QuiltView: a Crowd-Sourced Video Response System

Zhuo Chen, Wenlu Hu, Kiryong Ha, Jan Harkes, Benjamin Gilbert,
Jason Hong, Asim Smailagic, Dan Siewiorek, Mahadev Satyanarayanan

School of Computer Science, Carnegie Mellon University

ABSTRACT

Effortless one-touch capture of video is a unique capability of wearable devices such as Google Glass. We use this capability to create a new type of crowd-sourced system in which users receive queries relevant to their current location and opt-in preferences. In response, they can send back live video snippets of their surroundings. A system of result caching, geolocation and query similarity detection shields users from being overwhelmed by a flood of queries.

1. INTRODUCTION

The emergence of wearable devices such as Google Glass has ignited a debate about a "killer app" for them. Everyone wants one of these technically impressive and aesthetically elegant devices. However, their functionality today is roughly that of a smartphone. A user can perform voice-activated searches, place and receive phone calls, get directions, and take photographs and videos. Beyond novelty and coolness, however, their true value proposition is not clear. What payoff can we achieve from widespread deployment of Glass-like devices? That is the question we explore here.

We describe a crowd-sourced system called *QuiltView* that leverages the ability of Glass-like devices to provide *near-effortless capture of first-person viewpoint video*. Recording a video clip merely involves touching the shank of your Glass device. The extreme simplicity of video capture can be used to create a new kind of near-real-time social network. In this social network, users can pose brief queries to other users in a specific geographic area and receive prompt video responses. The richness of video content provides much detail and context to the person posing the query, while consuming little attention from those who respond. The QuiltView architecture incorporates result caching, geolocation and query similarity detection to shield users from being overwhelmed by a flood of queries.

HotMobile '14, February 26 - 27 2014, Santa Barbara, CA, USA

ACM 978-1-4503-2742-8/14/02. . . $15.00.
http://dx.doi.org/10.1145/2565585.2565589

2. MICRO-INTERACTIONS IN GLASS

Our work builds on a key design guideline for Glass application developers: they are encouraged to structure their applications in terms of *micro-interactions*. These are very brief episodes of display or audio stimulus from the system, followed by a quick and unobtrusive user response. As Lukowicz explains [10]:

The basic concept underlying the quick micro-interactions idea is the "two seconds rule." It says that anything that takes you more than two seconds or longer to accomplish, needs a good reason to be done. Anything that takes significantly less you will do without a second thought. Checking something on a smart phone or using a smart phone to take a picture takes much longer than two seconds. Google Glass on the other hand aims to allow you to do this in much shorter time.

Extensive interaction with Glass users is strongly discouraged because it distracts them from their situated contexts. Since user attention is at a premium in situated contexts, micro-interactions aim to be minimally disruptive. In the "snap, pause, tangent, extended" taxonomy of user distractions developed by Anhalt et al [1], a micro-interaction corresponds to a snap. This is the briefest of distractions, and a user can perform a snap activity without the cognitive inefficiency of a mental context swap.

This line of reasoning leads to the question: *"What high-value micro-interactions can one have with a Glass user?"* In other words, what are examples of high-value outputs that a Glass user can produce at low cost in terms of user distraction? One possible candidate is video capture in response to a brief query. A short query that fits within the small Glass display (say, 50 characters or less, which is about a third of Twitter's 140-character limit) can be understood in a single glance. With just a touch, the user can capture a 10-second video segment as a response to the query. Buried in that video is a wealth of information that the user does not have to interpret, convert to text, or explain. About the only investment of attention is to ensure that the scene being captured is relevant to the query. Transmitting the video back is, of course, much more expensive in terms of bandwidth and energy than returning text or even audio. However, when user attention is the most critical resource, the near-effortless capture and return of video is a better strategy. A good example of the kind of video we might expect in response to a query such as "What's exciting at the show?" was recently provided by designer Diane von Furstenberg. She wore Google Glass to capture and share her experience at the Spring 2013 Fashion Show in New York [4].

