Enhanced Exposure Exterminator (EEE) System

Nhat Nguyen (nvn@mit.edu),
Veronica Muriga (wmuriga@mit.edu),
Shashvat Srivastava (shashvat@mit.edu).

1. Introduction

Past human experience with pandemics has led to the increased demand in effective contact tracing and exposure notification systems, to mitigate the spread of infectious diseases by identifying people who have been in close enough contact with infected people. In this digital era, smartphones have the power to serve as the cornerstone of modern contact tracing systems, by leveraging technologies such as Bluetooth Low Emission (BLE) transmissions and WiFi. A contact tracing and exposure tracing system needs to identify infections and improve isolation compliance, while protecting user privacy. Analysis of historical contact tracing systems showed that they all faced some success, but with significant challenges in uptake due to privacy concerns, inability of EEEs to scale up to accommodate a large number of users, and lack of universal access to smartphones. In this paper we propose an app-based contact tracing system for a university similar in size to MIT, that prioritizes *accuracy*, *timeliness*, *privacy* and *low phone impact* to achieve the goal of swiftly identifying and responding to potential outbreaks on campus.

2. System Overview

Our exposure tracing system (EEE) prioritizes *accurate and timely exposure notification*, and e*fficient storage management*. EEE includes a central server and smartphones that support WiFi and BLE transmission. EEE is supported by WiFi routers that connect to the central server via a wired network.

EEE's main objective is to *accurately identify* exposures and *promptly notify* the exposed users. This is achieved in the following ways:

1. The definition of a contact event as people within 3m of each other for 20 minutes ensures accuracy. These distance and time parameters can also be easily adapted to suit different situations, or in case research comes up with a more effective combination of parameters.
2. The frequency at which users ping the server to get information on recently reported positive test cases depends on the use case, but we guarantee that it will never exceed 12 hours after the healthcare system reports a positive test case to the server.

Privacy in EEE is reinforced in the following ways:

1. No spatial information is collected. A system that tracks user locations cannot ensure that a user always carries their phone. Because individual voluntary effort is essential in both contact determination and isolation compliance, and constant surveillance violates user privacy, EEE does not implement location tracking.
2. Each BLE tag contains a temporary ID which is changed every 15 minutes.
3. All contact tracing information is stored on the user's phone for 2 weeks and discarded thereafter.

EEE has *little impact* on the smartphones, and requires minimal user input. This is achieved in two ways:

1. The set of BLE tags is regularly minimized to reduce the storage costs.

2. The number of server requests is adaptable. Thus, the overhead cost of waking the phone from sleep to conduct computation is minimal.

Furthermore, the overall computational costs of EEE overall is quite small.

3. System Design

3.1 System Components

The main components in EEE are smartphones and the central server. In this section, we discuss the role these two components play, and when and how they interact. These interactions are further illustrated in Figure 1 below.

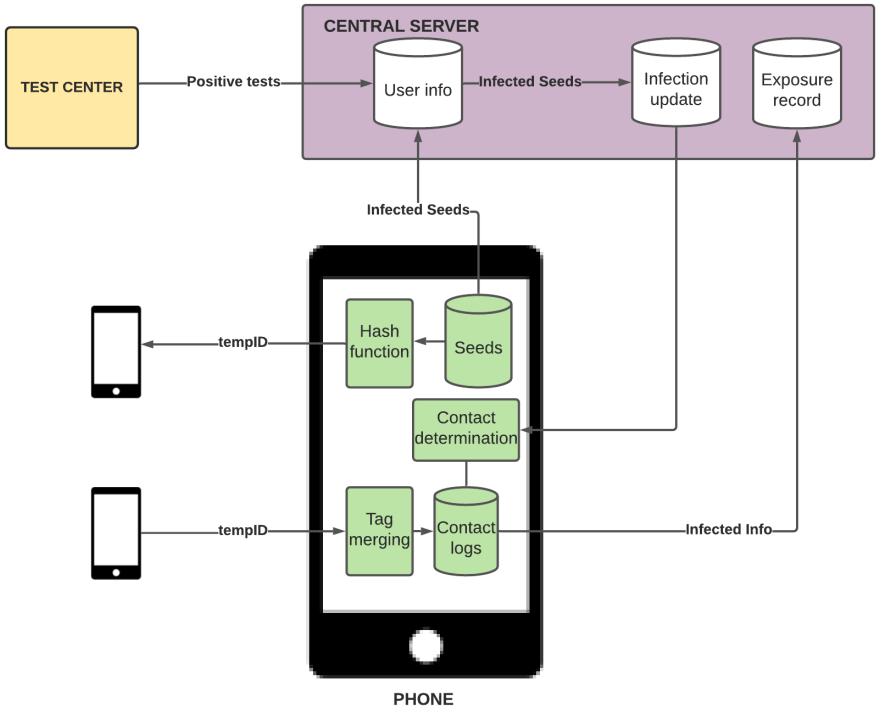

Figure 1: Illustrated Overview of System

3.1.1 Smartphones

EEE's app is installed on smartphones. The app uses *a hash function* and *a seed* to generate 1440 TempIDs **daily**, each 128 bits long. Our app will use each of these generated TempIDs as the phone's primary identifier for 15 minutes. This allows for user anonymity; even if attackers acquire TempIDs, it is impossible to reverse TempIDs to identify seeds. Furthermore, since each phone chooses a tempIDs randomly every 15 minutes, it picks 96 IDs at random without replacement from a total of 1440 tempIDs

so it's hard to guess which tempIDs are chosen, even if attackers have access to the seeds and the hash function.

