The Open INcentive Kit (OINK): Standardizing the Generation, Comparison, and Deployment of Incentive Systems

Noah Klugman
University of California, Berkeley

Santiago Correa
University of Massachusetts, Amherst

Pat Pannuto
University of California, Berkeley

Matthew Podolsky
University of California, Berkeley

Jay Taneja
University of Massachusetts, Amherst

Prabal Dutta
University of California, Berkeley

ABSTRACT

Incentives are a key facet of human studies research, yet the state-of-the-art often designs and implements incentive systems in an ad-hoc, on-demand manner. We introduce the first vocabulary for formally describing incentive systems and develop a software infrastructure that enables UI-based graphical generation of complex, auditable, reliable, and reproducible incentive systems. We call this infrastructure the Open INcentive Kit (OINK). A review of recent literature from several communities finds that of the one hundred and twenty-one publications that incorporate incentives, only thirty-one describe their incentive system in detail, and all of these could be implemented using OINK. We evaluate OINK in practice by using it for an active energy monitoring deployment in Ghana and find that OINK successfully facilitates thousands of individual incentive payments. Finally, we describe our efforts to generalize OINK for different research communities, specifically focusing on architectural decisions around extensibility to support unanticipated use cases. OINK is free and open-source software.

ACM Reference Format:
Noah Klugman, Santiago Correa, Pat Pannuto, Matthew Podolsky, Jay Taneja, and Prabal Dutta. 2019. The Open INcentive Kit (OINK): Standardizing the Generation, Comparison, and Deployment of Incentive Systems. In *THE TENTH INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGIES AND DEVELOPMENT (ICTD '19), January 4–7, 2019, Ahmedabad, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3287098.3287101

1 INTRODUCTION

Experiments involving human subjects are common in a variety of fields, including psychology, economics, and computer science, among others. These experiments often encourage user participation by providing incentives to participants [13, 29, 51, 62]. It is known that incentive system design choices can have significant effects on the results of a study [5, 13, 21, 53], yet incentive systems receive relatively little discussion in the literature and, even when they do, there is no standard vocabulary to describe them [25, 53].

While information technology (IT) has revolutionized many aspects of the scientific enterprise—innovating classical measurement techniques such as surveys [24] and improving data analytics [23, 56]—incentive systems have received little attention. Each experiment that uses incentives requires the researcher to develop methods to track when incentives are due, transfer the incentive, and provide accounting. In some cases, the cost of implementing a study-specific incentive system might be small—it may be easy to pay an individual for their participation in a survey—but implementation costs increase with the complexity of incentive triggers and the size of the participant population.

To facilitate the description and comparison of incentive systems in the literature, and to reduce their experimental overhead while increasing their quality, we introduce the **O**pen **IN**centive **K**it (OINK), the first work to propose a standard vocabulary for describing a generic incentive system and to support and automate the design, deployment, and management of end-to-end incentive systems.

OINK stems from the insight that very different studies share similar incentive system patterns. For example, a study that incentivizes a community health worker to visit a village (as reported from a mobile app tracking GPS) has a similar pattern to a study that incentivizes a driver to observe the speed limit (also as reported from a mobile app tracking GPS). OINK identifies and implements a core infrastructure that is common to many incentive systems, and offers a simple interface for implementing and sharing experiment-specific modules with other OINK users.

OINK supports high-quality, modular incentive systems for a range of simple to complex applications with minimal software programming. The system enables turn-key experiment planning and monitoring, including detailed transaction logging, visualizations of error states, and participation trends.

While incentive systems are used in a broad array of research settings, we initially introduce OINK as a tool for supporting research in development contexts, which have a history of innovative IT applications [27]. In particular, these settings challenge incentive systems because they have fewer sources of data or sensing devices and researchers may be working remotely with initially limited understanding of the context. These settings particularly benefit from the capabilities of OINK, including adapting to mobile money systems and simulating or modifying incentives to achieve a balance between effectiveness and manageability of incentives.

OINK Requirements

Reliable and Auditable. There is an ethical responsibility to provide all incentives promised and to ensure research funding is properly spent [11, 21]. An incentive system must always transfer incentives when expected. Further, it must provide bookkeeping to prove incentives are delivered as expected.

Broadly Accessible. The abstractions in this architecture must minimize the amount of work to generate incentive systems that fit the needs of a study. Anyone with a base level of technical literacy must be able to use the system without writing large amounts of code.

Extendable. The architecture must allow for functionality to be developed by the community to suit specific applications. Further, to ensure wide applicability, these extensions must be shareable and iterable between users.

Scalable. Incentive systems must be easy to scale. Further, the computing costs to run the system should be low regardless of the scale of an experiment.

Reproducible. When either the original designer or a third party replicates a study, the incentive system must not need to be reimplemented.

Secure and Accessible. Incentive systems must prevent the unauthorized transfer of incentives. Additionally, incentive system administration must be accessible to researchers without requiring low-level server administration.

2 BACKGROUND

We consider work that explores the impact and regulation of incentive systems as well as work that demonstrates similar examples of IT being applied to human subjects research. OINK is the first work to explore the principled generation and deployment of incentive systems.

Why Do Incentive Systems Matter?

Incentive system design is researched in many disciplines. The type of incentive chosen, the stimulus for incentive, and the method of incentive delivery have all been shown to have effects on presentation [13], selection bias [29], validity [9], response bias [51], as well as influencing other aspects of experimental design [5]. Thus, knowing more about an incentive system should be viewed as important in understanding a result [29]. OINK aims to make it easier for a community to include discussions of incentive systems design and implications by providing a common vocabulary for the literature.

Incentive System Requirements and Policy

In the United States there are federal guidelines provided for human subject incentive systems [36], and universities adopt their own additional guidelines in their respective Institutional Review Boards (IRB) [11, 38]. Because there is no standard best practice, OINK does not guide users towards a specific ethical incentive system design; instead it leans on the standard practice of IRB submission, comments, and revision. A principled system like OINK opens up the possibility for the export of standard-language descriptions of any incentive systems constructed with it to be submitted to an IRB, potentially removing hurdles towards regulatory approval.

Similarly, many different governments enforce regulations regarding the electronic transfer of funds [6, 8, 35, 37, 50]. OINK assumes that any mobile money payment API invoked will bear the regulatory responsibilities.

Bringing IT to the Scientific Method

Information technology is already applied to human subjects experiments. A somewhat similar service to OINK is Amazon Mechanical Turk (MTurk), which provides a structured way to pay "Workers" for completing "Human Intelligence Tasks" (HIT) [1]. Unfortunately, MTurk does not allow for easy enrollment of participants and therefore could not easily be repurposed for a study that requires the enrollment of a specific population. Further, MTurk provides only the single stimuli (completing a HIT) which does not cover the full set of possible incentive systems. For example, if a study has a non-monetary incentive like providing a cookstove, MTurk does not provide a mechanism to facilitate this transfer.

