6.033 Spring 2021 Alternate Design Project

See also: DP FAQ, DP Errata

*This design project is based on the 2017 DP, which is how we were able to create an alternate DP relatively quickly (and why the formatting of the this DP looks different from that of the exposure-tracing DP). Many of the components described in Section 2 existed in the 2017 DP; however, the goals of this system, and the parts we are asking you to design, are **not** the same as they were in 2017. A good solution to the 2017 DP would be at best an incomplete solution to this DP, and at worst, an actively bad solution.*

Though the scenario we are asking you to consider here is substantially different from the exposure-tracing DP, we have intentionally made the big questions similar: What data should your system store where, and for how long? How does data get from clients to servers? What is the overhead of that, and how is that data transfer affected in cases of high load? How does data get sent back to clients, and how does your system determine which clients need which data? How does your system take privacy into account?

Due Dates and Deliverables

There are five deliverables for this design project.

1. **DP Prep (DPP)**: In order to help you prepare with your team design effort, this assignment will require some guided analysis of the DP specification below. This assignment will be written by each student individually and is due March 12, 2021 at 5pm, EST.
2. **DP Preliminary Report (DPPR)**: This preliminary report will lay out your key design decisions, including both a functional system design and a sketch of any data structures, storage management, and/or network protocols required to achieve your design. It will not include any significant evaluation. It will be approximately 2,500 words and is due March 30, 2021 at 5:00pm, EDT.
3. **DP Presentation**: This presentation will address the feedback received on the DPPR, and any corrections or updates to the design project specification. It will also outline evaluation criteria and use cases you will use later for evaluating your design. The presentation will occur during the week of April 21-27,2021, EDT.
4. **DP Report (DPR)**: This will be your full report. It will include your final design, all diagrams appropriate for that, your evaluation of your design and a review of how effectively your design addresses the specified use cases. It will be approximately 6,000 words and is due May 11, 2021at 11:59pm, EDT.
5. **Peer Review:** In Tutorial your team will have done an early "review," providing informal feedback to another team on their design. For this peer review, you will individually review a few specific sections of that (same) other team's final report and will address some specific

questions about that report. It will be approximately 250 words and is due May 14, 2021 at 5:00pm, EDT.

Your assignment for each of the five parts above will be distributed in separate “assignment” documents.

The prep, preliminary report, final report, and peer review should be submitted via the submission site on the 6.033 website. As with real-life system designs, the 6.033 design project is under-specified, and it is your job to complete the specification in a sensible way given the stated requirements of the project. As with designs in practice, the specifications often need some adjustment as the design is fleshed out. Moreover, requirements will likely be added or modified as time goes on. We recommend that you start early so that you can evolve your design over time. A good design is likely to take more than just a few days to develop. A good design will avoid unnecessary complexity and be as modular as possible, enabling it to evolve with changing requirements.

Large systems are never built by a single person. Accordingly, you will be working in teams of three for this project. Part of the project is learning how to work productively on a long-term team effort. **All three people on a team must be in the same tutorial.**

Although this is a team project, some of the deliverables have individual components. See the individual assignment links for more information.

Late submission grading policy: If you submit any deliverable late, we will penalize you one letter grade per 48 hours, starting from the time of the deadline. For example, if you submit the report anywhere from 1 minute to 48 hours late and your report would have otherwise received a grade of A, you will receive a B; if you submitted it 48hours and 5 minutes late, you will receive a C.

You must complete the three team design project components, parts 2, 3, and 4 above to pass 6.033. For the other two (individual) components of the design project, the contribution to your overall grade will be whatever grade you receive on that component. Thus, if you choose not to do one or the other of them, you will receive an F for that component only as a contribution to your overall grade.

1 Introduction

The MBTA provides public transportation for the Greater Boston area via a network of buses, subways, commuter rails, and boats. For this project, we're going to consider only the buses.