Without consuming any additional user attention, video

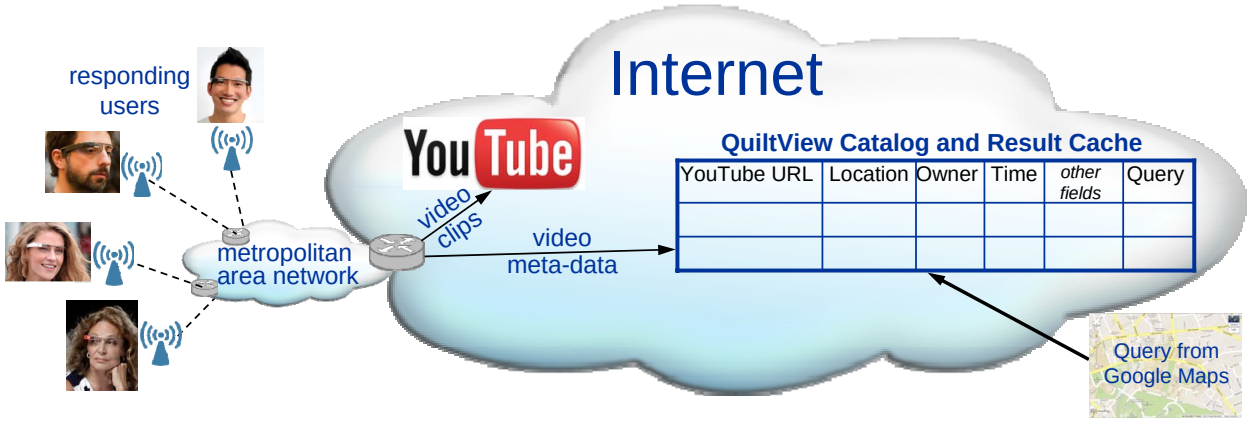


Figure 1: The Cloud-based QuiltView Architecture with YouTube and Google Maps Integration

responses could be accompanied by sensor information that provides context. For example, geolocation, orientation and accelerometer readings could be provided today. In the future, biometric sensor information such as heart-rate and gaze tracking could be included. In other words, at the user's discretion and in response to a specific query, a Glass user can easily respond with a wealth of relevant information. The one-touch effortlessness of this micro-interaction is unique to Glass-like wearable devices.

3. SIMPLE QUERIES, DEEP ANSWERS

"If a picture is worth a thousand words, then a video is worth a million" according to two YouTube guest bloggers from the marketing and advertising industry [13]. Many subtleties that would be lost in a verbal response are fully communicated in a video. Receiving a video response is like "being there" — the person asking the query is effectively transported to the scene of the response. He can use his own taste, judgement and knowledge rather than relying on those of an unknown responder in crowd-sourcing.

Imagine a future in which Glass devices become as common as smartphones. Glass users may be spread over a large area such as a city or a county. Consider a short query with deep semantics such as "How exciting is the party?" or "Is the beach crowded?" In response to such a query, it is faster and simpler for a number of Glass wearers to independently return brief video segments rather than to give detailed verbal responses. In any case, their concept of "exciting" or whether a beach is crowded may differ from that of the person who posed the query. With video responses, that person can judge for himself whether the party is exciting enough to attend, or whether the beach is too crowded. In other words, delivering raw data rather than interpreted data is preferable. This insight lies at the heart of QuiltView.

How could QuiltView make a difference in the real world? The vignettes below describe some potential use cases:

- **Traffic Emergency:** All of a sudden, four lanes of an interstate highway come to a complete halt. The cause of the stoppage is not visible on traffic cameras. Emergency response personnel do not know what to send in response, and where. Dispatching a police officer to do reconnaissance would not help since the officer would also be blocked by the traffic jam. Fortunately many drivers and passengers in the other direction of the highway are wearing Glass. As they pass the scene of the blockage, they receive a QuiltView query from the police. The videos sent in response soon help the emergency response center decide how best to respond.