The smartphone makes use of its Bluetooth Low Energy (BLE) transmission capability to transmit a BLE signal *every 250 ms* consisting of the TempID. A phone receiving a BLE signal will log the TempID received, and the timestamp that it is received at, and the signal strength of the received signal.

BLE tags are collected and stored,and every hour, *garbage collection* is done, where a background process compresses the set of tags, dumping tags that are too far away and condensing tags that appear multiple times into the time intervals in which they appear. These condensed tags are stored for two weeks and then deleted. To exceed 1GB of phone storage, we need ~ 14 million of these 70b tags. This requires a user to come in contact with 1000 people simultaneously for 1 hour, and is not feasible in light of current gathering restrictions.

To conduct exposure tracing, the app regularly requests updates from the server. These updates include a list of seeds used to generate TempIds as well as the time periods in which these seeds were used. Using these seeds, the app generates the list of TempIDs for each infected user, and uses those TempIDs to determine if there were any exposure events.

3.1.1.1 App Setup

To prevent unauthorized access of the app, the user is required to login or authorize themselves. The app sends an *init_request((userId, authorization)* to the server and receives an authorization token and an update schedule in response. If the authorization token ever expires, the app simply asks the user to login again. After this, the app launches a background process that requests updates from the server based on the update schedule.

3.1.1.2 Handling Rollover and Turnover

Rollover refers to the changing of the seed that a user will use for the next 24 hours to generate a TempID. Rollover of seeds happens at the same time for all users. Without knowing the seeds before and after the change, any exposure events during the rollover time may be missed. However, revealing that two seeds come from the same user sacrifices privacy. If this information is revealed, an infected user's seeds covering multiple-day periods are revealed. In essence, every infected user would effectively have a constant BLE tag throughout their quarantine period.

To solve this problem, our app assumes that all infected seeds within the same update list belong to the same user *during the rollover period only*. For example, if user A was near one infected user B for 10 minutes, and near a different infected user C for 10 minutes afterward, the contact determination process on user A's phone would log this as an exposure event. This eliminates the chance that any exposure event is missed, but might produce false positives. This is acceptable for two reasons. First, it could only happen during the rollover period, which by default is when most users are asleep (say, 4am), so its probability is minimal. Second, even though the user is not exposed to the same infected user for

more than 20 minutes, the user is most likely still at risk. Intuitively, staying near two infected users for ten minutes each should be as risky as staying near a single infected user for twenty minutes. However, we do not have data to back up this claim.

Turnover refers to the change in tempIDs every 15 minutes. Once a user is tested positive and their infected seeds are received by other phones, TempIDs for that user could be generated. If a number of TempIDs from the same user are found in sequence in a log, the contact determination algorithm aggregates the times and determines if there is an exposure.

3.1.2 Central Server

The core of the system design is a single central server. This server connects the users both to each other and to the central health authority. The app will communicate regularly with the central server to receive all seeds pairs of reported cases from the central server. The app then uses the hash function to generate the TempIDs from those pairs and checks its local logs to see if there are matches, and for those matches checks if they qualify to be contact events. We define a contact event as when two users are **within a range of 3 meters for 20 minutes or longer.** Due to the spreading of new variants that are proving to be more infectious, this definition might be modified to include a shorter range and time period, in tandem with state and federal guidelines. EEE allows the central health authority to change the definition of a contact event.

If an exposure event is determined, the app sends a notification to the central server with the details of the app user, and the user is also notified. The healthcare organisation has direct access to the server and can provide quarantine information and useful resources to users that the app determines to be exposed to an infected person.

3.1.2.1 Data Storage in the Central Server

We use relational databases because they are flexible and commonly used. The central server stores data in four relational databases, as shown in Figure 2 below, to compartmentalize the data into their use cases:

userDb stores user's living arrangements and classes. This database is necessary for proper exposure tracing. It exists so that EEE can determine exposure events for living groups and in-person classes.

statusDb stores the user's current status: whether the user is clear, in isolation, or infected. It also stores the times at which statuses changed as well as any testing results of users in isolation. As per the specification, changes in status (*i.e.* positive tests and exposure events) are stored for at least 180 days.

updateDb stores crucial exposure tracing updates that will be relayed to users and used to determine exposure events. It stores three types of updates:

1. *Infection updates* (or just *updates* for short): This includes BLE broadcast seeds of infected users along with the time periods in which those seeds were used.
2. *Policy updates: These originate* from the central health authority. The two types of policy updates, exposure tracing policy updates and isolation compliance policy updates, are discussed in the following section.
3. *Schedule-reset updates:* in case the central server fails to properly balance the schedules, or a large section of users are unable to follow the schedules. This update eventually gets relayed to the apps and instructs them to request a new schedule through an *init_request*.

isolationDb stores contact events from users intended to be in isolation. To preserve privacy, this database only stores the number of contact events an isolated user has each day. Furthermore, this count only starts after the user has been notified of their exposure. Future sections discuss the privacy and storage aspects in more detail.