To be a successful public transit system, the MBTA must meet various requirements for:

- **Availability.** Buses should operate during a reasonable part of the day—known as the *span of service*—and at a reasonable frequency. The bus network must also cover the entire Greater Boston area.
- **Accessibility.** Buses should be accessible to customers with disabilities; they must be ADA-compliant.[1]
- **Reliability.** Buses should depart on time, and maintain the timetable for scheduled stops.
- **Comfort.** Buses should not be too crowded.

The MBTA has designed their bus network to meet the first two requirements: they have data on the population of the metro area and have chosen the span of service and routes to cover the area, and the buses are equipped with the necessary ADA-compliant devices. But assessing whether the bus network is meeting requirements for reliability and comfort requires real-time tracking, as does enabling the system to respond to failures.

The MBTA is soliciting design proposals for incorporating passenger feedback into its real-time tracking infrastructure via a passenger feedback app. They want to use passenger feedback to calibrate their existing reliability and comfort requirements, and to communicate *in real time* to affected passengers about any changes in the service schedule.

This system has many constraints. The sensors on the buses, the communication network among the buses, the communication network between the passengers and the MBTA, and the capabilities of the server all place constraints on the amount of data that can be processed in a timely manner. The amount of data collected about passengers presents very real privacy concerns. The MBTA is also constrained by cost: they cannot hire new operators, purchase new buses, etc. The system will need to work well using the resources it currently has.

2 Background: Existing Infrastructure

2.1 The MBTA Bus Fleet

The MBTA operates a bus fleet of 1036 buses on 177 routes over the Greater Boston area. On average, 991 are active at any given time; the other 45 are reserved for failure recovery.

In this project, a bus "route" refers to a round-trip route, starting and ending at the same location. Though we typically imagine a bus traveling the first half of its route, turning around, and then servicing those same stops in the reverse direction, this is not the case. For one, the second half of a bus route may be slightly different to accommodate one-way streets. For two, when traveling in the opposite direction, a bus serves stops on the opposite side of the street. Routes do have a specified "midpoint": the point at which the bus, effectively, turns around to travel back to the start.

[1] https://www.ada.gov/

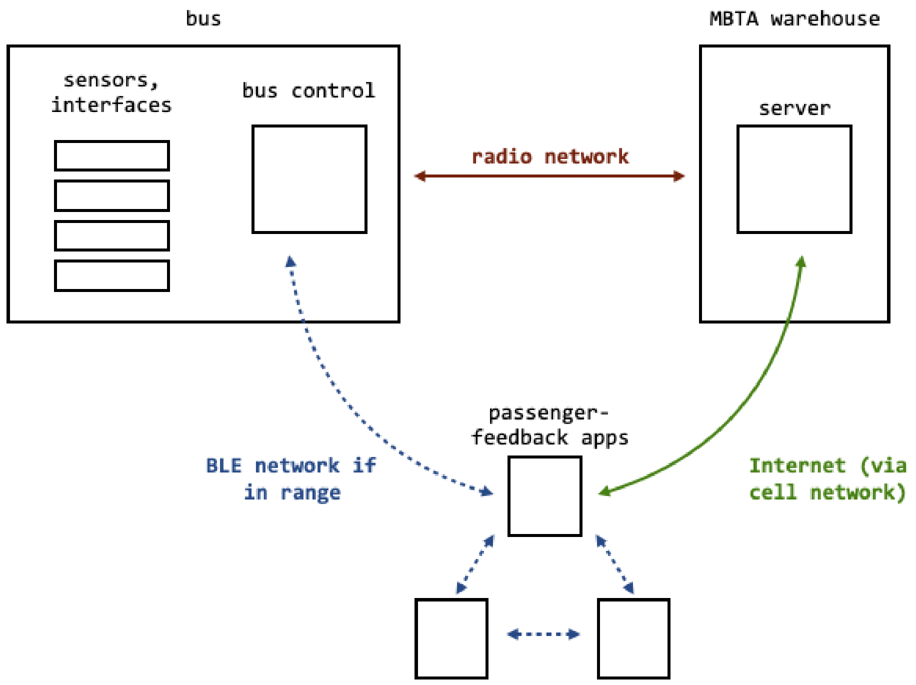

Figure 1: A diagram of the existing MBTA infrastructure involving a single bus and multiple passenger feedback apps. We describe the details of this diagram, as well as aspects not pictured, below.