IT is improving the generation and deployment of data collection systems, whether they are based on surveys [24], constrained geospatially [40], or incorporate sensors [7]. Open Data Kit (ODK), a suite of tools for data collection on mobile devices, has been adopted across multiple communities, and is used everywhere from 42 countries to the International Space Station. [39]. OINK takes its name as an homage to ODK. We hope this work identifies a similarly powerful abstraction for incentive systems as ODK described for data collection.

3 DESIGN AND IMPLEMENTATION

Before designing OINK, we derive a generic incentive system vocabulary from a literature review (see "Incentive Systems in the Wild"). This vocabulary then informs a generic incentive system architecture, the implementation of which we call OINK. We explain how each of the words in our vocabulary is implemented in OINK and introduce the enabling technologies. Finally, we share wireframe designs of the OINK user interface to illustrate how we envision non-computer scientists interacting with the system.

Incentive System Vocabulary

We propose a vocabulary to describe the generic incentive system. To derive this vocabulary, we review 390 papers and abstracts to identify descriptions of incentive systems in any detail. We then identity the minimal vocabulary needed to describe each incentive system end-to-end. Finally, by taking the intersection of each of these descriptions, we can identify words and construct our vocabulary:

- **Stimulus:** This is the trigger event for a transfer of an incentive. It contains information about how much incentive should be transferred on event and to whom. Example stimuli include the completion of a survey or the keeping of an app on a mobile phone for a pre-defined period of time.
- **Payment:** Payment is the logic to transfer an incentive. Example payments include a call to the PayPal API or the manual transfer of a good to a participant.
- **Identity:** Identity is maintained for both participants and field officers (when applicable). Example usage include ensuring that incentives are transferred to the correct person using the correct method of transfer, tracking participant consent, and keeping track of the behaviors of the research staff either for fraud detection.
- **Fraud:** Fraud is the undesired transfer of an incentive. Fraud detection ensures the transfer of an incentive only after verification conditions has been reached. Example verification conditions include setting a maximum number of hourly incentives or stopping transfer of incentives to participants located outside a pre-defined geographic area.
- **Alarm:** An alarm triggers if a condition within the incentive system is been met. Example conditions include payment error, system downtime, fraud, and budget exceeded.
- **Incentive System:** An incentive system is a system that keeps track of when a given amount of incentive should be sent (using stimuli and fraud detection functions), what incentive should be sent (using a payment function), to whom an incentive should be sent, and whether an incentive was transferred (using an alarm function on error).
- **Experiment:** An experiment is a collection of one or more incentive systems that share common participants.

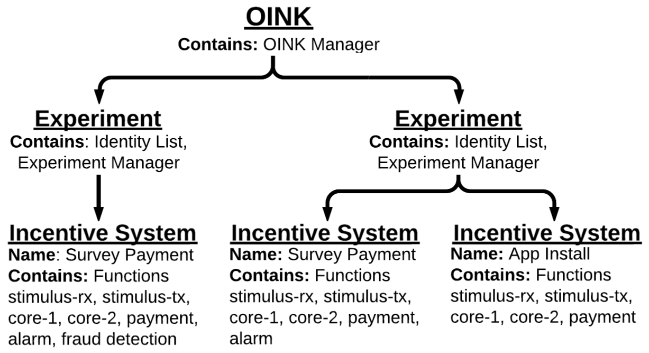

OINK
Contains: OINK Manager
Experiment
Contains: Identity List, Experiment Manager
Experiment
Contains: Identity List, Experiment Manager
Incentive System
Name: Survey Payment
Contains: Functions stimulus-rx, stimulus-tx, core-1, core-2, payment, alarm, fraud detection
Incentive System
Name: Survey Payment
Contains: Functions stimulus-rx, stimulus-tx, core-1, core-2, payment, alarm
Incentive System
Name: App Install
Contains: Functions stimulus-rx, stimulus-tx, core-1, core-2, payment

Figure 1: OINK system hierarchy. OINK sits at the highest level, followed by experiments and incentive systems. The OINK and experiment levels include managers, which provide administration, creation, and visualization at the respective level. Identity management is scoped to the experiment layer. Incentive systems are made up of multiple functions. Each function implements words from the generic incentive system vocabulary gathered from the literature. The system runs in a functions as a service (FaaS) runtime.

System Architecture

We introduce the OINK system architecture, which contains a runtime, high-level layers to support the administration of groups of incentive systems, and low-level layers that implement each word in the generic incentive systems described in "Incentive System Vocabulary." We then revisit the requirements introduced in "OINK Requirements" and describe the design decisions made to ensure each are satisfied. The OINK system hierarchy is shown in Figure 1 and is described in this section from the top down.

Runtime. OINK is implemented using a functions as a service (FaaS) runtime. FaaS is a cloud-based architecture that gained commercial traction in 2014 [4]. It provides serverless computing, allowing computation to occur without configuring and hosting a long-running runtime environment. The primary programming abstraction for FaaS is a function, which is invoked, wakes up, computes, returns, and then is destroyed. Throughout the rest of the discussion, "function" refers to a function within a FaaS runtime.

The current OINK implementation uses the Google Cloud ecosystem, depending on Google Functions [15] for the FaaS architecture, Google Firestore for the datastore [14], Google Cloud Messaging for downstream message passing [16], and Google Firebase Invites for in-app invitations [17]. We chose to use the Google ecosystem because our experiment involves an Android app, which integrates well. Using Google tools for data storage and message passing allows for a single authentication for access to the full suite of tools. There is, however, no technical reason why OINK could not be implemented using competing FaaS providers.

One limitation of FaaS is that it does not support long-running processes. Because of this, the OINK manager and the experiment manager—both of which are UIs and need to be accessable for as long they are being interacted with—are implemented as a webapp. The webapp authenticates with the FaaS runtime and datastore backend to load data and spawn new functions. If the webapp crashes, the FaaS runtime will continue hosting any active incentive systems.

OINK-Level: OINK Manager. This manages multiple experiments and is the entrance into the OINK UI. It allows environment variables to be set, including top-level authentication credentials required for administrating OINK, and provides an interface to create experiments. OINK can organize experiments with an arbitrarily large and complex number of incentive systems. The OINK manager is implemented as a webapp and a wireframe design is shown in Figure 3 (a).

Experiment-Level: Experiment Manager. This contains multiple functionalities and is implemented as a page of a webapp. It is where a user generates incentive systems (shown in Figure 3 (b) and described in Section 3). It allows for high-level management of multiple incentive systems. For example, an experiment can have a budget, which if exceeded pauses all incentive systems in that experiment. It is where a user can simulate an experiment to see how much incentive will be consumed with different incentive system designs (shown in Figure 3 (c)). Finally, it is where visualizations of the performance of the incentive systems are located.

Experiment-Level: Identity List. Each experiment implements a single identity database that is used by all the incentive systems in the experiment. The identity database is default read/create/update-only, which helps ensure consistent identities across incentive systems. Each row in the identity table contains (at a minimum) a unique identifier, an array of valid payment methods, timestamps at creation and modification, and participant status (enrolled, inactive, unenrolled).

Incentive System-Level: Overview. A single incentive system is shown in Figure 2 and each part is described in detail below. Every incentive system requires the choice and parameterization of exactly one stimulus function and at least one payment function. The core functions are included with every incentive system and require no parameterization. Along with these functions, multiple alarm and fraud detection functions can be added, each of which require parameterization.