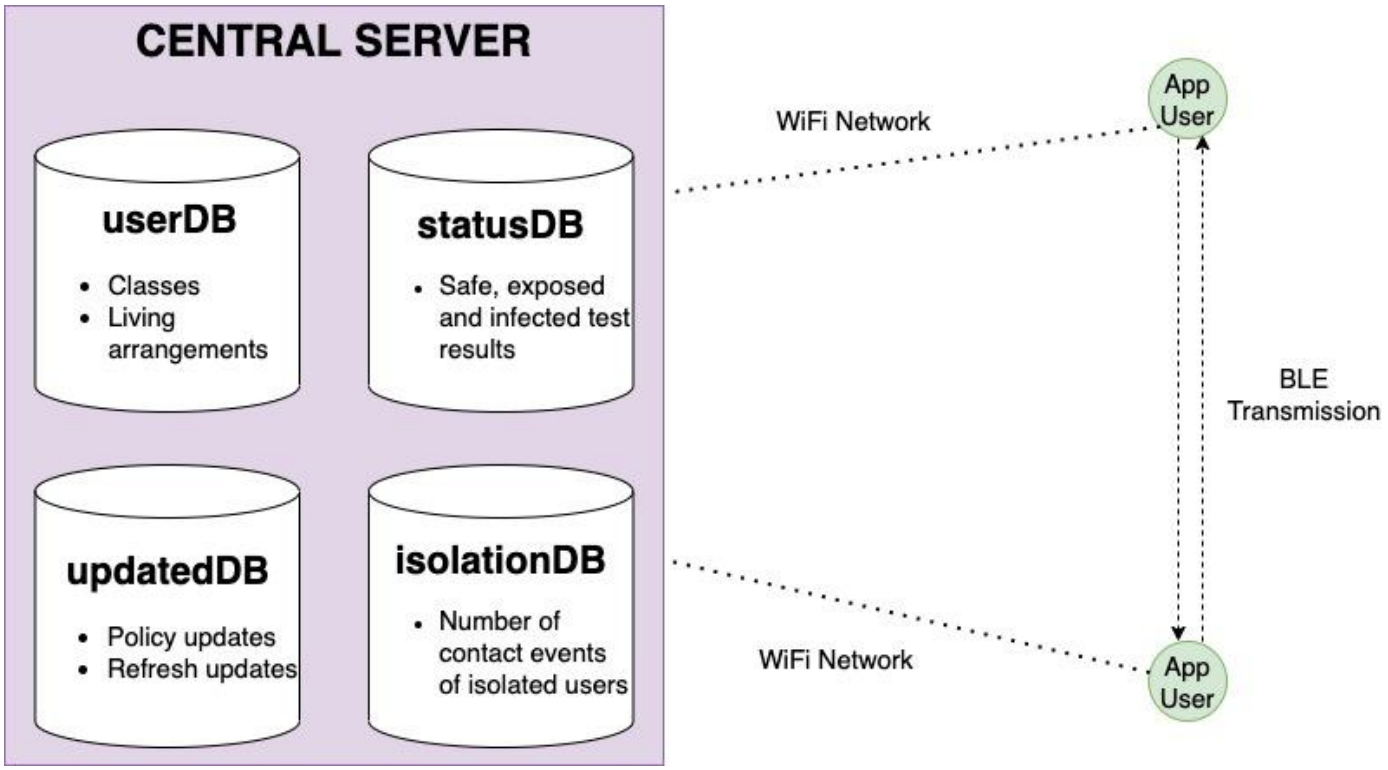

Figure 2: Databases in the Central Server, and Server-Client Communication Links

3.2 Interaction between the App and Central Server

The user's interaction with the central server can be broken down into four parts:

3.2.1 Setup

The userId and authorization token are sent to the server in every following interaction. Their inclusion, and the server's verification of these items, are implicit in each request; they are omitted to highlight the key details of each interaction.

The update schedule is a tool for distributing the load on EEE. The app requests updates from the server according to this schedule. For example, one such schedule could be to request updates from the server every two hours on the 13th minute. By choosing schedules for each app, the central server can distribute the load of handling update requests from many users. This is done to facilitate the goal of timely exposure notifications: since the server has a more evenly distributed load, the apps can request updates from the server more often, which in turn allows faster exposure notifications.

The schedule provided by the server is non-binding and merely a guideline. The server will still respond to requests even if the app does not adhere to the schedule. However, it is expected that most apps will adhere to the schedule. Timely exposure notifications are still the priority, so it would not make sense to ignore requests from users not following the ascribed schedule.

3.2.2 Requesting Updates

EEE provides a function *check_updates(timestamp)* that allows the central server to relay any updates to the users. The server sends all updates since *timestamp*, which is the previous time the app requests updates from the server.

