BearLoc: A Composable Distributed Framework for Indoor Localization Systems

Kaifei Chen, Siyuan He, Beidi Chen, John Kolb, Randy H. Katz, David E. Culler
Computer Science Division
University of California at Berkeley
{kaifei, siyuanhe, beidichen1993, jkolb, randykatz, culler}@berkeley.edu

ABSTRACT

Many indoor localization algorithms have been proposed to enable location-based applications in indoor environments. However, these systems are monolithic and not component-based. We present *BearLoc*, a distributed modular framework for indoor localization systems that provides (1) natural development abstractions for sensor, algorithm, and application components, and (2) easy and flexible component composition. We demonstrate the merits of BearLoc with an example use case. Our evaluation shows we can reduce developer lines of code by 60% while introducing acceptable network delay overhead.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Patterns (e.g., client/server, pipeline, blackboard), Domain-specific architectures*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.12 [**Software Engineering**]: Interoperability—*Distributed objects*

Keywords

Localization; Framework; Composability

1. INTRODUCTION

Location information is important for many Internet of Things (IoT) applications. GPS-enabled location-based services have brought tremendous benefits to people's lives. For indoor environments, GPS alone cannot be a solution. Researchers have explored alternatives such as signal fingerprints, triangulation, trilateration, dead reckoning, and data fusion algorithms.

These efforts focus generally on achieving accuracy rather than ease of development. They are designed and implemented as monolithic systems that cannot be broken into distributable, composable, and reusable pieces. These systems have impeded the portability and deployment of indoor localization technologies. To deploy an indoor localization system developed by others, a developer still faces a significant implementation challenge. This is because there is no intermediate system that supports abstraction and modularization, manages the interfaces, and ensures interoperability between different components.

In this paper, we present *BearLoc*, a distributed modular framework for indoor localization systems that provides natural abstractions and composability. We claim the following contributions in BearLoc:

- To our knowledge, BearLoc is the first modular framework for indoor localization systems. It forms an ecosystem for portable and composable indoor localization system components.
- We base BearLoc on the intrinsic separation of three system components: *sensors*, *algorithms*, and *applications.*
- We design and develop a simple *Binding Control Protocol* to easily compose components and maintain bindings, where sensor data is mapped onto locations through algorithms.

The rest of this paper is organized as follows: Section 2 gives an overview of typical indoor localization system architectures. Section 3 discusses our design decisions. Details of the BearLoc system architecture are in Section 4. Section 5 gives a use case to show how BearLoc simplifies development and deployment. Section 6 is an evaluation of our BearLoc implementation. Section 7 concludes the paper.

2. RELATED WORK

Researchers usually build their own systems to evaluate indoor localization algorithms. Because their focus is on algorithm performance, those systems are not designed for modularity and extensibility.

2.1 Client-Server Architecture

A client-server architecture is commonly used in indoor localization systems [2, 4, 5, 9, 11]. In these systems, the applications using indoor locations are clients, and a server performs the localization computation. To use the server, the client is required to collect sensor data and transmit to it. However, such client-server models are designed and implemented as monolithic systems. Reusing a part of them is not straightforward, such as redirecting the sensors to an alternative algorithm. This is also not a good match for integrating sensors across the physical environment.

IoT-Sys 2015, May 18, 2015, Florence, Italy.
Copyright © 2015 ACM 978-1-4503-3502-7/15/05 ...$15.00.
http://dx.doi.org/10.1145/2753476.2753478.

