Sinch: Searching Intelligently on a Mobile Device

by

Rajeev Nayak

S.B., Course VI M.I.T., 2010

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2010

Author .....
Department of Electrical Engineering and Computer Science
August 20, 2010

Certified by .....
Robert C. Miller
Associate Professor
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Sinch: Searching Intelligently on a Mobile Device

by

Rajeev Nayak

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 2010, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Sinch is an application that allows mobile device users to obtain answers to their questions without having to perform a web search in their mobile browser. Questions are answered by human beings using Mechanical Turk, an online labor market for simple human computation tasks. Workers on Mechanical Turk can search for answers using their desktop browsers, free from the numerous shortcomings of browsing the internet on a small mobile device while in a potentially distracting situation. Sinch contributes an URL-rewriting proxy browser that can be embedded in Mechanical Turk tasks to track browsing history, text selections, and other user events. It also introduces two ways to improve the credibility of human-generated answers: providing multiple answers to the same question and including a browsing history with each answer. The Sinch application also allows users to view answers in their original context on a web page, using a custom mobile browser to highlight the answer text on the web page and zoom into it. Evaluations demonstrate that Sinch is capable of delivering correct answers in a timely manner and investigate the effectiveness of providing multiple answers and a browsing history for each answer.

Thesis Supervisor: Robert C. Miller
Title: Associate Professor

Acknowledgments

First of all, I'd like to thank Rob Miller for being an amazing thesis supervisor. Without his insightful ideas and constant guidance, writing this thesis would not have been even close to possible. He was the driving force behind the project from the beginning, and through the entire process I never ceased to be in awe of his vast expertise and innovative mind.

I'd also like to thank Katrina Panovich, Michael Bernstein, Max Goldman, Adam Marcus, David Crowell, Greg Little, Jones Yu, and the rest of the User Interface Design group for helping me brainstorm and providing great feedback about my project. Their advice made me confident that my project was always continuing in the right direction.

A special thanks goes out to Jeff Bigham for his invaluable help with the implementation of my project. He graciously provided his code as an example for me to follow, and he promptly answered all of my annoying questions when I first started implementing and I was floundering with my limited web programming experience. I definitely couldn't have done any of this without him.

Finally, I'd like to thank my parents, Raghuveer and Anita Nayak, and my brother, Vinay, for supporting me through this entire process. My brother was always there to talk about basketball and take my mind off work, my dad provided an endless supply of bad jokes to entertain me, and most importantly, my mom tirelessly listened to my stressful rants and always made me feel better about everything. I can't express how much my family has helped me not only through the process of writing this thesis, but through my entire education.

Contents

List of Figures

Chapter 1

Introduction

There are many inherent problems with web searching on a mobile device. Mobile device browsers do their best to emulate their desktop counterparts, but many properties of mobile devices inevitably make web searching very cumbersome:

1. **Small screen size.** The screen of any mobile device is much smaller than a desktop monitor, which makes it very hard for users to find the information they are searching for. This is especially difficult when the answer to a search query is embedded within a page full of text.

2. **The "fat finger" problem.** Using a finger as a pointing device on such a small screen makes clicking on small targets much tougher. Users may have to navigate through multiple links to find what they are looking for, and this "fat finger" problem hinders that process. In addition, typing on a small keyboard is both slower and less accurate than typing on a normal-sized keyboard.

3. **Poor network connection.** Mobile devices frequently have bad network connections, whether it be 3G or WiFi. Just like the "fat finger" problem, this is also an issue when users have to navigate through many web pages when performing a web search.

4. **Situational disabilities.** People are often walking, driving, or otherwise distracted when they need information in a mobile setting. In these situations,

they cannot focus their full attention on their mobile device for an extended period of time, if at all.

This thesis describes Sinch, a mobile application that addresses these problems with web searching on a mobile device. The Sinch application allows users to type or speak questions that would have otherwise prompted a mobile web search. Sinch uses human computation, utilizing people who can perform web searches on their desktops in order to find correct and concise answers to these questions. These human-generated answers are returned to the mobile device users, eliminating the need for a mobile web search.

Various systems have already been created to deal with these problems with mobile web search. Services like ChaCha [3] allow mobile device users to text them questions. Their employees search for the answers to submitted questions and send the answers back via text. Other mobile search services provide answers using other sources of human computation, such as large online communities [5] or users' social networks [1]. Sinch uses a different approach to providing human-generated answers: Mechanical Turk.

Mechanical Turk is an online labor market developed by Amazon that allows users to pay people small amounts of money to perform simple human computation tasks. These tasks are posted by requesters: people who are seeking information that cannot be easily obtained using just a computer. Tasks are performed by Mechanical Turk workers, or "Turkers". Once a task is completed, the answer provided by the Turker is delivered to the requester and the Turker is paid their compensation. Using these tasks on Mechanical Turk, Turkers can be used to provide answers to mobile search queries for Sinch.

Unlike text messaging services, which are limited to providing short, plain-text answers, Sinch has more flexibility when returning answers. Sinch uses this flexibility to bolster the credibility of answers in two ways:

1. **Multiple answers are provided for each question.** Answers provided by human beings are not always accurate. In addition, questions can be worded

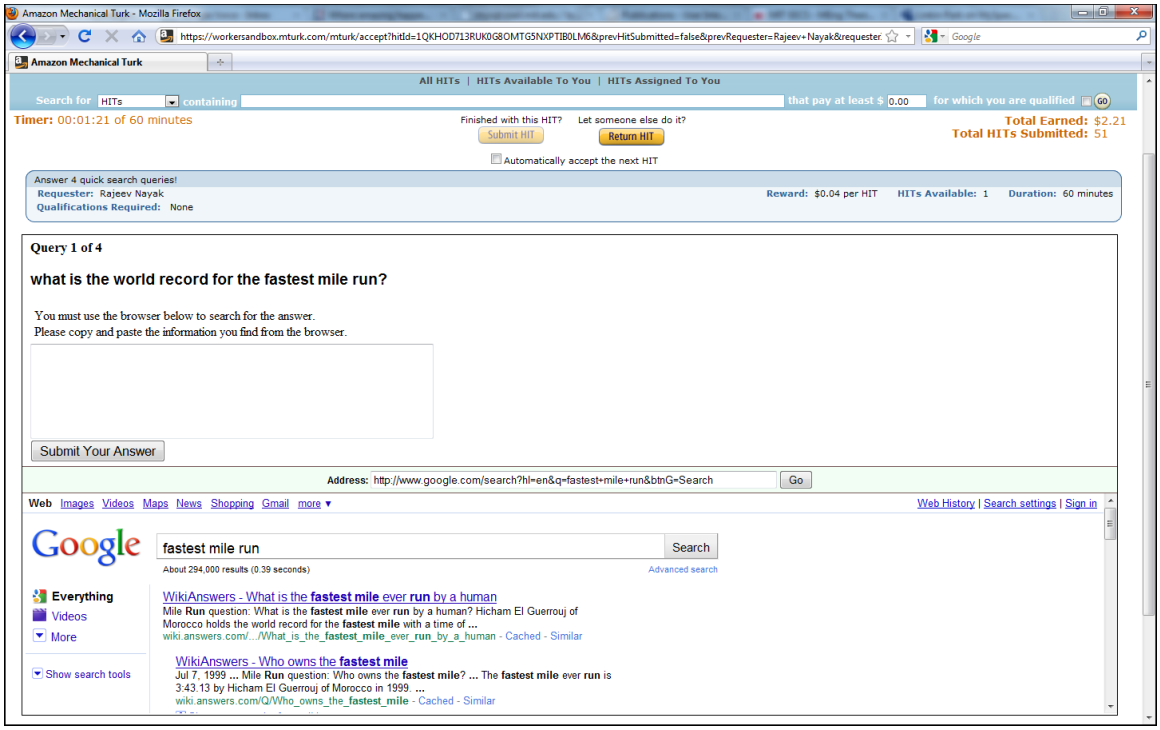

Figure 1-1: Answering a Sinch question in a Mechanical Turk task.

ambiguously and interpreted differently by people providing answers. Sinch provides multiple answers to each question in order to deal with these issues. Returning multiple answers increases the probability of at least one of them being correct. Furthermore, seeing multiple correct answers solidifies users' confidence in those answers.

2. **Users are shown the browsing history that led to each answer.** Viewing an answer provided by a human being in its original context on a web page gives it more credibility. The Sinch Mechanical Turk task includes an inline browser that Turkers are required to use. This browser tracks their browsing history and all of their text selections. Sinch users are shown the entire browsing history that led to each Turker-generated answer so that they can view the answer in its original context. In order to make this even easier, web pages from which the Turker copied and pasted text that contributed to their final answer are indicated. Users can view these pages in a custom mobile browser that highlights the copied and pasted text and zooms into it. This helps users

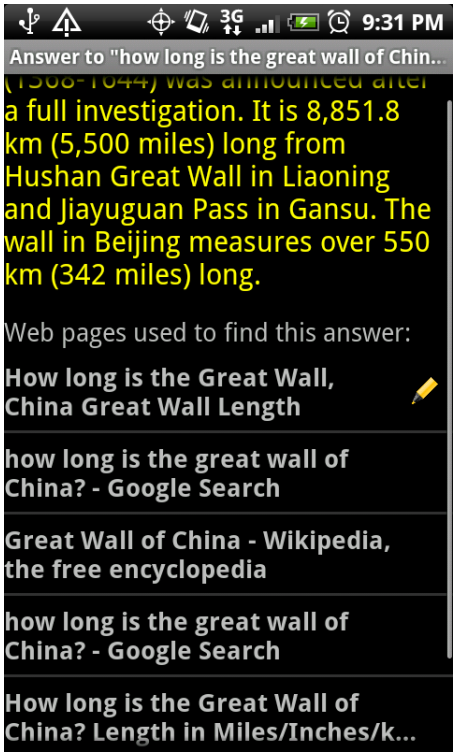

Figure 1-2: A Turker's browsing history displayed in the Sinch mobile application. The web page containing the answer text is indicated with a yellow highlighter icon.

find the answer in context as fast as possible, despite their mobile device's small screen size.

Two critical aspects of the Sinch system are the latency and correctness of answers. Users in mobile settings need answers quickly, but Sinch has an inherent latency in its human computation. The time that it takes Turkers to return an answer should be comparable to the time it would take a user to find that answer on their mobile browser. Furthermore, Turkers should be able to provide correct answers on a consistent basis. Even though multiple answers are provided for each question, the percentage of correct answers must be high enough to guarantee that at least one Turker-generated answer is correct for each question.

Two evaluations were performed in order to test the latency and correctness of the Sinch system. The first evaluation was run before Sinch was implemented. It compared the search times of Turkers to the search times of people using an iPod Touch when answering the same set of questions. This experiment established that Turker latency was acceptably low. The second evaluation was run after the Sinch system was complete. Turkers used the Sinch Mechanical Turk interface to answer questions, and the correctness of their answers was assessed. This evaluation also tested their browsing habits while searching for answers.

A third evaluation was run after the system was implemented. This evaluation tested the effectiveness of providing users with multiple answers and a browsing history for each answer. Users were shown questions and answers generated from the second evaluation. They were asked to determine whether a particular answer to a question was correct or not, and they were paid based on their speed and accuracy. While they assessed each question and answer, they had the option of viewing more answers to the question or viewing the browsing history for any answer. A significant number of users viewed either multiple answers or browsing histories before they were fully confident in their answer to the question.

The fully-implemented Sinch system provides a convenient way for mobile device users to obtain answers to questions without having to use their mobile browser. It contributes an inline browser that can be used in Mechanical Turk tasks to track

browsing history, text selections, and possibly more user events. Finally, it also introduces two ways to provide more credibility to human-generated answers: providing multiple answers to the same question and a browsing history for each answer.

The rest of this thesis details the design, implementation, and evaluation of Sinch. Chapter 2 covers other work related to Sinch. Chapter 3 covers the design of the system and its user interfaces. Chapter 4 covers implementation of both the mobile application and the Mechanical Turk task. Chapter 5 discusses the three-stage evaluation of Sinch. Chapter 6 provides a conclusion and chapter 7 outlines possibilities for future work on the project.

Chapter 2

Related Work

2.1 Question Answering Services

There are many existing services that attempt to provide answers to naturally phrased questions. As opposed to traditional search engines like Google which present users with a page of search results, these services return a concise answer, similar to an answer one might get after posing the same question to a human. Some of these services employ human computation, while others are completely automated.

2.1.1 Human-Powered Services

Most human-powered question answering services provide their answers in one of two ways: using a large online community or using a small set of designated question answerers. A large community brings a vast variety of expertise on different subjects, as well as a free method of generating questions. Designated answerers are more reliable than random individuals in a large online community, but this is because they are typically paid a compensation for answering questions.