The server responds to this request with the three types of updates:

1. Whether the user's status has changed from clear to exposed or infected (stored in *statusDb*)
2. Any possible exposure tracing policy updates (stored in *updateDb*)
3. Seeds plus time periods of any newly infected users (also stored in *updateDb).* For each infected user, the server sends all the seeds the user used for the last ten days, or since the user's last negative test, whichever is shorter.

If the user's status changes to infected, the server also includes the timestamps of the two tests: the last negative test for the user, and the most recent positive test.

Sending the list of seeds and time period pairs as a list sorted by the time period prevents users from being able to determine which seeds belong to the same user, enhancing privacy.

Although users are expected to make requests according to a known update schedule, the inclusion of the timestamp is crucial. The app may fail to follow the update schedule for unexpected reasons.

Sending the timestamp of the last update request ensures that no updates will be missed, aiding in the goal of accurate exposure notifications. Additionally, the timestamp ensures that updates are not sent multiple times to the user, reducing the load on the server. Again, this allows users to query the server for updates often, aiding in the goal of timely exposure notifications.

3.2.3 User Status Updates

Upon receiving an update from the central server, the app performs contact determination and may find that its user is either exposed or infected, and is required to notify the server via a function *send_update(new_status, seeds, times)*. If the user is infected, the app also sends relevant seeds that were used to generate tempIDs, along with the times at which those seeds were used. Upon receiving an update, the server stores the new user's status in *statusDb* and seeds and time periods in *updateDb*.

Whenever the server receives a set of seeds, this set is stored in a temporary location as opposed to being immediately added to the main table of *updateDb*. All seeds submitted from users are added to *updateDb* hourly. This feature protects the privacy of infected users. Hourly updates make it difficult to discern which seeds belong to the same user, since multiple user's seeds are added to the list all at once. This delayed-update mechanism is further discussed under privacy evaluation (Section 4.2).

When a user's status changes to exposed or infected, the app constantly notifies the user until it's acknowledged.

3.2.4 Isolation Compliance

EEE provides a function *upload_compliance_data(numberContacts, day*). Each day, every isolated user sends the number of contact events they had. This information was chosen to balance the goal of privacy with the goal of aiding in isolation compliance.

Uploading and storing more detailed information, such as the times of each contact event, would compromise privacy. Such information reveals details about the user's habits and personal lives. An alternative could be to simply report whether the user had any contact events in one day. The information gleaned from such a report is comparable to the standard infection and exposure reports already made. However, this approach may not be enough to truly aid with isolation compliance. Users may accidentally come into contact with people for too long, or there might be inaccurate reporting. A simple yes or no report for each day would not be enough to determine whether a user is maliciously refusing to stay isolated.

The approach taken balances these concerns. Information gleaned about user habits is small. Users that stay isolated should have zero or only a few contact events; said contact events can be chalked up to accidents or inaccuracies. On the other hand, a user refusing to stay isolated will likely come into contact with many users frequently. The server will record repeated days of having many contact events, and the malicious user can be identified.

One perceived flaw of this approach is that it does not detect users who do not stay isolated but also do not trigger any contact events. This is acceptable for two reasons. First, the only approach that can overcome this obstacle is the inclusion of geolocation tracking. We have already argued why the inclusion of geolocation tracking is not worth the reduction in privacy. Second, a user that does not trigger any contact events may not be a significant risk, even if they refuse to physically stay in one place. In fact, such a user could be considered technically isolated because they did not trigger any contact events.

Finally, we justify the inclusion of this isolation compliance feature itself. Recall that malicious users that refuse to stay isolated can leave their smartphones behind, avoiding any detection by any isolation compliance feature. This feature is still acceptable because it has no overhead: the computation done is essentially the same as the primary goal of determining if a user has been exposed. Thus, the inclusion of this feature has no overhead.

3.3 Interaction between the Health Authority and Central Server

The central health authority interacts with the central server in two ways:

1. Reporting positive test results to the central server. This is done through a function *submit_positive(results)*. *results* is an array of positive test results. Each result consists of a student id, two testing timestamps, and a submission timestamp. The first timestamp is the time of the last known negative test result while the second timestamp is the time of the positive test result. When *submit_positive* is called, EEE updates *statusDb* and stores the testing results.

2. Reporting any changes to the contact event policy. EEE provides a function *update_contact_policy(rule, type*), where the rule is a description of the new rule users must use.
    a. By default, a contact event is when two users are within three meters of each other for twenty minutes. However, in the future, epidemiological research may yield an improved rule or policy to determine a contact event. When this function is called with a new *rule*, *updateDb* is updated with the new *rule* and the current timestamp.
    b. There are two types of policy updates. The *exposure tracing policy* is used to determine when a user comes into contact with an infected user. The *isolation compliance policy* is used to determine when an isolated user comes into contact with another. Providing two types of policies allows the central health authority to be more flexible with their policies. For example, they might choose a stricter policy for isolation compliance.