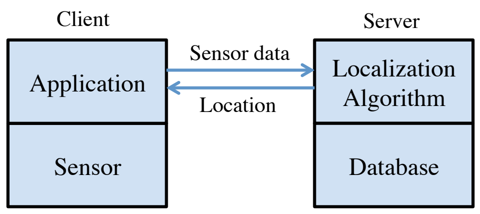

Figure 1: Typical Monolithic Client-Server Model of Indoor Localization System

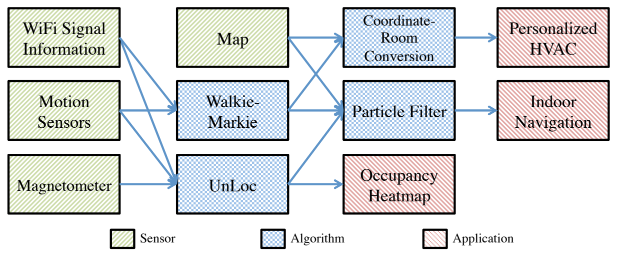

Figure 2: An Example of Decoupling and Composition of Indoor Localization Systems

2.2 Publish/Subscribe Network

A publish/subscribe (or pub/sub) network is a message-oriented network designed for asynchronous communication in distributed systems. It allows a subscriber to get messages, which are generated by publishers, that match its interests. In topic-based pub/sub networks, an interest is identified by a topic, which is an unique string identifier for a communication channel. This pattern is ideal for decoupling different components in terms of time, location, and synchronization [3]. Moreover, the asynchronous communication model works well for mobile devices and sensors that are power-limited as well as intermittently connected and addressed. However, to our knowledge, no one has taken advantage of this communication pattern and built a framework over it for indoor localization systems.

3. NEED FOR MODULAR LOCALIZATION SYSTEMS

Our goal is to make localization systems easier to develop, compose, and reuse than a client-server model. In this section, we discuss the problems of the typical client-server model and how we solve them in BearLoc.

3.1 Current Localization Architectures

Many current indoor localization systems are based on the client-server model, as shown in Figure 1. As an example, consider an indoor localization system using WiFi Received Signal Strength (RSS) as location fingerprints [1]. We could build a mobile phone application that generates these RSS values via an embedded WiFi chip. To get a location estimation, it sends the RSS values to a localization server, which maintains a database that maps fingerprints to previously surveyed ground truth locations. The server searches the database for the nearest fingerprint in Euclidean distance and returns the location thus found to the client.

Both the client and server are tightly coupled. There is no standard protocol for capturing sensor data schema and the interaction between the client and server. Suppose later there is an improved algorithm that uses n-grams of WiFi RSS as fingerprints [5]. We want to use this new algorithm simply by redirecting the application to a new server hosting this algorithm instance. However, if the new algorithm doesn't use the same protocol as the old, we have to change the client code.

Furthermore, indoor localization systems actually have more complicated configurations with respect to component distribution. Some systems have sensors that are not deployed on the mobile client device but dispersed throughout the physical environment. For example, vibration sensors [8] and cameras [6] deployed as building infrastructure can be used to identify people and infer their locations. Also, some applications are distributed between personal mobile devices and infrastructures. In building management systems, personalized Heating, Ventilation, and Air Conditioning (HVAC) control will need the locations of people. Shopping mall owners want to push advertisements to end users based on their locations. These heterogeneous scenarios require flexible composability among distributed components, which is not represented by the client server model.

3.2 Modular Architecture

It is ideal if deployed components can be reused and integrated for new applications with different settings. We achieve this using a framework that naturally decouples indoor localization systems and enables flexible composition. Figure 2 gives an example of how we would want it to work.

In this example, we have three applications: a personalized HVAC system, an indoor navigation application, and an occupancy heatmap visualizer. Each uses different algorithms for several reasons. First, they want different representation of locations. The personalized HVAC system wants a room number because it manipulates at the granularity of rooms. The navigation system and occupancy heatmap visualization need coordinates to visualize on the map. Furthermore, the navigation system requires higher accuracy and shorter response time than the other two. These applications need to pick an algorithm that fits their needs.

Among the algorithms, Walkie-Markie [9] uses peak WiFi RSS locations to calibrate dead-reckoning results. UnLoc [11] essentially uses a similar idea, except that it uses more types of landmarks from various sensors (e.g., WiFi, magnetometer). Because both of them output coordinates, the personalized HVAC system needs a simple algorithm that converts coordinates to room numbers using a map saved on a server. Also, the indoor navigation system wants to improve the accuracy by combining Walkie-Markie and UnLoc using a Particle Filter algorithm, so the algorithms are multiplexed.