The buses are designed to travel routes that have been pre-determined by the MBTA. These routes range in length from roughly 4 miles to 38 miles. Each bus has a unique, 48-bit ID assigned to it by the MBTA; these IDs do not change over the lifetime of the bus.

The MBTA also has 2000 bus operators, to drive the buses.[2] At the start of each day, an operator must drive their bus from the centralized MBTA warehouse out to the start of their assigned route. If a shift change happens during the day, a new operator will travel to the route's origin and take over the route. At the end of the day's service, MBTA operators drive the buses back to the centralized warehouse.

The MBTA aims to have each bus stop serviced at least once every twenty minutes during the span of service, and has scheduled its routes to meet this goal under normal operation.[3]

2.2 Devices on Buses

Each bus is equipped with a number of sensors that collect data, as well a small computer known as the *bus control* that reads that data and transmits it to the MBTA server via the system-wide radio (described in Section 2.5.1). The bus control has some amount of persistent storage available, and can also communicate with the passenger feedback app (described in Section 2.5.3).

[2]In reality, the MBTA has closer to 1700 bus operators, not 2000. We've given you more operators to work with because we have made some additional simplifying assumptions about the frequency of bus service.

[3]In the real world, each route has a different target frequency (and a different span of service). We've made a simplifying assumption here for the purposes of the design project.

2.2.1 Positioning Sensors

The **GPS sensor** on each bus uses the Global Positioning System (GPS) to calculate the bus's position (latitude and longitude). The GPS sensor updates its data every one second. Each piece of position data is 64 bits. This data is not automatically pushed to the bus control but may be read by the bus control at any point. For the purposes of this project, we will assume that GPS data is reasonably accurate: it always reports a bus's location to within five meters of its actual location.[4]

2.2.2 Passenger-Count Sensors

The bus is equipped with a **beam sensor**, which use two infrared beams to determine when a passenger passes through the door of the bus. Each time a passenger passes through the door, the beam sensor sends that event to the bus control. The event also indicates whether the passenger was entering or exiting.

Data from the beam sensors can be a bit noisy. Though they can distinguish between passengers entering or exiting, they often generate incorrect data when passengers are wearing large backpacks, using wheelchairs, or entering side-by-side. On average, beam sensors are 85-90% accurate. The MBTA buses are only equipped with beam sensors on the front doors of the bus, not the back doors.

2.2.3 Payment Interface

Also installed at the front of the bus is the **payment interface**. The MBTA accepts three forms of payment for buses: cash, tickets, and Charlie Cards.

When a passenger pays—regardless of the form of payment—the payment interface makes a record of the transaction and sends that data to the bus control. When a passenger uses cash, the transaction record only includes the timestamp (32-bit UTC timestamp) and the cost of the ride (32-bit floating-point number). When a passenger uses a ticket or a Charlie Card, the transaction record also includes the ID of the ticket or card (a 64-bit ID, read directly from the ticket/card), as well as information about whether the passenger is transferring from a different bus. The ticket/card IDs are unique, and generated when the passenger purchases the ticket or card.

For the purposes of this project, the primary difference between these forms of payment is that a passenger using a Charlie Card will use the same card over a long period of time (months, or perhaps years), because passengers can add monetary value to the cards. Tickets cannot be reloaded, and so are no longer used once their monetary value is zero.

2.2.4 Bus-Operator Interface

The **bus-operator interface** is used to display route information to the bus operator. In particular, it should always display the next three stops on the bus's route. This route data is available from the MBTA's centralized server (Section 2.3.1). The bus-operator interface can communicate directly with the bus control. The bus operator also has a radio transmitter and receiver that allow them to talk to someone at the MBTA (Section 2.5.1).

[4] In reality, this is not true. For instance, GPS tends to be inaccurate in urban canyons. To deal with this, buses use additional sensors to perform *dead-reckoning*, an alternate means of calculating position.