4. Evaluation

We evaluate EEE based on our design priorities of *accurate and timely exposure notification*, *user privacy*, and *minimal computation cost* on user phones. We also consider *communication overhead*, where we justify the amount and frequency of transmissions between user phones, and between phones and the server.

We assess the *adaptability* of EEE to different use cases, and identify edge cases which surpass the scope of EEE. Looking ahead, we realize the importance of this platform being easily extensible to account for new scenarios as the outbreak changes course or to enforce and assess new measures to curb it, and we evaluate how *scalable* EEE is to handle these scenarios.

4.1 Accuracy and Timeliness of Exposure Notification

Our operational definition of a contact event as people within 3m of each other for 20 minutes ensures accuracy. These parameters can be easily changed to deal with prevalence of a new virus variant, or if further research leads to new discoveries on how the disease spreads. EEE might use an inaccurate contact definition and produce false positive exposure notifications. In such events, better medical insights are required. Therefore we don't consider medical-related false positives as a metric in our system evaluation. If we assume we have the best definition of a contact event, the accuracy of EEE follows. Nonetheless, EEE monitors phone-determined exposure events and actual positive tests. When there are many false positive exposure notifications (many users are determined to be exposed, but then are tested negative), a low level of accuracy is detected, and EEE alerts the authorities to review the definition of contact event.

EEE also allows timely notifications, both in *worst-case* and *average-case* scenarios. Assume each seed has a size of 128 bits. Our server has two 10Gb Ethernet ports. The processing time starts at when the test results become available, and ends when an exposure event is determined and exposed users are alerted. The processing time consists of the times it takes for (1) the server to request and receive infected seeds, and (2) to send infection updates, and for (3) a phone to perform contact determination.

In a *worst-case* scenario, when 100 new infections are reported at the same time at 05:00 PM, and each infected user's last negative test happened 10 days ago (each user waits for an hour, then sends 10 infected seeds to the server upon scheduled time at 06:00 PM), the size of the infection update is 128 bit * 10 seeds * 100 infections = 16 kB. Assume the server has a terrible load-balancing update schedule (all 20,000 users request updates simultaneously at 07:00 PM), it needs to send 16 kB * 20000 updates = 0.32 GB over the network. With two 10Gb Ethernet ports, the server can still send updates instantly. After receiving the update, each app converts 100 seeds to 144000 tempIDs, and checks for matches on its local contact logs. With hash table lookups, this could be done in seconds. So while the worst-case

processing time might be 2 hours in total (from 05:00PM to 07:00PM), some phones might receive updates minutes after test results become available. A randomly chosen update schedule ensures that the *average processing time* is about 40 minutes.

4.2 User Privacy

In this section, we evaluate the privacy of infected users. EEE cannot provide any guarantees on the privacy of infected users. However, it still includes several features to boost the privacy of infected users.

Once users are infected, they upload the seeds since their last negative test. Furthermore, they continue to upload their seeds while they stay in quarantine. These seeds are sent to all users of EEE. Thus, infected users have effectively a constant BLE id every day during this time period. However, EEE takes steps to ensure that infected users do not have an effective constant id throughout the entire time period.

What this means is that while an adversary may be able to learn which BLE tags belong to the same user if they were broadcasted on the same day, the adversary will not be able to learn whether two BLE tags belong to the same infected user if they were broadcasted on separate days.

To get this property, EEE takes two steps.

First, EEE never sends pairs of seeds belonging to the same user. One motivation to send a pair of seeds could be to solve the rollover problem. If one knows the seed before and after the rollover time, all exposure calculated can be done easily. However, this approach would then link all seeds during the user's quarantine period, meaning that the adversary could learn every BLE tag sent by this user during their quarantine period.

Second, the list of seeds to send to each user is batch-updated hourly. Infected users upload their seeds to the server; an adversary continuously pinging the server for updates would notice whenever multiple seeds are added to the list. They would be able to deduce that those seeds came from the same user. By delaying the update to occur hourly, multiple user's seeds are added to the list all at once. The adversary will be unable to determine which seeds belong to the same user.

We can evaluate this effect in several use cases. First, consider the very-low use case: 8.9 infections per day with testing results coming evenly throughout the day. We assume that infected users stay in quarantine for at least 8 days. Furthermore, we estimate that the test results will arrive evenly throughout an 8 hour period.

Consider a user that finds out they are infected. Their last negative test result was two days ago, so they have to upload three seeds worth of data. If these seeds were "published" immediately, an adversary would be able to link these three seeds worth of data. However, EEE uses batching. There are about 72 quarantined users at any time, and these users must also upload their seeds. These seeds will be

uploaded during the same 8 hour time period as the infected user. Thus, this user will be able to "hide" their three seeds among nine other user's seeds.

The compressed very high use case performs even better. Every day, EEE finds out that 80 new users are infected at the same time. Estimating that roughly half these users can upload their seeds within the first hour and the second half can upload their seeds within the second user, each user can hide their seeds among at least 40 other user's seeds. This estimate does not even include the seeds from users that are already quarantined.

