributed across different machines.

Certain field personnel are equipped with battery-operated instant infrastructure backpack units. Equipment is commercial off-the-shelf hardware, so very large numbers of personnel can be outfitted easily. Custom SMesh software [1] helps maximize connectivity by dynamically reorganizing the network topology as personnel move about the field. The WebBee coordination server is an abstraction of several components that coordinate request handling, challenge-response management, policy examination, application hosting, and message dispatching. The database server manages all data interactions.

WebBee Coordination Server Component Detail

WebBee Master Server and Challenge Server Interaction

The WebBee master server negotiates traffic from clients between the challenge server and the application bridge. When a client request comes in, the WebBee master server stores it and asks the challenge server whether the client needs to be challenged. If the challenge server determines no challenge is needed, it tells the WebBee master server that it is OK to proceed. Otherwise, the challenge server issues a challenge through the master server to the client. The client's solution is sent back through the master server to the challenge server. If it is invalid, the challenge server informs the master server that no action is to be taken and the client is informed that the request was denied. If the solution is valid, the WebBee master server retrieves the client's most recent request and dispatches it to the application bridge. Our model, therefore, assumes that clients will only ever need a single request serviced at a time.

"... [communication] channels are more vulnerable to a compromise: a malicious agent could steal a first responder's cell phone and intercept communications. This can seriously undermine a first responder's effectiveness ..."

Security

Our security mechanism is broken into three separate subsystems: the challenge server, upload security, and download security. All are wrapped in a secure sockets layer.

The Challenge Server

The challenge server's job consists of *policies* and *challenges*. Policies encode conditions under which challenges are required, and are arranged in a hierarchy: If an agent passes one policy, there may still be subsequent policies that must be evaluated. The policy scheme for the WebBee coordination server is depicted in Figure 2 (see next page).

The first policy here is an *application-level* test. This special policy grants full access to certain applications, and demonstrates that WebBee supports both secure and non-secure applications. If the application must be challenged, a *temporal* policy is activated to determine if the client's last challenge-response has expired. If it has expired, the client is issued a challenge. The last policy is a *geospatial* policy: If the user has strayed far away from the set of last known global positioning system (GPS) coordinates, the client is challenged.

Policy intervals can be defined on a per-user basis, based on the level of security required for each client. At most, one challenge will occur through a traversal of this policy flowchart. Once the client solves the challenge, his or her GPS coor-