Incentive System-Level: Functions. Every incentive system is implemented as a group of functions, most of which are also words defined within the generic incentive system vocabulary. A function can be understood to be a standalone logical block within the incentive system. We introduce all of the function types below and their interactions in Figure 2.

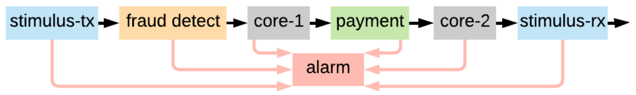
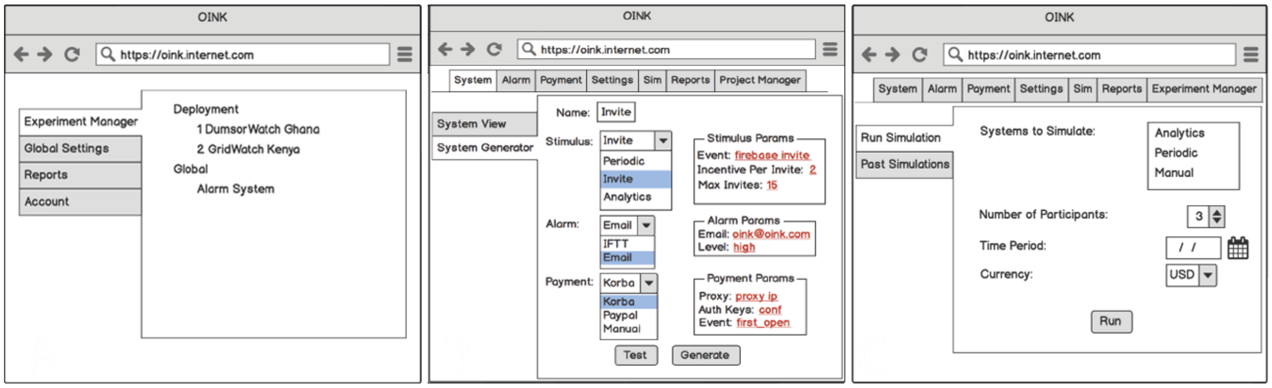


Figure 2: Simplified Incentive System Diagram. This diagram shows an OINK incentive system. The *stimulus-tx* function is triggered to start the transfer of an incentive. The request moves to the *fraud detection* function which accesses historic transactions to determine validity. It then moves to the *core-1* function which logs the request. The *payment* function transfers the incentive. Then the *core-2* function stores the result and wakes up the *stimulus-rx* function, which performs stimulus-specific cleanup such as notifying the user. At any point, any function can trigger an *alarm* function.

- **Core:** The core is the only area that does not contain words from the generic incentive system vocabulary. It is implemented by OINK and not modified by the user. It provides meta-functionality for all incentive systems including logging and error handling. There are two functions and two databases that make up the Core: function *core-1*, function *core-2*, database *transaction-tx* and database *transaction-rx*. Function *core-1* wakes up on write to the *transaction-tx* table by a stimulus function. The *transaction-tx* database holds transaction information and includes all information needed to trigger a payment function. The *core-2* function wakes up and logs to the *transaction-rx* table when a callback from a payment function returns, and will invoke the *stimulus-rx* function for any stimulus-specific reporting.
- **Stimulus:** A stimulus is a trigger that results in incentive transfers. Each incentive system contains only a single stimulus. OINK provides an interface for a stimulus to connect to an incentive system. For example, OINK provides a way for someone who has gathered a survey response to automatically trigger an incentive. The parameters for this stimulus would be the person to pay, the person who conducted the survey, the amount of money to pay, the desired method of payment, and when to pay (immediately, after upload and fraud check, etc.). This stimulus then exists over the lifetime of the incentive system and allows any authenticated user to initiate a transfer of an incentive.

  There are two functions required to implement each stimulus: *stimulus-tx* and *stimulus-rx*. The *stimulus-tx* function contains the logic to trigger OINK, the amount of an incentive that this trigger should release, who the incentive should be transferred to (as an index to the experiment-level identity table), how the incentive should be transferred, and how the result of a stimulus should be displayed back to the user. The *stimulus-rx* function writes the result of a payment into the *transaction-rx* database and optionally does stimuli specific clean up. For example, a stimuli generated from an action in a mobile app action might want to display a message back to the user after a successful payment in that app.



Name: Invite
Stimulus: Invite
Stimulus Params
Event: firebase invite
Incentive Per Invite: 2
Max Invites: 15
Alarm: Email
Alarm Params
Email: oink@oink.com
Level: high
Payment: Korba
Payment Params
Proxy: proxy ip
Auth Keys: conf
Event: first_open
Number of Participants: 3
Time Period: / /
Currency: USD

(a) The OINK manager. **(b) Generation of a incentive system.** **(c) Simulation of a incentive system.**

Figure 3: Three wireframes of the OINK system. (a) Shows the OINK manager with two available experiments (DumsorWatch Ghana and GridWatch Kenya). (b) Shows an experiment manager being used to generate an incentive system by selecting and parameterizing stimulus, payment, and alarm functions. This is generating the invite system (#2) from Table 2. (c) Shows the area in the experiment manager where the incentives consumed across multiple incentive systems can be calculated.

Functions implementing a small group of stimuli, including an HTTP REST API, IFTTT [41], a cron-based periodic system, and a Google analytics receiver [18], are provided in the OINK codebase as examples to demonstrate standard ways of triggering the incentive system.

- **Payment** A *payment* function provides logic to transfer an incentive to a user. For example, a *payment* function could be a wrapper to the third party payment API. As there is a one-to-one mapping of a stimulus to a payment, there is only a single *payment* function per incentive system. The inputs to a *payment* function include the amount of the incentive, the type of incentive, the time a payment was attempted, the reason an incentive is being generated, and the identity of the receiver. The output from a *payment* function is a confirmation that a given incentive transfer occurred or an error message and is collected by the *core-2* function and written to the *transaction-rx* table. Additionally, the *payment* function can be parameterized as a pass-through, allowing for an experimenter to opt-out of transferring incentives, which can be useful for testing. A starting group of *payment* functions, including the Korba airtime distribution API [26] and a manual payment over SMS, are implemented in the OINK codebase to demonstrate standard ways of constructing a variety of different types of *payment* functions.
- **Fraud Detection** The *fraud detection* function contains rules that can prevent a stimulus from triggering an incentive in order to prevent unattended transfer. For example, if an experiment detects T more incentives than expected, incentives can be paused. Further, fraud detection functions can access any of the databases in OINK to query historic state for undesired patterns. *Fraud detection* functions can query OINK databases for historic state. Multiple *fraud detection* functions can be defined for a single incentive system.
- **Alarm:** Any part of an incentive system can trigger an alarm. The input to an *alarm* function is a message and priority, and the output can be an API, webhook, or an internally-provided function to generate and transmit a notification. *Alarm* functions can be useful both for error tracking and for periodic updates to the research team. Multiple *alarm* functions can be included per incentive system. A small group of *alarm* functions are implemented in the OINK codebase including email and SMS [57].