- **Missing Child (Amber Alert):** A parent sees her child being grabbed and forced into a car, but is not near enough to stop the abduction. The police issues an Amber Alert with a description of the child and the car. This QuiltView query is received by all Glass wearers in the vicinity of the abduction. Many video responses are immediately received by the police, and they are soon able to apprehend the suspect and rescue the child.

- **Real-time Queue View:** Trying to decide whether to have dinner before a movie or after, a student sends a QuiltView query to see how busy nearby restaurants and movie theaters are. The video responses show the lines at the theaters, whether friends or family are already in line (and could purchase his ticket), and how crowded the restaurant lobbies are. Similar real-time information could be obtained about lines at flu shot clinics, sporting events, and crowds in buses.

- **Free Food Finder:** Late one afternoon, a hungry student wonders whether free cookies or donuts might be available nearby. He submits a QuiltView query about free food. Video responses from Glass wearers at different events on campus show not only how much food is left, but also the type of food and how many people are still waiting in line for what is left.

- **Scavenger Hunt:** A new Glass-based scavenger hunt game awards points for videos of specified items or events. One game specifies crowded restaurants, roadside food stalls, and particular types of cars. Because of the incentives, a rich collection of unique videos is captured over time and archived.

- **Time Machine:** A van with damage to its right front fender was involved in a robbery. The car is found abandoned and later discovered to be stolen. The police would like to recreate the crime commencing with the original theft of the auto. By applying image matching software on the QuiltView cache of videos from unrelated queries in the neighborhood of the crime, the police are able to identify the vehicle at specific times and places. From this evidence, the criminals are apprehended and later convicted.

4. SYSTEM ARCHITECTURE

QuiltView is a cloud-based service that is built using off-the-shelf Internet technology. Figure 1 shows the QuiltView architecture. At the heart of this architecture is a global catalog of users and queries, implemented with a SQL database. It includes details of all the queries that have been posed: their content, who posed the query, when, for what geographic target area, and so on. The catalog also includes details of all the responses that have been received: from whom, when, in response to what query, from what location, and other meta-data. Most importantly, it contains the YouTube link of the response. Both the uploading and viewing of videos are done using standard YouTube mechanisms that are wrapped inside QuiltView query and response software. The catalog only holds links, not video content.

The QuiltView catalog also includes details of users and their preferences regarding their willingness to respond to queries. These preferences may include specific topics, specific users or members of specific social network groups who are posing a query, the acceptable volume of queries during some period of time (e.g., queries per day or queries per hour), and the acceptable locations at which queries may be presented. A user can change these preferences at any time, with immediate effect. Any accounting information necessary to support the incentive model for crowd-sourcing will also be maintained in the catalog. For example, if a financial model similar to that of Amazon Mechanical Turk (AMT) is used, the accounting information would include the financial credits accumulated by a user as well as the financial reward offered for each query. In such a model, the reward would be displayed with the query so that a user can decide whether it is worth his while to respond.

The QuiltView catalog is used as a result cache that short-circuits query processing. During events of high public interest (such as the manhunt for the Boston marathon terrorists in 2013) there may be a flood of nearly identical requests for roughly the same information. If a query is deemed to be "close" to a recently-answered one and "recent enough," the cached results for the earlier query are returned. The user posing the query is first shown the earlier query, meta-data about the query, and the number of available responses. He has the choice of accepting the cached results or insisting that QuiltView obtain fresh results for his query. Result caching greatly improves the speed of responses from the viewpoint of the user posing the query. It also reduces the burden of repeated queries on Glass users. How best to define "close" and "recent enough" are important questions that we discuss later in the paper.

