Cloud-based Indoor Contact Tracing for Infectious Disease Control

Roy Karlo Nuyda
210235593
Dr. Arumugam Nallanathan
Queen Mary University of London

***Abstract*–With the coronavirus pandemic of 2019 (COVID-19), technological infection control systems became the arguably most important tool to contain emerging infections. While outdoor contact tracing has seen significant improvements, indoor contact-tracing represents a significant problem for health technologists to solve. Each space can require an individualized solution, complicating the path towards any one solution to scale. *WiFiTrace*, an open-source solution to indoor contact tracing developed by the University of Massachusetts Amherst, processes network logs from enterprise network systems to map out users' proximity to access points along the network. We propose a redesign of the original *WiFiTrace* into a modernized infrastructure supported by the Google Cloud Platform (GCP). Our redesign imagines the application as a web-service, where participants can upload their network logs to be processed into contact tracing reports without hosting any of the application infrastructure on-site.**

I. INTRODUCTION

The effect of the COVID-19 pandemic on society and technology cannot be understated: in the course of three years, the global effort to mitigate COVID-19's worst prompted a technology race to contain the spread of the disease. Perhaps the most ubiquitous infection control technique is contact tracing, the tracking of contact with the infected persons. Traditionally, contact tracing involves a trained workforce conducting manual searches to identify potential exposures. They can be notified by health authorities for quarantine and testing when appropriate.

In the early 2010s, research began into contact tracing using mobile devices, culminating in a full-blown proliferation of the technology during the COVID-19 pandemic (Cebrian, 2021). The contact tracing applications that characterized many countries COVID-19 response adapted communication infrastructure such as Wireless Personal Area Networks (WPAN) and Wireless Local Area Networks (WLAN). Historically the development of these contact tracing technologies has always been hampered by two factors: the lack of immediate need for widespread contact tracing, and the concern for user privacy (Cebrian, 2021). COVID-19 immediately changed the landscape and a race began to develop solutions to contain the spread.

By determining proximities based on spatial data, such as signal strength and geolocation, contact tracing applications are a powerful tool for mitigating infection spread (Cebrian, 2021). Solutions like the joint Apple-Google "Exposure Notifications" application leverage different technologies to determine the proximity of two persons. With a combination of Bluetooth and Global Positioning (GPS), these solutions are fairly accurate in outdoor settings (Sharoz, 2021) but ineffective in indoor settings. Bluetooth is vulnerable to interference indoors and cannot travel through certain mediums. GPS lacks the accuracy to conduct detailed contact tracing. The same technology used for outdoor contact tracing cannot be used indoors (Sharoz, 2021).

The lack of effective indoor contact tracing, as well as the increased incidence of infection indoors, makes indoor events a more severe liability than outdoor events. The inability to conduct reliable contact tracing puts certain essential industries and services, leaving society as a whole more vulnerable to pandemics. In this project, we propose adapting a pre-existing open-source solution, *WiFiTrace*, into an easily adoptable cloud framework hosted

on the Google Cloud Platform. The key benefit to *WiFiTrace* over other indoor contact tracing models is that it requires no action to set up–by processing network system data, the *WiFiTrace* passively gathers contact tracing data.

The overarching goal of this project was to develop a cost-effective and easily distributable application for indoor contact tracing. The architecture and corresponding infrastructure scripts can easily be distributed online for any cloud-literate practitioner to adapt. By focusing on a tested open-source solution like *WiFiTrace*, we increase the chances of adoption by removing the cost of the application itself as a consideration. The results, as described throughout this project, are easily replicated due to a clear set of setup instructions. Institutions needing to monitor their indoor spaces need only an active WiFi network for the system to start working.

II. RELATED WORK

Digital contact tracing systems based on mobile devices tend to use similar architectures, categorized as either centralized or decentralized (Sharoz et al, 2021). Typically, an application needs to be downloaded onto a mobile device. Centralized contact tracing systems involve individual mobile phones transmitting their data to a centralized database where contact tracing and risk analysis is performed (Sharoz et al, 2021). Decentralized contact tracing systems involve the mobile phone downloading a contact database and performing its activities locally. The majority of contact tracing applications for the COVID-19 epidemic are centralized systems of varying levels of intrusiveness depending on the locale. The digital contact tracing system's architecture must additionally be tailored to the privacy regulations of the government the application is made available on.

There are five signaling systems that have been suggested as potential candidates for contact tracing in general: Bluetooth, GPS, QR Codes, Zigbee, and WiFi (Sharoz et al, 2021). Bluetooth and GPS are the most popular outdoor tracing systems. The best example of a Bluetooth-GPS system is Apple-Google's "Exposure Notifications", which uses GPS to approximate global geolocation and Bluetooth to approximate local proximity. Bluetooth in particular has low energy requirements compared to even WiFI, but it has a low coverage area. GPS has a global coverage area, but lacks precision. The combination of these two signaling methods cover the other's weaknesses, and this combination informs many contact tracing systems in place today (Sharoz, 2021). However, neither product works particularly well in an indoor setting.

Novel internet of things (IoT) solutions have become available on the market. By using a combination of signals, these systems leverage personal devices and access points to determine indoor location. However, these solutions often require specialized equipment that is outside of the current industrial norms. Tags or specialized applications downloaded onto personal devices are often required to determine personal location, and a distributed system needs to be configured to capture the data. An example, *Blueiot*, uses asset tracking RFID tags to potential trace contacts, but the cost of setting up this system, as well as the professional cost for maintaining the system, have prevented it from achieving the widespread use needed to call it a true solution.