Yahoo! Answers [7] is an example of a service that uses the community approach to answering questions. Users can post a question on the Yahoo! Answers website, and other users will post answers to that question. In order to address the reliability issue, the community is allowed to vote on potential answers so that the best answer

for each question will eventually rise to the top. In addition, the question asker can personally select an answer to be marked as the best one. Sinch operates under the same assumptions as Yahoo! Answers; Turkers are not guaranteed to be reliable, so Sinch provides multiple answers to each question as well.

ChaCha [3] is a mobile question answering service that uses designated answerers, called Guides. Guides are hired by ChaCha, and they are paid to answer incoming questions. Questions are asked through text messages, and answers are returned using text messages as well. As opposed to Yahoo! Answers, ChaCha only provides a single answer to each question, because their Guides are trusted to be reliable sources.

2.1.2 Automated Services

One of the oldest automated question answering services is Ask.com, formerly known as Ask Jeeves. Ask.com has always encouraged the use of natural language questions, separating themselves from other search engines. However, just recently they revamped both their front-end and back-end to specifically focus on providing answers to questions posed in natural language [2]. Their new approach involves indexing existing answers on sites like Yahoo! Answers and ChaCha. This works in conjunction with a fallback onto human computation in case of failure. If an answer cannot be automatically generated, the question is posed to the Ask.com online community so that a human can provide an answer.

Wolfram Alpha [6] is a more recent automated question answering service. Although one of its primary goals is mathematical computation, it extends this focus into the question answering domain, allowing free-form input and returning specific answers. Its answers are presented in a unique format in comparison to other search engines. First, it returns a structural diagram of the parsed input question, showing a deconstruction consisting of the keywords that it extracted. Second, it represents the answer with a customized structure as well. For example, if the answer is easily readable in tabular format, it will automatically generate a table and provide extra customization options.

START [11] is a natural language question answering system developed at MIT

CSAIL. Like Wolfram Alpha, it provides a single, structured answer to each given question. However, instead of analyzing the question's structure, START uses keywords in the question to associate it with a question that it already knows the answer to. Since the same information need can be posed as a question in many different ways, START saves only one question for each unique information need. When another formulation of the same question is submitted to the system, it finds that saved question and then fetches the corresponding structured answer to return to the user.

Finally, Google has also recently added question answering features to their search engine [4]. The new functionality is mostly designed to handle trivia questions in particular. The unique feature that Google's question answering brings to the table is the citation of sources. Each answer is accompanied by a list of web pages where it was found, giving it credibility. Similarly, Sinch logs the browsing history of each Turker as they find an answer and presents the list of visited web pages to the mobile device user in addition to the answer.

2.2 Mobile Web Search

Information needs in mobile settings have been previously observed and analyzed. A study conducted by Sohn et al [15] examined the types of questions people need to address while they are away from their computers. They found that some of the most frequent information needs are answers to random trivia questions or locations of nearby points of interest. These question types work well with Sinch, because they can be answered concisely and easily by Turkers. The study also showed that many of these information needs were not urgent and could be answered at a later time, if at all. This also bodes well for Sinch, because users will not be put off by the latency in the system introduced by Turkers while they browse for the correct answer. Finally, over half of the people who decided to find the answer to their query at a later time did so because they were biking, driving, or busy with a task or a meeting. Sinch attempts to alleviate these situational disabilities in two ways. First, it allows users to speak their question into their mobile device, which can easily be done while

biking or driving. Second, users can wait for concise answers rather than searching and browsing on their own, which would require their attention over a period of time.

A previously built system called SearchMobil [14] addresses the problem of a small screen while searching on a mobile device. It utilizes a formatter called SmartView, which parses the HTML of web pages to divide it into logical regions. When a user navigates to a web page in their mobile browser, SmartView presents a fully zoomed-out version of the page with each region outlined. Users can click on a region to zoom into it. In SearchMobil, the search terms are automatically highlighted in the resulting web pages. In addition, each SmartView region is augmented with an annotation of how many search terms were found in it. This is intended to give the user an idea of which region is most relevant to their search. Since Sinch tracks all text that the Turker highlights while they browse, the exact locations of relevance are known. Thus, rather than highlighting all search terms and zooming out, Sinch highlights exactly what the Turker highlighted and zooms into it.

2.3 Collaborative Web Search

SearchTogether [13] is an application developed by Microsoft Research that facilitates collaboration with others while web searching. Users can create search sessions and share them with other users, so that they can either parallelize the searching effort synchronously or individually contribute to the search asynchronously. Each search session focuses on a single question that needs to be answered. As users perform search queries in a session, they are logged and presented to other users so that they are aware of queries that have already been used. When a user finds a relevant web page, they can comment on it and recommend it to the other searchers in that session.

Sinch is similar to a collaborative search client in that information about a search is shared between two people. However, the information flow is only one-way in Sinch. The mobile user should not be searching or browsing on their mobile device; they should just receive information about searches performed by Turkers. Sinch provides the mobile device user with a full browsing history for each answer returned by a

Turker, which encapsulates all of the search queries performed by the Turker, as well as all other web pages that they visited. Instead of letting Turkers manually recommend pages like SearchTogether, Sinch automatically discerns the relevant pages by tracking the Turker's text selections that contribute to their final answer.

2.4 VizWiz and TurKit

VizWiz [10] is an iPhone application that helps blind people find objects in their immediate surroundings. Users take a picture using their iPhone and ask about an object that they want help finding. When the question is submitted, the picture, the audio, and a transcription of the audio are posted in a task on Mechanical Turk. Turkers are presented with an interface that includes the picture sent by the iPhone user, the recorded question, and its text transcription. They enter the answer to the question, and this answer is returned to the iPhone application, which automatically reads it out to the user. Sinch uses the same application structure as VizWiz. It starts with a mobile phone user asking a question on their phone. That question is then sent to a Turker to answer it, and then the answer is returned to the mobile phone user.

VizWiz uses TurKit [12] to create tasks in real time on Mechanical Turk. TurKit is a toolkit that wraps around the Mechanical Turk API, allowing programmers to easily incorporate human computation into complex algorithms. It provides a Javascript library with easy access to functions that create new tasks on Mechanical Turk on demand. Sinch also uses TurKit to post tasks on Mechanical Turk when new queries are submitted through the mobile Sinch application.

The TurKit algorithm used by VizWiz to create new tasks is called quikTurkit [10]. The goal of quikTurkit is to provide nearly real time answers, since the users of VizWiz would ideally want their answers immediately. The quikTurkit algorithm runs in an infinite loop, checking the status of all active questions and all active tasks at each iteration of the loop. Since answers provided by Turkers are unreliable, quikTurkit sets a desired number of answers for each submitted question. The algorithm only

posts tasks if there is at least one question that has not reached the desired number of answers. After a certain period of time, all active tasks are removed and new tasks are posted to replace them. This increases the chances of new Turkers picking up the task, since most Turkers use the default sorting by most recent when browsing for tasks. Other customization options are also available to further improve the chances of recruiting new Turkers as fast as possible.

In addition to these measures taken to quickly recruit Turkers, VizWiz uses another trick to speed up response time. Instead of making Turkers answer one question per task, VizWiz includes a queue of multiple questions per task. Therefore, when a Turker picks up a task, they will have to answer numerous questions in succession before the task is complete. If a Turker is working on a task when a new question is submitted by an iPhone user, that question is automatically pushed to the front of that Turker's queue. As soon as the Turker finishes answering their current question, they will be asked to answer the newly submitted question. This greatly reduces the latency of Turker responses to questions. Sinch similarly demands nearly real time answers to queries, so it employs both quikTurkit and the question queue model to minimize the latency inherent in Mechanical Turk.

Chapter 3

Design

The Sinch system consists of three major user interfaces, reflecting the three-stage lifecycle of a question and its answers. These three stages are depicted in figure 3-1. First, the mobile device user must submit their question to the Sinch system. Next, Turkers must provide answers to the submitted question. Finally, the user must be able to view each of these answers on their mobile device, along with the web pages visited by the Turker who answered it. The mobile Sinch application consists of both the asking and viewing interfaces, while the answering interface is presented to Turkers on their desktops.

3.1 Asking a Question

Asking a question is the simplest aspect of the Sinch system. Therefore, the asking interface was designed for simplicity. Since asking a question is the first step in the Sinch process, the asking interface is the first thing shown upon opening the mobile Sinch application. As shown in figure 3-2, the user sees a text box, a microphone button, and a submit button listed under the "Ask" tab as soon as they open the application.

When a user clicks in the text box, it brings up the standard Android virtual keyboard, allowing them to type in their question. If the user instead clicks on the microphone button, the Android speech input modal dialog box pops up, allowing

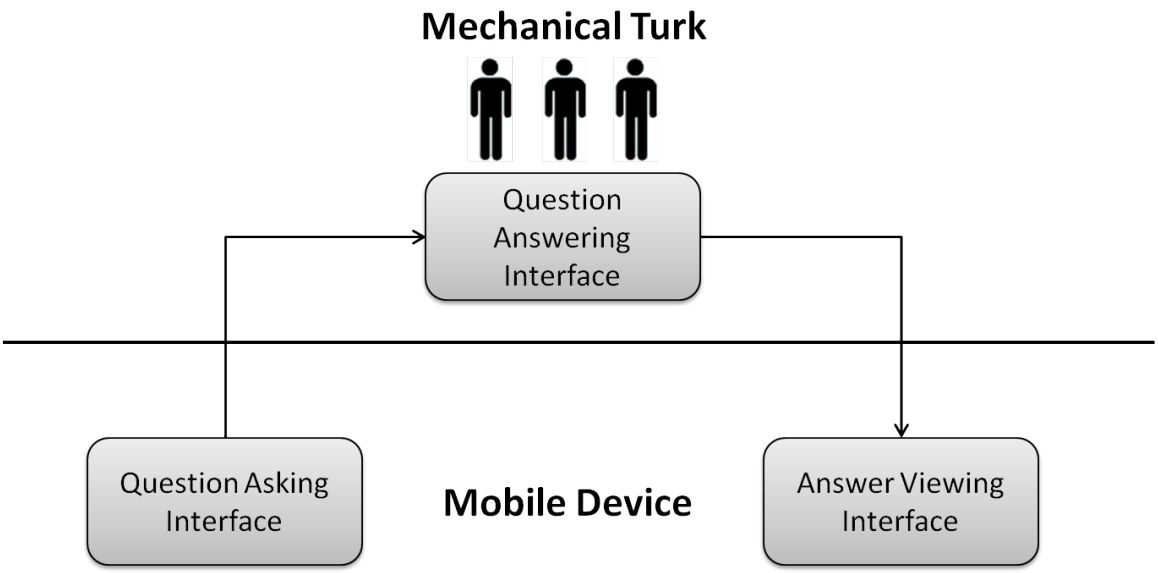

Figure 3-1: The three major user interfaces of the Sinch system: question asking, question answering, and answer viewing.

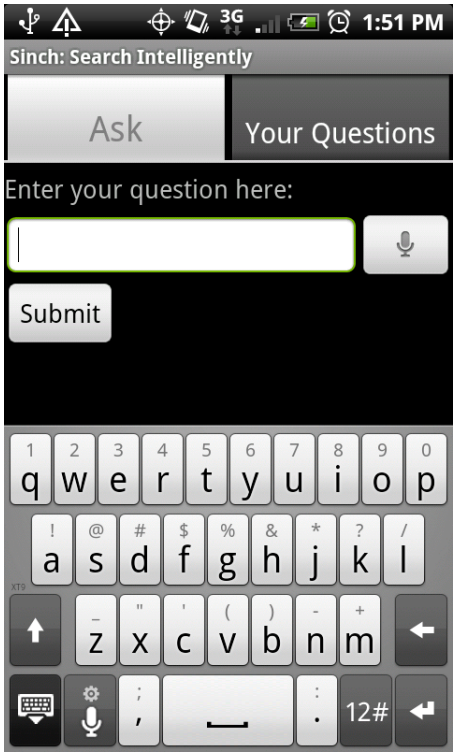

Figure 3-2: The interface for asking a Sinch question with the standard Android keyboard open.