2.3 Devices and Data at the MBTA

At its centralized warehouse, the MBTA has a single centralized server: a 64 core machine with 64GB of memory and 12TB of storage. This machine can communicate with the bus controls via the system-wide radio (Section 2.5.1). So far, the server has never failed.

The MBTA is expecting that your system is going to be used heavily. If you find that this single server is not enough, your design should argue why the MBTA should purchase additional storage.

2.3.1 Data

The MBTA already has a number of datasets that it must store:

- **Census data**, which the MBTA uses to plan its bus routes, to determine which areas have a high population density, among other things. This dataset is 600 MB.
- **Bus meta-data**, which stores information about each bus: its (unique) ID, its current operator and route assignment, and some additional information (the bus's make, model, etc.) This dataset is 10 MB.
- **Route and schedule data**, which stores information about each route. Each route has a unique 64-bit ID, as does each stop. The information about a route contains a list of the stops (including their IDs, lat/lon, and a text description of the stop) and the timetables for each day's trips on that route. Those timetables specify the times at which a bus should arrive at a particular stop. This dataset is 100 MB.
- **Alternate stop data**, which stores information (IDs, lat/lon, a text description) about locations that might be used as alternative stops. This dataset is not needed during normal operation, but may be useful as part of your failure-recovery plan. This dataset is 100 MB.
- **Operator data**, which stores information about each bus operator: their operator ID (a unique, 64-bit ID), whether they are currently on duty, the route they are currently driving, and a list of routes they have driven in the past month. This dataset is 10 MB.

You are responsible for specifying how the server stores these datasets. In addition, you are responsible for specifying how the server stores the data from the buses.

2.4 Devices and Data on the Passengers

The passenger feedback app is the biggest upgrade to the MBTA's real-time tracking system, and you will be able to control most of its design. For now, we will assume that all passengers have smartphones with the app enabled (which is of course not true in practice). The passenger feedback app is limited to 1GB of storage on each phone.

At a minimum, the MBTA wants the passenger feedback app to have three features:

1. Passengers can report that a bus is too crowded
2. Passengers can report that a bus is running late
3. The MBTA can send alerts to relevant passengers when there is an unexpected event (e.g., a route change)

It may be useful for the passenger feedback app to have additional features, or to get additional inputs from passengers. Though it's important to consider a passenger's experience, you do **not** need to worry about the details of the user interface of this app (we are not concerned about your graphic design skills). You can assume that your code has the ability to receive notifications when buttons are pressed, can push strings to the UI to be displayed, etc.

Each passenger feedback app has the ability to communicate with the bus control via BLE, with the MBTA server via the cell network, and with other passenger feedback apps via BLE (all described in Section 2.5.3).

2.4.1 Identifiers on Passenger Feedback App

Each phone has a fixed MAC address that can be accessed by the passenger feedback app. This address persists over the lifetime of the phone. It is possible for a phone to use a different identifier to identify itself on a network. For example, many apps that perform similar functionality use hash functions as part of their identifier scheme (see the Appendix), and in doing so, provide an identifier that changes over time. An identifier that changes over time can provide additional privacy to users at the expense of making data collection and analysis more complicated. In particular, if two phones A and B are communicating using two identifiers ID_A and ID_B, and then phone A switches to ID_C, phone B may interpret this as a new phone.

2.5 Networks

2.5.1 The MBTA Radio Network

Buses communicate with the MBTA warehouse via a trunked radio network (see Appendix 6), but for the purposes of this project, we will make some simplifying assumptions, as well as some upgrades, that allow you to ignore the complexities (and some drawbacks) of a trunked radio system while still reaping its benefits. Because this network is only used by MBTA buses, the structure of each transmission, as well as the addresses, are fixed.

Bus-to-server communication: Each bus is equipped with a radio transmitter and receiver; the warehouse also has a transmitter and receiver. When a bus needs to transmit to the server, it will do so on whatever frequency is assigned to it by the trunk controller; you do *not* need to specify any details about this request beyond that it happens. The MBTA's server will be listening on all relevant frequencies; it will receive all data that is sent to it via the radio network.