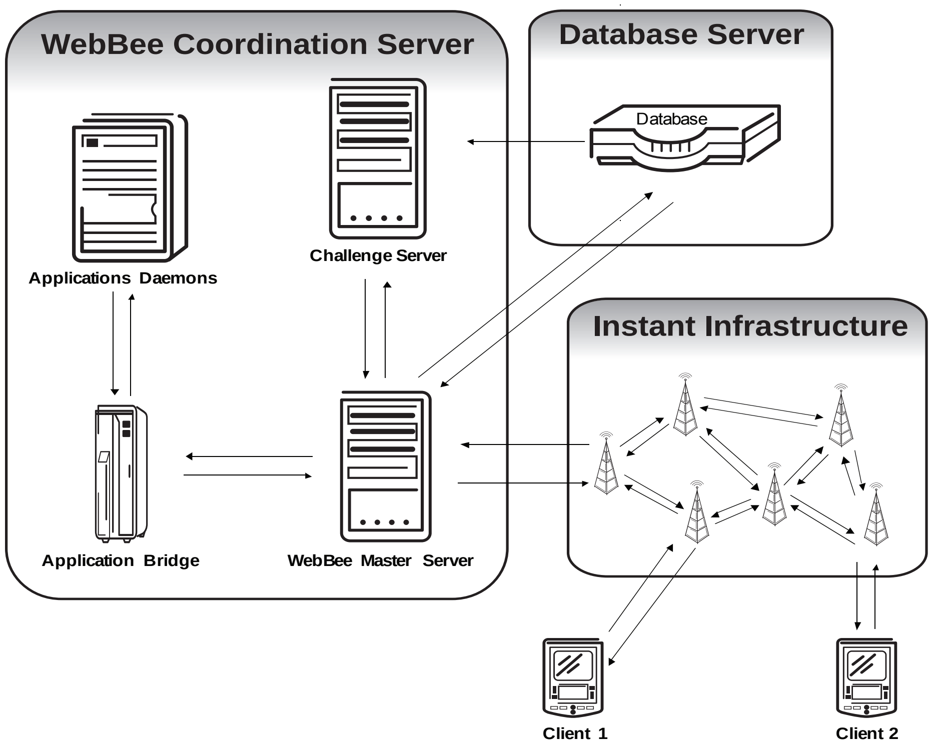


Figure 1: *WebBee Component Architecture*

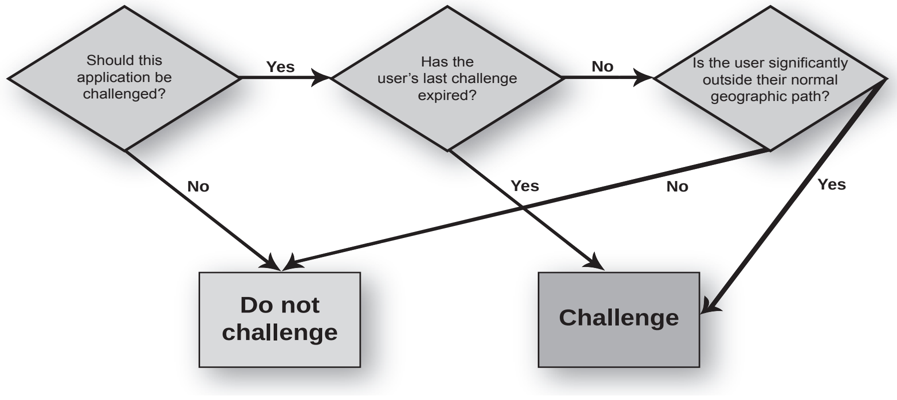

Should this application be challenged?
Yes
Has the user's last challenge expired?
No
Is the user significantly outside their normal geographic path?
No
Yes
No
Yes
Do not challenge
Challenge

Figure 2: *Policy Flowchart for the WebBee Coordination Server*

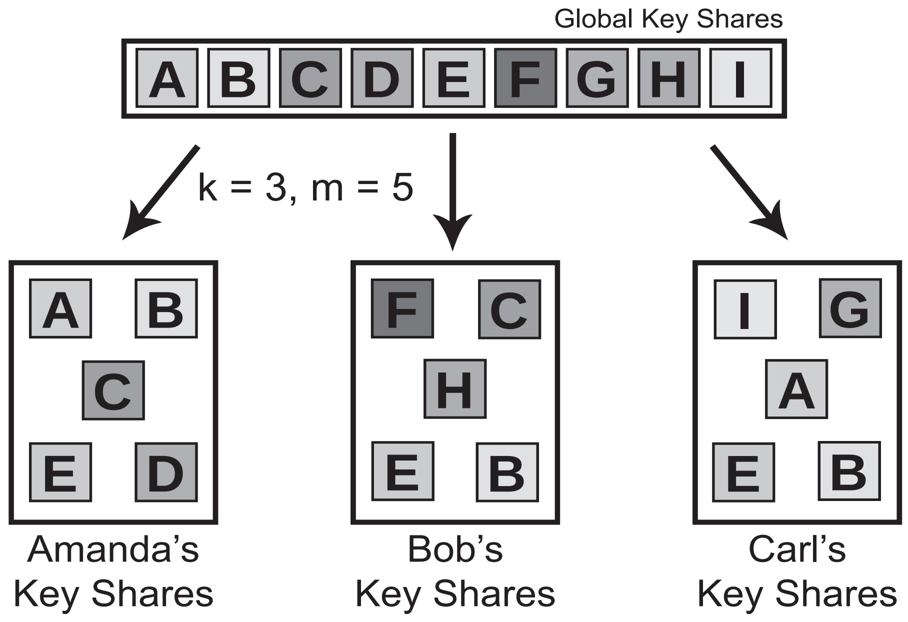


Figure 3: *Amanda, Bob, and Carl Initially All Have Valid Keyshares*

dinates and a timestamp are stored in the database.

When the policy flowchart determines that a challenge is required, the server randomly selects one of several possible challenges and issues it to the client. If the client solves it, then the request is serviced. Otherwise, the current and all subsequent requests will also be denied until the client successfully solves the original challenge. This eliminates malicious clients' ability to *game the system* by exploring the challenge space.

Currently, only text-based (e.g., password) challenges have been implemented. With the right hardware, the challenge system could be extended to issue other kinds of challenges, such as biometric challenges (e.g., fingerprint, voice, and/or retinal scanning).