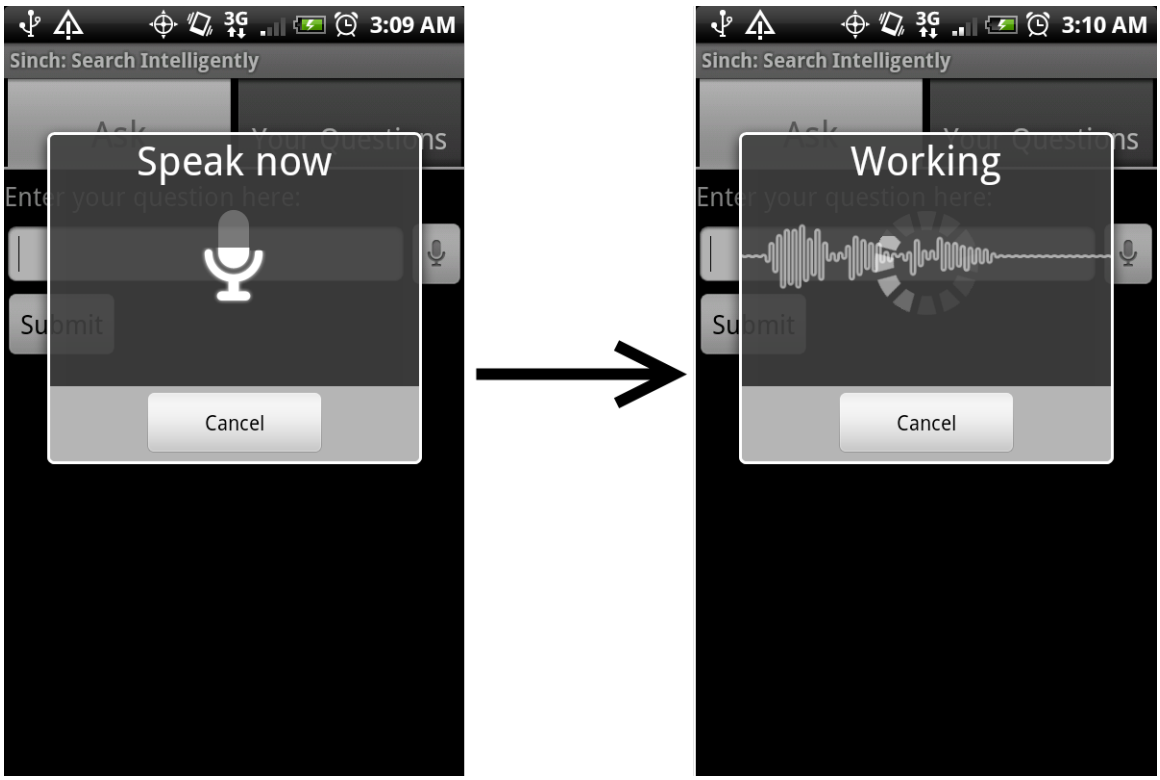

Figure 3-3: The Android speech input modal dialog box in each of its two states: recording and processing.

the user to record their question. This dialog box is displayed in figure 3-3. Once the user is done recording, the dialog box processes their input and disappears. The text box is then automatically populated with the result of the speech-to-text processing. If some of the words were processed incorrectly, the user can still click in the text box to bring up the keyboard and correct the mistakes. When the user is satisfied with the question shown in the text box, they can click on the submit button to send their question to Turkers for answers. After the button is clicked, the application switches to the interface that displays incoming answers for the question that was just submitted.

Design decisions were made about both the "Ask" tab and the microphone button. Rather than giving the user a longer, more descriptive title, the tab is labeled "Ask" in large lettering for simplicity and clarity. The microphone button is located in a prominent, easily accessible location since it is an important option given the nature of the application. While walking or driving, users would benefit from speech input

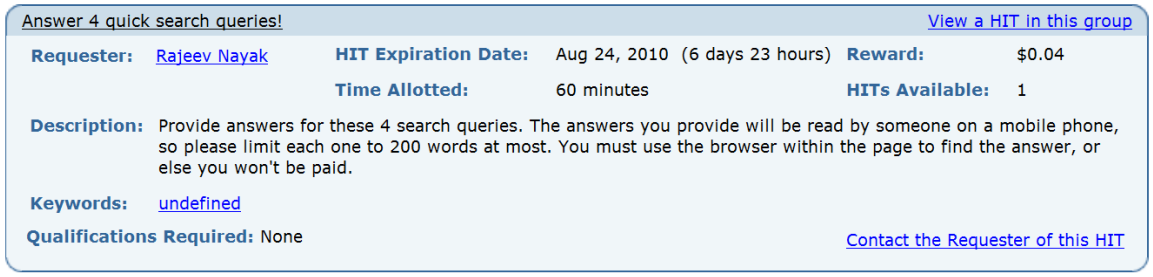

Figure 3-4: A single Sinch task as seen in the Mechanical Turk task browsing interface. This is the standard expanded form of a Mechanical Turk task listing.

rather than typing out their questions on the phone's keyboard.

The behavior of the microphone button was also a key design decision. Rather than taking the VizWiz approach of recording the audio and letting Turkers listen to it, the Sinch application immediately executes speech-to-text processing and does not save the audio. This method allows for simpler code and better space efficiency. However, it falters in cases where the speech recognition is erroneous but the audio is easily understandable by a human. In order to remedy this, the text output of the speech recognition is presented to the user, and they have the ability to record their question again or edit the text manually before submitting the question.

3.2 Answering a Question

When a mobile device user submits a question to the Sinch system, it is posted as a task on Mechanical Turk for Turkers to answer. As they browse through tasks using the Mechanical Turk browsing interface, Turkers will see the Sinch task listed with the title "Answer 4 quick search queries!" If they click on the task title, the task listing will expand and present them with the task description: "Provide answers for these 4 search queries. The answers you provide will be read by someone on a mobile phone, so please limit each one to 200 words at most. You must use the browser within the page to find the answer, or else you won't be paid." This expanded task listing is shown in figure 3-4. Turkers must click on the "View a HIT in this group" link to get to the task page.

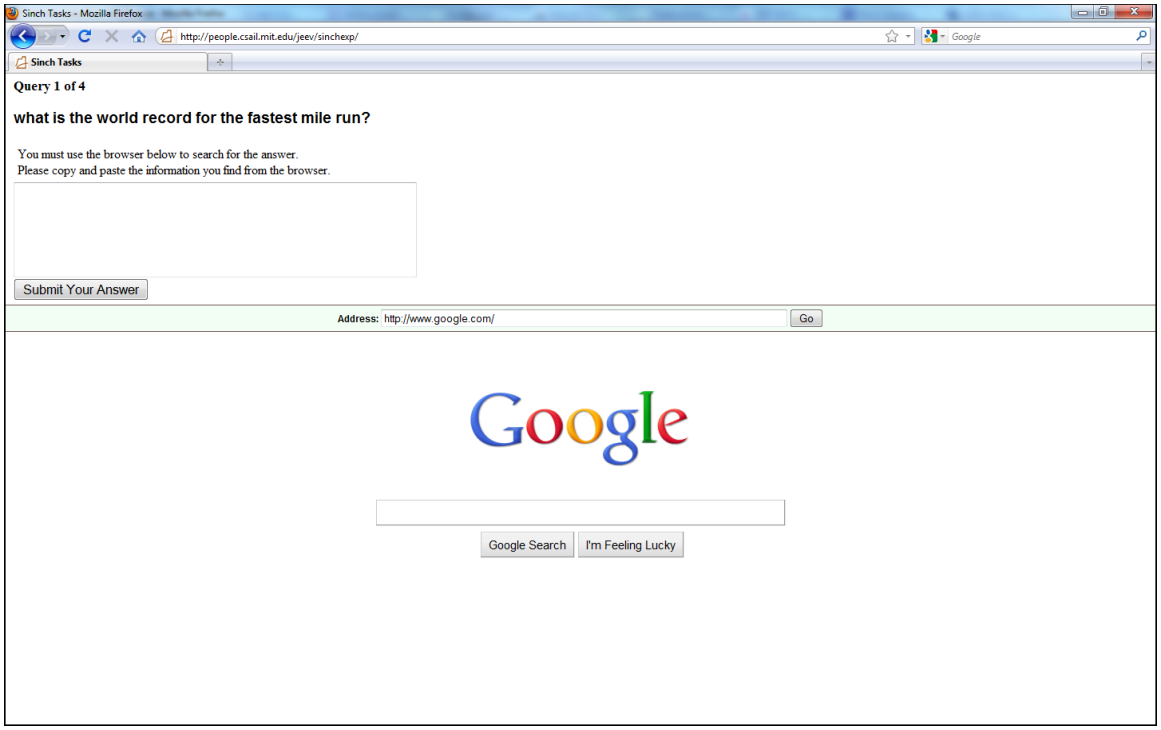

Figure 3-5: The Sinch Mechanical Turk interface.

As they first view a Sinch task, Turkers will see the web page in figure 3-5 embedded in the Mechanical Turk task window. The task consists of 4 questions, and the current question number is indicated at the top of the page. Underneath that is the question itself, displayed in large bold text since it is the most important piece of information on the page. Next is the text area where the Turker inputs their answer, along with a submit button and instructions: "You must use the browser below to search for the answer. Please copy and paste the information you find from the browser."

The bottom half of the page is occupied by the browser that the Turker must use to find the answer. It starts off on the Google homepage in order to encourage Turkers to immediately perform a web search and start looking for the answer. Turkers are also provided with an address bar so that they can type in a URL to navigate to any web page they want. As Turkers use the browser, the web pages they visit are logged by the Sinch system so that they can be returned to the mobile device user with the answer. In addition, all text selections in the browser are logged so that

they can be highlighted in the mobile device's browser, drawing the user's attention to the important text. This is why the instructions encourage Turkers to copy and paste from the browser.

After a Turker clicks the submit button, a new question appears and the browser returns to Google. If they click the submit button on their final question, then the task is automatically submitted on Mechanical Turk and they are returned to the task browsing screen.

3.3 Viewing Answers

The final stage of the Sinch system is when the mobile device user views the answers returned by Turkers. The answer viewing interface, outlined in figure 3-6, is a simple list-based drill-down interface.

Mobile device users can view all of their previously submitted questions by clicking on the "Your Questions" tab after opening the Sinch application. Under that tab, they will see a list of all of their questions, sorted chronologically with the most recently asked on top. This interface is on the top-left of figure 3-6. If a question has new answers that have not been viewed yet, the number of new answers will be indicated to the right of the question. For example, in the figure the question "what is the Stephen king story that stand by me was based on?" has one unviewed answer. Users can click on a question to drill down into a list of its answers.

The list of answers is on the top-right of figure 3-6. The question is displayed in yellow on top of the list of answers, which is sorted chronologically by most recently returned so that users can easily access the newest answers. If there are no answers yet, the interface in figure 3-7 is shown to the user instead. If users are impatient for answers, they can press the refresh button to check for new answers to the question. Otherwise, they can wait for an automatic notification when new answers come in. The notification appears in the notifications bar at the top of the phone's screen. In figure 3-7, a notification is displayed because a new answer was just submitted by a Turker.

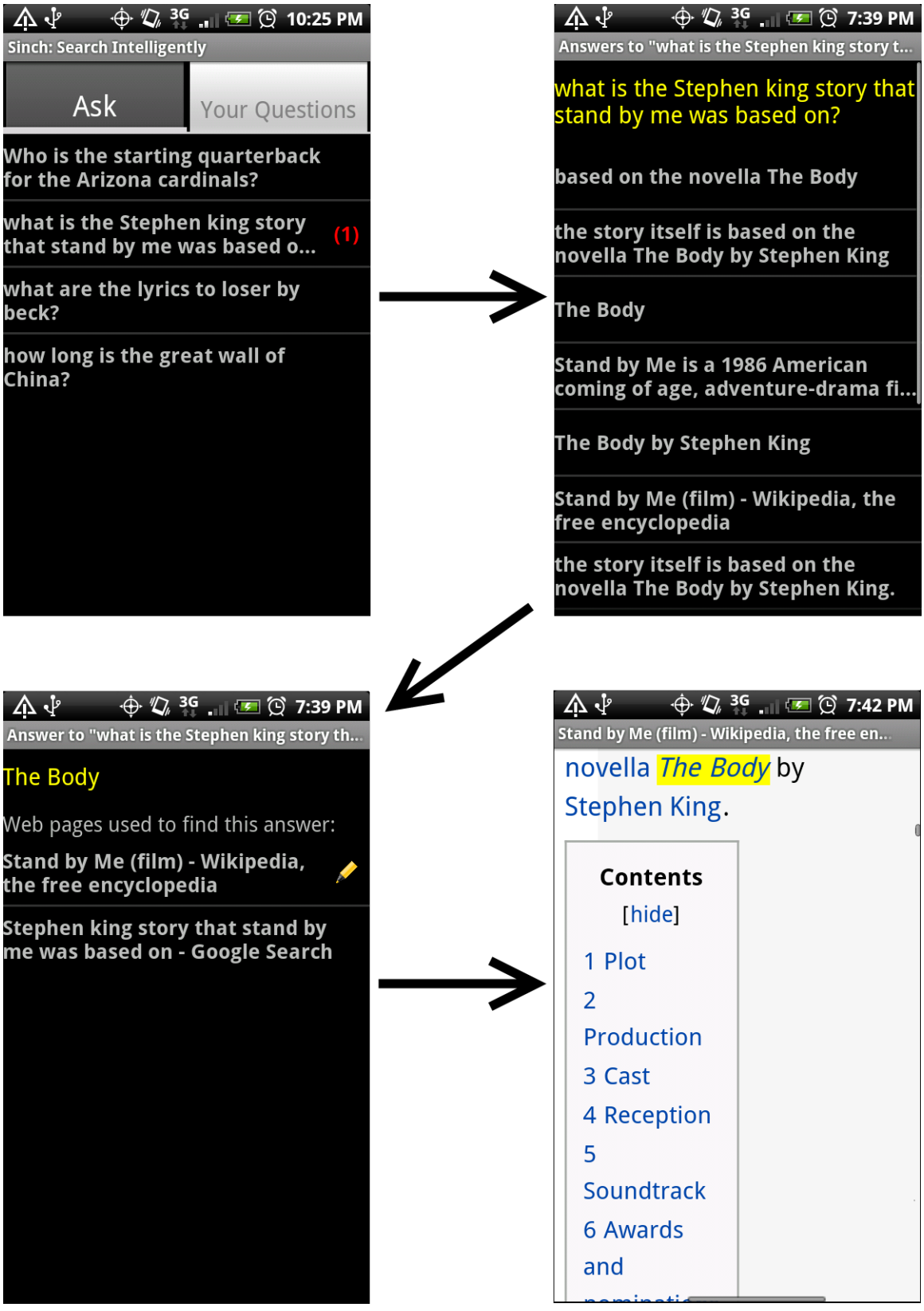

Figure 3-6: The four stages of the drill-down interface for viewing answers: list of questions, list of answers, answer details, and answer web page.