Of course, there are limitations. The MBTA's trunked radio network only has 10 frequencies allocated to it, and there are 1036 buses in the system. As more buses attempt to communicate at the same time, the delay to receive a vacant frequency will increase.

Server-to-bus communication: If the MBTA server needs to transmit data to a bus, things go a bit differently. The server has its own dedicated frequency—it doesn't have to worry about requesting anything from the controller—and all bus receivers are tuned to that frequency (that frequency effectively provides broadcast from the server to the buses). This means that buses can transmit data to and receive data from the server at the same time. Similarly, the server can receive and transmit data simultaneously. To determine whether a piece of data from the server is meant for it or for another bus, the bus receiver checks the destination address in the packet header (see below).

On each frequency, the network is capable of transmitting 16 Mbit/sec.[5] The average latency between the bus and the server is 10 ms, but again, as more buses try to communicate at once, that delay will increase.

For a network to work, each entity needs an address. In this network, each address is 48 bits; we'll use "radio address" to specify that this is their address on the radio network. The server's radio address is fixed as `0x00...0` (all zeroes). Buses are assigned a radio address at the beginning of each day and remain fixed for the day. By default, a bus's radio address is the same as its ID.

In addition to sending data to a particular bus, the server can broadcast data to all buses simultaneously by sending data to the address `0xFF...F` (all ones).

All transmissions in this network have the same format:

```
|src addr | dst addr | data |
```

Where:

- `src addr` is the 48-bit source address (either the bus's radio address or the server's radio address).
- `dst addr` is the 48-bit destination address (which could be a bus's radio address, the server's radio address, or the broadcast address). The bus's radio receiver uses this address to determine whether communication from the server is meant for it or for another bus (the server uses this address to determine which bus is communicating with it).
- `data` contains the actual data of the communication.

Using the MBTA Radio Network for Voice Communication

It is possible to use the network to transmit voice communications from a bus operator to a system administrator at the MBTA. Any audio data is transmitted in a standard format, which does not take up a large portion of network bandwidth. Because of how voice communication works, the two relevant frequencies (from the operator to a sys admin in the warehouse, and back) are occupied for the duration of the conversation.

2.5.2 The MBTA Warehouse Network

Whenever they are in the warehouse, buses can communicate with the server(s) via an in-warehouse wireless network. This is a faster network than the radio network, capable of 54 Mbit/sec.

2.5.3 BLE Networks

The passenger feedback app has the ability to communicate with the bus control via Bluetooth Low Energy (BLE), and with passenger feedback apps on other phones via BLE.

Each phone, as well as the bus control emits a BLE signal every 250msec. Although BLE signals can reach up to 50 or 100 meters (depending on the specific form) in freespace, the signals are significantly reduced when going through barriers such as walls (including the walls of a bus).

[5]This is the primary networking upgrade that we have made for you. In reality, radio networks are *much* slower. We've given you a network with speeds closer to 802.11 wireless in part because we want to see what cool things you do with it, and also because it's likely that in the near future such network speeds would be available for these purposes.

By default, each BLE broadcast is an extremely simple messages, primarily consisting of an ID. They can carry additional data, up to about 250B of data per packet (including the ID).

Each device within reach of such a broadcast has the ability to record a timestamp, signal strength, and the received payload for each such broadcast. In general, we estimate that to include all of the information needed to log one of these signals, each record will be 70 bytes long (or more, if the payload is larger than just the ID). This log includes the ID, timestamp, and signal strength, as well as some other ancillary information not needed in this particular analysis.

Because BLE signals do not reach very far and because the signal strength provides an estimate of distance away, these broadcasts provide the basis for estimating how far apart two phones are from one another, and perhaps from the bus control.

BLE does provide a means for direct, non-broadcast connections between the bus control and the passenger feedback apps. However, such connections add a significant amount of complexity to the system. If your system requires non-broadcast BLE, you will need a very strong justification for it.

2.5.4 The Internet

The passenger feedback app also has the ability to communicate with the MBTA server over the wider Internet, via their cell network. You can assume that the passenger feedback app knows the server's IP address. The server would not, by default, know every client's IP address.