Upload Security

In our scalable crisis management system, we are assuming that there are many downloads but relatively few uploads. With this in mind, we have decomposed our security requirements into upload and download security characteristics.

For 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, Mihir, and Miner [3].

Download Security: The Quorum System

For download security, scaling is an important issue. For clients, we want to require relatively few of their staff to have to acquire new keys during a change (e.g., departure or loss of device). The quorum system implements download security with these kinds of scalability concerns in mind.

In the quorum system, agents need to have a minimum number, k, of *keyshares* to securely read a message. At initialization, each agent receives m keyshares, where $m > k$, from a global keyshare 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, they must obtain a new set of valid keyshares from the global keyshare collection.

When the server broadcasts a message, it first encrypts it under a *message key*. This key, in turn, is itself encrypted s times. The s-encrypted message keys and the encrypted message are sent to all agents who decrypt the message keys using their personal keysets. If exactly *k* of the keys are identical, it is valid and the agent proceeds to decrypt the encrypted message with that decrypted message key.

Figures 3, 4, and 5 depict a scenario in which *k = 3* and *m = 5*. In Figure 3, Amanda, Bob, and Carl all have a quorum of valid keyshares. In Figure 4, when Bob leaves, three of Amanda's keyshares are invalidated, forcing her to obtain new shares. Carl only has two shares invalidated; he can continue to operate. Figure 5 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.

Application Bridge

The application bridge dispatches requests to the appropriate application daemon via an ID embedded in the request header. If a response is generated, it is sent back through the WebBee master server to the client. Gas Prices, Event Reports, and Agent Contingency and Action Coordinator (AC2) are three applications we have built using the WebBee framework.

Gas Prices

The Gas Prices application allows clients to determine the gas stations with the cheapest prices. A client initially sends a request containing his or her GPS coordinates. The Gas Prices daemon constructs a map through an implementation of the U.S. Census Bureau's Topologically Integrated Geographic Encoding and Referencing (TIGER) geographic information system (GIS) database [4], then queries a Web site that publishes up-to-date gas prices and sends it back to the client.

Gas Prices and other applications use the WebBee scraping engine to obtain data from the Web. For each application, a *scraping script* identifies the data components of interest in a Web page. Any static or dynamic data can be acquired—including text, images, and audio.

Event Reports

The Event Reports application (see Figure 6, next page) allows clients to log incidents that they observe in the field. Other clients are notified about these incidents only once they become geospatially relevant. Clients specify details about an incident by typing out a short message—as well as a radius in meters—on the handheld device. As other clients move in range, their handhelds are notified via the short messaging service (SMS). This relieves clients of having to sift through reports to determine which are immediately important, enabling him or her to react faster and more effectively.

A scenario is shown in Figure 7 (see next page). A report about a fire at the Chicago Mercantile Exchange (A) is submitted. One fire department unit (B) and two 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. By contrast, another police department (G) receives no alerts. As soon as G moves into range (if ever), they will receive the report.

Event Reports – Exploiting Database Triggers for Better Performance

Report notifications to clients are implemented through database triggers. The WebBee database server contains an