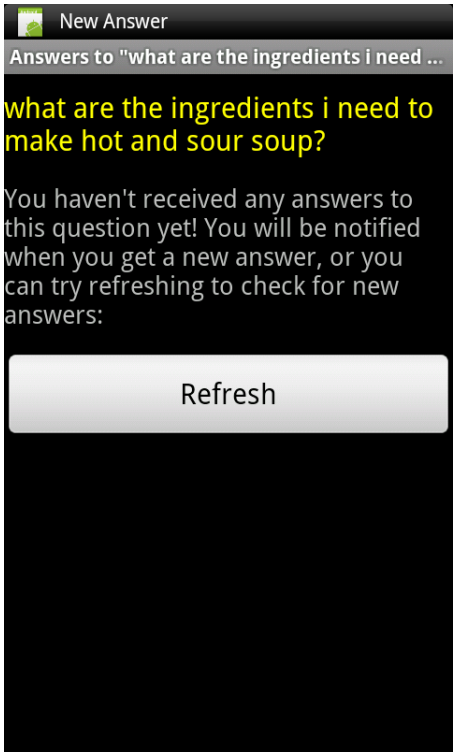

Figure 3-7: The answer list interface before any answers have been submitted. A notification at the top indicates that an answer has just been submitted.

By clicking on an answer in the list of answers, users can drill down into an answer details page, shown at the bottom-left of figure 3-6. The answer details page shows the full text of the answer in yellow at the top, followed by a list of the web pages visited by the Turker. The web pages are in reverse order for two reasons. First, it reflects the ordering of browsing history in all standard browsers. Second, the page(s) where the Turker found the answer to the question will probably be among the last pages that they visited, so they will show up closer to the top of the answer details page. Web pages that the Turker copied and pasted text from are indicated with a yellow highlighter icon on the right. This gives the mobile device users a better idea of which web pages are relevant.

Clicking on a web page title opens the web page in a browser window, shown on the bottom-right of figure 3-6. The web page view shows the web page title in the top bar and loads the desktop version of the web page, just as the Turker would have seen it. If the Turker copied and pasted text from the web page, the original text will be highlighted and the browser will automatically zoom into it so that the user can easily see the answer text in its original context. In the example shown in the figure, the answer text, "The Body", was copied and pasted from the Wikipedia page "Stand by Me (film)". The browser automatically highlighted the text in the Wikipedia page and zoomed into it.

3.4 Design Iteration

The Sinch system design was iterated on three times since the original version. The first iteration focused on adding new features to the application, while the next two made the interface more learnable and efficient.

3.4.1 Version 1

In version 1 of Sinch, the question asking interface in the mobile application only consisted of the input text box and the submit button. Instead of a microphone button, the speech input was included as a menu option. Users would have to press

the "MENU" button on their phones to bring up the menu, and then they would have to click the "Speech Input" menu item to trigger the speech recognition.

The Mechanical Turk interface was also very similar to the final version. In version 1, the only difference was the instruction text. The copy and paste logging was not included in the Sinch system yet, so the text omitted the instruction to copy and paste text from the browser.

The answer viewing interface in the mobile application was lacking in many features that are present in the final version. The question list and answer list were exactly the same, but the answer details page was a bit different. Instead of displaying web page titles in the list of visited pages, the URL's were used. In addition, there were no highlighter icons, and the default starting web page for all Turkers, "http://www.google.com/", was included in the URL list. Upon clicking on a URL, users were taken to the normal browser, not the modified one that highlighted text and zoomed in.

3.4.2 Version 2

The two major features added in version 2 were notifications and text highlighting. Aside from factoring in these new features, the interface did not change.

After using version 1 of the Sinch application, it was apparent that notifications were essential to the user experience. There is no way to avoid the latency that Turkers introduce into the Sinch system, so users of the mobile application have no choice but to wait for an indeterminate amount of time for answers to come back. In version 1, the only way to check for new answers was by repeatedly pressing the refresh button on the page that displayed the list of returned answers for a particular question. In addition, since each answer list corresponds to a single question, it would only check for new answers to that certain question.

Version 2 solved this problem with notifications. As soon as a new answer was returned by a Turker, the phone would vibrate, the phone's notification light would blink, and a notification would pop up at the top of the phone's screen. Clicking the notification would take the user to the Sinch application. The unviewed answer

counts were also added to the question list in this version of the application, so that users would easily be able to tell which questions received new answers after clicking on the notification.

Another problem with version 1 was the difficulty in finding the answer text within the provided list of web pages. After opening the pages in the browser, it was often hard to find where the answer came from, especially when dealing with long, text-heavy websites like Wikipedia.

This problem was solved by introducing text highlighting. In version 2 of Sinch, it was assumed that the last text selection made by a Turker was the final answer to the question, since they would have no reason to select any further text after finding the answer. Therefore, the final text selection on the last visited web page was logged for each Mechanical Turk task. In the mobile application, a new button was added to the answer details page just above the URL list. Clicking on this button opened the last web page the Turker visited in a modified browser that highlighted the Turker's last text selection and zoomed into it.

3.4.3 Version 3

Version 3 focused on tweaking the text highlighting feature and making it more usable. Instead of just logging the final text selection made by the Turker, version 3 of Sinch logged all text selections. The changes were motivated by questions like "What are the birth dates of the starting five players on the Miami Heat?", for which the answer might require highlighting multiple text regions. Moreover, the answers might not all be on the same web page, so the Turker might copy and paste relevant text from multiple web pages.

This issue was fixed in version 3 by taking into account all text selections that contributed to the final answer text. Turkers were instructed to copy and paste from the browser, and once they submitted their final answer, all of the text selections that were a part of that answer were saved by the Sinch system.

In the mobile Sinch application, the answer details page changed once again. The button introduced in version 2 that opened the last visited webpage and highlighted

the Turker's last selection was removed entirely. Instead, each of the links in the list of web pages opened up in the special browser rather than the default browser. If the web page had any saved text selections in it, they were highlighted in the special browser. The browser also automatically zoomed into the first of the selections on the page. Pages that had selections were indicated by a yellow exclamation point in the web page list so that users would know which pages were relevant to the answer.

The answer details page underwent one other minor change in version 3. After using version 2, it was apparent that URL's often did not communicate the content of the web pages well enough. Instead, the URL's in the list were replaced by web page titles, which gave users a better idea of the Turkers' browsing history.

3.4.4 Version 4

The first two design iterations were a result of self-evaluation of the interface. For the third and final design iteration, version 3 of the mobile application was given to users to try out. They were asked to submit a question, wait for answers generated by Turkers, and view the answers. During this process, they were encouraged to think aloud as they used the interface. Observations of their behavior led to a few final design changes.

1. **The speech input was undiscoverable.** None of the users found the speech input option in the menu on the question asking page. As mentioned before, speech input is an essential feature of Sinch, and it must be discoverable. This led to the decision to include a microphone button located prominently right next to the input text box.

2. **The text highlighting feature was unclear.** Users were confused by the meaning of the yellow exclamation points in the web page list, and it often had to be explained to them. This was made clearer by using an image of a yellow highlighter instead of the yellow exclamation point.

3. **The listing of the Google web page on the answer details page was unnecessary.** While thinking aloud, a few users complained that it was annoy-

ing to always see Google at the end of every list of web pages. This happened because the browser in the Mechanical Turk task always started at the Google homepage and it was logged as the first visited web page. The answer details page was changed so that the web page list omitted the initial Google page.

4. **The text highlighting occasionally made text unreadable.** The text highlighting feature set the background of selected text to yellow. However, some Turkers selected light-colored text, which was unreadable by users when the background was changed to yellow. This issue was fixed by not only changing the background color to yellow, but also changing the text color to black.

Chapter 4

Implementation

The Sinch system has four main components, the database, the mobile application, the TurKit script, and the Mechanical Turk task. The database is hosted on the MIT CSAIL MySQL server. It stores all of the questions and answers, including auxiliary data like Turkers' visited web pages and text selections. The mobile application is developed on the Android platform, and it provides users with the means to submit questions and receive answers. The TurKit script reads the database and posts new tasks to Mechanical Turk based on how many new answers are needed. Finally, the Mechanical Turk task presents Turkers with questions from the database and saves their answers back into the database. These component interactions are summarized in figure 4-1.

4.1 The Database

The Sinch MySQL database consists of four tables: questions, answers, web pages, and text selections. Each table stores multiple pieces of information that are critical to the Sinch system.

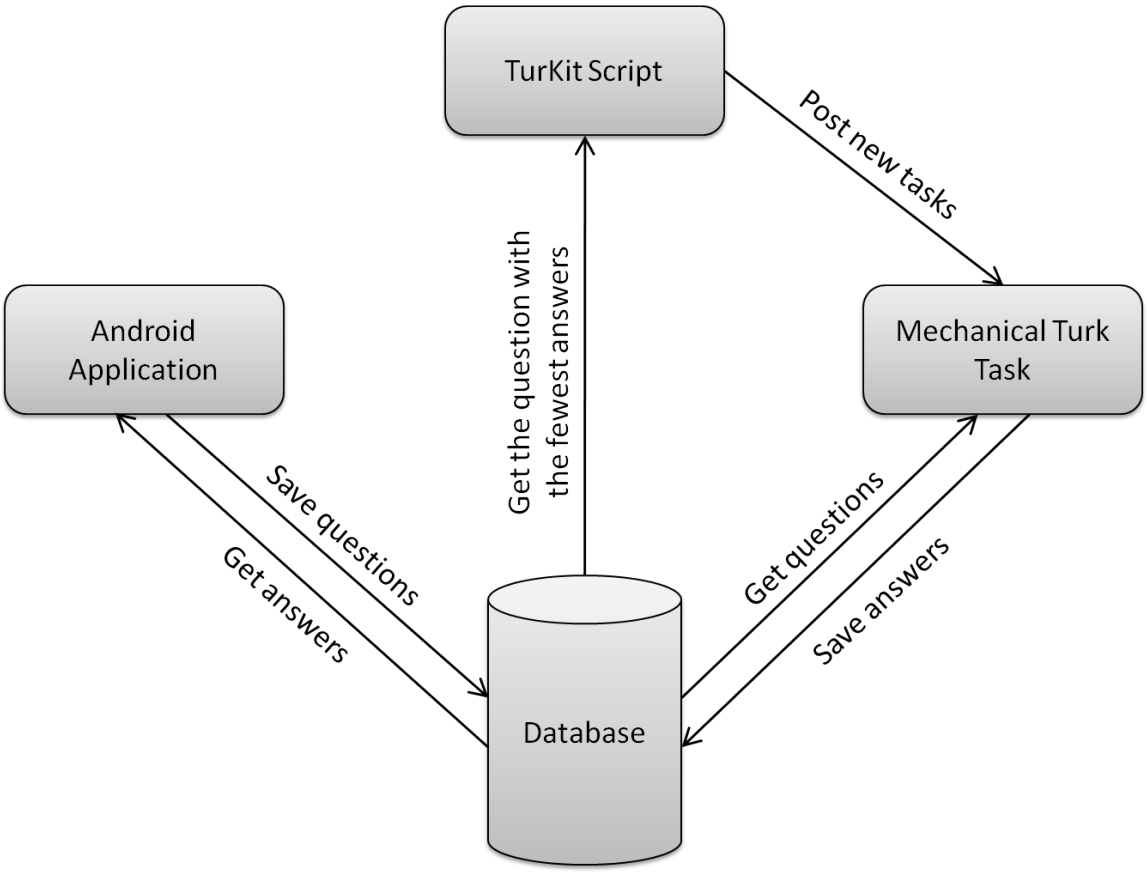

Figure 4-1: The four major components of the Sinch system architecture.