As with the algorithms, sensors are also multiplexed. The sensors on the mobile device provide the data required by both Walkie-Markie and UnLoc. The map is used by both the Coordinate-Room Conversion algorithm and the Particle Filter algorithm.

Unlike in a client-server model, a component can run anywhere based on its need. For example, the map is a sensor that runs on a server, even though other sensors run on a mobile device. Also, the personalized HVAC system runs inside a building management system, whereas the indoor navigation application runs on a mobile device.

It is not simple to set up such a system using the client-server model. Now we look at what features are required for this degree of composition flexibility.

System Partitioning

BearLoc breaks up the system into pieces that can: (1) run on various distributed hardware, such as mobile devices and infrastructure components; and (2) clearly represent different roles in an indoor localization system, so developers can implement them independently. We argue that indoor localization systems essentially involve three different aspects, as shown in Figure 2: sensors, algorithms, and applications.

Sensors are the components that generate any location data, which are not limited to physical signals. For example, these can be WiFi chips along with the drivers that can read RSS values, or a textbox UI that takes strings describing locations from human input. Algorithms map sensor location "readings" like WiFi RSS or user input location into a particular location and its representations, such as a room or coordinate. An algorithm can be as simple as converting a coordinate to a room number based on a building map. Applications use the locations from algorithms for their particular tasks. By abstracting these three types of components, BearLoc enables sensors, algorithms, and applications to evolve independently.

Component Composition

In addition to system partitioning, components must also interact. There are three design decisions in BearLoc that enable easy and flexible composition. First, components have to agree on a data schema. BearLoc provides a library of sensor and location schemata. Second, BearLoc uses a topic-based pub/sub network designed for message-oriented asynchronous communications. This enables much flexibility in component composition, such as multicast and many-to-many communication. Third, we define a *Binding* concept that specifies how components can be "wired up," and design a *Binding Control Protocol* to initiate and maintain bindings. We will discuss Binding in detail in Section 4.1.

Sensor and Algorithm Multiplexing

It is not uncommon that a sensor or an algorithm is multiplexed as in our example. For many stateful algorithms like a Kalman Filter or a Particle Filter, they must maintain the relationships between input data to output topic while being multiplexed. BearLoc captures this relationship using a binding, and also provides an *Algorithm Manager* that implements this multiplexing. The algorithm manager maintains one algorithm instance process for every binding; therefore developers can focus on the algorithm itself. A sensor does not have such issues, because it is a source of data that everyone should see unaltered. Multiplexing a sensor is then handled by the pub/sub network multicast.

4. SYSTEM ARCHITECTURE

The BearLoc architecture is shown in Figure 3. From bottom to top, there are three layers: a topic-based pub/sub network, the BearLoc framework, and component implementations by developers. BearLoc provides wrappers for all three categories of components: sensors, algorithms, and applications. Developers build and deploy their components using relevant wrappers.

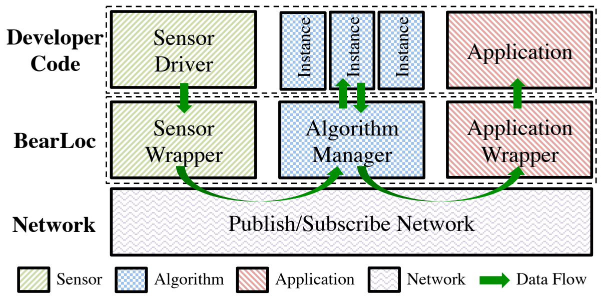

Figure 3: BearLoc Framework Architecture

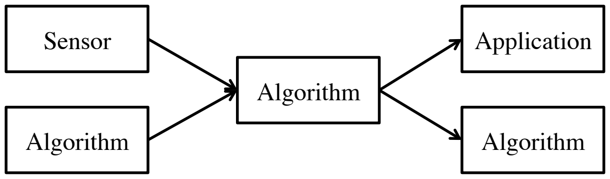

Figure 4: BearLoc Binding Definition

All components are distributable, and can run on any device or location. For example, a sensor driver can run on a low-power mote or a web browser, and an application can run as a part of an HVAC system or on a light actuator. Even though an algorithm typically runs on a server, BearLoc doesn't disallow deploying algorithms on local devices for localization without Internet connection.