Queries are posed using a web interface to Google Maps. Just as one can perform "Search nearby" today in Google Maps, a user zooms into a geographic region and then poses a query such as "Have you seen my dog?" along with a thumbnail image of his pet. The zoomed-in region on the map implicitly defines the scope of the query. Within this geolocated scope, a subset of the Glass users who have opted-in to receive QuiltView queries on a relevant topic (such as "pets" or "community help") receive the query. How the subset of users is chosen is an important QuiltView design feature. The goal is to respect user preferences rigorously, and within that constraint to spread the burden randomly across users. The size of the subset has to be determined based on an estimate of how many users will respond to the query. For example, if three responses are needed and experience has shown that a 60% response rate is typical for the parameters of the query, QuiltView has to present the query to five users. While this simple approach is a good starting point, dynamic adaptation of estimates based on actual experience is possible and more sophisticated user selection mechanisms (such as those based on user reputation) can be envisioned for the future.

Some constraints on queries are necessary. As mentioned earlier, the length of queries is limited to about a third of the Twitter limit for easy display in Glass. There is also a limit on the size of the zoomed-in region on a map — otherwise a single query could be of unacceptably large scope, such as covering the whole planet. When a user receives a query, he can decide to respond with a brief video clip of an appropriate part of the scene around him. Each video clip is uploaded into YouTube, and its link is displayed in the list of responses to the query. Meta-data about this video clip, matched to the original query, is entered into the global QuiltView catalog.

5. PROTOTYPE IMPLEMENTATION

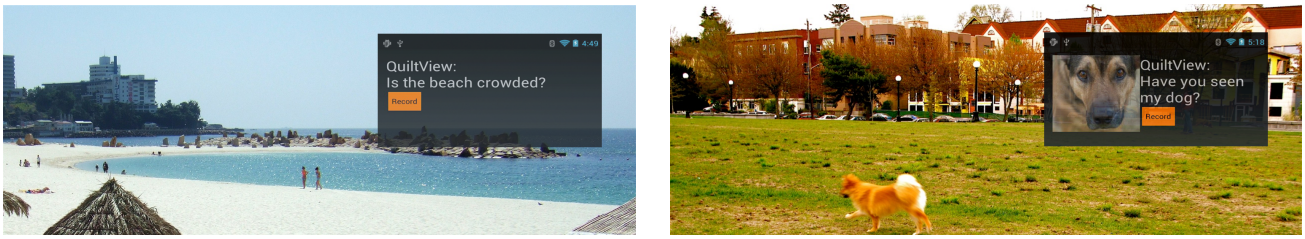
We have implemented a complete prototype of the QuiltView architecture shown in Figure 1. The prototype incorporates all the steps mentioned in Section 4: posing a query using Google Maps, checking for cache hits in the global catalog, sending a notification message to Glass clients, one-touch recording and uploading of video responses to YouTube, and returning responses to the user who posed the query. We describe the client and server implementation in Sections 5.1 and 5.2 below. The query interface and workflow are described in Section 5.3. Our implementation of result caching, including similarity detection in queries, is described in Section 5.4. Load balancing across users is described in Section 5.5. Finally, Section 5.6 describes how we combine synthetic users with real Glass users to explore scalability issues.

5.1 Glass Client

Our approach to implementing the client side of QuiltView is strongly shaped by our goal of one-touch response. The easiest approach would have been to use the *Mirror API,* as recommended by Google [8]. By using this API, a QuiltView server can easily call appropriate RESTful endpoints on a Mirror server in order to communicate with a Glass device. Although this approach can simplify our implementation, the limitations of the Mirror API compromise our one-touch goal. Therefore, the Glass client in our prototype is implemented with the *Glass Development Kit (GDK)* [6], which provides much richer functionality than the Mirror API, and supports creation of native Glass clients.