Figure 4: *Bob Leaves*

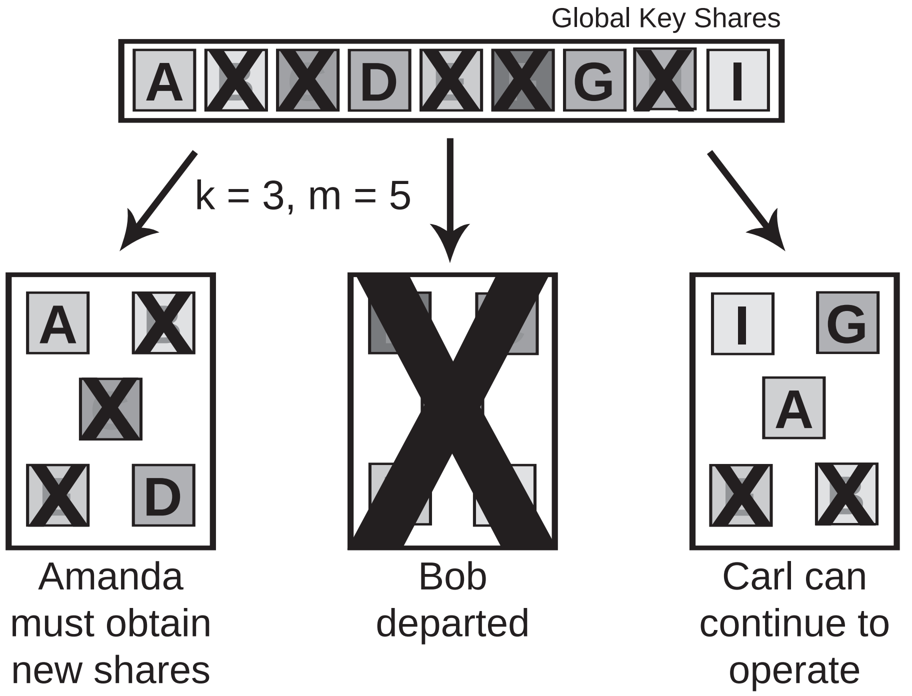
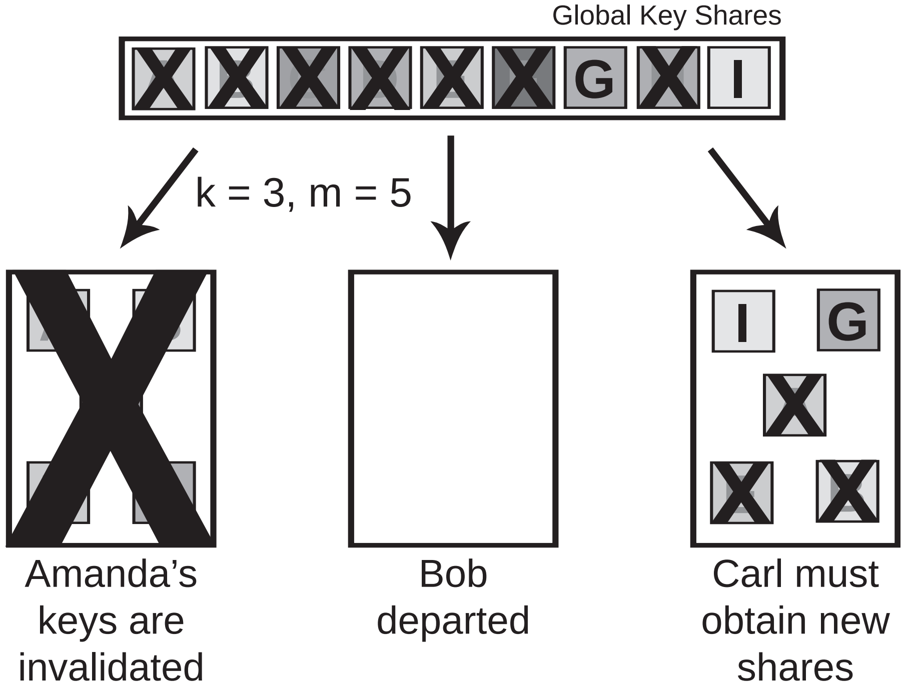
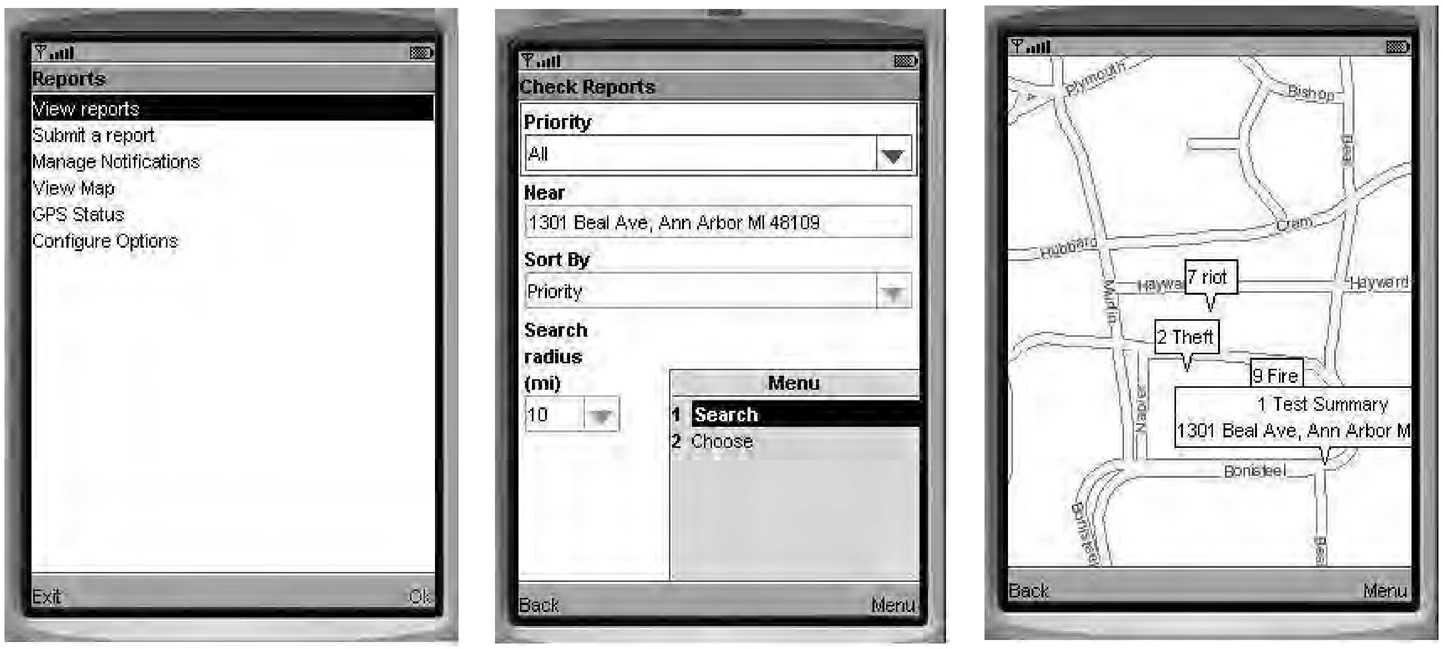