4.1 Data Flow: The Concept of Binding

A *binding* specifies how an algorithm is "wired up" with other components, such that data flow through it to generate estimated locations. The concept of binding is illustrated in Figure 4 and a typical data flow between components is shown in Figure 3. In a binding, an algorithm takes data from existing sensors or other algorithms and publishes locations to applications or other algorithms. More complex configurations, such as chaining and multiplexing, can be composed using multiple bindings. A functioning indoor localization system consists of both running components and bindings connecting them together. As BearLoc is built upon a topic-based pub/sub network, a binding is created by publishing to an algorithm's control topic with several topics of existing sensors/algorithms and a new output topic. Other applications and algorithms can get the estimated locations by subscribing to the new output topic. This procedure is defined by our Binding Control Protocol in Section 4.2.

4.2 Control Flow: Binding Control Protocol

BearLoc provides a *Binding Control Protocol* between algorithms and applications for easy binding creation and maintenance. An application can start a new binding as follows:

BearLoc Binding Control Protocol

```
1: When an algorithm starts, it subscribes to its control topic.
2: An application sends a Start Binding request to the control topic, which contains (1) a list of existing sensor/algorithm output topics, (2) a new algorithm output topic, (3) a keep-alive topic, and (4) optional algorithm-dependant configuration data.
3: The algorithm (1) subscribes to the sensor/algorithm output topics and the keep-alive topic, and (2) publishes new locations to the algorithm output topic.
4: The application periodically sends keep-alive messages to the algorithms it is interested in.
```

In Step 2, there are four elements in a start binding request. The first two tell the algorithm where to "wire" the inputs and output respectively. The keep-alive topic is used for applications to preserve the binding and continue processing sensor data. The optional configuration data is useful for algorithms using shared sensors to filter data for specific targets. For example, a vision-based human tracking algorithm may only report locations of particular people. After a new binding is created, other applications and algorithms can subscribe to this algorithm and get location updates as well.

The first two steps require the application to know the algorithm specifications such as control topic, input sensor list, and configuration options. This should be simplified using a sensor and algorithm discovery service. We leave this to our future work.

4.3 Components

4.3.1 Sensor

In Figure 3, the sensor driver is a data generation implementation of a *Sensor Driver* interface defined in BearLoc. The interface specifies the sensor class, and lets the sensor wrapper register a callback function for data updates. When the sensor generates new data, the sensor wrapper relays it to the sensor's topic on the pub/sub network.

BearLoc provides a library of commonly used sensor classes. A sensor class specifies the data schema of a sensor, and provides data serialization and deserialization functions. To ensure interoperability between sensors and algorithms, sensor developers must use the sensor classes supported by BearLoc.

4.3.2 Algorithm

An algorithm manager implements the binding control protocol and multiplexes an algorithm. For every start binding request it receives, it starts one algorithm instance, which is an executable process of its algorithm. The algorithm executable is implemented by algorithm developers using a BearLoc algorithm interface. The interface is an RPC server wrapper that invokes localization computation methods. An algorithm manager relays sensor data to an algorithm instance and then directs localization results back as an RPC invocation.

An algorithm manager also subscribes to the keep-alive topics for all bindings. A binding expires when no message is published to its keep-alive topic after a timeout period. Once a binding expires, the algorithm manager kills the associated algorithm instance, and unsubscribes from its topics.

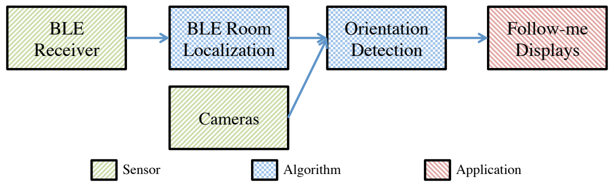

Figure 5: Composition of An Indoor Localization System for Follow-me Displays

We can see an algorithm manager maintains one-to-one mappings from bindings to algorithm instances. By managing algorithm instances, BearLoc hides the multiplexing overhead from algorithm developers.