Current software support for Glass also limits the energy efficiency of our QuiltView client. Since we expect requests to be relatively rare (a few an hour, perhaps, for a typical user), a push-driven notification mechanism based in the cloud would be more energy-efficient than polling by the client. Unfortunately, *Google Cloud Messaging (GCM),* which is the standard service to push updates from the cloud to Android devices [9] is not available yet for Glass. In our current implementation, the client has to poll the QuiltView server periodically to check for new queries. We plan to switch to GCM-based notification as soon as possible.

The QuiltView client software runs as a background service on a Glass device. Figure 2 shows two examples of what



(a) Simple Text Query

(b) Query with an Image

These illustrations conform to Google's guidelines for Glassware screenshots [7]: in each case, the small box on the top right is a screenshot of the Glass display, while the background is the real-world scene visible to the user. The figure on the left corresponds to a simple text query, while that on the right corresponds to a query accompanied by an image .

Figure 2: What a QuiltView User Sees When Receiving a Query

a user sees when he receives a query. The example on the left is a simple text query, while that on the right includes an image. The query message remains on the Glass screen for about fifteen seconds. While the query is displayed, the user can briefly touch the side of the Glass device to record and upload a ten-second video response. If the user does not wish to respond to the query, he can simply ignore it.

5.2 QuiltView Service

QuiltView is implemented as a web-based service in a single virtual machine at Amazon EC2 East. We expect this to provide adequate scalability for the near term. Standard load balancing and scaling mechanisms for web-based services can be used in the future to cope with increased load. In addition, since users and queries both exhibit significant spatial locality, standard distributed systems techniques can be used for partitioning and replicating the QuiltView service across multiple Amazon data centers.

The QuiltView service is available at `https://quiltview.opencloudlet.org`. Glass clients use SSL to communicate with this service. To distinguish between Glass devices, we use a device serial ID provided by the Android SDK. Our current implementation assumes that each Glass device is unique to a specific user — a reasonable assumption for such an intensely personal device. The device-user binding is established when a user registers with the QuiltView service. Only registered users can pose queries and provide responses. After registration, users authenticate via Mozilla Persona [11]. This decentralized authentication system, based on the BrowserID protocol, allows a user to verify his/her identity via a participating email provider's OpenID or OAuth gateway. No new password creation is involved.

The QuiltView server is a Django application that provides a web front-end to QuiltView users. The back-end is a MySQL database that stores the global catalog shown in Figure 1. The tables of this database contain information such as user registration details, user preferences, YouTube video links, query history, response history, query rewards and credits, and per-user timing and response quality information (for future user-reputation extensions to QuiltView).

5.3 Query Workflow

The QuiltView user interface is in two parts. First, the user defines the geographic scope of a query by zooming into an appropriate region of Google Maps as shown in Figure 3(a). The zoom level has to be 15 or deeper, corresponding to a roughly 3-mile by 2-mile bounding box on a typical desktop monitor. A *location share link* on this map encodes the latitudes and longitudes of its bounding box, as

Queries	Matched Results
Is there a thunderstorm?	What is the weather? Is it raining? Is it cold outside?
How is the thesis defense?	Are the professor's questions hard?

(a) Examples of True Positives

Queries	Wrongly Matched Results
Is it sunny?	What is happening in India?
Is the new year celebration fun?	Is the Mardi Gras exciting?

(b) Examples of False Positives

Queries	Missed Results
How is the party?	Is anyone drunk?

(c) Examples of False Negatives

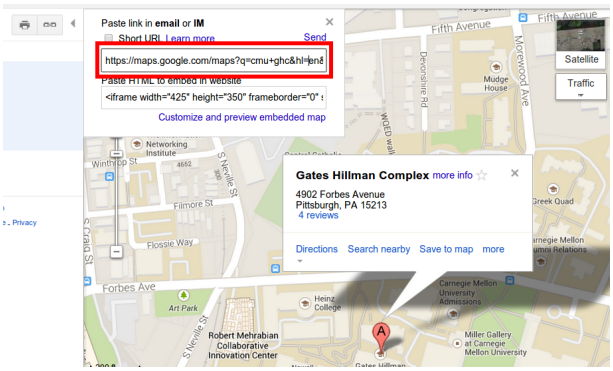
Table 1: Approximate Query Matching Algorithm

well as the zoom level and other relevant details. The user copies and pastes this link into the query interface shown in Figure 3(b). He then types the text of his query, uploads any associated image (such as a picture of a missing child or pet), and adds final details such as reward offered and desired timeliness of responses.