The weaknesses of this approach are that the benefits depend on the reliability of users and the conditions of the use case. For example, in an extremely low use case situation, say 1 infected user per day, users may not be able to hide their seeds among many users. Additionally, if a user uploads their seeds at a time that does not match most other users, that user will not get any privacy benefits.

4.3 Computation Cost

To ensure EEE has low impact on users' smartphones, we limit the bulk of computation to garbage collection and exposure computation given distance and time parameters. Garbage collection is necessary to reduce the amount of storage space taken up by logging multiple identical BLE tags. Since a TempID is logged 4 times every second, and each tag logged is 70 bytes long, in an hour a phone can log about 1MB of the same TempID. Garbage collection ensures this redundant data does not build up.

There are two ways in which the cost of exposure computation can be taken away from the phone. Both of these ways would improve battery life, but we argue that the advantages of our current implementation outweigh this advantage. First, the central server could do all the computations. However, this would sacrifice privacy. Second, we could forego hashing and use fixed identifiers in order to reduce the amount of computation done. Hashing for 100 new infections would translate to 144000 computations and hash table lookups. However, this can be done in seconds, even on a smartphone. The additional resources consumed by hashing does not significantly improve the power used by the application.

4.4 Communication Overhead

Communication in EEE can be divided into three main categories:

1. Communication between the server and phone during setup when the server authenticates the user during setup via the *setup(userId, authorization)* function.
2. Communication between phones via BLE signals includes the TempID tag and a timestamp, all of which are about 70b long. BLE transmissions have a medium data rate of ~200 kbps[1] and a

[1] Does bluetooth drain your battery? - senion: Smart office solution. (2019, September 16). Retrieved May 09, 2021, from https://senion.com/insights/bluetooth-drain-battery/

battery consumption of 0.01 to 0.25 w, which is relatively low considering an average phone may have a battery power of about 12.4 watt-hours[2].

3. Communication between a phone and the server to get updates of newly reported positive test cases. When pinging the server via WiFi to request updates of all positive reported cases since its last ping, via the *check_updates(timestamp)* function, a phone will also send a timestamp, which is about 15b long. Upon receiving updates from the server, if the app realizes that its user is infected, the app sends the new status of the user, the seeds the user used to generate TempIDs since their last negative test or for the last 10 days, whichever is shorter, and the times that those seeds were used, via the *send_update(new_status, seeds, times)* function. WiFi drains the battery much faster than BLE transmissions, but since these server communications happen at a maximum 24 times a day if an app pings the server hourly, we determine that this is an acceptable battery cost.

Some instances of communication overhead in EEE are timestamps sent between phones via BLE transmissions and between phones and the server via WiFi. EEE relies on timestamps to make accurate determinations of exposure events by determining from the TempID timestamps, which users were in contact range for more than 20 minutes.

We identified the importance of an app checking for contacts with infected users beyond just those that report positive test cases since the last server ping. This serves as another crucial part of isolation compliance, in case infected users are not isolating accordingly. Instead of having the server concatenate new positive test cases to all positive test cases currently being tracked and sending this updated list to users, we transfer the tracking to individual apps, to keep track of presently infected users. Apps then update this list by removing users after two weeks which is the defined infectious period for the disease. This reduces the overhead it would incur on the WiFi network to transmit the list of all currently infected users.

EEE therefore has minimal communication overhead, and we have determined that all existing overhead is absolutely necessary.

4.5 Adaptability

4.5.1 Adaptability to Pre-Defined Use Cases

We assess the adaptability of EEE given three use cases: when reported infections are very low, very high and when infections hit a compressed very high range. We design the frequency at which phones ping the server for updates of infected users around these use cases. When positive case numbers are very

[2] Negroni, C. (2016, December 26). How to determine the power rating of your gadget's batteries. Retrieved May 09, 2021, from https://www.nytimes.com/2016/12/26/business/lithium-ion-battery-airline-safety.html#:~:text=A%20battery%20in%20a%20smartphone,and%20only%20under%20certain%20conditions.

low, the phone pings the server every 12 hours to get newly infected users; when positive case numbers are very high, this frequency increases to every one hour; and in compressed very high numbers, the phone pings the server every five hours for updates. During setup, an app also randomly chooses a time to start pinging the server, to reduce synchronization in the times that users ping the server.

In defining these frequencies, we considered the given infection rates and daily new cases defined for each use case, as well as the fact that the reporting of new positive cases is spread evenly over time for very low and very high case numbers. At very low case numbers, the number of daily new cases is 8.9 and they are reported evenly throughout the day, meaning that on average a new case is updated every three hours. We choose to keep the frequency to 12 hours in this case to prevent overload on communication networks to prevent a case where apps ping the server at highly synchronized times leading to network congestion. At very high case numbers, there are about 80 new cases every day and they are again reported evenly throughout the day, meaning that on average a new case is updated every eighteen minutes. In this case we tradeoff having a larger amount of network traffic for fast exposure notification, and apps ping the server every hour for updates.