Not only must these contact tracing systems work, they must adhere to the privacy regulations of the country they are deployed in. In effect, the design requirements for each application vary from country to country, as laws are often written non-uniformly. What user data the system can collect, store, analyze, and share often differs from country to country. Western countries, in particular, have difficulties implementing contact tracing systems because privacy laws often prevent these measures from becoming mandatory (Hoepman, 2022). Requiring elective active user engagement often leaves enough room for users to simply opt out of the contact tracing system altogether. Depending on the institution implementing the contact tracing system, the ability to enforce contact tracing can be difficult, limited, and even impossible.

A third issue in the design of indoor contact tracing systems is the need to individualize each system to the location it serves. The layout of the building or environment as well as the technology available are key design considerations. Developing a solution for a hospital, where the use of signaling technology might be limited, versus a school, where users may be missing a mobile device to track. Perhaps the most important barrier to consider are the technical limitations of indoor spatial recognition (Poslad, 2009). It is inherently too difficult, if not impossible, to track spatial location indoors with the variety of materials, barriers, and levels for an algorithm to consider. An indoor environment cannot be considered one open plane.

These cumulative challenges have prevented the wide adoption of indoor contact tracing systems. The cost and manpower required to implement a system often outweighs its potential benefit; it is often easier to simply close the premises than to trace contact indoors. *WiFiTrace* is by no means a universal solution to the problem altogether. Yet with the control of infectious disease, perfection is not required to determine success. If the system is able to contain infections at some meaningful degree, then it should be considered successful.

III. METHODOLOGY

WiFiTrace is an open-source contact tracing application developed by a team at the University of Massachusetts Amherst. It utilizes a command-line interface (CLI) built in Python for passive network-based contact tracing. It reads system logs from enterprise network routers, then runs a graph-based contact tracing algorithm based on proximity to an access point. The user can specify contact tracing per location or per proximity to a certain user.

The system then generates a CSV file with contact tracing data for review and analysis. The result is a stateless application that takes a single input then produces a single output without requiring setup beyond the setup of the WiFi network and the application itself.

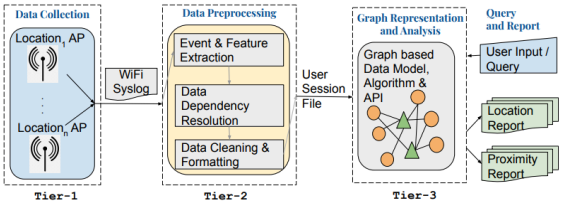
WiFi is a series of protocols based on the IEEE 802.11 standards. Because the data generated on a network device following IEEE 802.11 is largely the same, the system can be adapted to several different enterprise network systems. The system then requires no user-side application or specific tagging device to operate, so long as users connect to the network, *WiFiTrace* can generate a relatively accurate approximation of their location. By using a widely adopted wireless network, *WiFiTrace* lowers the curve for implementation. Being open-source, the program can be used and adapted rather freely, according to its common license.

By imagining *WiFiTrace* as a web-service rather than as a command line application, the system becomes more usable by non-technical users. On the front-end, users can log into their accounts, designate a MAC address to contact trace, upload a network system log file, then receive a contact tracing report without having to host the program on their own machines. A prototype of this user interface is included in Appendix A. During a pandemic, where traffic for contact tracing can number in the millions, we need a means of completing multiple workloads at the same time. We propose improving the scalability of the service with Kubernetes.

Kubernetes is an open-source orchestration system developed by Google and now maintained by the Cloud Native Computing Foundation (Kleppman, 2017). Kubernetes automatically deploys, manages, and scales applications by separating them into groups called clusters. Each cluster consists of a master node which acts as control panel or switchboard, receiving requests which are then sent to worker nodes. The worker nodes complete the jobs. When they are not working on jobs, the worker nodes remain dormant, costing less resources to maintain. The main advantage of Kubernetes is the management of computing resources according to request load (Kleppman, 2017). For large-scale web applications, Kubernetes has become the preferred way in industry to manage computing resources. An example of Kubernetes infrastructure is included in Appendix B.

We describe a naive implementation of this design in the implementation section of this paper using the Google Cloud Platform. Similar solutions can be generated using Amazon Web Services or Azure, but we have found that the Google Kubernetes Engine (GKE) has by far the best support for Kubernetes orchestration. With GKE, Kubernetes is native, meaning that managing clusters after Kubernetes updates is far easier. As a managed Kubernetes instance, GKE allows the user to set different metrics, such as CPU usage, to scale the number of nodes. As more requests enter the system, GKE activates more worker nodes to handle these requests, increasing the efficiency of the application.

The current state of *WiFiTrace*'s source code splits the application into two main components, the preprocessor and the main contact tracing program. The preprocessor takes a network system log as an input then produces a session file with the data from the system logs packaged for processing. The main contact tracing application takes the session file as input, conducts contact tracing, and then produces a csv of the potential infections. A chart of the project's dataflow in its source code is listed below:



For this project, these two components can be imagined as two separate services. The preprocessor as a generator of session logs and the contact tracer as an analyzer of those logs. The problem with the current WiFiTrace source code is that it is configured to handle only one job at a time. By treating these two services as separate services, we can process session files and contact traces in parallel. This requires intermediate short-term storage where the preprocessor stores the session files and the contact tracer retrieves session files. A user can either make a session file request, where they upload network system log data and then generate a session file. From their session files, they can request contact traces either by person or access point. In particular, it is not always the case that contact tracing needs to be run. Session files can be created for specified time periods in preparation for possible contact tracing. The preprocessor can be packaged differently depending on the enterprise network provider, as each merchant produces different sets of network system log data.