Once the query is submitted, the QuiltView server first checks to see if there are any relevant cached results. This involves a query similarity check (described in Section 5.4) and a timeliness check based on requester's preference of accepted staleness (Figure 3(b)) to eliminate obsolete cache hits. The query content of relevant hits are presented to the user, along with the YouTube URL of responses for each. The user can decide whether one of these queries is close enough to his query, and whether any of the returned responses are adequate. In that case, the query terminates without contacting any Glass users. Otherwise, the user can force delivery of the query to Glass users, and they can optionally respond as described in Section 5.1. How the right set of users is selected will be described in 5.5.

5.4 Query Similarity

For result caching to be effective, it is important to be able to detect queries that mean roughly the same thing. Otherwise there will be very few hits in the cache because two users are unlikely to phrase the same query using exactly the same text string. Hence, "similar" in this context means semantically close, not literally identical.



(a) Location URL from Google Maps

(b) Querying Interface

Figure 3: Composing a QuiltView Query

Query similarity detection is a very deep and open-ended problem, with roots in natural language processing, machine learning, and artificial intelligence. As a proof of concept, our prototype builds on an open source framework for unsupervised semantic modeling called *Gensim* [14] and uses the LDA (Latent Dirichlet Allocation) model for topical inference. Using a text corpus that is about 9 GB in size when compressed and encodes the entire English edition of Wikipedia, QuiltView gives an acceptable quality of similarity detection at reasonable speed.

Table 1(a) gives some examples of cache hits that are true positives; that is, many users would accept cached video responses to the queries on the right as acceptable results for the query on the left. Table 1(b) gives some examples of false positives; that is, QuiltView indicates similarity but is wrong. Fortunately, the query workflow ensures that a false positive is self-correcting. The queries corresponding to the erroneous hits are presented to the user, who immediately dismisses them and insists on QuiltView obtaining fresh results. Erroneous hits never lead to insidious use of wrong results. Table 1(c) gives some examples of false negatives; that is, QuiltView indicates no hits but the cache contains some matching queries. This case only represents a lost opportunity; QuiltView is unable to take advantage of the cached results and unnecessarily contacts users.

5.5 User Load Balancing

In distributed systems, "load balancing" typically refers to the utilization of server machines. In QuiltView, however, it refers to the cognitive burden placed on users who receive queries. Even if a user chooses not to respond, the distraction caused by receiving a query can be significant.

User preferences about receiving queries may vary considerably, depending on the individual, time, location, query topic, and reward offered. For example, a user might not want to receive any query at specific locations such as home or library, or anywhere after 9pm. QuiltView provides for flexible expression of such user preferences using a JSON specification. A user can specify a maximum number of queries that she/he is willing to receive for a given time (e.g., 5 queries per day). To meet this preference, the system keeps track of the query sent time and responded time for each user. When a new query is posed, QuiltView first identifies relevant Glass clients based on their locations and the geographic scope of the query. Then, it factors in their preferences to obtain a smaller set of eligible users. From this subset of eligible users, QuiltView randomly chooses the desired number of users and delivers the query to them.

