Smarter Push Notifications in Multi-Device Environments

Theodore F. Morse · Sungho Jo

The number of internet-connected devices a person interacts with has increased. With this increase comes an increase in the number of devices available to notify the user of new information. This paper proposes an algorithm for intelligently selecting the primary device that should first interrupt the user with new information. This algorithm fuses information such as activity level and movement from all of a user's devices to predict which device is best. This algorithm is then implemented as a push-notification service. Using this service, and implemented clients for Android and Google Chrome, we test this algorithm to establish both its ability to predict the user's preferred device and that it reduces the annoyance level of users. Our results confirm that our algorithm has the ability to select an appropriate device, in the future more accurately predict the correct device, and finally that it is less annoying than a naive approach.

Keywords: Mobile devices, Notifications, Push-services, Networking, User context acquisition

I. INTRODUCTION

The early start of the internet and the introduction of email in the 1970's has transformed the computer into far more than a simple computing device. The computer became a communication device as well. With communication comes the desire to know when new information is available.

For years, before the introduction of mobile phones, the computer was the primary device to find out about this new information. Computers were relegated to the office and, as time went on, at home as well. When mobile phones and pagers were introduced, a new, smaller device that could notify the user came into existence. Even so, the data that the user consumed on each device was separate. For example, people consumed text messages and pages on mobile phones and pagers while they consumed email and other internet-based information on computers. However, this too changed with the introduction of the smartphone and the ability to be connected to the internet via a mobile device.

With the introduction of internet-connected smartphones came the ability to have the information that was formerly relegated to that of the computer spill over into the mobile side as well. Smartphone users could be notified of any new email, chat messages and other important information just as easily on their computers as on their smartphones. However, in the beginning, individual applications were required to setup the way they fetched this new information on their own. Some applications took the route of polling a server periodically to find new information, while others set up persistent connections to servers in order to receive data as a push-based mechanism.

It was quickly realized that every application having its own way to get new information negatively impacted devices in various ways. For example, waking up the network hardware for every application to poll or send

논문번호: TR14-116, 논문접수일자:2014.12.23, 논문수정일자:2015.05.29, 논문게재확정일자:2015.10.22
Theodore F. Morse, Sungho Jo(Corresponding author): Korea Advanced Institude of Science and Technology

keep-alive information to their respective servers depleted the battery faster. For this reason, the major smartphone operating systems set up their own centralized push mechanism for application and system-related messages to be passed down to the phone from a central server in the form of a persistent network connection. [1], [2] This negated the need for every application to setup its own protocol which both increased battery life and eased application development.

In recent years, the introduction of tablet computers and smart watches caused the number of possible devices to consume this pushed information to explode. This presents new challenges for delivering this information. Among these challenges is how to handle the issue of multiple devices. Up to this point, this has been handled on a per-application basis. Naive applications such as email may simply notify all of the devices at once. In the worst case, this would cause possibly more than three devices around the user to simultaneously blink and make noise. For some users, this may not be an issue. A simple silencing of the offending devices will ensure that they are not annoyed further. For others, multiple devices may be comforting. However, scaling up the number of devices may reach a tipping point where silencing the offending devices takes more time than it is worth. Furthermore, if silencing devices, one may forget to re-enable them, possibly missing important notifications in the future. Thus, while user preference is important, the ability to be smarter about which device to notify the user on may become a requested feature. Currently, this puts the onus on each application to handle this issue on their own.

This situation closely analogs the time when applications were creating their own unique ways of pushing or polling for new information from their respective servers. For every new application, they may eventually have to deal with the same type of problems. Thus, it would behoove us to research the best possible way to solve such a problem and build it into the lower levels of such mechanisms already in place in order to reap the same types of benefits that were gained by centralizing the push-mechanism. Furthermore, this centralization is of great benefit to the user as well.

Thus, this paper aims to provide further research into this problem in order to find a better solution than what is currently deployed. Furthermore, research of this nature may eventually make its way into the base push-mechanisms of today's technology, freeing future application developers to solve more interesting problems, as well as giving users a consistent and more intelligent mechanism for receiving new information.

The problem to be solved may be summarized as follows: We want to pick the best device out of the possible devices a user owns. This best device is called the primary device. This paper aims to solve the problem of selecting the correct primary device in which to first notify the user of new information. For the purposes of this paper, the correct primary device is the device that both annoys a user the least, as well as being the preferred device by the user to receive notifications on.

We attempt to solve this problem through a smarter algorithm for determining the user's current context. Based on this context, we select the best primary device and send new notifications to only this device.

This paper is organized into multiple sections. Section I provides an introduction to the problem and gives reasoning for why it is needed along with an overview of the methods and objectives used to solve the problem. Section II gives an overview of the existing literature in the areas related to the problem at hand, including activity tracking, annoyance levels and current commercial offerings related to the problem. Section III goes into detail about the proposed algorithm and the system built to test the algorithm. Section IV provides details about the experimental setup and protocols used to obtain the results to test our algorithm. Section V details the results of the experiments conducted and dives into a discussion about the results. Finally, in Section VI we conclude and state future work that should be undertaken to improve upon the results found.

II. RELATED WORK

Work related to this paper can be divided into several areas. While no work has been found to investigate the exact problem studied in this paper, there is work related to this field of study.

There have been multiple studies related to creating software architectures for multiple devices, but not in the context of notifications or primary device selection. Chmielewski and Walczak [1] researched software middleware for intelligently adapting software based on context and device. Pierce and Nichols [2] tackled the issue of multiple devices belonging to one person by elevating ownership to be a first class property. They extend Extensible Messaging and Presence Protocol (XMPP) [3] to allow services between devices for synchronizing and sharing information.