4.1.1 The Questions Table

The questions table stores the questions submitted from the Android application, along with a unique phone identifier. The unique identifier ensures that when a phone requests new answers, the server will only return answers to questions asked from that phone.

4.1.2 The Answers Table

The answers table stores the answers given by Turkers who complete a Sinch Mechanical Turk task. In addition, it contains a foreign key mapping each answer to a question from the questions table.

4.1.3 The Web Pages Table

The web pages table stores all web pages visited by Turkers in the Mechanical Turk interface. In addition to the web page URL's, it also stores their titles so that they can be displayed on the answer details page in the Android application. The index of the web page in the Turker's browsing history is stored as well, so that the ordering can be preserved when displaying the list of web pages to the Android users. The web pages table has a foreign key mapping each web page to an answer.

4.1.4 The Text Selections Table

The text selections table stores information about each text selection made by Turkers using the browser in the Mechanical Turk interface. The selected text is saved, accompanied by information about the location of the selection. The XPath of the DOM nodes where the selection started and ended is stored in the table, as well as the starting and ending character offsets in each of these nodes. The nodes and offsets together can uniquely identify the position of the text selection. Each selection is mapped to a web page using a foreign key.

4.2 The Mobile Application

The Sinch mobile application is built on the Android platform, which uses a Java API. There are three types of Java classes used in the Sinch application: a database class, a service, and activities. The database class is a Java class that acts as a wrapper around a SQLite database. The service is a background task that runs even when the application is not open. It repeatedly polls the Sinch server for new answers. Activities are classes that represent what users actually see on their screens. Each different page in the application has its own activity class.

4.2.1 The Database Class

The database class wraps around a SQLite database that is almost identical to the central Sinch MySQL database. It uses the same four tables: questions, answers, web pages, and text selections. The class includes methods for data insertion and retrieval that are used by the service and the activities.

4.2.2 The Service

The service starts up with the application and runs in the background continuously, even if the application closes or the phone goes into sleep mode. As soon as it is started, the service begins to periodically request new answers from the main Sinch database. When a new question is submitted, the service starts polling the server at 10 second intervals. The rate of polling then backs off exponentially with respect to the amount of time since the last question was submitted. Each time the service requests new answers, it sends a GET request to a PHP script running on the MIT CSAIL web servers, including the unique phone identifier and the unique id of the last answer received by the phone. The script returns all answers corresponding to the phone's id that were submitted after the last answer received by the phone. The service inserts these new answers, along with their web pages and text selections, into the application's database.

When new answers are received, the service issues a notification to the Android

operating system using the Android NotificationManager class. The notification appears in the operating system's notification bar at the top of the phone's screen. In addition, the phone vibrates and its LED notification light starts blinking.

4.2.3 The Activities

The Sinch application includes five different activities, each corresponding to a different page in the application. The activities allow the user to submit a question, view all of their questions, view all answers to a particular question, view a particular answer with a list of visited web pages, and view a web page that highlights key text and zooms into it. As users navigate through the application, different activities come to the foreground. The application flow shown in figure 4-2 indicates the actions that trigger activity switches.

Submitting a Question

The question submission activity is the first activity presented to users when they open the Sinch application. When users submit a question, the activity sends a GET request to the Sinch PHP server with the question text and the phone's id, and the server returns the question's unique id in the central Sinch database. Once this id is received, the question submission activity saves the question in the local SQLite database and switches to the activity for viewing answers to the question.

Viewing Questions

The question viewing activity is activated when the user clicks on the "Your Questions" tab after opening the application. It extends Android's built-in ListActivity class, which allows it to easily display a list corresponding to SQLite database items. As it loads, the activity makes a call to the database class and populates its list with all questions in the database in reverse chronological order. If any of the questions have unviewed answers, the activity displays the number of unviewed answers next

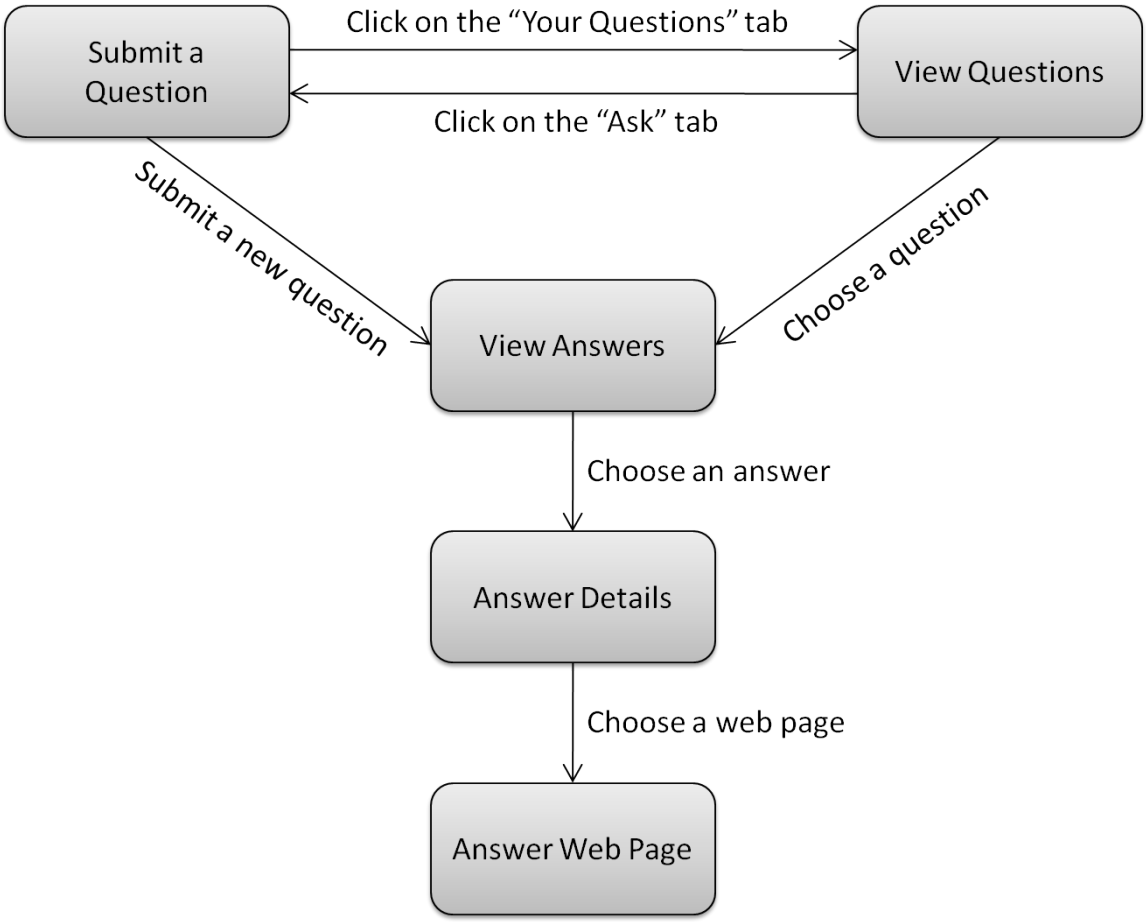

Figure 4-2: The five activities in the Sinch Android application.

to the corresponding question in the list. When the user clicks on a question, they are taken to the answer viewing activity.

Viewing Answers

The answer viewing activity is always given a single argument on startup: the unique id of a question. It makes a call to the database class to get all answers to the question with the given id. Just like the question viewing activity, it extends ListActivity, and it populates its list with answers in reverse chronological order. The question text is added in a large yellow font above the list by adding a HeaderView to the list. Clicking on an answer triggers a switch to the answer details activity.

Answer Details

The answer details activity receives the unique id of an answer when it is invoked. It also extends ListActivity, and it populates the list with all of the web pages corresponding to the given answer. The list entry for each web page that includes text selections is accompanied by a yellow highlighter icon. The activity also adds the full answer text and the string "Web pages used to find this answer:" as HeaderViews above the list. Clicking on a web page title opens the final activity, the answer web page.

Answer Web Page

The answer web page activity uses a WebView: an Android view that allows applications to display web pages. When the activity first starts, it sets the WebView's user agent string to "Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6". It then loads the given URL and displays the title at the top of the screen. Behind the title, a progress bar indicates the status of the loading web page. Once the page is loaded, the activity injects JavaScript code into the web page. The JavaScript code uses the text selection data from the database

to determine the DOM nodes and offsets that begin and end each selection on the web page. The positions of these DOM nodes in the WebView should be consistent with the text selection data since the user agent was modified before the page was loaded. The script then uses the node and offset information to highlight each selection, changing the text background color to yellow and the text color to black. Once all of the selections are highlighted, it calculates the positional offset of the first text selection and zooms into it.

4.3 The TurKit Script

Sinch uses the quikTurkit [10] script, developed for VizWiz, to post new Mechanical Turk tasks. TurKit is a JavaScript wrapper around the Mechanical Turk API, so quikTurkit is written completely in JavaScript. The quikTurkit algorithm manages all of the Sinch Mechanical Turk tasks, posting new ones or removing old ones depending on how many are needed. It runs in an infinite loop, checking on the tasks at each iteration.

The goal of quikTurkit is to maintain a certain amount of tasks at all times, depending on the status of the questions in the Sinch database. The quikTurkit script sets a desired number of answers for each question; it doesn't stop posting tasks until at least three answers are returned for every question. This increases the chances of receiving a correct answer, since Turkers are not all guaranteed to return the right answers.

The script queries the Sinch database for the question with the fewest answers so far. If this number of answers is less than three, quikTurkit sets the desired number of tasks to three minus the number of answers. Otherwise, the desired number of tasks is zero, since no new answers are needed.

A multiplier can be applied to the goal value as well, since more tasks will increase the probability of a Turker picking up a task. Sinch uses a multiplier of three. As an example, when a new question is submitted, it will have zero answers, so three new answers will be required. Since the multiplier is three, the desired number of tasks

will be nine.

At each iteration of the loop, quikTurkit checks the database of questions and recalculates this desired number of tasks. If the current number of active tasks is less than the desired number, then it posts new tasks in order to reach the desired number. If the current number is greater than the desired number, then it removes the oldest tasks.

One last optimization that the algorithm provides is the reposting of stale tasks. If a task has been posted for more than a minute and it has not been picked up by a Turker, it is removed and replaced with a new one. This helps because most Turkers sort by newest while browsing for tasks. Once a task is too stale, most Turkers will never see it again.

4.4 The Mechanical Turk Task

The Sinch Mechanical Turk task consists of two main components: an answer form and an inline browser. The task itself is a web page that TurKit embeds in the Mechanical Turk task page as an iFrame. This web page is hosted on the MIT CSAIL web servers.

The answer form consists of a question number, a question, an answer box, and a submit button. When the task first loads, an Ajax POST request is sent to a PHP script, which is also hosted on the MIT CSAIL servers. The script returns the question in the Sinch database that has the least amount of answers. This question is presented in the answer form, and the question number is set to 1 since it is the first question in the task.

When a Turker clicks on the submit button, the page makes another POST request to the same PHP script. However, it includes the unique id of the question that was just used, so that the script will return a different question. This ensures that the Turker will not answer the same question multiple times in the same task. As the Turker continues to submit answers, the POST requests include a list of all completed questions, ruling them out as future possibilities. When the fourth and final answer

is completed by the Turker, the entire Mechanical Turk task is submitted using the TurKit library.

The inline browser uses PHProxy, an open-source URL-rewriting proxy built with PHP. The proxy is embedded in the task page using an iFrame, and JavaScript is injected into the proxy so that it can log events and communicate with the outer task page. Every time a new web page is loaded in the proxy, its URL and title are sent to the outer page, which maintains a list of web pages. In addition, every time the Turker makes a text selection in the proxy, information about the selection is communicated to the outer page. This information includes the absolute location paths of the start and end DOM nodes using only position predicates in abbreviated XPath syntax. The relevant start and end offsets within those nodes are also included.

When a question is answered, all of this logged information is saved to the Sinch database through a PHP script. As soon as a Turker presses the submit button, before the next query is loaded, the page makes an Ajax POST request to the PHP script. It sends the script the answer text and the entire list of visited web pages. Before it sends the text selection information, it cross-checks all of the text selections against the answer text. Text selections that are not included in the answer text are deemed irrelevant, and they are discarded. All other text selections are sent in the request and saved to the database.