Revisiting Requirements. We revisit each system requirement in "OINK Requirements" and describe how these requirements are satisfied by the OINK design.

Requirement: Reliable and Auditable. We implement OINK on top of the Google FaaS runtime [15]. This bootstraps reliability off Google who, like each of three largest commercially-available FaaS runtime providers, employ dedicated staff to keep their FaaS runtime operational. Recognizing that the FaaS runtime creates a log each time a function is invoked and destroyed, we design OINK to invoke a new function whenever data is moved through an incentive system. This ensures a record at the entry and exit point of each stimuli, fraud, payment, core, and alarm function.

Requirement: Broadly Accessible. The FaaS runtime by definition removes the need for the configuration and maintenance of a long-running server to host OINK, reducing the barrier of entry for non-technical users. Further, we designed OINK with input from non-computer science researchers and the first deployment of OINK supports an interdisciplinary project. We received early feedback on the usability of OINK based on wireframes of the user interface (shown in Figure 3) and will focus on usability as future work.

Requirement: Expandable. To support large numbers of different incentive system configurations, OINK is designed so that each word in the incentive system can be completely customized. This is done by coding a function for that word, which can be a technically challenging task. Thankfully, once a custom function is written, the FaaS runtime allows for it to be shared between users to be incorporated into their incentive system design. We hope that once popular functions have been implemented once, it will be unlikely that a user will have to implement an extension to OINK. We are motivated by the emergence of people sharing useful FaaS functions [33].

Requirement: Scalable. Every commercially available FaaS runtime operates at scale and supports millions of function invocations out-of-the-box. This allows OINK to scale without user intervention. As more incentives are transferred, more functions will simply be called. Because every major FaaS platform allows at at least one million free invocations monthly [2, 15, 30], OINK will experience minimal operating costs for many experiments.

Requirement: Reproducible. OINK can be reproduced without requiring any low-level server configuration. This is possible both because the FaaS runtime will be the same for all users, and because all major FaaS providers provide databases and virtual servers that can be launched pre-configured to host the OINK databases, OINK manager and experiment manager webapps. Further, each part of OINK is independently reproducible because the same function will run on any FaaS runtime (requiring a lightweight adaptation layer to the specific FaaS runtime environment).

Requirement: Secure and Accessible. OINK bootstraps security and access control off the authentication system of both the FaaS service and the payment API. Each of the three FaaS services evaluated provide abstractions to protect and provide levels of access [2, 15, 30]. By keeping all functions and logs within this system, data generated and the code being run are secured by the FaaS service provider. Some stimuli themselves can bootstrap security with these same protections; for example, an app developed with Firebase Cloud Messaging can use Google APIs to authenticate with an OINK system using Google Functions as its FaaS backend out-of-the-box [16]. OINK bootstraps payment security off methods implemented by payment APIs which may include OAuth [48, 52], HTTP basic auth [47, 54], HMAC signing [26, 59], whitelisting [26], and certificates [43].

4 EVALUATION

We evaluate incentive systems in the wild by performing a literature review of 175 papers across three development-related ACM communities. We also consider work containing descriptions of incentive systems discovered from an examination of 215 publications from medical journals [53]. We evaluate the performance of OINK under an ongoing deployment managed by the authors where participants are incentivized for nine different types of activities. OINK is being used to implement all incentive systems for this deployment and has issued over three thousand incentive transfers. Finally, we select the most complicated incentive system found in our literature review — taken from *A Feasibility Study of an In-the-Wild Experimental Public Access WiFi Network* which appeared in the fifth ACM DEV — and discuss how OINK could be configured to support this work [49].

Incentive Systems in the Wild

To uncover the taxonomy of incentive systems in the technology for development community, we perform a literature search.[1] We select papers using the Google Scholar query "source:Computing source:Development" and select the first 150 papers [20]. We scan each paper to decide whether they had no incentive system (papers using existing datasets, wireless network protocols, etc), or if an incentive system was present. We also select prior CHI and ICTD proceedings selected based on titles that seem likely to have human factors research. Finally, we select papers that contain incentive system descriptions from the medical community as identified by Stovel et al. [53]. We then classify each paper's incentive system as either triggered by a survey or by a deployment. This was done because of the intuition that a survey might have a simpler incentive system that would be less likely to be reported in the literature (our review refuted this). This line can be fuzzy at times and some works do both. For these cases, we make a subjective decision and place the work in a single category. We further categorize works based on their discussions of the incentive system: no mention, some mention, or discussion. When a paper mentions giving someone technology or asking someone questions but does not go further, we consider this "no mention." When a paper mentions the work being done in a school or implies that participants volunteered, we classify as "some mention." When a paper mentions either what people received or specifically what triggered an incentive, we classify as "discussion." Not a single paper talked about how the actual incentive was transferred. The results of this classification are summarized in Table 1.

From Table 1 we can make a couple of observations. Broadly, we find a majority of papers with incentive systems do not describe these systems either at all or beyond trivial details. Of all papers found involving human subjects, 32% contain at least a minimum amount of information about their incentive system and 26% speak in detail. Grouping this by communities, we see that out of the CHI and ICTD papers (selected for

[1]The full citation of the papers selected and their classifications can be found at https://www.github.com/lab11/OINK/papers/ictd18/citations.csv.

Type	Survey			Deployment			Both
	No Mention	Some Mention	Discussed	No Mention	Some Mention	Discussed	No Human
Papers	15	11	22	19	18	5	27
Abstr	8	1	2	9	9	2	36
Total	23	12	24	28	27	7	63

Table 1: Discussion of incentive systems found in 390 publications: 150 ACM DEV papers and shorts, 215 medical education research papers, 3 ICTD Papers, and 22 CHI Papers. Note the small number of papers that discuss incentive systems in detail. For each paper that discusses an incentive system, we find that the OINK incentive system vocabulary can fully describe the incentive system presented.

our review based on titles suggesting human studies), 84% had human subject studies. Of these, 28.5% mention their incentive system and 42.8% have a discussion of the incentive offered. We discover 59 papers (39%) from the set of ACM DEV papers (which were selected for our review regardless of title) that have some amount of human subject work, 21% of which mentioned incentive systems and only 6% of which described incentive systems thoroughly.

Scenario 1: DumsorWatch Pilot

DumsorWatch is an experiment in Accra, Ghana where participants are surveyed in-person, a mobile app is installed on participants’ everyday-use Android smartphones. and grid reliability sensors are installed in some households. All incentives are provided as airtime top-ups.

Experiment Walkthrough. Table 2 provides an overview and explanation of all incentive systems within the experiment, each of which are described using the OINK vocabulary. This experiment demonstrates the breadth and complexity of the incentives enabled by OINK.

Incentive Walkthrough. We take a closer look at a single incentive system: app_invite (incentive system 4 in Table 2). This incentive system transfers credit to a user when they use the DumsorWatch app to invite an additional participant to download the app. For each function we discuss its implementation and break out its parameters.