The level of annoyance and interruption has been studied for single devices and for groups, but not in the

context of multiple devices under a single owner. In Wilson and Miller [4], the HCI aspect of notifications was studied. They looked at how to decrease the interruption factor of notifications when they are displayed to the user by introducing gradual-awareness notifications. Bani-Salameh et al. [5] looked at decreasing interruptions in the context of social software development.

Commercial systems exist already, used by most smartphones today. Android devices use Google Cloud Messaging (GCM) [6] to maintain a connection to servers run by Google in order to receive messages from servers belonging to individual applications. However, until recently, GCM had only provided ways to send to a list of devices, with no way of grouping devices by user. Recently they have started what they call, ''user notifications,'' which aims to allow grouping of a set of devices owned by a user. However, presently, this feature only aims to aid in the dismissal of shared notifications and does not suggest a primary device.

Apple created the Apple Push Notification Service (APNS) [7] for their iOS and Mac OS X product lines. Much like GCM, APNS simply provides an internet-based connection point in order to send a message to a device. Any grouping must be done at the application-level.

Flores and Srirama [8] showed how these commercial systems could be replaced by a pure XMPP-based protocol, useful for inter-operability between the major smartphone operating systems, but does not look at any multi-device intricacies related to messaging.

Along the same lines, different applications have developed different ways of handling the idea of multiple clients. This comes up most frequently in chat applications. Some applications, such as Kakao Talk, use a combination of last device used, but default to the phone client if the computer has been idle for more than a user-specified amount of time or the screensaver activates. Other applications, such as Google Hangouts, from empirical investigation, appear to notify the device that last used the application, broadcasting to all devices after a certain time-out period. These algorithms may choose the wrong device, or unnecessarily delay notifying the user. Protocols such as XMPP allow devices to set a priority, routing messages to the device with the highest priority. However, priorities are self-reported and have no insight into the overall context of the user. Thus a better primary device selection algorithm is important.

Finally, for other parts of our algorithm, Ravi et al. [9] looked at different ways of classifying accelerometer data into different activities such as walking, running as well as interesting features to use for such classifications. Kwapisz et al. [10] worked on activity recognition using the accelerometer in cellphones. Additionally, Gellersen et al. [11] looked at fusing sensors to gain a better understanding of a user's context for smart artifacts. Patel et al. [12] did an important study on how far away the average person's cellphone is from them at any given moment, allowing us to create the movement correlation technique we present in Section III.

III. METHOD

Suppose that a user has three different devices on which they consume information and receive notifications on. The first device is a smartphone, the second device is a tablet computer and the third is a desktop or laptop computer. When a user receives a new email, one or more of the devices try to notify the user of this fact. This act of notification typically involves a sound and/or vibration in the case of a smartphone or tablet as well as a visual pop-up on all three devices. This simultaneous ringing is not optimal for several reasons. First, there is no guarantee that the user is near any device other than possibly the smartphone. Second, if the user is near more than one of the devices, the level of annoyance perceived by the notification increases. Instead, it would be better to notify the user on only one of the devices, falling back to a different device in the case that they do not respond to the notification.

The problem then arises as to which device should be used to notify the user first. This is called the primary device. The primary device is the device that is best used to get the user's attention. This device changes by situation, and thus one device cannot be labelled as the primary device without taking context into account. For example, while using the computer, it makes more sense to notify the user of new email on the computer since they are both using it and it is a more comfortable device to use compared to a smartphone, despite both being, possibly, within reach of the user. These types of situations must be accounted for in any algorithm that tries to determine the primary device to notify the user on.

The algorithm proposed in this paper takes these things into account by fusing activity and movement information from all of the devices into a model to predict which device should be the primary device. This primary device is selected as each notification comes in, allowing the primary device to change in any given situation.

This section is broken up into subsections. Each subsection will go into detail about the different pieces of

Table 1. Device status values and device categories

Status	Category	Description
OFFLINE	All devices	Device is not connected/online.
IDLE	Computers	The device is not being used.
LOCKED	Computers	The device's screensaver is active.
STATIONARY	Phones, Tablets	The device is idle and not moving.
STASHED	Phones, Tablets	The device is idle and on the body of the user.
MOVING	Phones, Tablets	The device is idle and being carried.
ACTIVE	All Devices	The device is being used. The screen is on.

the proposed algorithm. Afterwards, these pieces will be brought together to form the outline of the proposed algorithm. Finally, several sections will discuss the implementation details of the system used to test the algorithm.

1. Device Status and Ranking

Our algorithm is heavily dependent on the self-reported status history of each device. The devices report time-stamped status changes. Between status updates, it is assumed that the devices maintain the same status. If a device unexpectedly goes offline, this is automatically recorded by the server -- as a device which suddenly goes offline is unable to report its status as offline.

Depending on the type of device, the possible status values differ. For example, computers give more coarse-grained status values that smartphones and tablets.

For an overview of the different types of statuses that are recorded for a device's history, see Table 1. Status values are broadly categorized as ''active,'' ''idle'' or "unavailable." Active statuses typically mean the device in question is being used. For smartphones and tablets, this occurs when the screen is on. In the case of computers, an ''active'' status is when there is active input to the computer such as with the mouse or keyboard. Idle statuses indicate that the machine is online, but is not currently being used. For computers, this means that there has been no activity on any input device for at least one minute. In the case of mobile phones and tablets, which typically contain an accelerometer, the "idle" status can be further subcategorized into ''stationary,'' ''stashed'' and "moving." See Section III.2 on more information as to how these statuses are determined via the accelerometer. An additional status that is limited to computers is the "locked" status. This status value indicates that the screensaver has activated and a password may be needed to resume using the device. This status is currently not used in the main algorithm and is treated as an "idle" status.