Chapter 5

Evaluation

The effectiveness of the Sinch system hinges upon the validity of four major hypotheses:

1. **The latency of Sinch is not significantly greater than the time taken to find an answer using a mobile device's browser.** The main goal of Sinch is to deal with the problem of mental focus in a mobile situation. However, the immediate need for answers is always present. Although Sinch allows users to submit their questions and immediately turn their attention elsewhere, it still needs to provide timely answers.

2. **An acceptable percentage of Turkers are capable of providing the correct answer to any given question.** Sinch provides multiple answers to each question, so some wrong answers are acceptable. However, at least one answer to each question absolutely must be correct.

3. **Viewing the browsing history that led to a correct answer increases confidence in that answer.** Each Turker's browsing history is presented to users so that they can verify the Turker's answer. If users blindly trust correct answers without checking the browsing history, then providing it is useless.

4. **Viewing multiple correct answers increases confidence.** Another way that Sinch expects users to verify answers is by comparing them to other an-

swers. If users are very confident after receiving the first correct answer, then providing multiple answers is useless.

These hypotheses were tested in three separate evaluations.

5.1 Evaluating Latency

The first hypothesis was tested before Sinch was implemented in order to determine the feasibility of the system. If the latency of Turker responses turned out to be too high, then there would be no point in developing the actual system. The latency was evaluated by asking Turkers to answer questions and asking people to find answers to the same questions using a mobile browser.

A corpus of 30 questions was used in this experiment. The difficulty of the questions varied, and each was classified into one of three difficulty levels:

1. After copying the question text into Google, the answer shows up directly on the search results page. (e.g. "What song has the lyrics planet earth turns slowly?")

2. After copying the question text into Google, finding the answer requires drilling down into the first web page in the search results list. (e.g. "How do you open a pomegranate?")

3. After copying the question text into Google, the answer is neither on the search results page nor the first web page in the search results list. (e.g. "How many a cappella groups does mit have?")

The corpus was generated by coming up with original questions and testing them until there were 10 of each difficulty level. All 30 questions are listed in table A.1.

5.1.1 Turker Latency

The first half of the evaluation determined the amount of time Turkers took to answer each question. 24 tasks were posted on Mechanical Turk, and each task consisted of

6 questions. Since the Sinch Mechanical Turk interface was not built yet, Turkers were only shown the question text, two input text boxes, and a submit button. They were asked to paste their answer into the first text box and the URL of the web page where they found the answer into the second text box. The amount of time it took to answer each question was logged.

5.1.2 Mobile Search Latency

The second half of the evaluation determined the amount of time people took to answer each question using a mobile browser. 10 different users were given an iPod Touch, and they were instructed on how to use its browser. They were told to use the browser to find the answers to the questions given to them, and announce that they were done only when they were fully confident of the answer they found. Each user was given three questions: one of each difficulty level. The question difficulties were counterbalanced to account for bias. Users were timed from when they turned on the iPod Touch until they announced that they were done.

5.1.3 Results

The average search times for each question difficulty level on each platform are summarized in figure 5-1. For the Mechanical Turk tasks, the average search time was 113 seconds and the median was 80. For the iPod Touch, the average time was 101 seconds and the median was 77.

Although these times are very similar, the Mechanical Turk task time does not reflect the true latency of the operation, since Turkers will not begin answering a new question immediately after it is submitted. Instead, they would probably be in the midst of answering another question. The expected time until a Turker finishes their current question is 50 percent of the time it takes a Turker to fully answer a question. Therefore, the expected time until an answer is received for a newly asked question is 150 percent of the time it takes a Turker to answer a question. This pushes the average time to 169.5 seconds and the median to 120. This is still an acceptable

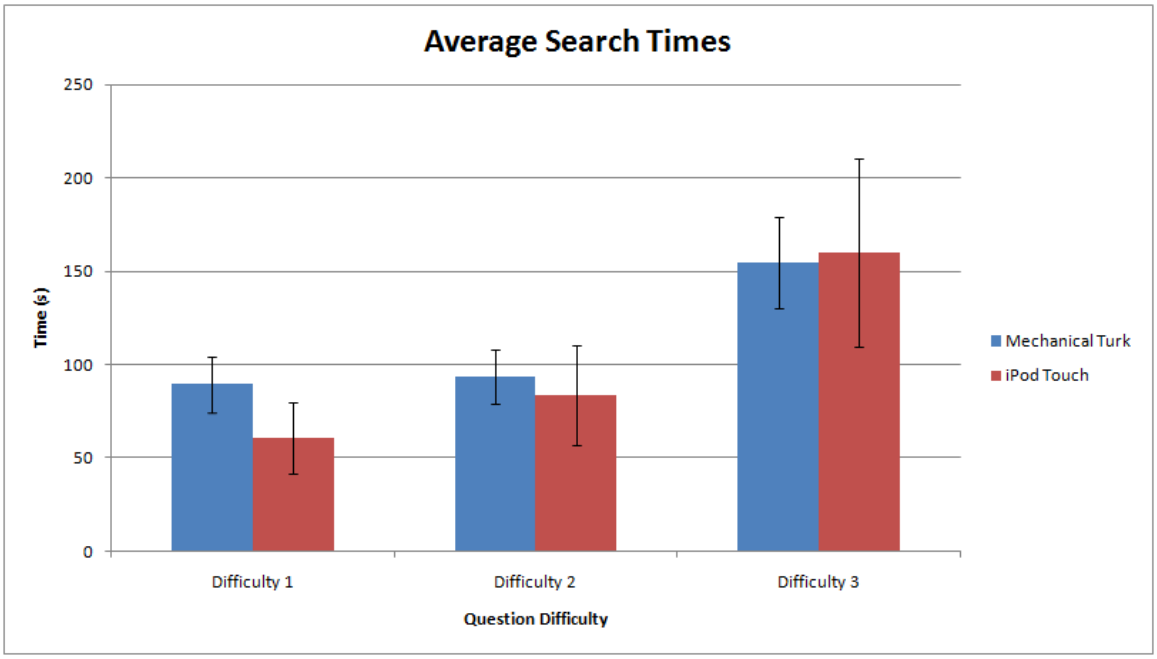

Figure 5-1: Average search times of Turkers and iPod Touch users over the three question difficulties.

latency, given the high latency of the iPod Touch browsing tasks.

It can also be observed from the graph that Turkers started to do better than the iPod Touch users as the questions got more difficult. This can be attributed to the inherent weaknesses of browsing on a mobile device. It probably took the iPod Touch users more time to find what they were looking for on each web page, and web pages probably loaded more slowly on the iPod Touch. For the difficult questions, these effects were magnified by the amount of browsing that was required.

The error bars show a much greater variance for the difficult questions as well. This was most likely caused by the randomness inherent in difficult web searching. Slightly different search queries can elicit different lists of search results. In particularly difficult web searches, certain formulations of the search query can yield more useful results than others, causing people who fail to keep rewording their query until they get a useful result. The randomness in this repeated rewording can lead to high variance in search times.

5.2 Evaluating Correctness

The second experiment evaluated the second hypothesis: the ability of Turkers to answer questions correctly. This experiment was carried out after the final version of the Sinch system was complete, so it also tested the effectiveness of the Mechanical Turk interface. For this experiment, a new corpus of 24 questions was assembled. Since the experiment's focus was testing how well Turkers could answer the types of questions that would be submitted through Sinch, all of these new questions were gathered from credible sources. Five of the questions came from ChaCha [3] and two of the questions came from Yahoo! Answers [7], both online question answering services. The rest of the questions were taken from formal studies of web searching: eleven were taken from the study of mobile information needs conducted by Sohn et al [15] and six were taken from a Google paper that studied search behavior [8]. The 24 questions covered a majority of the categories of mobile information needs outlined by Sohn et al. Each question, along with its category and source, is listed in table A.2.

In order to test the limits of Turkers' answering abilities, some of these new questions were even more difficult than the questions of difficulty level 3 in the latency experiment. Questions like "which us president named his child after the child's grandfather's college buddy?" could not be easily answered by copying and pasting the question into the Google search box. Instead, these questions usually required multiple search query rewordings in order to find the answer. Each of these questions was tagged as extremely difficult, and each Turker was given no more than one of these questions to answer.

5.2.1 Experiment Design

The experiment was deployed on Mechanical Turk using the final version of the Sinch Mechanical Turk interface. 52 unique Turkers were each given four questions to answer. The first question of each task was a stock question: "what is the world record for the fastest mile run?" A stock question was used because Turkers were

able to view the first question in the task preview page, before they accepted the task. Therefore, they might search for the answer to the first question on their own before accepting the task, which would skew the timing data. Also, they would not be able to use the proxy browser, so their browsing history would not be tracked. All data for this first question was discarded.

The other three questions comprised of two normal questions and one extremely difficult question for each Turker. The position of the difficult question was randomized for counterbalancing purposes. For these three questions, the answers, completion times, and visited web pages were all logged.

5.2.2 Results

Each Turker answered three questions, so their number of correct answers ranged from 0 to 3. The number of answers that each of the 52 Turkers answered correctly is depicted in figure 5-2. Since each Turker was given two normal questions and one difficult question, the majority of Turkers unsurprisingly answered two out of the three questions correctly. The average number of correct answers was 1.88, which means about 63 percent of all answers were correct.

The phenomenon of Lazy Turkers [9] is apparent in these results. Lazy Turkers are Turkers who try to do the minimum possible work in order to get paid. In this case, the five Turkers who got zero answers correct were all Lazy Turkers. Upon examination of their answers and web page history, it was apparent that all they did was search for the exact question text on Google, copy all of the text on the entire search results page, and paste it into the answer box. Without these Lazy Turkers, the percentage of correct answers would increase to 70. In order to deal with this problem in the future, Lazy Turkers could be blacklisted from completing any Sinch tasks once they are discovered.

The fraction of answers that were correct for each question is shown in figure 5-3. Each question received between 8 and 10 responses from Turkers, and each question had at least one correct answer provided. Although most of the questions had a significant number of correct answers, 4 of them had a success rate of less than 33

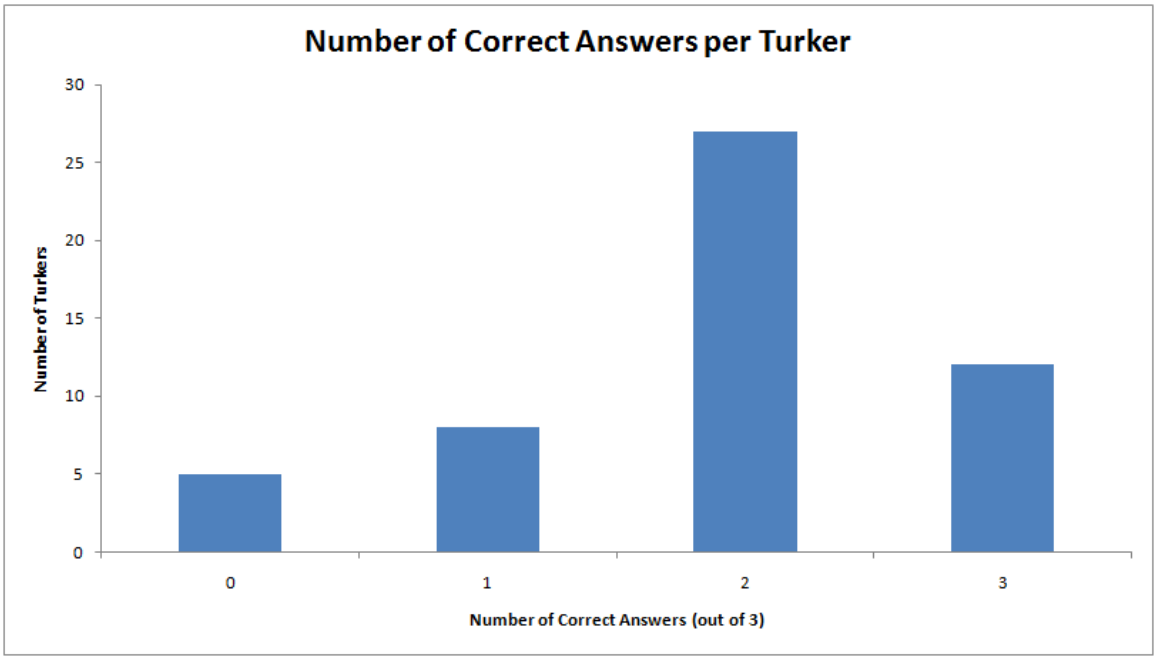

Figure 5-2: The number of correct answers that each Turker provided.