The stateless nature of *WiFiTrace* makes it an ideal candidate for Kubernetes, as the application does not need to save a state at any point. Rather, the main two components of the application complete separate jobs. This greatly simplifies the data storage component of the application - when a session or contact tracing file is created, we only need to store it temporarily. Files that have not been accessed after a short amount of time are deleted, simplifying our storage requirements. At scale, storing hundreds to thousands of session files that can contain thousands of records will be costly, even on a Cloud Platform.

We refer to the official Google Cloud Platform documentation (Google, 2022) for implementation as a guide to implementing this software. Namely, we leverage GCP's recommendation for the refactoring of monolithic applications into microservices. Appendix C includes a design of the *WiFiTrace* infrastructure hosted on the GCP, including the service names.

As a web-service, this version of *WiFiTrace* not only accounts for requests by a single user, but for multiple users making requests simultaneously–the single largest problem for a contact tracing platform imagined as a web-service is the sheer number of requests that can occur at the same time. By using GKE, we can scale the number of worker nodes according to multiple requests, processing jobs both at the preprocessor at contact tracing levels.

By leveraging Docker containers, we can create different configurations of *WiFiTrace* without management references in the code itself. Modifications to the preprocessor for particular network providers can be encapsulated in the preprocessor image independent of the other components. We can create special instances of the application using unique URLs to direct traffic to these modified containers. There are other cloud-provided tools, including storage of the session and csv files, that can be leveraged on a case-by-case basis. These options are discussed in the Further Work section.

IV. REQUIREMENTS

Every user must be able to fulfill three separate tasks: submit system logs for preprocessing into a session file, query the system for contact tracing proximity around a person, and query the system for contacting tracing proximity around an access point. The non-functional requirements include security, privacy, performance, and reliability. For the basic *WiFiTrace* application, the hardware components include a WiFi networking system, a mobile phone connected to the WiFi system, and personal platform to run the preprocessing and querying program. The software requirements include Linux, C (depending on the network devices, Linux and C assumes a Cisco router), and Python.

A. Functional Requirements

- Interface: The system needs to be able to access, process, and display system log data from a WiFi networking system.
- Capacity: the system needs to store processed information
- Analytics: the system needs to generate data in a form amenable for data analysis, such as JSON or csv.

B. Non-Functional Requirements

- Privacy: User Data must be anonymised to a certain extent and protected health information must only be accessed when required
- Adaptability: the system must be able to be customized to the situation / location it is introduced in, ie an office or a school.
- Reliability: the system must be able to generate contact traces reliably
- Efficiency: due to the large dataset, the system must be able to produce contact traces efficiently.

C. Hardware Requirements

- Because our system largely operates on existing network architecture, there are no novel hardware components. For a function like contact tracing, being able to produce a feasible solution on top of already existing infrastructure without introducing additional hardware components is arguably a benefit. Users will be able to adopt the system at decreased cost and manpower.
- A WiFi-enabled Smartphone that is able to produce a MAC address (Assuming in this case, a mobile phone). Typically, on mobile phones, a card with antennas is attached to the system on a chip (SoC) which connects with different networks, including WLAN. Specifically,
- WiFi System
- The WiFi system needs to be to maintain a WLAN to the IEEE 802.11 standards.
- Computer

D. Software Requirements

- Python 3.6, including the libraries pandas and flask.
- An IDE, VSCode, for developing Python code.
- The code base for WiFiTrace, available on github (Triveldi et al, 2021), which is written in Python.
- Operating System on the personal device, iOS or Android most typically. iOS is written in a combination of C, C++, Objective-C, Swift, and assembly language. Android is written in Java, C, C++, among other languages.
- Operating System on the WiFi device acting as an access point. For Cisco

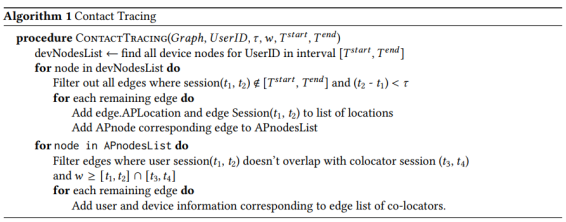
routers, the operating system is a Linux system written in C.

- Access to the Google Cloud Platform, either on a desktop Cloud Shell instance or online through the console.
- Google Cloud Platform, specifically the Google Kubernetes Engine services.

E. Algorithm Requirements

WiFiTrace uses a graph-based algorithm to conduct efficient contact tracing. Both the devices on the WiFi log and each access point (AP) are modeled as nodes. Edges are annotated with the time interval (t1, t2) associated with the start and end times of the association between the device and the AP. The pseudocode for the algorithm is presented in figure 1 (Triveldi et al, 2021). The algorithm produces two collections of information, a location report and a proximity report. The records in these reports contain the following pieces of information: Timestamp, AP Name or ID, Device MAC ID, event type, and an optional Username. It has been shown that this graph algorithm performs better than a naive linear search (Triveldi et al, 2021).