4.3.3 Application

An application uses a localization service by initiating a binding and registering a callback function for location updates through the *Application Wrapper*. The application wrapper implements the binding control protocol. It also subscribes to the algorithm output topic, and invokes the registered callback function on new locations.

5. CASE STUDY: FOLLOW-ME DISPLAYS

To demonstrate how developers use BearLoc, we now describe one use case. Imagine a building outfitted with network connected displays throughout its interior. As users move between different locations in the building, they may interact with different displays at different points in time, depending on which are nearby. The implementation of such a follow-me display application relies on localization services.

We can build this system as shown in Figure 5, which involves complex compositions like chaining and multiplexing. It has one Bluetooth Low Energy (BLE) receiver deployed as a BearLoc sensor in each room listening for advertisements from users' wearable devices[1] and publishing them. A BLE room localization algorithm collects all advertisements and searches for an ID configured during binding creation. To ensure the person is facing the display in her room, an orientation detection algorithm using the camera in the room is triggered. Upon confirmation that a user is facing the display, the orientation detection algorithm outputs the room number to the follow-me display application. Note that every user of this application builds such a chain, and the BLE receivers are shared among multiple BLE room localization algorithms.

Because BearLoc modularizes this system, people can build each component independently. We now describe how different developers will use BearLoc.

Sensor developers can set up the BLE receivers and cameras using the BearLoc sensor wrapper. In particular, their implementations need to pull data from BLE receivers and camera drivers, instantiate data objects of sensor classes provided in BearLoc, and call the notification function. This is set up without considering any other components.

Algorithm developers write programs that implement the BearLoc algorithm interface, which contains only one method called "localize". For example, BLE room localization algorithm developers must implement a "localize" method that scans input advertisements and, if the target user is found,

[1]Such as a Fitbit Flex: `https://www.fitbit.com/flex`

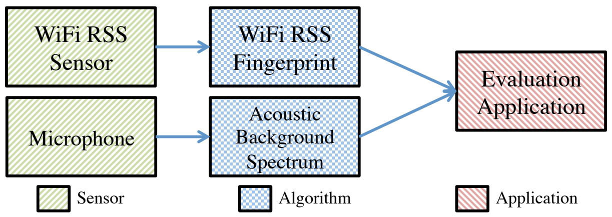

Figure 6: Evaluation Setup for Data Flow Time Measurement

returns the new location. This development only involves the computation itself with no multiplexing concern. For every algorithm, algorithm developers start an algorithm manager, which registers the path of the algorithm's executable. In the follow-me display example, two algorithm managers are created for BLE room localization and orientation detection respectively.

Apart from implementing application specific logic, application developers must compose components by creating bindings. The follow-me display application needs to create two bindings for every user: a BLE room localization binding, and an orientation detection binding. This is done by calling the "start_binding" method provided by the BearLoc application wrapper.

6. EVALUATION

We attempt to answer two questions: how much effort can BearLoc save for developers? And how much overhead does BearLoc introduce?

6.1 Experiment Setup

We implemented a sensor wrapper and application wrapper for Android in Java, and an algorithm manager in Python. The algorithm manager talks to algorithm instances via the Cap'n Proto[2] RPC Protocol. We use MQTT[3] as the pub/sub network. Specifically, the Paho[4] MQTT client and the Mosquitto[5] MQTT broker are used.

We implemented a WiFi RSS sensor and an audio sensor for Android in Java, a WiFi RSS fingerprint-based algorithm [1] in Python, and an Acoustic Background Spectrum (ABS) algorithm [10] in MATLAB with a Python wrapper. We also built an evaluation Android application that uses both location algorithms. Our experimental setup is shown in Figure 6.