Furthermore, in our algorithm, devices are given a "type" attribute that specifies what type of device they represent. Possible values are ''computer,'' ''tablet,'' "phone," and "watch." Each type is given a rating of how convenient the device is to use. For example, a tablet is more convenient to use than a phone due to its bigger screen. In this way we can rank devices. The ranking of devices, in ascending order, is as follows: watches, phones, tablets, and finally computers.

2. Movement Determination from an Accelerometer

For devices that contain an accelerometer such as smartphones and tablets, more accurate idle information can be obtained. This enhanced idle information can be of great use when choosing a primary device. The three subcategories of an "idle" status that pertain to phones and tablets are ''stationary,'' ''stashed,'' and ''moving.'' "Stationary" means that a phone is merely sitting on a table or someplace off the person of the user and is not moving. It gives no indication of its proximity to the user, merely that it is not moving. "Stashed" means that the phone or tablet is being held by the user, possibly in their pocket, but the user is not moving. This can happen if the person is sitting with the phone in their pocket. "Moving" implies that the phone is both on the user and the user is actively moving, such as walking.

Figure 1 shows some example accelerometer data collected from a Samsung Galaxy Nexus smartphone for each of the three statuses. The three data sets represents the acceleration in m/s^2 along the x, y and z axes. The data was collected at a rate of approximately 62 Hz. In Figure 1, box A represents data when the phone was stationary. Box B represents data while the phone was stashed. Finally, Box C represents data form walking. As can be seen from the graphs, the difference between

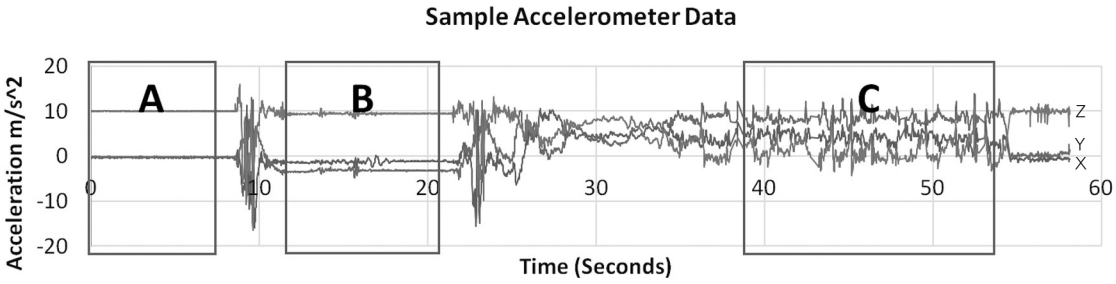

Figure 1. Sample Accelerometer data. Box A: stationary. Box B: stashed, Box C: walking/moving

"stationary" and "stashed" is minimal, yet it exists and can be detected. Furthermore, there exists a large difference in magnitude between "stashed" and "moving." Thus, a classifier was written to distinguish between these cases.

Loosely based on [10] and [9], a minimum of 50 Hz sampling rate was requested. Due to the Android operating system, an exact sampling rate is not guaranteed, and a 62 Hz sampling rate was typically achieved. However, 62 Hz is above the minimum of the 50 Hz required. A window of 256 samples were gathered, representing approximately four seconds of movement. This window was then evaluated for movement classification. Every 256 sample window of data overlapped the previous window by 128 samples. Each window is classified as to which status it should take between "stationary," "stashed," and "moving." If two consecutive windows classify to the same movement state, the device is reported to have that status. Data is collected and evaluated until a decision can be reached. Thus, after at least six seconds, approximately, a new status can be confirmed. In the future this delay could be shortened. However, a tradeoff occurs between false positives and the time to confirm a status. For example, given an average pace of 120 steps per minute, there would be approximately two steps per second. If a user has a cellphone in their pocket, then this means we record the major movement of one leg only, thus reducing it to one step per second. Thus, more than one step would be needed to confirm that the user is indeed moving and not simply shifting their body. Therefore, we chose a longer period to collect data and confirm our classifications. This period has been confirmed to be sufficient according to [9], who also used an overlapping window.

The classification between the different status values is accomplished by a simple linear classifier based on the standard deviation of the given sample window. For standard deviation values in the range of 0 to 0.085 m/s^2, the device was considered "stationary". For values between 0.85 m/s^2 and 1.5 m/s^2, the device was considered "stashed". Anything above was classified as "moving." These threshold values were determined empirically. The classifier required that two axes' standard deviations fell into the same range before a determination was made on the entire sample.

We determined these values by analyzing the collected accelerometer data. After selecting thresholds, we then verified these thresholds were sufficient through experimentation.

In this way, the accelerometer of the mobile devices are able to give valuable information about their idle status.

3. Special Situations to Consider

There are a few cases which must be considered when developing an algorithm for selecting a primary device. These cases include: One device active, more than one device active, and no devices active. Each of these cases are discussed in turn.

First, the case where only one device is active should be the easiest to consider. The active device represents the current device the user is interacting with, and thus the notification in theory should go to this device. As will be discussed in Section V, this may not be a valid assumption, but is the assumption used in our algorithm for the purposes of this paper. Furthermore, issues of multiple people using the same device were ignored in this paper.