Stimulus-TX: Firebase Invites provide a turn-key mechanism for sending download invitations to contacts in a phone either over email or SMS [17]. When the user triggers an invite in the app, the DumsorWatch app writes to a Firestore collection using the Firebase Android library [14]. An on_create trigger on this table wakes up the app_invite *stimulus-tx* function. This function contains the logic of the identity of the person receiving the incentive, the method of transfering the incentive, the amount to pay per invite and the threshold number of invites allowed per user.

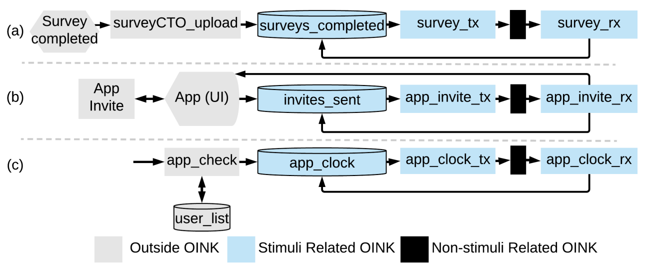


Figure 4: Three different stimuli from DumsorWatch. In (a) a completed survey is uploaded which writes to a collection (surveys_completed) and wakes up the survey_tx *stimuli-tx* function. In (b) a user sends a download invitation in an app, writing to a collection (invites_sent) and waking up the app_invite_tx *stimuli-tx* function. Finally, in (c) a cron job (app_check) wakes up on a timer, checks to see how long users from the identity table have been active, and then writes users active longer than a threshold to a collection (app_clock) waking up the app_clock_tx *stimuli-tx* function. The grey shapes show logic that occurs outside of OINK and demonstrate that different stimuli interface with OINK using a common interface, a single write to a collection. The blue shapes show the OINK stimulus logic, and the black box represents the rest of the incentive system. Mapping these to Table 2, (a) is #9: surveyCTO, (b) is #2: app_invite, and (c) is #3 or #5: keeping either the app (app_clock) or sensor (sen_clock) installed for a period of time.

Fraud: The *fraud* function detects if a user has triggered an amount of invites greater than allowed. If this threshold is not hit, the fraud block transfers data to the core and marks the stimulus collection "enqueued". If the *fraud* function rejects a transaction, it marks the stimulus database as “fraud”, sends an alarm, and stops the transfer of data.

Core-1: The *core-1* function receives a payment record and is responsible for logging, error handling, retries, and guaranteeing that the record will be updated upon success or failure of incentive delivery.

Payment: This incentive system uses the Korba service to deliver incentives. The Korba *payment* function is responsible for mapping the generic payment information from OINK (whom and how much) to the format expected by the Korba API [26]. In practice, the Korba API exposed a limitation of the FaaS architecture. For security, Korba requires that all payment requests come from a whitelisted IP, and FaaS systems do not support static IPs. As a workaround, we deploy a dedicated proxy server for OINK requests to Korba.

Core-2: The Korba API includes a callback URL, which calls the *core-2* function upon payment completion. If the response is successful, it updates the transaction database and invokes a *stimulus-rx* function. If the payment failed, users can configure several responses. For this incentive system, the payment system is configured to retry after a delay up to five times, after which point payment is paused and the *alarm* function is called to notify the research team.

	1	2	3	4	5	6	7	8	9
name:	app_install	app_invite	app_behavior	sen_install	sen_behavior	sen_collect	debug	FO_manual	survey
stimulus-tx:	first_open	app_invite	app_clock	sen_query	sen_clock	sen_pick	debug	FO_manual	surveyCTO
fraud detection:	user_thresh	user_thresh	none	none	none	user_thresh	none	user_thresh, FO_thresh	user_thresh, FO_thresh
payment:	Korba	Korba	Korba	Korba	Korba	Korba	none	SMS	Korba
stimulus-rx:	first_open	app_invite	app_clock	sen_query	sen_clock	sen_pick	debug	FO_manual	surveyCTO
alarm:	email	email	email	email	email	email	email	email	email

Table 2: The DumsorWatch experiment incentive systems described by the OINK vocabulary. Each column is a single incentive system used in the DumsorWatch deployment. Each row is a function hosted with the FaaS architecture. The incentive systems send airtime when: (1) installing an app for the first time, (2) a user invites a friend to install the app, (3) the user keeps the app installed, (4) a sensor is installed in a household, (5) a sensor is kept in a household, (6) a sensor is collected, (9) a survey is completed. Additionally, (7) allows for debugging and (8) lets a field officer manually transfer airtime for unanticipated reasons. The user_thresh and FO_thresh fraud detection functions ensure no more than T transactions occur for a user or field officer.

Stimulus-rx: This function is responsible for handling the completion (or failure) of a *payment* function. For the app_invite incentive system we want to notify the user that sent an invite that their payment has been disbursed, so the *stimulus-rx* function spawns a message back to the app using Firebase Cloud Messaging [16].

Alarm: The *alarm* function takes a message and sends an email to the project team. Eventually, we envision a more nuanced set of alarms, with varying priorities and mechanisms.

Other Incentive Systems in DumsorWatch. We briefly discuss parts of the other incentive systems in DumsorWatch, each of which is shown in Table 2. In (1) the *stimulus-tx* function first_open triggers when an app is first installed. The *fraud detection* function user_thresh only allows a user to receive a "first install" incentive once by looking at historic data on the user. The *payment* function Korba uses the Korba Payment API to send airtime topup [26]. On a successful payment response from the Korba API, the first_open *stimulus-rx* function sends a confirmation message to back to the app. The *alarm* function email sends an email if any part of this chain fails. (3) and (6) both use cron-jobs to pay incentives after users have kept either the app or the sensor installed for a period of time. (7) shows a configuration that allows for the ad-hoc manual transfer of an incentive in a principled manner, ensuring that this transfer is logged. (8) and (9) use two fraud-detection functions; one that checks the recipient and one that checks to behavior of the field officer (i.e. don't let a field officer report more than N surveys a day). Finally (8) demonstrates how a field officer can send an SMS to create a record in OINK after manually transferring an incentive.

Field Performance. OINK has transferred nearly three thousand incentives during three months of deployment. It has had experienced 100% uptime. Further, it has proven to be fault resilient in other ways, with the alarm functions letting the team know quickly when there are problems (a common problem turns out to be the reliability of the Korba API) by sending emails with a targeted error message and an index to trace the lower-level logs. Over the course of the deployment there have been multiple reasons to audit particular payments, each of which OINK has alerted the research team about and provided sufficient logs to address. The research group is interdisciplinary and includes an graduate student in economics, who has been able to interact with OINK for day-to-day management activities. This deployment continues to scale and the research team will continue to use OINK.

Scenario 2: A paper from the ACM DEV literature

We infer the design of an incentive system from the description in "A Feasibility Study of an in-the-wild Experimental Public Access WiFi network" [49]. This paper explores constraints in providing free public WiFi network in a low income neighborhood in a medium-sized British city. We choose this paper from our survey of incentive systems because it contains the most complicated incentive system we were able to find. Figure 5 shows our mapping of this incentive system to OINK. The authors describe their incentive system as follows:

> For their time and the inconvenience, participants were offered compensation in the form of shopping vouchers. . . The size of compensation was scaled in line with the inconvenience suffered: sharers who made their broadband available to others. . . received higher compensation (100 pounds) than those citizens who simply used the. . . service (50 pounds).