6.2 Lines of Code

To assess the ease of deployment, we use *lines of code* as an approximate metric for developer effort. We implemented the WiFi RSS algorithm as a server in Python with Twisted[6]. We also built both Audio and WiFi RSS sensors for Android in Java. Both sensors report data to the server over HTTP.

Table 1 summarizes the lines of code of all components written by developers, with and without BearLoc. BearLoc reduces nearly 700 lines of Java code for the two sensors.

[2]`https://capnproto.org`
[3]`http://mqtt.org/`
[4]`https://eclipse.org/paho/`
[5]`http://mosquitto.org/`
[6]`https://twistedmatrix.com`

	Audio Sensor	WiFi RSS Sensor	WiFi RSS Algorithm
Without BearLoc	856	925*	737
With BearLoc	173	184	350

* It shares 722 lines of codes with the audio sensor.

Table 1: Lines of Code Comparison of Components With and Without BearLoc

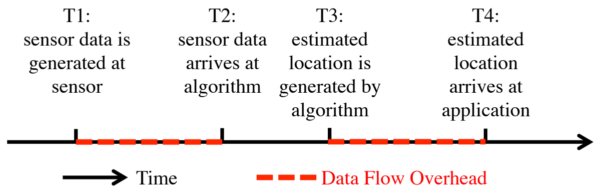
T1: sensor data is generated at sensor
T2: sensor data arrives at algorithm
T3: estimated location is generated by algorithm
T4: estimated location arrives at application
Time
Data Flow Overhead

Figure 7: Data Flow Overhead

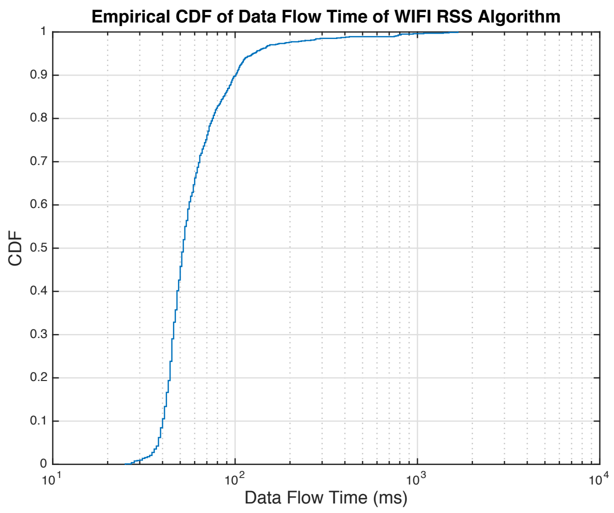

Figure 8: CDF of Data Flow Overhead of WiFi RSS Fingerprint-based Localization

This is primarily because the data queuing and retransmission is moved to MQTT, and sensor data schemata are provided by the BearLoc sensor classes library. The WiFi RSS algorithm saves about 390 lines of code by moving the Twisted server to the Bearloc algorithm manager. In total, BearLoc reduces lines of code by 60% for developers.

6.3 Data Flow Overhead

We measure data flow overhead, defined as the time from when sensor data is generated to when the location is received by the application, subtracting the algorithm computation time. It is illustrated in Figure 7. The data flow overhead captures the network delay overhead in a data flow introduced by distributing components in BearLoc.

The measurement is done for both WiFi RSS and ABS algorithms with MQTT QoS value 1, which indicates the messages are delivered to subscribers at least once. The WiFi RSS sensor scans every 2 seconds, and the microphone records for 1 second with 4-second intervals. The sensors