Because the phones can communicate with the bus control via BLE, this means that phones can technically communicate with the MBTA server in two ways: directly, via the cell network, or indirectly, via the bus control (which could forward that data to the MBTA server via the radio network).

2.6 People

The MBTA has 2000 **bus operators** and 10 **system administrators** who are responsible for managing the warehouse operations and data. The system will be used by the MBTA's 446,700 daily bus passengers..

3 Requirements for Your System

The existing MBTA infrastructure already imposes some requirements on your system: you cannot exceed the storage or processing power of any component, nor the maximum speed of the networks involved. Moreover, your system must work at scale. Assume that, in the worst cases, all 1036 buses will be running at once, all routes will covered, and the routes will span the entire Greater Boston area.

We realize that we are identifying a set of goals for this system, some of which may be contradictory, so part of your challenge will be to find a defensible compromise among them. Overall, the goals span four areas:

1. Functionality: accuracy and timeliness in both data collection, response to failures, and alerts to changes
2. Ease of use (for passengers, operators, etc.)

Standard	Target
Reliability	75% of the time, buses should meet their frequency-of-service demands (arriving at their origin, midpoint, or destination timepoints within three minutes of the scheduled arrival time)
Coverage	At least 75% of the Boston metro population should live within .5 miles of at least one bus stop.
	At least 85% of low-income households should be within .5 miles of at least one bus stop.
Comfort	96% of the time, the maximum passenger-to-seat ratio should be ≤ 1.4.

Table 1: MBTA Service Targets

3. Low-impact on phones
4. Privacy

3.1 Fleet Monitoring

The MBTA wants to monitor their system for availability, reliability, comfort, and quality. They will do this monitoring using the data available on their server, most of which your system provides. They will need to answer questions such as the following:

- **Frequency of service:** For the duration of the span of service, are buses departing on each route at least once every twenty minutes on average?
- **Coverage:** How many citizens in the metro area live within .5 miles of a bus stop? The MBTA plans its route coverage to meet their target (see Table 1) in the presence of no failures.
- **Reliability:** How many buses are arriving at their origin, midpoint, and destination timepoints within three minutes of their scheduled arrival time?
- **Comfort:** What is the maximum passenger-to-seat ratio for each ride?
- **Total Load:** How many total people are using the buses? How many people are using the bus within particular geographic areas?
- **Value to network:** How many passengers who use a bus transfer from one bus to another? The number of transferring passengers gives the MBTA a sense of how valuable its network is.

3.2 Target Estimation

The MBTA currently operates within the service targets described in Table 1. With the fleet-monitoring data that your system provides, the MBTA is able to assess whether it's meeting these targets at any given point in time. These targets are part of how the MBTA decides whether part of its system has failed (see Section 3.3).

The MBTA would like to use data from the passenger feedback app to assess whether it should update its targets. Your system should enable system administrators to compare real-time data with customer feedback, to answer questions such as the following:

- When a passenger reports that a bus is late, how late was the bus?

- When a passenger reports that a bus is too crowded, how crowded was the bus?

Doing this will require a way to infer which bus the passenger is giving feedback on. Note that some passengers may give feedback when they are not actually on a bus (they may be, e.g., waiting for a bus that has not arrived yet).

3.3 Failures

Beyond basic fleet monitoring, the MBTA will also use your system to detect and recover from failures. They are concerned with three types of failures:

1. **High demand.** In a high-demand scenario, there is more demand for at least one bus route than the route provides (indicated by a passenger-to-seat ratio higher than the MBTA's target). This scenario can occur when there are more people using a route than usual or when the route is not meeting its frequency target, perhaps because a bus breaks down. When there is high-demand on a route, the MBTA must choose whether to add more buses to that route to compensate.
2. **Route unavailability.** A portion of a route can become unavailable for unexpected reasons: fire or police activity, construction, etc. When this happens, the MBTA must decide how to re-route the buses *and* how to alert passengers to the route change.
3. **Unexpected routes.** In some scenarios, the MBTA may have to add routes to its bus service. For instance, when a portion of the subway system isn't running, the MBTA will add routes that parallel the subway routes. You can assume that the MBTA knows what those routes should be; the problem here lies in making sure the MBTA can meet the demand that an influx of subway passengers will bring.