Figure 1. Pseudocode for Contact Tracing Algorithm



Algorithm 1 Contact Tracing

procedure ContactTracing($Graph, UserID, \tau, w, T^{start}, T^{end}$)
devNodesList ← find all device nodes for UserID in interval $[T^{start}, T^{end}]$
for node in devNodesList **do**
Filter out all edges where session$(t_1, t_2) \notin [T^{start}, T^{end}]$ and $(t_2 - t_1) < \tau$
for each remaining edge **do**
Add edge.APLocation and edge Session(t_1, t_2) to list of locations
Add APnode corresponding edge to APnodesList
for `node in APnodesList` **do**
Filter edges where user session(t_1, t_2) doesn't overlap with colocator session (t_3, t_4) and $w \geq [t_1, t_2] \cap [t_3, t_4]$
for each remaining edge **do**
Add user and device information corresponding to edge list of co-locators.

Modifications to the algorithm to include the RSSI-derived positioning estimate could be handled in different ways. The simplest would be adding the estimate as a separate value as an annotation for each edge, where t^start, t^end, and l (read as location) are the values on each edge. Whether this level of accuracy is necessary depends on the environment the system is deployed in. For larger environments where the distance between each AP is larger, precise position may be necessary. For smaller environments with more APs, modifying the algorithm to include an estimated position. One concern with this type of data structure is the time and space complexity. The graph algorithm is already an improvement over a linear search in terms of time, but the worst case space complexity is O(V^2) when all nodes are connected. Given that the dataset could number well into the tens of thousands, there needs to be considerations for space. We can reduce the number of user nodes on the graph by increasing w, which represents the amount of time the user spent connected to the

F. Security Requirements

Because the system operates on passive WiFi logs, there are no client-side requirements. To gather data users may be prompted to sign in on WiFi–it certainly is not a requirement. The users of the application are entirely the administrators of the network who only need to generate reports and manage user data. The biggest source of tension for this project are in the privacy requirements, which are subject to legislation (Poslad, 2009). User health and location information is strictly on a need-to-know basis, and that information is only released when there is a strong match for possible infection. Standard network defense features, such as a firewall, can be used to protect user information on the network level. Application-side features, including two-part user authentication, will likely be necessary on the application level to increase the system's security.

V. IMPLEMENTATION

The implementation for WiFiTrace on the Google Cloud Platform as the components are stateless. Stateless applications are applications that do not save client data for the next use, as opposed to a stateful application, which stores data for the next session. It is much easier to migrate applications onto containers when they are stateless (Google, 2022). Much of this project's implementation features come from Google's official documentation.

A. Initializing Google Cloud Project

Google Cloud workloads are organized into structures called projects, with permissions to these modules controlled by user accounts called roles. For this particular project, we do not need much apart from starting our initial project then using the default administrator role, because we only assume one user. The name of the project is WiFiTrace test, and from there, we begin preparing our code for migration. An important point not to forget is "Enable Billing" on the Google Cloud account console, as the GCP instance will not work if there is no billing account set up.

B. Refactoring the Source Code

The first step in migrating an application to the Google Cloud Platform is to create docker containers for the code. GCP has a built-in Kubernetes configuration tool called Cloud Build, which allows you to containerize your application into GCP without using a desktop docker instance. Cloud Build creates a container image which can be loaded onto a GKE instance, which can be designated at the start of the build. The benefit to working with cloud services is the integration of the tools. When a connection needs to be made from one component to another, GCP often has a native route available to do so.

C. Building a Front End

A prototype of a simple front-end for the program is provided, with the outputs and inputs.

D. Deployment

From the Google Cloud Platform console, we can specify the address of our GKE on the front-end, so that user jobs are sent accordingly. We include a mockup of this front-end in Appendix A.

E. Analysis

Deploying a simple application like *WiFiTrace* requires little legwork on GCP, which is a primary driver for cloud migration. By supporting automatic scaling and descaling of resources, the computing resources required by the application are better managed in the long run. Anticipating a surge in requests during an event like a pandemic, we can allocate more worker nodes to ensure the application remains available.

VI. EVALUATION

The lack of readily available testing data in the form of network system logs from multiple users affected our ability to properly test implementations of the system. Based on an analysis of the algorithm, we evaluate *WiFiTrace*'s implementation as it answers the project's central problem of indoor contact tracing.

A. Complexity Evaluation

The first component, the preprocessor, reads the system logs which are processed as a list, which can be pictured as A. The preprocessor then reformats the contents of A into a list, stored in a separate data structure B, so the time and space complexity of these operations should be O(A + B), as the program only writes and reads each item in a single pass. The preprocessor then creates a session file, which is read by the second component, the main contact tracing application. The contact tracing application generates an unstructured graph data structure based on the session file list, which can be represented in terms of time as O(A). Once the initial graph is built, the system then performs a search on the graph to obtain potential contact traces, which is O(V + E), V and E being the number of vertices and edges. The total time of the algorithm between the write and read equals O(2V + E), with V being both the items of the list and the corresponding nodes on the graph.

Because we are not dealing with quadratic algorithm complexities, *WiFiTrace* will run smoothly for most small inputs, but once the application begins reading larger inputs, the system will likely take at the very least hours to trace a single contact without significant concurrency support. Refactoring the

code to process each file in chunks is likely required to scale the application for large-scale events.

WiFiTrace currently produces a single csv file, which is not scalable. Read operations on CSV files take significant time, as a data structure representing the values of the file must be constructed along with the file. In Further Work, we propose refactoring the code to another language that better supports concurrency, such as Golang or Rust, with an added graph-based NoSQL database like Neo4J to store the graph data structure of contacts.