The next case is when more than one device is active at the same time. In this case a decision must be made between the active devices. In this case, one can take the type of each device into consideration. Types are ranked according to ease of use. Thus, in the case that more than one device is active at once, a simple and effective approach ranks the active devices according to type and picks the device that is the most convenient to use among

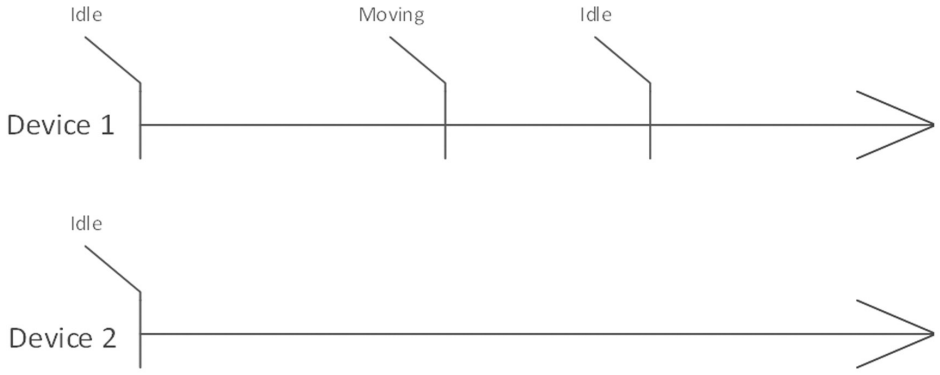

Figure 2. A timeline of two idle devices, one of which moves. Time flows in the direction of the arrow.

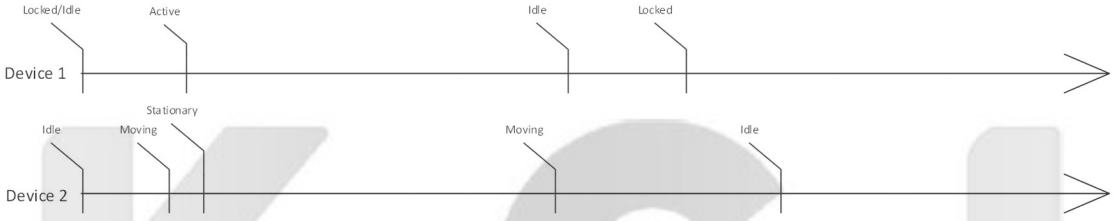

Figure 3. A timeline depicting two devices. Device 1 is used while Device 2 lays idle next to it. Later, Device 2 is taken and leaves Device 1 alone.

the active devices to be the primary device.

The final case is when all devices are idle. The simple approach is to say that the primary device is the most recently used device. This fails under two scenarios. The first case is when two or more devices were used at approximately the same time, but the user leaves, carrying only one of the devices. However, because both devices are idle, a pure comparison of idle time is not appropriate. Movement information, however, can help make a determination. This scenario is depicted in Figure 2. As one follows the timeline across, the figure depicts when the devices have the given labeled status. Following the devices until the end of the depicted timeline, clearly the device that has moved is a better device to send to, since it has traveled with the user. Thus, a device that is moving can be considered less idle than a device that is stationary.

When calculating idle time, it is useful to calculate a weighted idle time. Each "idle" status value is given a weight that scales down the effective idle time of the device. A "moving" status has a weight of 0.5, while a "stashed" idle status has a weight of 0.75. "Stationary," "idle," and "locked" statuses all have weights of 1.0. When the idle time of a device is calculated, it is calculated from its status history from the last time it was "active." The skewed idle time is calculated as such:

$$time = \sum_{i=2}^{n} W_{s_i}(t_{i-1} - t_i)$$

Where i is the current status out of n status values since it was active, each with timestamp t, and status s, and each status has a weight W_{s_i}. In this way, a device that has been moving has a lower idle time than a device that has been stationary for the same period of time. Thus, the primary device is the least idle device, given this skewed idle time. Going back to Figure 2, this would correctly choose the primary device as the device that the user took with them.

For the purposes of our algorithm, the exact values of the weights is mostly insignificant. The important property is that the weights increase with the increased "idle-ness" of the device. Thus, as long as the weight for

"Idle" is greater than that of "Stashed," our algorithm works.

The second case is when a user leaves the active device, taking another device with them. This scenario is depicted in Figure 3. It is important to note that in this scenario, the user has not activated the device they take with them, and thus a pure comparison of idle times, even skewed idle times, would select the primary device as the device the user abandoned. However, yet again, the introduction of movement information helps guide the primary device selection algorithm

Consider again the scenario in Figure 3. First, when the user goes to initially use the computer, there is a short amount of time between when the computer becomes active and the phone becomes stationary or stashed. In this case, we can imagine the user has sat down at the computer and placed their phone either in their pocket or next to the computer. At this moment, we can consider the two devices co-located due to the correlation between status updates. Additionally, at the end of Figure 3, when the user leaves the computer to go somewhere else, the same, yet reversed, situation can be seen. Within minutes of the computer updating its status to "idle," the phone updates its status to "moving." In this way, we can consider the computer and phone no longer co-located. The phone has effectively taken over as the primary device.

Our algorithm actively looks for this correlation between devices when selecting a primary device. In the case where it finds a mobile device that has moved since another device has gone idle within a three minute window, it considers the mobile device's idle time to be the skewed idle time since the previously active device went idle. Also, a constant amount of time is subtracted from the moving device's idle time in order to negate the order in which the active device and moving device and moved and became idle. In this way, the mobile device effectively takes control and becomes the primary device for any future notifications, baring no other device becomes active.