In the compressed very high use case where about 80 new cases are all reported at 5 pm, we minimize strain on communication networks by asking phones to ping the server every five hours for updates. The challenge here is that the transmission of these ~80 cases need to be transmitted to all users in a timely fashion, but without completely congesting the network to a point of collapse. We decided to avoid splitting up the cases being reported to users- for example only sending 5 cases every time updates are requested- because in a high use case, it is imperative for users who are exposed to be notified quickly. We determine that 5 hours is an acceptable window for the network to experience high traffic.

4.5.2 Adaptability to Large Changes in Use Cases

A large change in the use case that would lead to an overload of EEE would be when requests to the server are highly synchronized and reported cases are very high, causing the network to break down, or the server to crash. In this case, users upon pinging the server will get no response back, and an error report will be sent to the system administrator to notify them that the server is down. Upon startup, the server will now send all the cases reported during the crash, as well as cases that the server had immediately before the crash, which may have been corrupted if the server crashed while transmitting these to a user.

Another large change that would not be accommodated by EEE is the case where a phone crashes and it no longer transmits BLE signals. Here a user would not be identified as a contact in case they come in contact range of another user. There are two challenges to get around here:

1. Missing messages from the server: While the phone is down, messages from the server will not be received by the app. To get around this, at startup, the app would ping the server with the last timestamp that it got the seeds for infected people; if the server determines this timestamp to be greater than 24 hours ago, the server will send the last recorded batch of infected contacts.

2. Missing BLE transmissions from users in range of the crashed phone: BLE transmissions from phones in contact range of the crashed phone will not register on the app, and this may lead to potential exposure events not being logged. To minimize the likelihood of this happening, we add 10 minute intervals to all BLE messages logged before the crash and after the crash when the phone comes back online.

Figure 3 below demonstrates the app crash recovery process.