3.3.1 Failure Detection and Recovery

Your system is responsible for detecting the first type of failure (a high-demand scenario). Your system is **not** responsible for detecting whether the second (route unavailability) or third (unexpected routes) types of failure occur. In these cases, an MBTA system administrator will be alerted (via some other means, for instance a police scanner).

Depending on the particular scenario, the MBTA may take any of the following actions:

- Re-route one or more buses. That is, calculate a new path from a bus's current location to the endpoint of its original route, and send the bus on this new route. This response is most useful in the case where a portion of a route has become unavailable; note that this is *not* the same as moving a bus from one route to another.
- Put a currently-idle bus into operation, sending it from the MBTA warehouse out to a route. The idle bus can start its service at any stop on the route (i.e., not just the beginning of the route); once it has started, it will traverse the entire route as it would normally. Keep in mind that it will take a bus some time to travel from the MBTA's warehouse to the route.
- Pull a bus off of one route and add it to another. Buses can only be pulled off a route when they reach the route's origin, destination, or midpoint. As in the previous scenario, it will take a bus some time to travel from one route to another.
- Do nothing.

Your system is **not** responsible for deciding what action to take. However, it must enable any combination of the above actions, and consider how to alert affected passengers and bus operators about changes. Doing so may involve specifying what data needs to be updated, what messages need to be sent and to where, etc.

3.4 Passenger Experience

3.4.1 Performance on Phones

It is important to passengers that the passenger feedback app not significantly degrade the operation of other apps on the phone. This can take the form of memory, storage, computation, communication, and general battery drain. Note that communicating via the cell network generally drains battery faster than using BLE (though the cell network has benefits over the BLE network as well).

Because of this, your system should move the burden of performance, power and resource utilization to the central server whenever justifiable.

Most passengers also will not want to have to type detailed input into the passenger feedback app, and would prefer to simply push buttons to make a report.

3.4.2 Privacy

MBTA passengers care about the comfort and reliability of the MBTA's service, but also about their own privacy. A system such as the one you're designing has the potential to track passengers very closely, especially if passengers are giving unique, persistent identifiers within the system.

Since you are designing the passenger feedback app, you control a great deal about how this identifier works: where and when it's sent, whether it changes (and if so, how), etc. Your design should take passenger concerns for privacy into account. Will passengers be comfortable allowing the passenger feedback app, and potentially the MBTA, to access certain information? Is there information that passengers can learn about *other* passengers? What are the consequences of that?

4 Design Trade-offs, Use Cases, and Other Considerations

4.1 Design Trade-offs

While designing your system, you will be faced with a number of design trade-offs. We've summarized a few below:

- **Data accuracy:** In general, the more data you collect, the more accurately you can estimate various quantities. How will you trade-off the accuracy of your data with the performance of your system (for instance, the overhead of storing and transmitting data)?
- **Alert relevance:** The MBTA doesn't want to spam passengers with irrelevant alerts. In the ideal case, it would send alerts about service disruptions to *exactly* the set of passengers who are affected by such disruptions. How will your system attempt to determine this set of passengers, especially knowing that some of them might be waiting for a bus, and not on a bus?

- **Privacy vs. Improvements:** In general, the more data you collect about passengers, the less privacy they have. There may also be data that the passengers are comfortable sharing with the MBTA, but not with other passengers. On the other hand, there is a need for the MBTA to understand how its infrastructure is performing, so that it can make improvements. The collection of data over time can also enable the MBTA to spot historical trends to make larger changes (adding new routes, etc.).

4.2 Use Cases

As part of your proposal, you should describe how your system would work under each of the following use cases. This is not an optional part of the design project; if your system cannot handle one of these cases, you must explain why.