4. Algorithm for Selecting the Primary Device

The algorithm for selecting the primary device is a three-stage algorithm based on the status history of each device.

When a notification is received that is bound for the primary device, the algorithm first collects all the devices which are online. That is, they have a status other than "offline." If there are no devices online at that time, the message is stored and delivered to the first device that comes online and no further processing is done.

Second, the algorithm finds all the devices that are active. Among these devices, it finds the devices that ranks the highest and selects that device as the primary device and does no further processing.

Third, the algorithm computes the skewed idle time of each device. For every device that can be considered mobile (phones and tablets), a simple search is done to correlate them with the other non-mobile devices. If such a correlation is found, their skewed idle time is adjusted and the algorithm continues. Finally, the algorithm selects the device with the lowest skewed and adjusted idle time as the primary device. The notification is then sent to the primary device that was selected.

5. Implementation of a Push-Notification Service

The algorithm described in Section III.4 was embedded within a simplified custom push-notification service. This service, much like the existing commercial offerings by Google and Apple, allows all the devices to maintain an open connection to a server. For a graphical overview of the architecture, see the left side of Figure 4. At the heart of the system is the MQTT [13] protocol. This is a TCP/IP based publish-subscribe protocol the other major piece of the architecture is the "brain" of the push-notification system. This system is responsible for subscribing to all of the status information topics and fanning out any notifications to the appropriate devices. The next subsection will cover this piece more extensively. Finally, there are the various devices, which are connected to the MQTT server and listen and send status information to the MQTT server. For our implementation, we used the open-source mosquito [14] MQTT server running on a virtual private server in the cloud.

Within MQTT, data is published to topics. Topic names form a tree structure. To receive data, one registers to these topics. MQTT has a wildcard syntax to allow one to register to more than one topic at a time. Within this system, each device has its own "sub tree" of the topic tree to receive and send messages on. For example, status updates are published under this tree. Any notification or message that needs to be sent to the device is published to a topic under this tree. Additionally, the topic tree has a few "virtual" topic branches. These topics are topics that only the brain is subscribed to. The brain is then responsible for fanning out the received messages to the

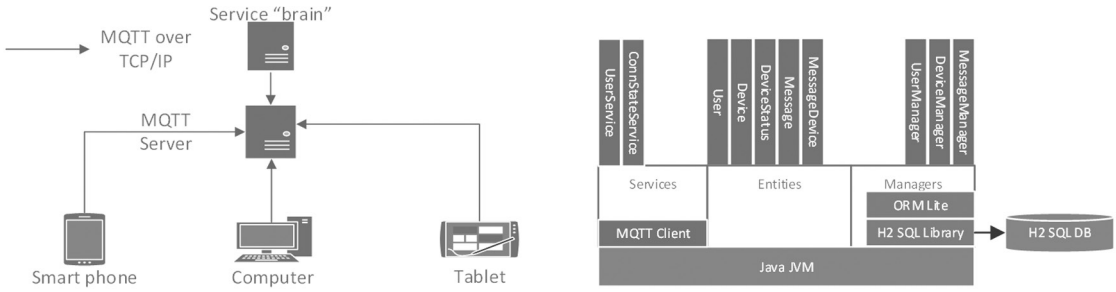

Figure 4. (Left) An overview of the network architecture.
(Right) An overview of the software architecture of the brain of the push service.

appropriate topics. For example, there is a virtual topic branch of all of the users. When data is published to any of these topics, the brain selects the primary device and re-publishes the data on the appropriate topic under the device's tree, which the device will then receive.

Everything within our system has a unique identifier. Devices, users and messages are all assigned a 128-bit random Universally Unique Identifier (UUID) [15].

All data published is a valid UTF-8 encoded JSON [17] object. Invalid data is silently ignored.

5.1. Brain

The brain of the notification system is implemented as a client of the MQTT server. See the right side of Figure 4 for an overview of the system architecture of the server. Working from the bottom up, the brain uses an H2 [18] SQL Relational Database to store all of the pertinent data of the system. This includes all of the devices registered, the devices' status histories, which user a device belongs to, queued messages and information about which device has received which message, or whether the message still needs to be sent to a device. Within the brain, a service-based architecture was implemented. Each service is responsible for defining which topics within the MQTT topic tree they need to be subscribed to. Thus, when data is published to any of the interested topic trees, the data is routed to the appropriate service. Services can handle data on their subscriptions either synchronously or asynchronously.

Within the brain there are two main services: the UserService and the ConnStateService. The UserService is responsible for the user virtual topic tree. This is where our proposed algorithm lives. Since the user virtual topic tree is required to fan out to an appropriate device, when a message is received to send to a user, our algorithm selects the appropriate primary device and forwards the message. The ConnStateService is responsible for recording the time-stamped status updates sent by the devices. It is also responsible for passing along any messages that were queued while the device was offline.

Along with these services that are primarily outward-facing. There are a number of "managers" that deal with managing the different entities. For example, there is a UserManager that aids in retrieving and updating users.

In between services and managers are the basic objects that organize the system. These objects mainly represent entities in the database.

5.2. Android Client

The Android client supports Android phones and tablets using Android version 4.3 and higher. It is implemented as a persistent service with hardly any UI elements to it. The UI elements that exist were created for the user study described in Section IV. The main service of the application keeps the MQTT connection alive. It displays notifications based on the messages that it receives and updates its status.