Figure 5: *Amanda Reports Lost or Stolen Handheld*





Priority: All

Near: 1301 Beal Ave, Ann Arbor MI 48109

Sort By: Priority

Search radius (mi): 10

Figure 6: *Mobile Client Screenshots for the Event Reports System*

information server, which is a Postgres database with a PostGIS [5] extension that is integrated with an instance of a visualization server in an application daemon. The visualization server renders map data for visualization [6] in concert with an instance of a TIGER database [4]. When a client enters an event report region, the database triggers the insertion of a new record into a special table. Meanwhile, the event reports daemon monitors this table. If there are any new entries, the daemon creates an SMS and sends it to the target user. The heavy lifting for this mechanism is done through an extension of Postgres triggers (Figures 8 and 9 show an example for alpha-numeric and spatial range triggers), resulting in fewer queries and better performance.

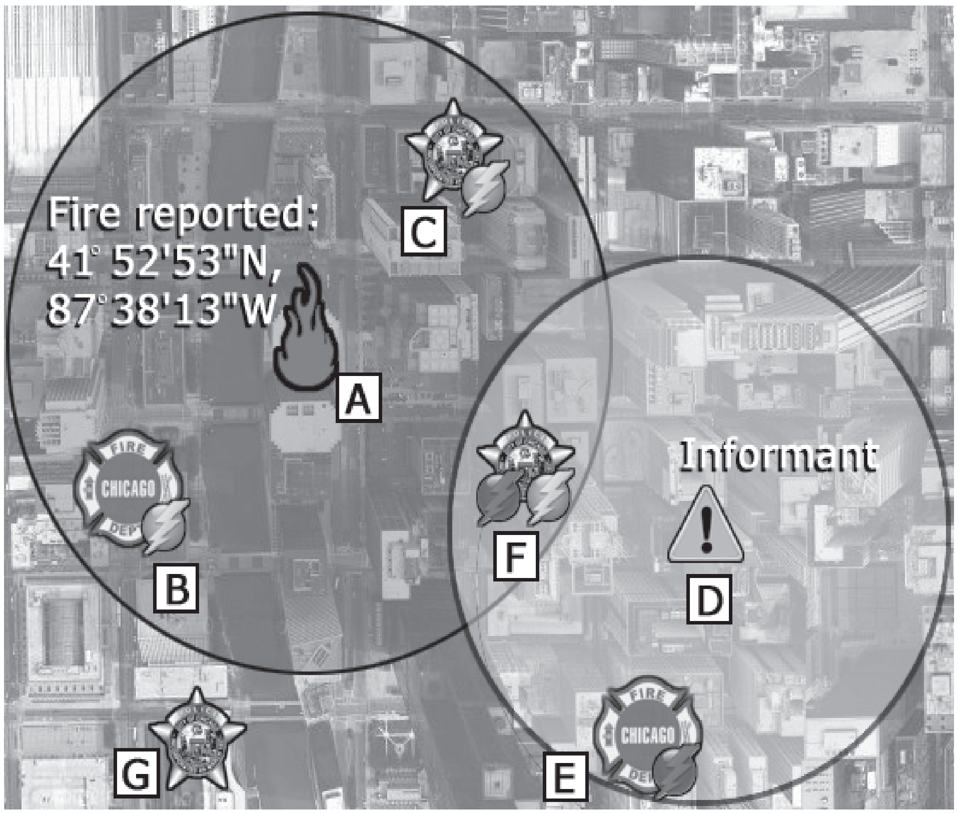
Trigger support in Postgres is table-based and comparatively primitive: with *n* table triggers, an update will cause *n* operations to occur, resulting in decreased performance if updates are frequent. Also, Postgres does not provide out-of-the-box support for multi-table triggers. This becomes a problem, for example, with mixed notifications.