Testing on different sizes of datasets needs to occur in order to truly determine the application's accuracy and efficiency, but from the preliminary results delivered by its original researchers, the application is fairly accurate.Once the program is determined to be accurate enough, the scope of the program's operations is simple enough that a short testing schedule would be sufficient to determine its feasibility as a long-term project.

B. Usability Evaluation

WiFiTrace, as a technology, has several limitations that should be well-understood before use. As connecting to the WiFi network on the premises is voluntary, without requiring everyone on premises to connect to the network, there is no guarantee everyone in the building will join the contact tracing scheme. Also, the system works by having adequate access point coverage. If there is no signal at any point in the building, then there would be no contact tracing data for that location. WiFi can suffer interference with its signals, and its location sensing is inaccurate in multi-story buildings. Because *WiFiTrace* leverages an existing network, it is a vastly more simple implementation than the IoT contact tracing solutions available on the market.

For the simple use of contact tracing on a single plane, *WiFiTrace* is an effective and cost-efficient solution. By hosting the service on GCP, we can leverage the adaptability of GCP's Kubernetes engine to customize the instances of the application according to the region, or even create specialized instances for certain users. This is particularly important as security and privacy regulations vastly differ from country to country. Overall, we believe *WiFiTrace* hosted as a web-service is an efficient means to implementing indoor contact tracing, even if there are limitations in its use cases.

VII. FURTHER WORK

There are a number of changes that could be made to further improve *WiFiTrace* that leverages the power of scalable cloud infrastructure. The first is concurrency support. We could refactor the code to support the splitting of the job in components, allowing the virtual machines to utilize multiple cores. For particularly large datasets, more virtual machines could be recruited to process chunks of data, greatly increasing the performance of the system. Python has concurrency support, but we lean towards a language like Golang which has a much simpler means of supporting concurrency. We likely keep the main contact tracing application in Python because it uses the well-supported *pandas* library, but move the preprocessor to Golang.

The output of the contact tracing application could also be ported out to a cloud native data solution, such as Cloud Storage, or Big Query, which is the data warehousing and analytics service provided by Google. Users could look up previous contact tracing reports, and health officials could leverage GCP's analytic capabilities to perform analyses on the contact data.

REFERENCES

Cebrian, M. 2021. "The past, present, and future of digital contact tracing". Nature Electronics, Vol 4, no 2 - 4.

Google. 2022. "Google Cloud Documentation". Available at: https://cloud.google.com/docs (Accessed: August 2022)

Hoepman, J.H., 2020. A critique of the google apple exposure notification (GAEN) framework. arXiv preprint arXiv:2012.05097.

Kleppman, Martin. 2022. Designing Data-Intensive Applications. O'Reily Media Incorporated, New York.

Mazuelas, S. 2009. "Robust Indoor Positioning Provided by Real-Time RSSI Values in Unmodified WLAN Networks". IEEE.

Sharoz, Muhhamad et al. 2021. "Covid-19 digital contact tracing applications and techniques: a review post initial deployments". Transportation Engineering. Vol 5.

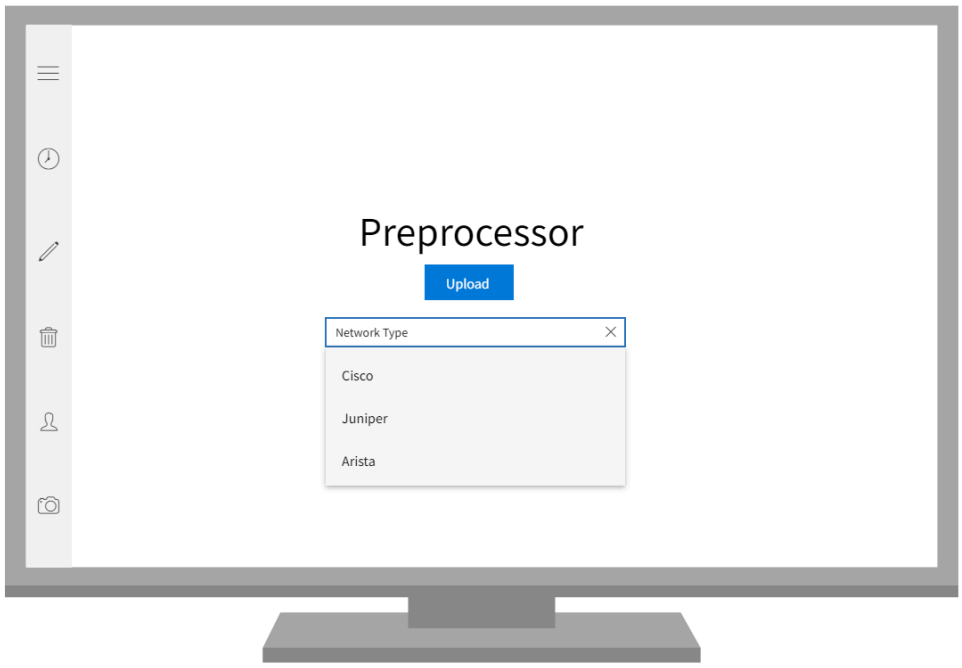
Stefan, Poslad. 2009. Ubiquitous Computing. John Wiley & Sons, West Sussex.