A secondary service within the application keeps track of the activity of the device. Every 30 seconds, the phone starts querying the accelerometer according to Section III.2 in order to update the device's idle status. For newer devices that support a sudden motion sensor, a sudden movement trigger also starts a reading of the accelerometer, as this would give a tighter bound on when the user actually started moving. Any change in status is then reported on the phone's status topic.

5.3. Desktop Client

A desktop client was written as a Google Chrome

application. Being a Google Chrome application, the client was written in Javascript. The client relies on Google Chrome to report the computer's idle status. The decision was made to implement this as a Google Chrome application in an effort to make the UIs identical for the user study, as Google Chrome recently gained the ability to receive GCM [6] messages in the browser.

IV. EXPERIMENTAL SETUP

In order to test the algorithm defined in the previous section, a user-study was completed. The purpose of the user-study was to gain two metrics. First, that the algorithm picks a primary device that matches the user's expectations. Second, that, compared to a naive approach, our algorithm was subjectively less bothersome or annoying to the user.

To approach this, an identical set of applications for the desktop and Android were created using the existing GCM framework. Additionally, a website was created to allow "fake" messages to be sent through either system, taking the form of an email or chat notification containing a title and an optional message.

The applications for both systems were installed on three devices: a Google Galaxy Nexus smartphone, a Google Nexus 5 smartphone and a laptop computer. The Google Nexus 5 has a bigger screen when compared to the Google Galaxy Nexus smartphone and was thus labeled as a tablet instead of a phone for the purposes of testing.

A total of seven scenarios were created that modeled a typical day of an office worker or graduate student. These scenarios are explained below:

① The user has just woken up in the morning via their cellphone's alarm. They have a tablet and phone available, their computer remaining off for the time being. While getting ready for the day, they receive an urgent email from the office. For this scenario, the notification is this email.
② Upon the receipt of the email in scenario 1, they decide to reply to it via their tablet, as their computer remains off and the tablet is a more convenient interface to reply on. While replying to the email, another email arrives, notifying them again.
③ Having finished replying, they go to the office and start working on their computer in the office. Now there are a total of three possible devices. An email arrives while working.
④ While they are working, they receive word of a new tablet game. They momentarily pause their work to download the game on their tablet. While using the tablet, they receive an email.
⑤ It is break-time at the office. The user decides to get coffee. They leave their computer, taking their phone with them to go purchase coffee at a nearby cafe. While purchasing the coffee an email arrives.
⑥ The user has finished working and is at home. The user is relaxing and playing a computer game. While playing the computer game an email arrives.
⑦ Finished with their game, they go out to a local convenience store to grab a late-night snack, taking their phone with them. While walking the last email arrives.

For each scenario, two notifications were sent, totaling 14 notifications total. The first notification was sent via GCM and simulated the naïve approach of sending to every applicable device. The second notification was sent via our system.

For each notification, the user was asked two questions:

① For the given scenario, if a notification were sent, which device would be the ideal device to receive the notification on?
② Given the notification that was just received, on a scale of 1 to 10, how bothersome or annoying was the notification?

The above scenarios were each chosen to test a specific portion of our algorithm and judge its effectiveness in different situations. Scenario 1 aims to test the basic idle time-based selection, and confirm that a user in fact does want it sent to the phone. Scenario 2 and 3 tests basic selection when 1 device is active. However, scenario 3 differs in that the possible devices has increased, as well as where the user is. Scenario 4 tests the case when 2 devices are active at the same time. Scenarios 5 and 7 test device correlation from movement. Scenario 6 is identical to scenario 3, but the circumstances have changed in that the user is much more engrossed in their current activity than in scenario 3.

V. RESULTS AND DISCUSSION

1. Results

A total of 26 participants were tested. The participants

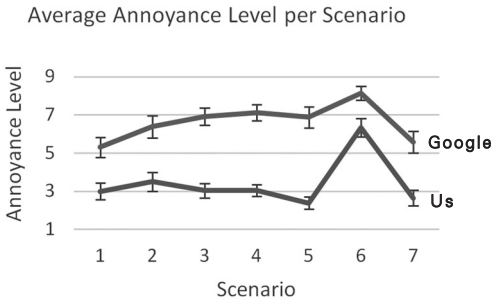

Figure 5. A breakdown of annoyance levels by scenario with bars representing the standard error

Table 2. Algorithm Accuracy

Scenario	Algorithm's Choice	No. Users prefer the Computer (%)	No. Users prefer the Tablet (%)	No. Users prefer the Phone (%)	Algorithm Accuracy
1	Phone	2 (7.7%)	1 (3.9%)	23 (88.5%)	88.5%
2	Tablet	2 (7.7%)	16 (61.5%)	8 (30.8%)	61.5%
3	Computer	22 (84.6%)	0 (0%)	4 (15.4%)	84.6%
4	Computer/Tablet*	4 (15.4%)	18 (63.2%)	4 (15.4%)	42.3%
5	Phone	0 (0%)	0 (0%)	26 (100%)	100%
6	Computer	4 (15.4%)	0 (0%)	22 (84.6%)	15.4%
7	Phone	0 (0%)	0 (0%)	26 (100%)	100%

ranged in age from 23 to 57 years, with an average age of 28±6.8 years. All participants were informed about the experiment and what questions would be asked of them, confirming they understood. In the case of a language barrier, an assistant was present to translate. Throughout the test, many users gave valuable feedback as they selected their answers. These answers were invaluable to gain insight as to why some results were lower or higher than expected.