To address these problems, we have implemented a *trigger meta table*, which encodes relationships between trigger class identifiers and ownership, and is referenced before trigger evaluations. Consider the mixed notification: "NOTIFY me WHEN I come WITHIN 2 miles of a gas station WITH a gas price LOWER THAN $3.50." When the user's location is updated, the trigger meta table is examined on the user ID trigger class identifier. When gas prices are updated, entries in the meta table are examined on the gas station ID and the trigger class identifier. Performance is up to eight times faster than without the meta table for alpha-numeric triggers (Figure 8), and up to 10 times faster for spatial range triggers (Figure 9). Performance increases as the total number of triggers increases.

Agent Contingency and Action Coordinator

Another application that we have built is an AC2 application, which provides a full-text, voice, and picture messaging system. Messages may be sent directly to individual clients or by *radius*. The radius message mechanism works as follows: The sender specifies his or her GPS coordinates and radius in meters within the message header. When the message is sent to the server, all agents' last known GPS coordinates are examined. The message is sent to all agents in the defined circle. Radius messaging might be useful, for example, for

Figure 7: *An Example Event Reports Scenario*



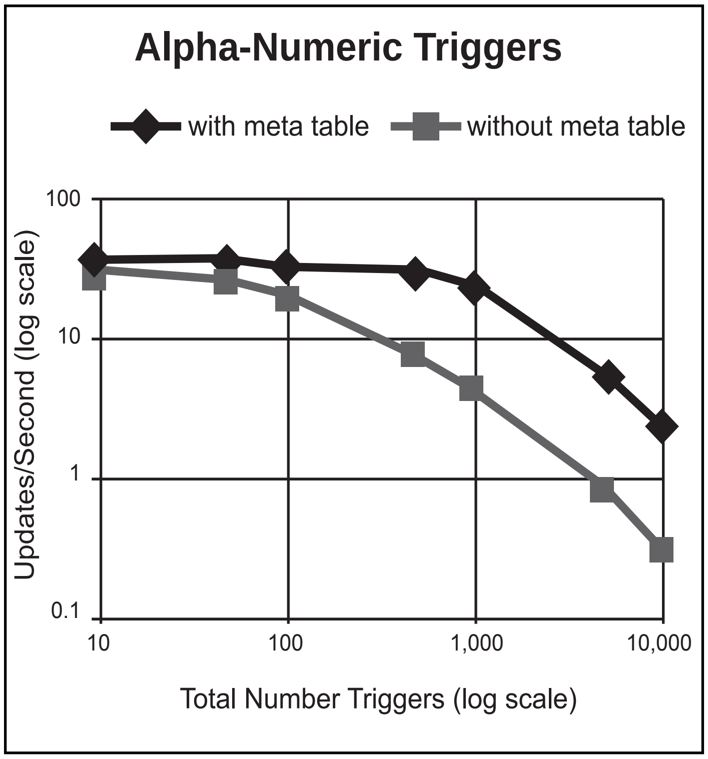


Figure 8: *Meta Table Performance Comparison for Alpha-Numeric Triggers*

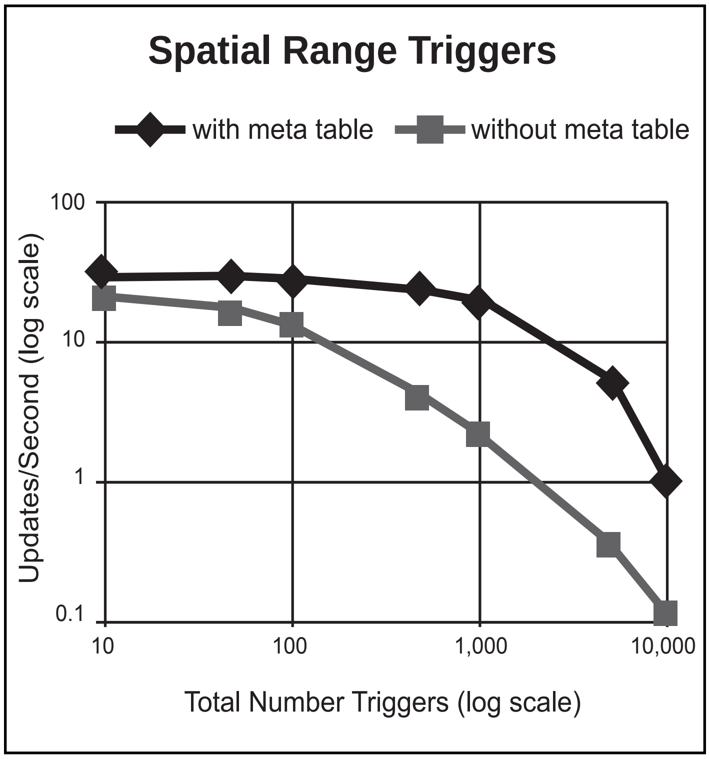


Figure 9: *Meta Table Performance Comparison for Spatial Range Triggers*

the dissemination of orders to all agents within a specific location.

Another innovative feature of AC2 is *message withdrawal*. If a client has sent a message and then later circumstances change and they no longer want the message to be read by other agents, they can withdraw the message; it will be removed from the inboxes of all agents to whom the user sent it. This is useful in situations in which agents have decided a reported incident has stopped being of interest. For example, if an agent initially reports seeing a suspicious package, but later determines that it is not a threat, they can withdraw the message to prevent confusion among the other agents. All messages—including withdrawn messages—persist in the WebBee server log so as to provide a traceable audit trail.