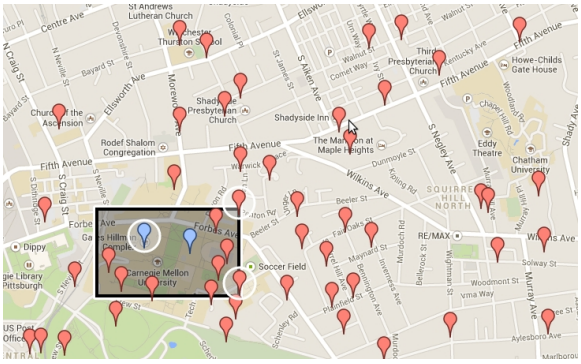


Figure 4: Synthetic and Real Glass Users

5.6 Synthetic Glass Users

QuiltView is designed for a future when there will be hundreds or thousands of Glass users in a city-size area. At present, only very limited numbers of Glass devices are available. To accelerate our experience with QuiltView mechanisms under stress, we have developed Python software that emulates a human Glass user. A QuiltView server cannot tell whether it is interacting with a real user or a *synthetic user* — the network protocol behavior of the two are identical. To return a video in response to a query, a synthetic user randomly chooses one from a set of `.mp4` files. We provide a tool to create and randomly distribute a specified number of synthetic users over a geographic area specified by a location link from Google Maps. Just like real users, these synthetic users connect with the QuiltView service and respond to queries. They do not, however, pose any queries.

Figure 4 shows 50 synthetic users (red markers) along with 2 real Glass users (blue markers). The shaded area identifies the set of users who are within the geographic scope of a query. Within this area, QuiltView delivers the query to a subset of users (real or synthetic) as described in Section 5.5. Synthetic users randomly decide whether to respond, and their responses are combined with those from live users. At present, synthetic users do not move; a future extension would be to include a mobility model such as the random waypoint model [2].

Using synthetic users will enable us to study the scalability of the QuiltView infrastructure without waiting for the widespread deployment of Glass-like devices. It will also help us to create reproducible benchmarks that can help us

compare alternative design choices and to evolve the implementation for improved scalability. Such experimentation can prepare the way for large-scale field studies with live users, which will be the ultimate test of QuiltView.

6. RELATED WORK

QuiltView is unique in its use of video for crowd-sourced responses to queries. As discussed in Section 3, many subtleties that would be lost in a verbal response are fully communicated in a video. At the same time, a video response is nearly effortless for a Glass user. Through a system of result caching, geolocation and query similarity detection, QuiltView improves scalability and shields users from being overwhelmed by a flood of queries. We are not aware of any other system with this unique combination of attributes.

Closest in spirit to QuiltView is a now-defunct system called Aardvark [17, 15] that connects users to a social network of friends and family who can respond to questions. Queries and responses are both in text, with no use of images or video. As in QuiltView, MicroBlog [5] users receive queries relevant to their location. However, the geo-tagged blogs they post in response are not micro-interactions. Further, the long response times for posting a blog entry suggest that queries are less likely to receive a near-real-time reply. QuiltView's per-query control over result freshness bears resemblance to the consistency model supported by LazyBase [3]. The query-driven and crowd-sourcing aspects of QuiltView bear resemblance to mobile micro-task services such as Field Agent, GigWalk, NeighborFavor, and TaskRabbit that have been studied by Musthag et al [12].

7. CLOSING THOUGHTS

QuiltView represents a new kind of crowd-sourcing system in which users return raw video as their response to a specific query. Such a system could, in principle, be built using smartphones or other mobile devices. However, the QuiltView workflow is designed to take advantage of the specific strengths of a Glass device, namely low user distraction and low cognitive load for this style of interaction.

The simplicity of recording and uploading a video in QuiltView is likely to raise privacy concerns. In our design, we have followed Google's guideline to have the Glass screen turned on while recording, so that people being recorded are aware. The procedure of touching the Glass device before recording is explicit confirmation from the Glass wearer that he is willing to share the current scene. In the future, Glass users may, based on their preferences, share video clips only with selected friends as in Google+ or Facebook. Automatic denaturing, as described in GigaSight [16], can also be helpful. This mechanism removes sensitive parts of a scene, such as human faces, before uploading.