Annoyance levels for our system were lower than the naïve system. On average the naïve system garnered a score of 6.62±2.65 with a standard error of 0.015. Contrasting with our system, which had an average score of 3.42±2.38 with a standard error of 0.013. Figure 5 shows a graph of the annoyance levels by scenario.

As can be seen from the graph, for each scenario, our system was less bothersome to the user than the naïve approach. There is an interesting increase for scenario 6 which will be discussed in the next section.

See Table 2 for an overview of the results of our system for predicting the same primary device as the user specified. As can be seen from Table 2, the success of our algorithm varies from 15% accuracy to 100% accuracy with a total average of 70.3%. The discussion next section will cover some reasons for this wild variance in accuracy. In Table 2, scenario 4 uncovered a bug that was not caught during testing, and thus the result changed for the remainder of the tests.

2. Discussion

On average the users felt less annoyed with our system than the naïve one. Scenario 6 was the exception to this due to the user's perceived concentration level. Generally, when a user is playing a game, they are concentrating much harder on their current activity compared to general office work. Thus any notification is a bigger distraction and more annoying to the user. In this regard, the algorithm should become more aware of the context of the user as they use a computing device. That is, the algorithm should assess the state of activity on the computer. If it indicates that a high-concentration or full-screen activity is happening, a different primary device should be selected, or the notification should be postponed until the user is in a better state to receive it, provided the notification is not urgent.

In scenarios 5 and 7, the user only had their phone with them, and thus would not hear any other device that

may have been notified at the same time. Thus, it is interesting to note that the annoyance level with the naïve approach is higher for scenario 5 than 7. Recall that in scenario 5, the user had left the office while in scenario 7 the user is at home. Users said they were generally worried about privacy. Even though they could not hear the notification being sent to the computer and tablet, they assumed it would be sent, and thus worried about privacy. They were worried about other people around their computers becoming curious as to the noise and looking. However, at home they feel safer and thus were less concerned with other people noticing.

The more ripe area for discussion is in the algorithm's primary device prediction percentages. From Table 2 we can see that the percentages ranges wildly from 15% to 100%. In an ideal world, this would not be acceptable. The reasons for this lie in the assumptions of the algorithm as well as the variability of people's preferences. For example, there were two subjects that preferred their phone to be the primary device no matter what scenario they were in.

In scenario 1, the algorithm was mostly correct. There were a few users that would have preferred another device, even if that device wasn't available at the moment. The reason for this stems from the fact that certain devices are preferred for different communications. Even though more accessible devices were available, email was seen as a computer-related communication. However, usually users preferred to use their phones for chatting.

Scenario 2 was slightly worse that scenario 1, coming in at about 61% accuracy. Users were generally conflicted between receiving notifications on their tablets vs their phones, even if they were using the tablet at the time. In this case the algorithm was correct to select the tablet as the primary device, but it's important to bear in mind that user preference should be taken into account in the future.

Scenario 4 was a case where the algorithm misjudged what people actually wanted. Even if a computer and tablet are both active, the user would prefer to receive a notification on a less comfortable device. This may be due where their concentration is at the time. The primary device should be where the user is concentrating, but only if they are not concentrating too hard.

Scenarios 5 and 7 were the most straightforward, due to the user only having one device with them at a time. This gives credence to the necessity to take movement and colocation into consideration when selecting a primary device.

However, it should be noted that scenarios 5 and 7 have the highest probability of destroying the trust a user has in the system. If the system guesses incorrectly and sends to the wrong device, especially in these scenarios, a user could possibly miss a notification. Thus, a secondary device should also be selected, or all devices should be broadcasted to. In the worst case, we would be no worse than the naïve approach. While cascading was outside the immediate scope of this particular study, it is an important feature for any system put into production in order to retain the trust of the user.

Finally, scenario 6 was the worst of the accuracy rates, being only 15%. As discussed before, when users are concentrating hard, such as when playing a game, they generally dislike being interrupted. However, should they have to be interrupted, they want the primary device to be the device they are not concentrating on. That is, any notifications that appear should not detract from their current activity. Thus, again, more context of the computer is needed to be gathered to select a secondary primary device. Further correlation should be considered between co-located devices in order to select the appropriate secondary device.

Overall, the level of annoyance was lower than a naïve approach, giving credence to the importance of picking a primary device. However, the algorithm as it stands is less than perfect as it picks a primary device that deviates from people's line of thought. However, even if the algorithm picks the wrong device, it never picked a device that was not near the user. That is, the user never missed a notification, even if that notification was sent to the less than ideal device. This is in contrast to other systems which may delay notifications due to their cascading between possible devices.

Even with the above results looking promising, more information may be needed to prove that this system would be ideal outside of planned scenarios. In a planned scenario, a user may decide that the ideal device is their computer, but in real life realize a phone is the ideal interface. This is a limitation of the type of user study done. Thus a longer user study may be needed to verify the validity of the algorithm over a longer period of time.

Additionally, it could be noted that by testing the system against a simplistic broadcast notification, the annoyance levels were lower only because less devices were notified at once. If a random device were picked as the primary device, the annoyance levels could be similar. While certainly plausible, it also gives credence to the necessity of picking a primary device, even if the device is chosen at random. Further study may be needed to confirm this. If true, the second part of the study becomes more important. If less devices should be notified, then selecting

the right device becomes more important.

In conclusion, while the user study completed shows that the initial results are promising, further user studies are needed to address further dimensions of the problem.