- **Normal operation:** The pool of idle buses is large, all routes are running smoothly, and there are no unexpected failure conditions.
- **High demand:** There is a Red Sox game at Fenway park, and the D branch of the Green Line is not running due to a track problem. Because of the length of that branch and the passenger demand, the pool of idle buses alone will be insufficient for handling this demand.[6] The MBTA may deploy all idle buses as well as re-assign routes. Multiple groups of passengers are affected: those waiting on routes that will receive idle buses and those waiting on routes that will have buses re-routed.
- **Construction:** There is unexpected construction that affects multiple bus routes. Passengers on these bus routes, as well as passengers waiting for these bus routes, are affected.
- **Historical Data:** The MBTA wants to investigate a complaint from a passenger. The passenger reported that their bus was over thirty minutes late one day last week.

Also consider how your system would scale beyond the MBTA. Suppose that a larger city wants to adopt your system. Will your system be able to adapt to such growth?

5 Evaluation

We will think about evaluation in the later stages of the design process, but it is helpful to start thinking about it early in the process as well. Here is a set of questions you might ask yourself:

1. How long should various kinds of data be archived where?
2. How much storage capacity will you need for that much data?
3. What are the factors in determining how much storage is needed?
4. How much data needs to be transferred under what conditions?
5. How quickly will the system respond? How quickly will a passenger be alerted to a change in service? Why would this matter?
6. Are there situations in which the system will be overloaded? In what ways and under what conditions?

[6] The insufficiency of the idle pool of buses is a real problem for the MBTA.

7. There are things that may change with time; for example, the MBTA may request additional features to the passenger feedback app, Does your design allow for changes in requirements?

6 Appendix - Trunked Radio Networks

Radio networks are wireless networks. In any wireless network, two users cannot transmit on the same frequency at the same time; when this happens, their transmissions "collide" and cannot be decoded.

Many wireless networks get around this problem by assigning different users different frequencies. When two users transmit on two different frequencies, their transmissions do not collide, and can both be decoded at the receiver in a process known as frequency division multiplexing.[7] The number of simultaneous transmissions is limited by the number of frequencies available to the system, among other things.

A downside of this model is that, since each frequency is assigned to a different user from the start, a user's frequency goes to waste when they aren't transmitting; other users don't have a means to "take over" that frequency for a period of time.

Trunked radio systems get around this limitation by not assigning frequencies up front. In a very simple trunked radio system, when a user wants to transmit, it requests a frequency from a centralized controller (there is a special frequency for communicating with the controller: the control frequency, or "control channel"). The controller finds a vacant frequency and gives it to the user to transmit on until its communication is complete.

The benefit of trunked radio systems is that frequencies are not as easily wasted. However, these systems can be quite complex.

7 Appendix - Hash Functions

A *hash function* maps an input string x, which can be any length, to an output string y, which is a fixed length (let's say 256 bytes). We will often say that y is the "hash" of x. This mapping has a few properties:

1. It's very hard to reverse; given y, it's virtually impossible to determine x
2. It's virtually "collision-free"; given two different strings, x_1 and x_2, it's *extremely* unlikely that the hash of x_1 will be equal to the hash of x_2. In fact, for our purposes, you can assume that such a collision will never happen.

The way this mapping is computed is out of scope for now, but suffice to say that hash functions are very carefully designed; only a few functions with these properties exist.

Sometimes hashes are used in conjunction with random (or pseudorandom) numbers to generate multiple hashes of the same string, for example a sequence of hashes. In this case given carefully selected number r, we can compute y_1, the hash of x concatenated with r. We can then repeat that and take the hash of y_1 and r to generate y_2. And so on. Often x is called the seed and r is called the key. As long as one can keep track of the sequence of random numbers, this can be a useful way to continually generate some sort of identifier from a string x.

You may not find hash functions useful at all for your design project; you should feel no pressure to use them. But since some apps use them as a means of creating IDs, we wanted you to have

[7] Technically, one also needs the frequencies to be "far enough apart" for this to work.

a quick primer. We'll talk about hash functions more in-depth in the last part of 6.033, because they're used for many things.