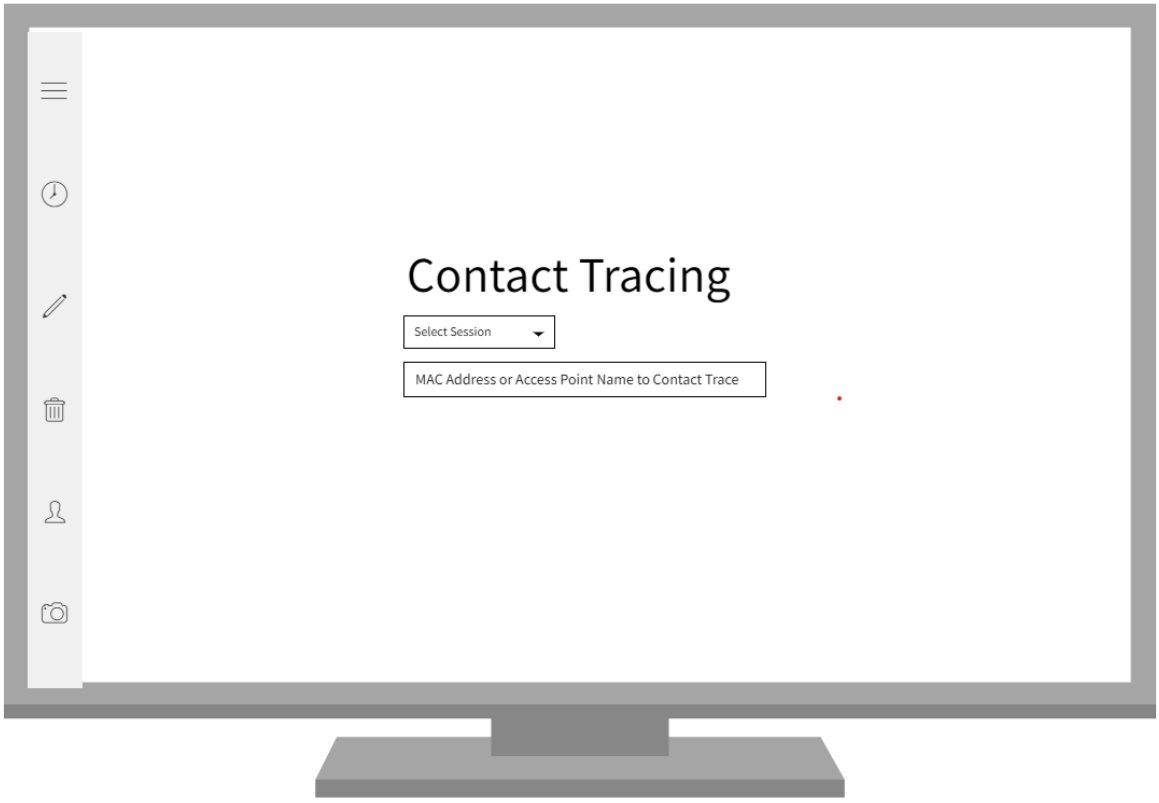
Trivedi et al, 2021. "WiFiTrace: Network-based Contact Tracing for Infectious Diseases Using Passive WiFi Sensing". ACM Interact. Mobile Wearable Ubiquitous Technology Vol 5. No1, Article 37.

Appendix A

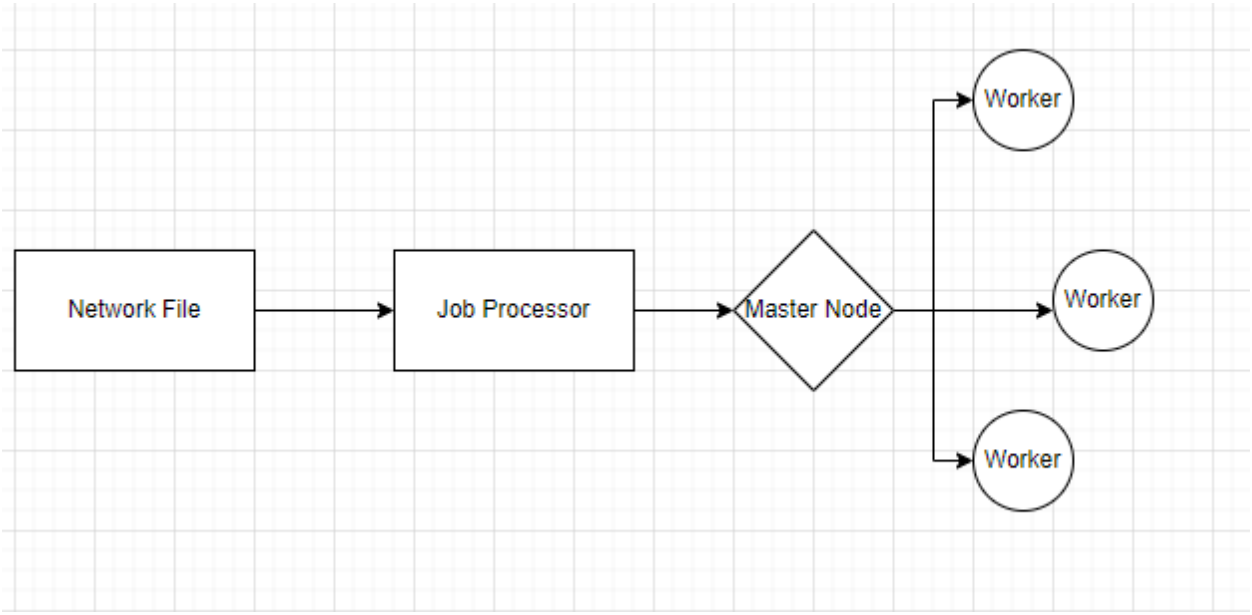

WiFiTrace Portal
username
password


Welcome to WiFiTrace
Please Select a Function
Preprocessor
Contact Tracing
Account