VI. CONCLUSION

This paper presents an algorithm to select a primary device out of a group of devices belonging to a user. This algorithm takes into account activity status, idle time and movement, fusing it together to make the best decision for a given situation. This algorithm was tested within a custom push-notification service implementation using MQTT for the publish-subscribe protocol. Clients were created for Android and the computer. A user study was completed to test the validity of both primary device selection and which devices our algorithm selects as the primary device. Results were mixed, with annoyance levels decreasing, but primary device selection guiding future work.

Future work includes changing the algorithm to take into account personal preferences as well as enhancing the resolution of activity statuses on devices. Furthermore, greater correlation should be attempted to detect co-located devices in order to guide secondary device selection, should the primary device be a poor choice due to activity levels. Additionally, secondary devices should be taken into account in case the user has not acknowledged the notification in order to ensure the user can continue to trust the system. Notification priorities should be taken into account and studied further. Notifications which are urgent should be dealt with in a different manner than normal notifications. For example, high priority notifications might be broadcast to every device instead of only the primary device in order to negate any chance that the notification is missed. Also, an additional user study should be performed over a longer period of time to ensure that the algorithm in this paper is useful outside of pre-planned scenarios.

〔References〕

[1] Google, Inc, "Google Cloud Messaging for Android," [Online]. Available: https://developer.android.com /google/gcm/index.html.

[2] "Apple Push Notification Service," Nov. 2014. [Online]. Available: https://developer.apple.com/library /mac/DOCUMENTATION/NetworkingInternet /Conceptual/RemoteNotificationsPG/Chapters /ApplePushService.html.

[3] Jacek Chmielewski and Krzysztof Walczak, "Application Architectures for Smart Multi-device Applications," in *Proceedings of the Workshop on Multi-device App Middleware*, 2012. pp. 1-5.

[4] Jeffrey Pierce and Jeffrey Nichols, "An Infrastructure for Extending Applications' User Experiences Across Multiple Personal Devices," in *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, 2008. pp. 101-110.

[5] Peter Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP)," Internet Engineering Task Force (IETF), 2011. [Online]. Available: http://tools.ietf.org/html/rfc6122

[6] Thomas Wilson and Robert Miller, "Reducing the cost of interruption using gradual awareness notifications," [Online]. Available: http://groups.csail.mit.edu/uid/projects/slowgrowth/ gradual-Awareness.pdf.

[7] H. Bani-Salameh and C. Jeffery, "Notifications Management in Distributed Development Environments: a Case Study," in *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, May, 2014. pp. 49-55.

[8] Huber Flores and Satish Srirama, "Mobile Cloud Messaging Supported by XMPP Primitives," in *Proceeding of the Fourth ACM Workshop on Mobile Cloud Computing and Services*, 2013. pp. 17-24.

[9] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore and Michael L. Littman, "Activity recognition from accelerometer data," in *AAAI*, 2005. pp. 1541-1546.

[10] Jennifer R. Kwapisz, Gary M. Weiss and Samuel A. Moore, "Activity Recognition Using Cell Phone Accelerometers," in *SIGKDD Explor. Newsl.*, Mar., 2011. pp. 74-82.

[11] Hans. W. Gellersen, Albercht. Schmidt and Michael. Beigl, "Multi-sensor Context-awareness in Mobile Devices and Smart Artifacts," in *Mob. Netw. Appl.*, Vol. 7, No. 5, Oct., 2002. pp. 341-351.

[12] Shwetak N. Patel, Julie A. Kientz, Gillian R. Hayes, Sooraj Bhat and Gregory D. Abowd, "Farther Than You May Think: An Empirical Investigation of the Proximity of Users to Their Mobile Phones," in *UbiComp 2006*: *Ubiquitous Computing*, Vol. 4206. 2006. pp. 123-140.

[13] International Business Machines Corporation (IBM),

"MQTT V3.1 Protocol Specification," Nov. 2014. [Online]. Available: http://public.dhe.ibm.com /software/dw/webservices/ws-mqtt/mqtt-v3r1.html.
[14] Roger Light, November 2014. [Online]. Available: http://mosquitto.org.
[15] P. Leach, M. Mealling and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," Internet Engineering Task Force (IETF), 2005. [Online] Available: http://tools.ietf.org/html/rfc4122
[16] S. Josefsson, "The Base16, Base32, and Base64 Data Encodings," Internet Engineering Task Force (IETF), 2006. [Online] Available: http://tools.ietf.org/html/rfc4648
[17] ECMA International, "The JSON Data Interchange Format," October, 2013. [Online]. Available: http://www.ecma-international.org/publications/ standards/Ecma-404.htm.
[18] Thomas Mueller, "H2 Database Engine," 2014. [Online]. Available: http://ww.h2database.com.
[19] Gray Watson, "OrmLite - Lightweight Object Relational Mapping (ORM) Java Package," [Online]. Available: http://ormlite.com/

Theodore F. Morse

He received the B.S. degree in Computer Science from University of Evansville, USA, in 2007 and the M.S. degree in Computer Science from KAIST, Korea, in 2015. His research interests include smart device communications and the internet of things.
E-mail: tmorse@gmail.com

Sungho Jo

He received the B.S. degree from the School of Mechanical and Aerospace Engineering, Seoul National University, Korea, in 1999 and the M.S. degree in Mechanical Engineering and the Ph.D. degree in Electrical Engineering and Computer Science from MIT, USA, in 2001 and 2006, respectively. From 2006 to 2007, he was a Postdoctoral Researcher with the MIT Media Lab. Since December 2007, he has been with the School of Computing, KAIST, where he is currently associate professor. His research interests include neural signal-based interfaces, human-robot interaction, and wearable computing.
E-mail: shjo@kaist.ac.kr
Tel:+82-42-350-3540