Conclusion

WebBee is a robust, mobile, scalable communications and coordination framework that can handle several applications at various levels of security. The challenge-response and quorum systems are scalable mobile security paradigms that are appropriate for our system. The implementation of a policy hierarchy strikes a nice balance between client situation-dependent security and future extensibility. Finally, database optimizations—like trigger meta tables and streamlined indexing—impart significant performance gains to our system.◆

References

1. Distributed System and Networks Lab at Johns Hopkins University. "SMesh." 2008 <www.smesh.org>.
2. Anderson, R. Invited Lecture. Fourth Annual Conference on Computer and Communications Security, 1997.
3. Bellare, Mihir, and Sara K. Miner. "A Forward-Secure Digital Signature Scheme." Lecture Notes in Computer Science 1666: 431-448, 1999.
4. United States Census Bureau. "Topologically Integrated Geographic Encoding and Referencing System." United States Census Bureau. 2008 <www.census.gov/geo/www/tiger>.
5. Refractions Research. "PostGIS." 2007 <http://postgis.refractions.net/>.
6. MapServer. University of Minnesota. 2008 <http://mapserver.gis.umn.edu>.

Note

1. Primary Investigators: Sugih Jamin, Zhuoqing Mao, T. V. Lakshman, Sarit Mukherjee, Jignesh Patel, and Limin Wang. Students, past and present: Brendan Blanco, Hyunseok Chang, Yun Jason Chen, Søren 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, and Byung Suk Yang.

About the Author

Sugih Jamin is an associate professor of computer science at the University of Michigan. His research interests lie in Internet measurement, protocol, as well as infrastructure design and deployment. He has earned many awards including the Sloan Foundation Fellowship, the National Science Foundation Faculty Early Career Development Award, and the White House Presidential Award. Jamin is also the chief technical officer and co-founder of Zattoo, a prominent startup that develops technology for delivering television content over the Internet. In addition to WebBee, his research at the University of Michigan includes building protocols, architectures, and mechanisms to support new uses of computer networks, as well as measuring, studying, and characterizing Internet topology and traffic.

The Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48109-2121
Phone: (734) 763-1583
Fax: (734) 763-1260
E-mail: jamin@eecs.umich.edu