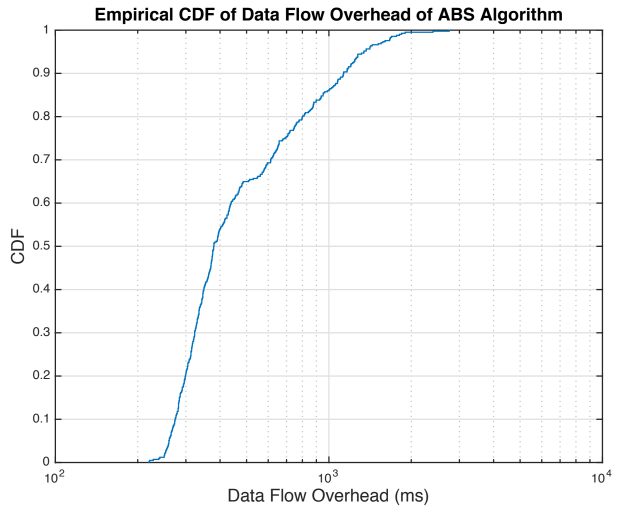

Figure 9: CDF of Data Flow Overhead of Acoustic Background Spectrum

and evaluation application run locally on an LG G2 Mini Android phone. The WiFi RSS algorithm and the MQTT broker both run on an Amazon EC2 server, and the ABS algorithm runs on a laptop.

Because our sensors and application run on the same device, we can directly measure $T4 - T1$ in Figure 7. The algorithm computation time, which is $T3 - T2$, is recorded in the algorithm manager and sent back to the application along with the location result. Then the data flow overhead is calculated as $(T4 - T1) - (T3 - T2)$.

Figure 8 and Figure 9 show the CDF of data flow overhead of the two algorithms respectively. 90% of data flows in WiFi RSS have overhead less than 0.1 second. BearLoc does not add more overhead apart from MQTT [7]. Nearly 85% of data flows in ABS have delay overhead less than 1 second. The overhead here comes from audio recorder operations and heavy audio data copying and serialization. Given these, we conclude that the BearLoc framework introduces acceptable network delay overhead.

7. CONCLUSION

In conclusion, we have argued that BearLoc simplifies indoor localization system development with natural component abstractions and an easy composition mechanism. Starting with BearLoc, we aim to build an ecosystem for indoor localization system components. Furthermore, we hope these ideas can also be applied to more IoT systems that have similar needs to indoor localization systems.

Acknowledgements

We thank Qualcomm Inc. and Intel Corp. for their gifts. This work is supported in part by the National Science Foundation under grant CPS-1239552 (SDB) and the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP), a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

8. REFERENCES

[1] P. Bahl and V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM, 2000 Proceedings IEEE*, volume 2, pages 775–784. Ieee, 2000.

[2] P. Bolliger. Redpin-adaptive, zero-configuration indoor localization through user collaboration. In *Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments*, pages 55–60. ACM, 2008.

[3] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[4] X. Jiang, C.-J. M. Liang, K. Chen, B. Zhang, J. Hsu, J. Liu, B. Cao, and F. Zhao. Design and evaluation of a wireless magnetic-based proximity detection platform for indoor applications. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 221–232. ACM, 2012.

[5] Y. Jiang, X. Pan, K. Li, Q. Lv, R. P. Dick, M. Hannigan, and L. Shang. Ariel: Automatic wi-fi based room fingerprinting for indoor localization. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 441–450. ACM, 2012.

[6] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *Visual Surveillance, 2000. Proceedings. Third IEEE International Workshop on*, pages 3–10. IEEE, 2000.

[7] S. Lee, H. Kim, D.-k. Hong, and H. Ju. Correlation analysis of mqtt loss and delay according to qos level. In *Information Networking (ICOIN), 2013 International Conference on*, pages 714–717. IEEE, 2013.

[8] S. Pan, N. Wang, Y. Qian, I. Velibeyoglu, H. Y. Noh, and P. Zhang. Indoor person identification through footstep induced structural vibration. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 81–86. ACM, 2015.

[9] G. Shen, Z. Chen, P. Zhang, T. Moscibroda, and Y. Zhang. Walkie-markie: indoor pathway mapping made easy. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 85–98. USENIX Association, 2013.

[10] S. P. Tarzia, P. A. Dinda, R. P. Dick, and G. Memik. Indoor localization without infrastructure using the acoustic background spectrum. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 155–168. ACM, 2011.

[11] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury. No need to war-drive: unsupervised indoor localization. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 197–210. ACM, 2012.