Incentive Walkthrough. One possible incentive system design supporting this work imagines a web-form being used as a manual stimuli. A field officer enters the participant ID, their ID, whether a participant shared their broadband, and how much they shared. When the web-form is submitted, it would generate a HTTP POST, which OINK would receive using a *stimulus-tx* function as pass to the *payment* function. The *payment* function then would generate a SMS message to the field officer with the correct amount of voucher to transfer.

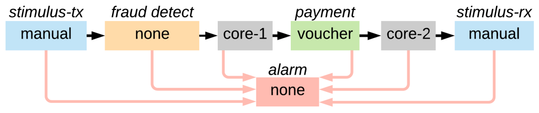


Figure 5: Incentive System from "A Feasibility Study of an in-the-wild Experimental Public Access WiFi network" [49]. One design using OINK to the support the incentive system described in [49], which transfers vouchers to participants based on the amount of WiFi bandwidth they share.

To finish, the field officer enters the participant ID, their ID, and whether the incentive was transferred into the web-form. On submit, OINK would receive another HTTP POST using a *stimulus-rx* function and mark the transaction complete. This full incentive system is shown in "Figure 5."

What Could OINK Have Added? This paper does not talk about fraud detection or alarms. We imagine a basic *fraud* function to ensure that no user is paid twice. Similarly, we imagine a *alarm* function that sends an email if the transfer of incentives fails or if fraud is detected.

OINK allows for higher frequency payments than the authors used. For example, the experiment only paid people at the end. Perhaps multiple midpoint payments would have improved participation rates [5]. Similarly, OINK allows for higher resolution stimuli. For example, the experiment also only had two voucher levels; perhaps more levels would have uncovered more types of public access WiFi users [60].

5 DISCUSSION AND FUTURE WORK

Architecture Limitations

New Features Require Custom Code. Stimuli and payments require a fair amount of software development to first be incorporated into OINK, which will lead to technical hurdles for early adopters. For example, the app-invite described in Table 2 contains over a hundred lines of code written specifically for that stimulus. Similarly, the Korba payment function took considerable engineering effort learning and debugging with the Korba API. We imagine that as OINK matures, a set of templates will emerge to simplify this process. To bootstrap this, we provide example stimuli, payment, fraud, and alarm functions in the publicly available codebase that demonstrate our best practices. A partial list of these include a SurveyCTO stimuli [55], a Google Analytics stimuli [18], a SMS based stimuli, a manual stimuli, a IFTTT stimuli [41], a manual payment, a Korba payment [26], and an email alarm. OINK is architected such that once this work has been done, these functions can be reused across experiments.

Static IPs and Whitelisting. None of the three commercial FaaS services currently provide a static IP service, although it is a commonly requested feature [10]. This is a problem when using third party APIs that use whitelisting in their security systems. For our OINK implementation of the Korba API (which uses whitelists), we configure the *payment* function to call a whitelisted server rather than the Korba API directly. This server then forwards the request to the Korba API and the response back to OINK. Requiring an always on whitelisted proxy server violates the serverless architecture of OINK.

Latency. All three FaaS providers have latency guarantees in of milliseconds, which could be too slow.

Infinite Loops. None of the three FaaS providers have explicit protection against infinite loops created by functions that call themselves. Some protection is provided by a hard limit in the number of functions that can be run in a unit of time, but this can lead to unexpectedly high FaaS costs.

Incentive Systems As Their Own Intervention

Communicating Incentive System Design Patterns. The argument can be made that because incentive systems can impact the results of an experiment, each experiment that involves incentives should report the design of their system [53]. In our literature review we find that only 30% of papers include descriptions. Perhaps the barrier towards including more incentive system descriptions will be lowered by our definition of a standard vocabulary to describe incentive systems. Or, perhaps, OINK itself could export descriptions about the incentive systems it is running to be included in the literature.

Similarly, Computer Science has been working towards better adherence to Institutional Review Board (IRB) approval requirements for publication [12]. We aim to work with different IRB's to establish an method of automatically exporting descriptions of experiments designed using OINK in a format that could be presented as part of an IRB protocol.

Experimental Design. Both by increasing the resolution of stimuli and by allowing tighter control of the frequency of incentive transfer, OINK could enable previously difficult to conduct experiments. For example, if a researcher was interested in incentivizing a user to check their blindspots more while driving, OINK could allow stimuli from an eye tracker. Or, a researcher could study how incentives influence behavior over the course of a day by using OINK to provide different amounts of credit for the same action over the course of the day. Programmers could generate code with relative ease implementing either of these examples, but OINK would allow people less comfortable coding to run these experiments.

Simulation. The principled implementation of an incentive system opens up an opportunity for tools to be developed that allow for the quick simulation and iteration of incentive systems. These tools could be helpful to scope different experimental designs when faced with budgetary constraints.

Incentive System Research. Many crowd-sourcing works explore incentive systems design to attract and hold crowds of specific sizes or expertise [61]. OINK can ease the implementation and evaluation of these novel incentive systems.

Other Uses for OINK

OINK can be extended to many different domains. For example, OINK could distribute bonuses at a company by using key performance indicators as a *stimuli* function, and payroll as a *payment* function. It could be used to track non-tangible incentives using a design where a public health worker uses OINK to email words of encouragement every time a patient arrives at a clinic on time by using the arrival time as a *stimuli* function, and an email service as a *payment* function. Or, it could be used for non-human scientific studies by dispensing a food when a button is pressed (a common incentive when studying animals [28, 44, 46]) with a button press as a *stimuli* function and actuator control as a *payment* function.

Future Work

Full Release. OINK is in an alpha state; the most mature parts of the implementation are those required to support the DumsorWatch deployment in Ghana. It is mature enough for experienced computer scientists to deploy and manage, but not yet a turn-key solution for domain scientists.

Community. To minimize development effort, OINK users should have access to multiple pre-implemented functions that support the majority of incentive systems. Some functions are provided in the OINK code base, but many more will have to be implemented. We plan on hosting OINK tutorials to attract a community of early adopts to contribute to this effort.

Usability Testing. The OINK system has been designed based on input from by a small group of researchers. We are confident that OINK will be made better by learning from larger and more diverse groups of users.

Cross-community Literature Review. OINK aims to be a tool for many different scientific communities. It has been implemented to support the communities we are most familiar with. However, we expect branching out will reveal edge cases not previously encountered, potentially requiring revisions to our incentive system model.

Expansions. We enumerate a list of potentially useful extensions for OINK left for future development. Possible *stimuli* expansions include libraries that respond to events on mobile devices [3, 17, 32], web user interfaces triggered by user interaction [19], and events generated from data-streams collected by sensors [42]. Possible *payment* expansions include support for different mobile money APIs [6, 43, 52, 54], non-monetary non-tangible incentives such as sending a supportive email [34, 53], or non-monetary tangible incentives such as releasing food to an animal [44, 45]. Possible *alarm* expansions include services that generate emails [34], SMS messages [57], and graphical displays [22]. Possible *fraud* expansions include services that detect anomalies [31] or provide methods for additional authentication [58].