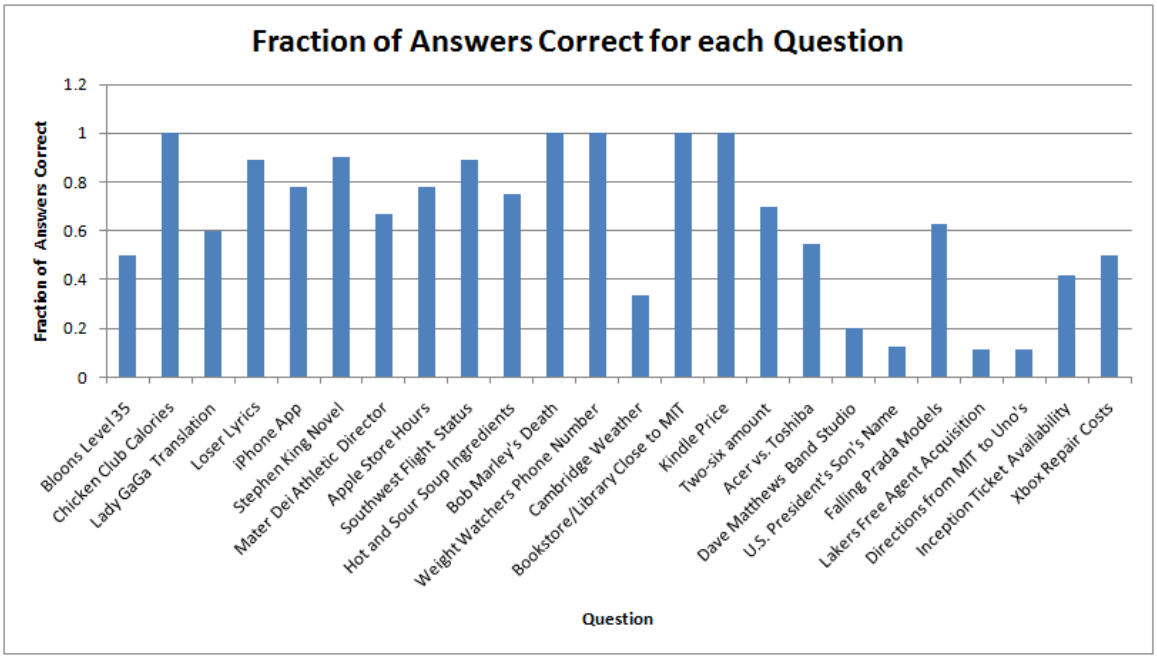

Figure 5-3: The fraction of answers that were correct for each question. The questions are shown in abbreviated form; the full questions can be found in appendix A.

percent. This suggests that the Sinch application should provide more than 3 different answers to a single question in certain cases. This could be accomplished by either setting the desired number of unique answers to a greater value or allowing users to request more answers to specific questions.

One of the 4 questions with a low success rate suffered from an undetected bug in the Sinch Mechanical Turk task. The question "how do I get from mit to uno's pizza?" required Turkers to return a list of directions. However, the proxy browser in the Sinch Mechanical Turk task was not able to load Google Maps properly. This bug hindered Turkers from obtaining the answer to this particular question, and only 1 Turker was able to provide the correct answer. With the bug fixed, questions like this will hopefully be able to elicit more correct answers.

Aside from measuring correctness, this experiment provided insight into how Turkers used the Sinch Mechanical Turk interface. It was the first time Turkers were able to use the proxy browser, and all of the visited web pages were logged. Figure 5-4 shows the number of web pages visited by each Turker in order to find each answer, including the initially provided Google home page. The average number of pages visited was 4.45 and the median was 3.

The graph is generally decreasing, except for a large spike at 3 and a minor spike at 5. The large spike at 3 corresponds to Turkers who performed a Google search, navigated to a web page from the search results, and found the answer on that web page. The minor spike at 5 corresponds to Turkers who did not find the answer on the first web page they clicked on, so they went back to the search results page, clicked on a different search result, and found the answer there. One other notable data point is the maximum of 33 visited web pages, which shows that some Turkers are willing to work very long and hard to deliver correct answers. Unfortunately, the Turker who visited 33 web pages was attempting to answer the driving directions question and was forced to search for an answer without the use of Google Maps, due to the undetected bug in the proxy browser.

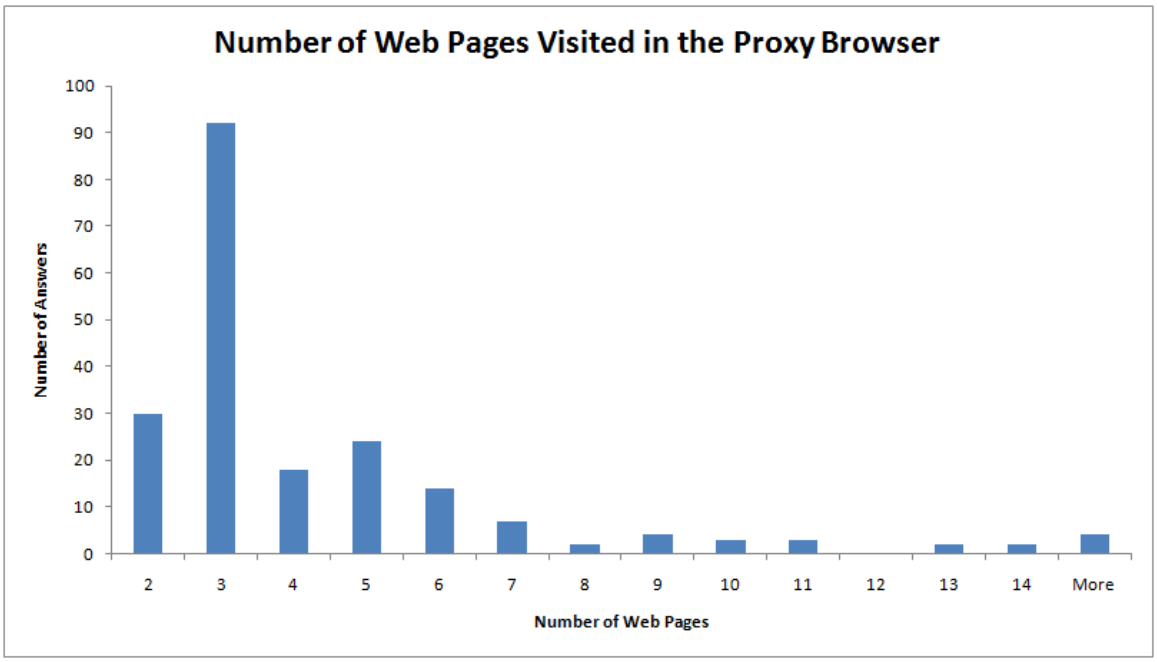

Figure 5-4: The number of web pages visited by Turkers while finding answers.

5.3 Evaluating Confidence

The third and final evaluation tested the last two hypotheses: would viewing the browsing history that led to an answer or viewing multiple answers to the same question improve confidence? This evaluation used the same corpus of questions as the correctness experiment, and it utilized the answers and browsing history generated by the Turkers who participated in that experiment. This experiment was also run on Mechanical Turk, placing Turkers in the role of mobile users.

5.3.1 Experiment Design

The experiment was designed to evaluate confidence indirectly. Rather than asking Turkers to evaluate their own confidence in answers, it put each Turker in a situation where they had to determine the correct answer to a question and be confident about it. The Mechanical Turk task was deployed to 40 unique Turkers. In the task, each Turker was given 5 questions, and for each question they were given a random answer to that question that was provided by a Turker in the previous experiment. Questions from the previous experiment with time-dependent answers were omitted, as well as

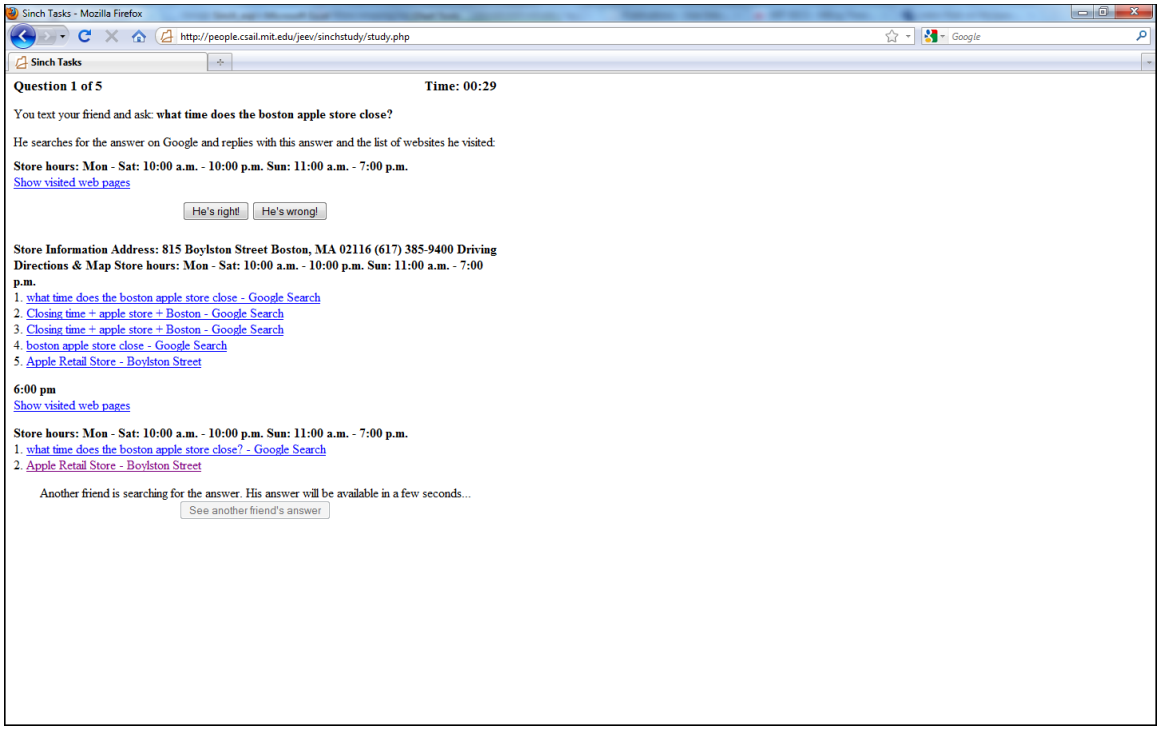

Figure 5-5: The Mechanical Turk task web page for the confidence experiment, shown after the Turker has already revealed 3 additional answers and expanded 2 web page lists.

questions that did not receive more than 1 correct answer. The Turker's task in this experiment was to determine if the given answer was correct or incorrect. The task web page is shown in figure 5-5.

The task is presented as a game, where Turkers have to answer questions correctly to earn money. They are told that they are in a scenario where they have no web access and they have to determine the answer to a question. They text the question to some of their friends and wait for answers. They receive the first answer, but they need to determine if that answer is correct or not. They can look at the list of web pages that their friend visited to find the answer, and they also have the option of waiting for their other friends to respond and viewing their answers so that they can cross-check them against the first answer.

The entire set of 5 questions is timed, and Turkers earn more money the faster they answer the questions. However, if they get more than one of the five answers wrong, they make no money. This simulates a real-life situation where a mobile device

user needs to obtain a piece of information as soon as possible, but they also need to be confident that it is correct. Turkers are encouraged to answer quickly because of the time constraint, but they have to be confident about their answers because of the penalty for guessing. A timer on the top-right of the task page acts as a constant reminder that time is ticking.

Each answer is accompanied by a hidden list of web pages. Turkers can click the "Show visited web pages" link to view the list of web pages. They can click on a link in the list to open the web page in a new tab. This simulates the behavior of the Sinch Android application, in which users have to click on an answer to see the corresponding list of web pages, and they have to click on a web page title to see the web page.

Every 5 seconds, a new answer is made available to the Turker. This answer is also randomly chosen from the answers logged in the previous experiment. When the question first appears, Turkers will see a disabled button that says "See another friend's answer" and a message that says "Another friend is searching for the answer. His answer will be available in a few seconds..." After 5 seconds have elapsed, the button is enabled and the message says "Your other friend's answer is ready!" When the button is clicked, a new answer appears with its own expandable list of web pages, the button is disabled, and the 5 second timer is reset.

When previewing the task, Turkers are shown an instructions page outlining the scenario and the incentives for answering the questions quickly and correctly. Upon completing the 5 questions, they are shown their total score and the amount of time they took.

5.3.2 Results

For each question, the number of extra answers viewed, the number of web page lists expanded, and the number of web page links clicked were all logged. This data is shown in figure 5-6. The average number of extra answers viewed was 0.57, the average number of web page lists expanded was 0.42, and the average number of web page links clicked was 0.09.