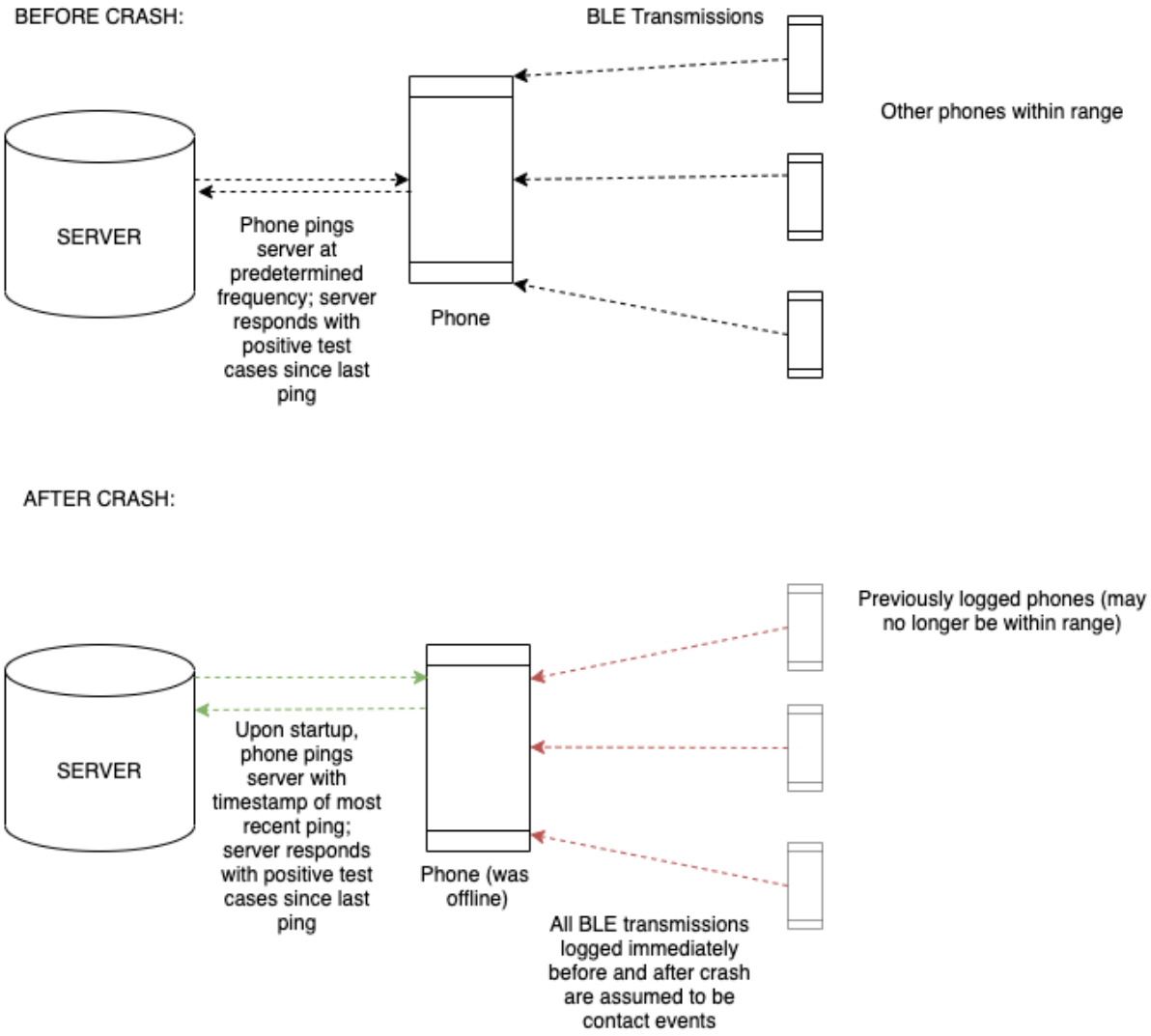

Figure 3: App Recovery After Crash

BLE does not provide perfect reliability. Phones may occasionally miss a broadcasted message. However, apps log BLE messages every 250msec, so occasionally missing a transmission does not have a large impact on reliability as the likelihood for missing a transmission for an extended period of time is low. As

long as the same TempID is logged more than once within an interval of one second, the garbage collector will condense them into the same contact interval.

Our design hinges on users having smartphones- in case a person does not have a phone or does not have the app installed, they are not considered a user and are not accounted for in EEE. In the worst case, a user may come into contact with an infected person who is not an active user, and the user would not be notified about this exposure. In this situation, it would be up to the university to decide whether students having smartphones and installing the app is a requirement, and how to enforce this across the student and faculty population.

4.6 Scalability

There are two main ways in which we anticipate EEE needing to scale:

1. Enrollment of more users: In the case that the university decides to expand EEE to include neighborhoods around the campus, or families of faculty members that live off-campus, then the number of users will increase. The number of exposure events will also increase, since people living off-campus are exposed not only to the campus environment but also to the metropolitan area of the city. The spec defines their infection rates as “double that for the university”. In this case, the size of the infection update (whose maximum size for 100 daily infections as calculated above is 16 kB) increases. However, EEE should be able to handle a *20 times increase in user number* (i.e. 400,000 users) if its update schedule is load-balanced across each 60-minute interval. Even in a compressed very high infection use case with 4000 daily infections, the infection update’s size is 0.64 MB. So the server (with a 20GB outbound capacity) can still send all 400,000 updates (total size 256GB) within 15 minutes. Note that an increase in user size doesn’t necessarily affect a phone’s storage. Since the amount of contact logs stored on a user phone only depends on the number of people around them, and it takes one hour for 1000 different tempIDs to fill up 1GB of contact-log storage, it’s physically impossible for a user to constantly stay within 2-meter proximity of that many people.

2. New data types: As the outbreak evolves and possible symptoms of the disease are identified, the university may want to prompt users to self-report if they experience such symptoms. Further research may also lead to discovery of vaccines, and the university may want to keep track of which users have been vaccinated, which vaccination they receive and how far along in the vaccination process they are. In this case, if the additional data fundamentally changes the size and timing of server-initiated infection updates, as well as contact-log storage and contact determination process, EEE might experience *network delays*, or exceed *phone storage*. However, since EEE’s current data currency consists of seeds and tempIDs, EEE can add new auxiliary data types such as new symptoms, virus variants data and vaccination data to its messages without having to change the fundamental communication protocols and database structures.

5. Conclusion

With EEE, our app-based exposure system, we have achieved the goals laid out in Section 1:

1. EEE maintains *accurate contact determination* in determining contact with an infected person by its very definition of a close contact as a TempID logged being within a range of 3 meters for 20 minutes or longer. *Timeliness* is ensured by apps pinging the central server at a minimum every 12 hours to check for newly reported positive cases. This frequency increases to 3 hours when the number of positive cases reported is high and remains at 12 hours when the number of reported cases is low.

2. The *privacy* of the users is maintained by the 15-minute turnover in TempIDs which reduces the likelihood of external parties identifying the logged contacts. The central server that the administration and health organisation have access to also cannot access logged TempIDs in smartphones if the app does not locally match those TempIDs to a positive case and determine from the signal strength that the user was in close contact with that case.

3. *Computational impact* on the smartphones is minimized by picking out a good hash function with an even distribution of inputs to mapped outputs, and by apps generating TempIDs once daily and picking one at random every 15 minutes as opposed to running the hash function to generate a TempID every 15 minutes.

We believe that EEE provides an effective and robust contact tracing and exposure notification system that meets the present needs of both the university and users and can be readily changed to accommodate a range of new scenarios as the outbreak progresses.

6. Author Contributions

The ideation of EEE was a result of collaboration between all three team members. **Shashvat Srivastava** worked primarily on designing the server databases and ensuring the privacy of users both in the face of malicious attacks and unintentional leaks. **Veronica Muriga** worked on the client app design and assessing the computation cost and adaptability of the system. **Nhat Nguyen** worked on server-client interaction, evaluating the accuracy and timeliness of notifications, and the scalability of the system, as well as coming up with the creative name EEE.

7. Acknowledgements

We are deeply grateful to Professor Katrina LaCurts for making 6.033 both insightful and interesting, Professor Hari Balakrishnan for giving us honest and helpful feedback that enabled us to identify and rectify gaps in our design early on, and Michael Maune, our WRAP instructor, for guiding us in honing our technical writing skills over the course of the semester.