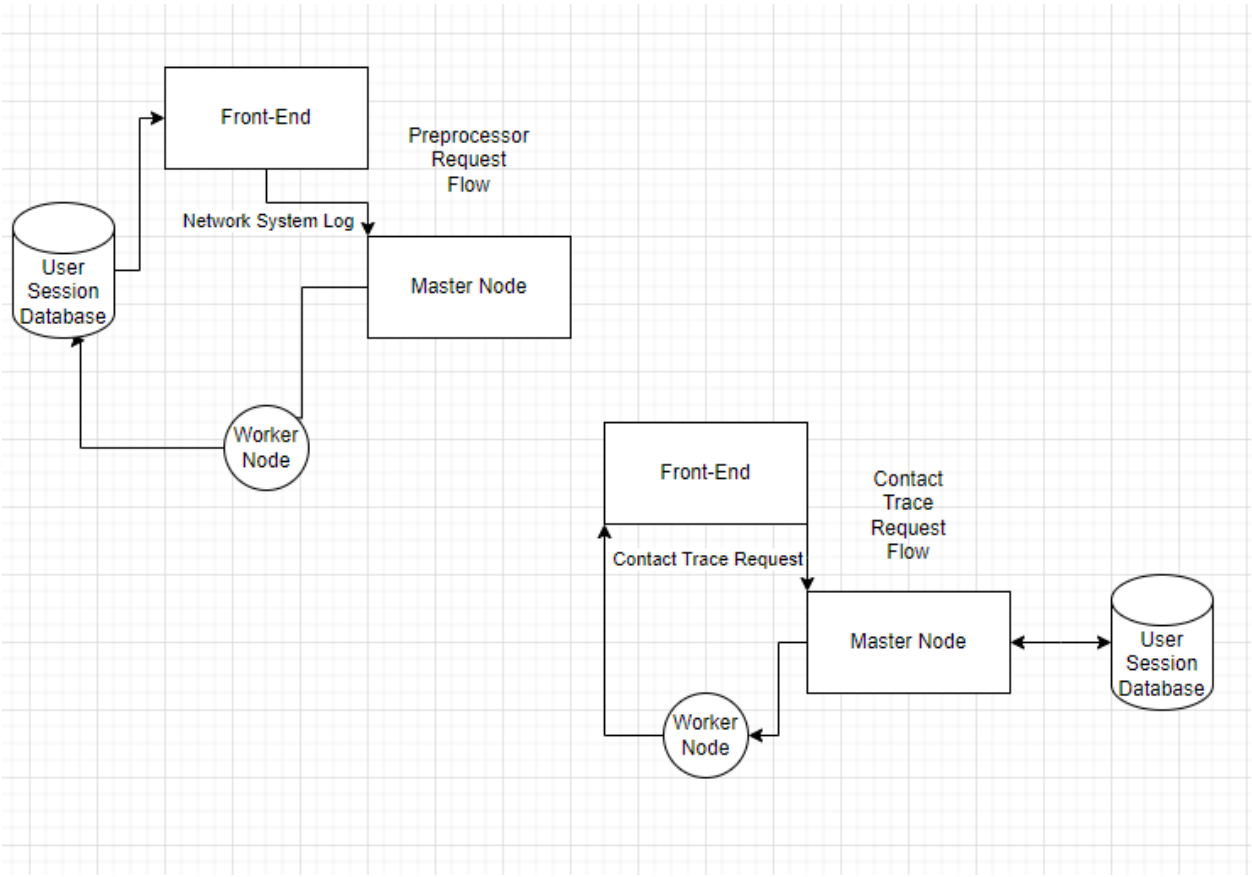

Preprocessor
Upload
Network Type
Cisco
Juniper
Arista


Contact Tracing
Select Session
MAC Address or Access Point Name to Contact Trace

Appendix B



Appendix C



MSc Project - Reflective Essay

Project Title: Cloud-based Indoor Contact Tracing to Combat Infectious Disease
Student Name: Roy Karlo Nuyda
Student Number: 210235593
Supervisor Name: Dr. Arumugam Nallanathan
Programme of Study: Computing and Information Systems

1.1 Strengths

- The design included here uses a very novel and relevant technology to solve an immediate problem related to infectious diseases. As we conducted our literature review, we became aware of the absence of indoor contact tracing schemes on the market today, especially those that do not require the user to download an application, use QR codes, or hold a tracking device.

- In order to configure the cloud implementation, we needed to deeply understand the existing application architecture as well as the networking fundamentals to ensure the platform can be migrated successfully, essential skills for cloud engineers working on infrastructure migrations.

- The solution pays great attention to the scalability and distribution of the service, improving upon the open source code.

- Leverages GCP's Kubernetes engine for automatic scaling and management of computing resources.

1.2 Weaknesses

- Lack of expert evaluation or user evaluation. We did our best to evaluate the program based on our knowledge of networking limitations and algorithm performance.

- Simple UI.

- There is little about the actual implementation - this is more of a prototype and feasibility study.