We have built a working prototype of QuiltView, as described in Section 5. Our next step (subject to availability of sufficient number of Glass devices for real users) is to validate our implementation in live use. We envision a combination of field studies and lab-based experiments. Through this empirical process we hope to answer many questions.

For example, what are appropriate incentive models for participation in QuiltView? How annoying and distracting do users find queries that appear in their Glass displays? Can we use audio cues to minimize distraction by picking an optimal moment to present a query to a user? How constrained do queries have to be for easy response and scalability? What type of queries do people like to post in the real world? How close to real-time can we make the query-response loop? How rich a spectrum of opt-in preferences do we need in practice? How effective is result caching? What are typical time scales over which cached results are useful? How energy efficient is QuiltView?

These are, of course, only a subset of the many questions that come to mind about QuiltView. As is typical in experimental research, the early results and insights from our system will serve as a guide to further exploration. While we may not be able to answer all these questions definitively, we expect to gain valuable insights into this novel use of Glass and create a disruptive technology for mobile computing.

8. REFERENCES

[1] J. Anhalt, A. Smailagic, D. Siewiorek, F. Gemperle, D. Salber, S. Weber, J. Beck, and J. Jennings. Toward Context-Aware Computing: Experiences and Lessons. *IEEE Intelligent Systems*, 16(3), May/June 2001.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Dallas, TX, 1998.

[3] J. Cipar, G. Ganger, K. Keeton, C. B. Morrey III, C. A. Soules, and A. Veitch. LazyBase: Trading Freshness for Performance in a Scalable Database. In *Proceedings of the 7th ACM European Conference on Computer Systems*, Bern, Switzerland, 2012.

[4] Ellie Krupnick. Diane Von Furstenberg's Google Glasses Bring Geek-Chic to Fashion Week. *The Huffington Post*, September 2012. `http://www.huffingtonpost.com/2012/09/10/diane-von-furstenberg-google-glasses-fashion-week_n_1870028.html`.

[5] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. Micro-Blog: sharing and querying content through mobile phones and social participation. In *Proceedings of the 6th international conference on Mobile systems, applications, and services*, New York, USA, 2008.

[6] Google. Glass Development Kit. `https://developers.google.com/glass/gdk`, 2013.

[7] Google. Glassware Launch Checklist. `https://developers.google.com/glass/distributing/checklist`, 2013.

[8] Google. The Google Mirror API. `https://developers.google.com/glass/`, 2013.

[9] Google GCM. `http://developer.android.com/google/gcm/index.html`, 2013.

[10] P. Lukowicz. Head-Mounted Displays: From Cyborgs to Google Glass. *Wearable Technologies*, May 2013. `http://www.wearable-technologies.com/2013/05/head-mounted-displays-from-cyborgs-to-google-glass/`.

[11] Mozilla. Introducing BrowserID: A better way to sign in. `http://www.mozilla.org/en-US/persona/`, October 2013.

[12] M. Musthag and D. Ganesan. Labor Dynamics in a Mobile Micro-Task Market. In *Proc. of CHI 2013*, Paris, France, May 2013.

[13] T. Olson and J. Loquist. If a picture is worth a thousand words, then a video is worth a million. `http://youtube-global.blogspot.com/2010/10/if-picture-is-worth-thousand-words-then.html`, October 2010.

[14] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010.

[15] M. G. Siegel. The Killings Continue at Google: Aardvark Put Down. *TechCrunch*, September 2011. `http://techcrunch.com/2011/09/02/google-kills-aardvark/`.

[16] Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Satyanarayanan, M. Scalable Crowd-Sourcing of Video from Mobile Devices. In *Proceedings of the 11th International Conference on Mobile Systems, Applications, and Services (MobiSys 2013)*, Taipei, Taiwan, June 2013.

[17] Wikipedia. Aardvark (search engine). `http://en.wikipedia.org/wiki/Aardvark_%28search_engine%29`, 2013.