One notable external service, especially for alarms and stimuli, is IFTTT, which allows for the graphical construction of applets that incorporate multiple APIs [41]. For example, IFTTT can automate the configuration of a system that sends an HTTP POST packet when a phone enters a location. This HTTP POST could then be used as an OINK stimulus to trigger an incentive, allowing for construction of a complicated incentive system with writing minimal (if any) code. Similarly, IFTTT can respond to incoming web traffic, allowing OINK alarms to integrate easily with third-party APIs.

Implementation using other FaaS Implementations. There are three major FaaS providers each who could support OINK [2, 15, 30]. A researcher using services from one cloud may want to run OINK in that same environment. Porting OINK to different FaaS environments remains future work.

6 CONCLUSIONS

Incentive systems are critical to human subjects research, often impacting the results of the experiments themselves. Their architectures, however, are not well discussed in the literature, and current incentive systems are complicated to construct, difficult to scale, and limited without integration with a large number of stimuli and payment APIs. We claim that these challenges impede human subjects research. To address this problem we present the Open INcentive Kit (OINK), a software system that provides out-of-the-box automation, autogeneration, scalability, security, and accountability for incentive management. We believe that OINK can add structure to the incentive design process, potentially leading to broad adoption by human subjects researchers. This enables incentive systems that can be shared and reused across a community, ultimately modernizing this critical component of human subjects research.

7 ACKNOWLEDGMENTS

This work was supported in part by the Development Impact Lab (USAID Cooperative Agreement AID-OAA-A-13-00002), part of the USAID Higher Education Solutions Network. We thank Joshua Adkins for his technical guidance, Meghan Clark, Zakir Durumeric, William Huang, and Lane Powell for their edits, Da Seglah and Nana Bonku Eshun for their technical support with Korba, Kwame Abrokwah, Susanna Berkouwer, and Catherine Wolfram for being early adopters of OINK, the anonymous reviewers for their insightful feedback and our anonymous shepherd for their guidance.

REFERENCES

[1] Amazon. 2018. Amazon Mechanical Turk. https://www.mturk.com/. (2018). (Accessed on 07/22/2018).

[2] Amazon. 2018. AWS Lambda – Serverless Compute – Amazon Web Services. https://aws.amazon.com/lambda/. (2018). (Accessed on 02/28/2018).

[3] Amazon. 2018. AWS Mobile. https://aws.amazon.com/mobile/. (2018). (Accessed on 03/02/2018).

[4] Amazon. 2018. Document History - AWS Lambda. https://docs.aws.amazon.com/lambda/latest/dg/history.html. (2018). (Accessed on 03/02/2018).

[5] Colin F Camerer and Robin M Hogarth. 1999. The effects of financial incentives in experiments: A review and capital-labor-production framework. *Journal of risk and uncertainty* (1999).

[6] Indian Governement Cashless India. 2018. Unified Payments Interface (UPI). http://cashlessindia.gov.in/upi.html. (2018). (Accessed on 03/02/2018).

[7] Rohit Chaudhri, Waylon Brunette, Mayank Goel, Rita Sodt, Jaylen VanOrden, Michael Falcone, and Gaetano Borriello. 2012. Open data kit sensors: mobile data collection with wired and wireless sensors. In *Proceedings of the 2nd ACM Symposium on Computing for Development*. ACM, 9.

[8] PCI Security Standards Council. 2018. Official PCI Security Standards Council Site. https://www.pcisecuritystandards.org/. (2018). (Accessed on 03/02/2018).

[9] Robin P Cubitt, Chris Starmer, and Robert Sugden. 1998. On the validity of the random lottery incentive system. *Experimental Economics* 1, 2 (1998), 115–131.

[10] James Daniels. 2017. Google Cloud Functions static IP - Google Groups. https://groups.google.com/forum/#!topic/firebase-talk/o9Br8GtYBN4. (2017). (Accessed on 03/02/2018).

[11] Committee for Protection of Human Subjects. 2017. Compensation of Research Subjects. https://cphs.berkeley.edu/compensation.pdf. (2017). (Accessed on 02/28/2018).

[12] Simson L Garfinkel and Lorrie F Cranor. 2010. *Institutional review boards and your research: a proposal for improving the review procedures for research projects that involve human subjects and their associated identifiable private information*. Technical Report. NAVAL POSTGRADUATE SCHOOL MONTEREY CA.

[13] Uri Gneezy, Stephan Meier, and Pedro Rey-Biel. 2011. When and why incentives (don't) work to modify behavior. *Journal of Economic Perspectives* 25, 4 (2011), 191–210.

[14] Google. 2018. Cloud Firestore. https://firebase.google.com/docs/firestore/. (2018). (Accessed on 03/01/2018).

[15] Google. 2018. Cloud Functions - Serverless Environment to Build and Connect Cloud Services. https://cloud.google.com/functions/. (2018). (Accessed on 02/28/2018).

[16] Google. 2018. Firebase Cloud Messaging. https://firebase.google.com/docs/cloud-messaging/. (2018). (Accessed on 03/01/2018).

[17] Google. 2018. Firebase Invites. https://firebase.google.com/docs/invites/. (2018). (Accessed on 03/01/2018).

[18] Google. 2018. Google Analytics Solutions - Marketing Analytics & Measurement. https://www.google.com/analytics/#?modal_active=none. (2018). (Accessed on 03/02/2018).

[19] Google. 2018. Google Forms App Script. https://developers.google.com/apps-script/reference/forms/. (2018). (Accessed on 03/02/2018).

[20] Google. 2018. Google Scholar. https://scholar.google.com/. (2018). (Accessed on 03/02/2018).

[21] Christine Grady. 2005. Payment of clinical research subjects. *The Journal of Clinical Investigation* 115, 7 (2005), 1681–1687.

[22] Grafana. 2018. Grafana - The open platform for analytics and monitoring. https://grafana.com/. (2018). (Accessed on 03/02/2018).

[23] Bin Guo, Zhu Wang, Zhiwen Yu, Yu Wang, Neil Y Yen, Runhe Huang, and Xingshe Zhou. 2015. Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm. *ACM Computing Surveys (CSUR)* 48, 1 (2015), 7.

[24] Carl Hartung, Adam Lerer, Yaw Anokwa, Clint Tseng, Waylon Brunette, and Gaetano Borriello. 2010. Open data kit: tools to build information services for developing regions. In *Proceedings of the 4th ACM/IEEE international conference on information and communication technologies and development*. ACM, 18.

[25] Matthew Jadud. 2016. IRB Reviews Required. (2016). https://doi.org/10.1145/2993223.2993229

[26] Korba. 2018. Korba - Our Lives Simplified. http://korbaweb.com/library. (2018). (Accessed on 03/01/2018).