2. Presentation of Possibilities for Further Work

- We could refactor a few pieces of code into a language with higher performance, like Golang or Rust, in particular the preprocessor. The main contact tracing application should stay in Python, because the algorithm runs using pandas to build the dataframes. That particular library is more well-supported than any other equivalent.

- We could consider improving the accuracy of the location-sensing with a novel technique, like this triangulation method based on relative signal strength values.

- We could add in an analytics dashboard to analyze contact tracing data after the contact tracing is completed.

- We could do further research on the privacy regulations of a single country then implement that into the project.

- A larger scale project would be to store the values produced by the contact tracing application into a graph-based database that could be accessed in the future. This would require refactoring the contact tracing algorithm and the output of the contact tracing application, as well as require some configuration on the backend. In this respect, the web-service would be a truer "web-service" in the style of many modern websites.

3. Critical analysis of the relationship between theory and practical work produced

During this project, I confronted a challenging problem to understand for someone with limited programming experience. There were multiple areas I needed to learn and understand to create a feasible implementation of this application:

3a. Programming

As a Computing and Information Systems student, I did not have exposure to complex programs like *WiFiTrace* during my course, so I first needed to understand the Python code provided, including the libraries invoked. At first I planned on converting the entire code to another language (from Python to Golang), but this proved unfeasible because of the sheer amount of knowledge required of both languages to even begin migrating the code. There is no guide or website that informs you of library equivalencies between languages–each library and function call is often another trip to Google. This exercise (attempting to convert Python code to Golang) made me more familiar with Python than I was before the project, as I learned the "Pythonic" ways to handle certain situations as well as a more "Go" way of handling situations. After reading through the entirety of the source code, I realized Golang may be a suboptimal choice for this application, because the main algorithm utilizes *pandas*, and the main algorithm is over 500 lines long. The end result was a prototype, feasibility study, and architecture design for a cloud implementation of the application.

3b. Networking and Computer Architecture

Working with a WiFi-based contact tracing tool pushed me to learn a lot about WLAN in a very short amount of time. I relied heavily on the *Mobile Services* module I took during the second term, as well as Professor Poslad's book *Ubiquitous Computing*. By deep diving into the source code, I needed to learn what data networking hardware collects and the differences between network manufacturers. Understanding basic Linux architecture in order to configure the Kubernetes clusters was also a challenge, as I had relatively no experience with Linux machines prior to this project. Learning about containers and Kubernetes architecture is a vital skill for the kind of work I would like to pursue, so this project worked well to help me prepare.

3c. Cloud Computing

My interest in creating a cloud implementation came from some certification work I did after the term ended. This is the area of computing that I will likely being working in after I complete this degree, so I invested a great deal of time learning its intricacies I became pretty knowledgeable of GCP, so it was a natural fit for me to design an architecture compatible with cloud computing, leveraging the automatic scaling of Kubernetes clusters. I needed to learn how to host an application on GCP, which GCP service to choose from, and how to connect different GCP services together.

WiFiTrace is not the typical program hosted on cloud, so there was a lot of knowledge required. Initially, I planned to use Cloud Run, which is a managed container service on GCP–I realized the basic architecture of *WiFiTrace* would need to be completely refactored in order to host it as is on Cloud Run; we would need to create a REST API for this to work, so I opted for Kubernetes Engine instead, because I

could still feasibly run the command line interface with Kubernetes. Working with Kubernetes required me to learn about Docker, so I was able to learn two key technologies in a single project. For the web-services model, there was some requisite system design knowledge I needed, which required outside research. My Mobile Services module was probably the most useful module in this respect, as we covered distributed systems.

4. Awareness of Legal, Social Ethical Issues and Sustainability

I added in a few sections about the privacy laws affecting contact tracing in the paper itself, but in short, the legal system in which this system is implemented greatly affects implementation. The kind of data that mobile devices are able to gather–then use–differs from country to country. How personally-identifiable data (PID) is stored also differs from country to country. In many cases, we would need to add more levels of encryption to ensure the data cannot be accessed unless for the express purpose of contact tracing. As I reflect back on this project, it would be enormously difficult to roll out a global solution without creating several different instances of the application. Every country would need different (but perhaps similar) implementations depending on their privacy laws. We would likely need to add a robust authentication system for users, especially on the web-service, then review the system regularly for security flaws. One of the reasons why Cloud is such an attractive option versus an onsite server is that the cloud provider does some of the security legwork, especially when it comes to the management of the actual servers. The ability to virtually manage multiple instances (then direct traffic using a load balancer) is the primary driver of Cloud in this case.

The issue of privacy is far more complex than just legality. Whether we should be tracking locations at all is a serious issue in every country, especially after COVID. As the number of tools to contain the spread of COVID-19 increased, so did the number of ways to digitally monitor a population. For example, the technology behind *WiFiTrace* could be used to determine indoor location for a number of reasons, including law enforcement. Passively capturing data is also a moral gray area; *WiFiTrace* operates by passively picking up the proximity of users on the WiFi Network. Developers need to seriously consider the ramifications of the technologies we create, even if they are at first dedicated to a noble cause. These technologies will live on even when COVID-19 becomes a recurring disease. Legal systems to some degree exist to protect privacy, but there are also legal systems which infringe on privacy. Should we allow users to opt out of tracing and still use a network? For private companies, do the same privacy regulations apply as they do to public institutions? There are a host of legal issues that accompany this application.