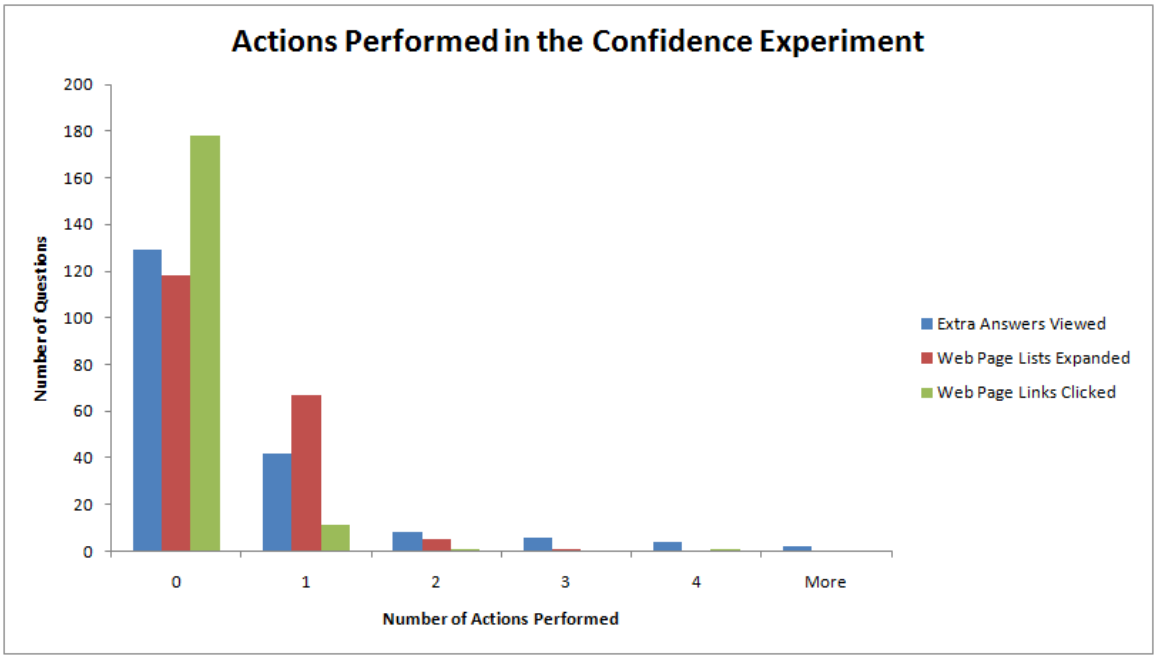

Figure 5-6: The number of new answers viewed, web page lists expanded, and web page links clicked for each question answered by Turkers in the confidence experiment.

Although these averages are low, a significant number of Turkers performed at least one action before submitting their answer to a question. For 32 percent of the questions, the Turker viewed at least one new answer, and for 38 percent of the questions, the Turker expanded at least one list of web pages. The decision to view new answers or web page lists depends on both the Turker and the question. Some Turkers never viewed any new answers or web pages, and for easier questions almost no Turkers viewed new answers or web pages. Another factor that might have biased Turkers was the use of the word "friend", which may have given them extra confidence in the answers.

Nevertheless, the conclusion that can be drawn from this is that viewing multiple answers and browsing history improves confidence for certain users and certain questions. Therefore, these two features of Sinch will not always be utilized, but their presence is valuable in a non-negligible percentage of situations.

A concern in this experiment was that Turkers might search for answers themselves in their own browser to improve their confidence rather than viewing more answers or web page lists in the task. This would defeat the purpose of the experiment,

since users would only submit questions to Sinch if they were unwilling to search for the answers themselves. The data from the correctness experiment showed that the median time it took Turkers to search for answers in their browser was 96 seconds. Of all questions answered in the confidence experiment where the Turker did not view any extra answers or expand any web page lists, only 2 of them took longer than 40 seconds to submit. This shows that the experiment was not significantly biased by Turkers ignoring the task and searching on their own.

One other interesting piece of data is that Turkers clicked on at least one web page link for 7 percent of the questions, less than one-fifth of the number of questions where Turkers expanded at least one web page list. This suggests that just viewing web page titles is usually enough to convince a user that an answer is correct.

Chapter 6

Conclusion

Sinch is a mobile application that uses human computation to provide answers to questions via a desktop web search. By eliminating the need for a mobile web search, Sinch solves all of the problems with searching in a mobile browser. Turkers return concise answers that are easily readable on a small screen. Furthermore, when the Sinch application allows users to view the answer text in its original context on a web page, it automatically highlights the answer and zooms into it so that the user does not have to deal with searching through the web page on their small device. The "fat finger" problem is addressed by allowing speech input for questions, bypassing the need for the on-screen keyboard. Outsourcing the web search to a Turker on their desktop allows mobile device users to avoid navigating through multiple pages in their browser, trying to click on small links with their finger. This web search outsourcing also fixes the problem of poor network connections. Finally, situational disabilities are dealt with by allowing speech input and providing concise answers. Users can quickly speak their question and direct their attention back to the task at hand until an answer is returned by a Turker.

Sinch utilizes Mechanical Turk to obtain human-generated answers to submitted questions. Each Sinch Mechanical Turk task entails answering a series of questions by performing a web search in an embedded browser. The browser embedded in the task is a URL-rewriting proxy than can track browsing history, text selections, and other user events. This proxy browser can be used in future Mechanical Turk tasks

to assess various internet browsing habits.

Sinch introduces two ways to provide more credibility to human-generated answers: providing multiple answers to the same question and a browsing history for each answer. Human-generated answers are prone to error, so returning multiple answers to a single question increases the probability that at least one of them is correct. In addition, users become more confident in an answer when they see similar answers to back it up. Seeing the browsing history that led to an answer also improves a user's confidence in that answer, because they can see the answer in its original context on a web page.

All of these unique contributions in conjunction make Sinch an innovative question answering application that overcomes the shortcomings of web searching on a mobile device.

6.1 Future Work

Many new features can be added to the Sinch system. The next logical step is to support location-based Sinch questions. All Android phones provide location information to applications, so the Sinch application could have an option to include the user's location when they submit new questions. This could be especially useful for questions regarding directions or nearby points of interest. The Mechanical Turk interface could be modified to support this as well. If a location is submitted with a question, the proxy browser could initially load Google Maps and set a marker at the Android user's location, instead of just loading the Google home page.

The Android application could also be modified to support multimodal queries. If a user comes across an object or a location that they want to identify, the application could allow them to take a picture with their phone and ask a question about the picture. The Mechanical Turk interface would have to be modified to display pictures to Turkers.

Sinch currently only deals with question answering web searches. However, it can easily be adapted to also deal with another prevalent type of web search: a search for

a particular web page. For example, users might want to find the legitimate music video for a song on YouTube or a website where they can buy obscure comic books. Since Sinch already logs the web pages that Turkers visit, handling this new type of search query is just a matter of giving Turkers another method of submitting their tasks. Instead of forcing Turkers to write in an answer, Turkers could simply navigate to the desired web page and submit the task. Then the mobile device user would be able to see the Turker's browsing history, ending with the user's desired web page.

Finally, the Sinch application could be ported to the iPhone. Android was chosen as the developing platform because it supports automatic speech recognition and it has a Java API. However, if Sinch were to be widely adopted, an iPhone version of the application would be essential given the large amount of iPhone users.

Bibliography

[1] Aardvark, 2010. http://www.vark.com/.

[2] Ask.com reverts to its q.& a. origins, 2010. http://bits.blogs.nytimes.com/2010/07/27/ask-com-reverts-back-to-its-q-a-origins/.

[3] ChaCha, 2010. http://www.chacha.com/.

[4] Google mobile answers questions and settles bar bets (with sources), 2010. http://lifehacker.com/5560872/google-mobile-answers-questions-and-settles-bar-bets-with-sources/.

[5] Mosio, 2010. http://www.mosio.com/.

[6] Wolfram Alpha, 2010. http://www.wolframalpha.com/.

[7] Yahoo! Answers, 2010. http://answers.yahoo.com/.

[8] A. Aula, R. M. Khan, and Z. Guan. How does search behavior change as search becomes more difficult? In *Proc. CHI 2010*, pages 35–44. ACM, 2010.

[9] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich. Soylent: A word processor with a crowd inside. In *Proc. UIST 2010*, To Appear.

[10] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. VizWiz: Nearly real-time answers to visual questions. In *Proc. UIST 2010*, To Appear.

[11] B. Katz, G. Borchardt, and S. Felshin. Natural language annotations for question answering. In *Proc. FLAIRS 2006*, pages 303–306. AAAI, 2006.

[12] G. Little, L. B. Chilton, M. Goldman, and R. Miller. TurKit: Human computation algorithms on mechanical turk. In *Proc. UIST 2010*, To Appear.

[13] M. R. Morris and E. Horvitz. SearchTogether: An interface for collaborative web search. In *Proc. UIST 2007*, pages 3–12. ACM, 2007.

[14] K. Rodden, N. Milic-Frayling, R. Sommerer, and A. Blackwell. Effective web searching on mobile devices. In *Proc. HCI 2003*, pages 281–296. British HCI Group, 2003.

[15] T. Sohn, K. A. Li, W. G. Griswold, and J.D. Hollan. A diary study of mobile information needs. In *Proc. CHI 2008*, pages 433–442. ACM, 2008.

Appendix A

Questions

Table A.1: The 30 questions used in the latency experiment grouped by difficulty level.

Question	Difficulty
“When will halley’s comet next appear?”	1
“What is lady gaga’s real name?”	1
“What song has the lyrics planet earth turns slowly?”	1
“What is the world record for the fastest mile run?”	1
“What is the name of tina fey’s character on 30 rock?”	1
“When was mit founded?”	1
“What does cbs stand for?”	1
“How big is the ipad?”	1
“What is shaq’s shoe size?”	1
“Who is the ceo of coca cola?”	1
“How do you open a pomegranate?”	2
“How do you fill out a check?”	2
“What was the last us city to host the summer olympics?”	2
“What is the weather in cambridge ma?”	2
“When is daylight savings time this year?”	2
“How long did conan o’brien host the tonight show?”	2
“What is the record for the fastest rubik’s cube solve?”	2
“When did the first nintendo system come out?”	2
“Which nba player has won the most championships?”	2
“How many players are on a football team?”	2
“How much does an ice cream sandwich cost?”	3
“How many days were there in the year 1752?”	3
“What is the standard size of a crossword grid?”	3
“How many a cappella groups does mit have?”	3
“Where is the nearest bank of america atm to the corner of massachusetts ave and newbury st in boston?”	3
“What time does legal sea foods in the prudential center close?”	3
“How much does mistborn book 1 cost on amazon?”	3
“How do you get from mit to harvard square?”	3
“How much does it cost to get 50 gb on dropbox?”	3
“What is the notebook of the week on newegg.com?”	3

Table A.2: The 24 questions used in the correctness experiment, including their categories and sources.

Question	Category	Source
"How do you beat level 35 of bloons in player pack 2?"	Trivia	ChaCha
"How many calories are in a chicken club?"	Trivia	ChaCha
"What does the french part in the song bad romance mean?"	Trivia	ChaCha
"What are the lyrics to loser by beck?"	Trivia	ChaCha
"What is the name of the iphone app that tells you what song you are playing when you hold your phone to the speakers?"	Trivia	Google CHI paper
"What is the stephen king story that stand by me was based on?"	Trivia	Google CHI paper
"Who is the athletic director at mater dei high school in santa ana, california?"	Trivia	Google CHI paper
"What time does the boston apple store close?"	Business Hours	Mobile Info paper
"What is the current status of southwest flight 2021?"	Travel	Mobile Info paper
"What are the ingredients I need to make hot and sour soup?"	Recipes	Mobile Info paper
"What did bob marley die of, and when?"	Trivia	Mobile Info paper
"What is the phone number for weight watchers?"	Phone Number	Mobile Info paper
"What will the weather be like in cambridge, ma this weekend?"	Weather	Mobile Info paper
"Where is the nearest library or bookstore to mit?"	Point of Interest	Mobile Info paper
"How much does the kindle cost on amazon's website?"	Shopping	Mobile Info paper
"How much is a two six of alcohol?"	Trivia	Yahoo! Answers
"Is acer better than toshiba at making laptops?"	Shopping	ChaCha
"What is the name of the dave matthews band studio located outside charlottesville?"	Trivia	Google CHI paper
"Which us president named his child after the child's grandfather's college buddy?"	Trivia	Google CHI paper
"What are the names of the two prada models who fell at milan fashion week 2008?"	Trivia	Google CHI paper
"Did the los angeles lakers have any free agent acquisitions this summer?"	Sports	Mobile Info paper
"How do I get from mit to uno's pizza?"	Directions	Mobile Info paper
"Are inception tickets available tonight at boston common?"	Movie Times	Mobile Info paper
"How much does it cost to send a broken xbox 360 to microsoft for repair without a warranty?"	Trivia	Yahoo! Answers