[27] Stanley Lippert. 1968. A comprehensive approach to human factors in developing countries. *Human Factors* 10, 6 (1968), 649–662.

[28] Margaret S Livingstone, Warren W Pettine, Krishna Srihasam, Brandon Moore, Istvan A Morocz, and Daeyeol Lee. 2014. Symbol addition by monkeys provides evidence for normalized quantity coding. *Proceedings of the National Academy of Sciences* 111, 18 (2014), 6822–6827.

[29] Edwin A Locke. 1968. Toward a theory of task motivation and incentives. *Organizational behavior and human performance* 3, 2 (1968), 157–189.

[30] Microsoft. 2018. Azure Functions–Serverless Architecture. https://azure.microsoft.com/en-us/services/functions/. (2018). (Accessed on 02/28/2018).

[31] Microsoft. 2018. Azure Machine Learning Anomaly Detection API | Microsoft Docs. https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/apps-anomaly-detection-api. (2018). (Accessed on 09/30/2018).

[32] Microsoft. 2018. Working with the App Service Mobile Apps managed client library. https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-dotnet-how-to-use-client-library. (2018). (Accessed on 03/02/2018).

[33] Phil Nash. 2018. philnash/useful-twilio-functions: A set of useful Twilio Functions. https://github.com/philnash/useful-twilio-functions. (2018). (Accessed on 03/01/2018).

[34] Nodemailer. 2018. Nodemailer. https://nodemailer.com/about/. (2018). (Accessed on 03/01/2018).

[35] Peoples Bank of China. 2018. Peoples Bank of China Issues Barcode and QR Code Payment Rules. http://www.pbc.gov.cn/english/130721/3456052/index.html. (2018). (Accessed on 03/02/2018).

[36] Department of Health and US Government Human Services, Office of Inspector General. 2000. Recruiting Human Subjects: Sample Guidelines for Practice (OEI-01-97-00196; 6/00). https://oig.hhs.gov/oei/reports/oei-01-97-00196.pdf. (2000). (Accessed on 03/02/2018).

[37] Central Bank of Kenya. 2005. Kenya Electronic Payments and Settlement System (KEPSS) Rules and Procedures. https://www.centralbank.go.ke/wp-content/uploads/2016/08/KepssRules1.pdf. (2005). (Accessed on 03/02/2018).

[38] University of Massachusetts Research Administration and Compliance. 2018. IRB Guidelines | Research and Engagement | UMass Amherst. https://www.umass.edu/research/compliance/human-subjects-irb/guidance. (2018). (Accessed on 03/02/2018).

[39] OpenDataKit. 2018. Open Data Kit - Deployments. https://opendatakit.org/about/deployments/. (2018). (Accessed on 03/01/2018).

[40] OpenMapKit. 2018. OpenMapKit Website. http://openmapkit.org/. (2018). (Accessed on 03/01/2018).

[41] Steven Ovadia. 2014. Automate the internet with "if this then that" (IFTTT). *Behavioral & social sciences librarian* 33, 4 (2014), 208–211.

[42] Particle. 2018. Particle Guides | Webhooks. https://docs.particle.io/guide/tools-and-features/webhooks/. (2018). (Accessed on

03/02/2018).

[43] PayPal. 2018. Creating and managing NVP/SOAP API credentials. https://developer.paypal.com/docs/classic/api/apiCredentials/?mark=API%20security. (2018). (Accessed on 03/02/2018).

[44] Irene M Pepperberg and Ken Nakayama. 2016. Robust representation of shape in a Grey parrot (Psittacus erithacus). *Cognition* 153 (2016), 146–160.

[45] petnet. 2018. Petnet.Io Smartfeeder - The SmartFeeder feeds your pet the right amount at the right time, automatically! https://www.welcome.ai/products/hardware-iot/petnet-io-smartfeeder. (2018). (Accessed on 09/30/2018).

[46] Maria Elena Miletto Petrazzini. 2014. Trained quantity abilities in horses (Equus caballus): A preliminary investigation. *Behavioral Sciences* 4, 3 (2014), 213–225.

[47] Razorpay. 2018. Libraries & Integrations o Razorpay. https://docs.razorpay.com/docs/libraries. (2018). (Accessed on 03/02/2018).

[48] Safaricom. 2018. Developers' Portal | API Documentation. https://developer.safaricom.co.ke/docs#authentication. (2018). (Accessed on 03/02/2018).

[49] Arjuna Sathiaseelan, Richard Mortier, Murray Goulden, Christian Greiffenhagen, Milena Radenkovic, Jon Crowcroft, and Derek McAuley. 2014. A feasibility study of an in-the-wild experimental public access wifi network. In *Proceedings of the Fifth ACM Symposium on Computing for Development*. ACM, 33–42.

[50] Peter D Schellie. 1979. Electronic Fund Transfer Act. *The Business Lawyer* (1979), 1441–1452.

[51] Eleanor Singer and Cong Ye. 2013. The use and effects of incentives in surveys. *The ANNALS of the American Academy of Political and Social Science* 645, 1 (2013), 112–141.

[52] Square. 2018. Square Connect API Documentation. https://docs.connect.squareup.com/api/oauth#credentials. (2018). (Accessed on 03/02/2018).

[53] RG Stovel, S Ginsburg, L Stroud, RB Cavalcanti, and LA Devine. 2018. Incentives for recruiting trainee participants in medical education research. *Medical teacher* 40, 2 (2018), 181–187.

[54] Stripe. 2018. Stripe API Reference. https://stripe.com/docs/api#authentication. (2018). (Accessed on 03/02/2018).

[55] SurveyCTO. 2018. Homepage. https://www.surveycto.com/. (2018). (Accessed on 07/22/2018).

[56] R Core Team et al. 2013. R: A language and environment for statistical computing. (2013).

[57] Twilio. 2018. Twilio - Communication APIs for SMS, Voice, Video and Authentication. https://www.twilio.com/. (2018). (Accessed on 03/02/2018).

[58] Twilio. 2018. Two Factor Authentication for Identity Management via SMS or Voice. https://www.twilio.com/use-cases/two-factor-authentication. (2018). (Accessed on 09/30/2018).

[59] WeChat. 2018. [WeChat payment] ordinary merchant access documents. https://pay.weixin.qq.com/wiki/doc/api/index.html. (2018). (Accessed on 03/02/2018).

[60] Dale Whittington, John Briscoe, Xinming Mu, and William Barron. 1990. Estimating the willingness to pay for water services in developing countries: A case study of the use of contingent valuation surveys in southern Haiti. *Economic development and cultural change* 38, 2 (1990), 293–311.

[61] Xinglin Zhang, Zheng Yang, Wei Sun, Yunhao Liu, Shaohua Tang, Kai Xing, and Xufei Mao. 2016. Incentives for mobile crowd sensing: A survey. *IEEE Communications Surveys & Tutorials* (2016).

[62] Xinglin Zhang, Zheng Yang, Zimu Zhou, Haibin Cai, Lei Chen, and Xiangyang Li. 2014. Free market of crowdsourcing: Incentive mechanism design for mobile sensing. *IEEE transactions on parallel and distributed systems* (2014).