MASSACHVSETTS INSTITVTE OF TECHNOLOGY

Improving MBTA Bus Reliability with Real-time Tracking

Stephen White, Jon Beaulieu, Rishad Rahman

{stwhite, jbeau16, rrrahman}@mit.edu

May 8, 2017

6.033 - Computer Systems Engineering
Prof. Katrina LaCurts

*

Section: R04
RI: Prof. Martin Rinard

*

WRAP Instructors:
Jessie Stickgold-Sarah / Juergen Schoenstein
Friday, 1PM

1 Introduction.

The MBTA currently provides public transportation to the Greater Boston Area with a wide variety of vehicles including busses, trains, cars, and even boats. Coordination of such a fleet requires a dependable and flexible system. This paper focuses specifically on improvements to be made to the MBTA bus system's real-time tracking system. The primary goals of the MBTA's bus service include availability and accessibility, but the aims of this system are are designed to improve reliability and passenger comfort. Standards of twenty-first century travel have dictated needs for the latest technology in order to maintain the expected quality of service. Real-time systems using automatic payment and passenger counting, GPS tracking, and constant wireless communication all must be coordinated to deliver the best rider experience.

The following explains how a centralized system using servers, fault recovery algorithms, and real-time tracking could be combined with the MBTA's existing infrastructure to provide passengers with higher availability, less overcrowding, and more accurate stop information. The existing system fails to utilize location data to automate monitoring of the fleet's over 1000 operating busses.

2 System Infrastructure.

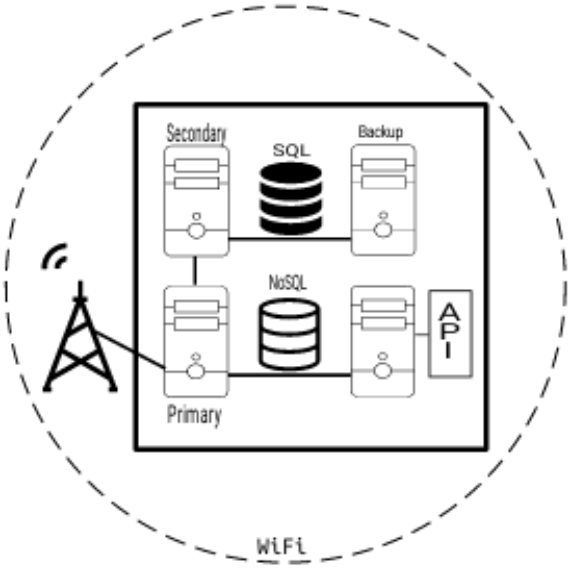
Today, the MBTA bus system services 177 daily routes, ranging in length from 4 to 38 miles, to provide public transit to 446,700 daily passengers. The authority aims to service every stop along every route at least once every 20 minutes, though often traffic, weather, and mechanical issues prevent the MBTA from meeting this goal. Improvements to the system would leverage the precision of the real-time tracking, enabled by a constant flow of data about the busses, to decrease the impact of these failures. The system discussed in this paper is constrained to use the existing infrastructure of the MBTA. Below, the components of the existing infrastructure are detailed.

2.1 Bus Fleet

The MBTA employees 2000 bus operators to drive their fleet of 1036 busses. Each of the busses is equipped with an on-board computer connected to a radio transceiver. This computer system, known as the *bus control* is connected to a number of components: an infrared beam passenger counter on the front door, security cameras, and a digital payment interface. All of these components collect data on ridership, but only payment information is stored on the bus control's 124MB hard drive. These records are stored in a .csv file until they are transferred to the servers.

Additionally, the bus control can access an onboard GPS navigation system to obtain current coordinates, send updates to the bus-operator-interface about route information, and transmit/receive data to/from the MBTA warehouse. In case of emergencies or unexpected scenarios, the bus operator also has access to a voice transceiver to communicate directly to the MBTA warehouse using the bus control's radio transmitter.

2.2 MBTA Warehouse



(**Figure 1**) Each of the 4 servers are explained below. Notice that only the Primary server has direct access to the radio transceiver.

The control center of the MBTA's system resides in their centralized warehouse, which also serves as the bus garage. The warehouse is equipped with a wireless network, which can communicate to the internet, the MBTA servers, and the buses (when in the garage) at 54MB/s. Four servers (connected via ethernet) comprise the datacenter, each with 10TB of storage and 16GB of memory.

- **Primary:** The primary server is connected to the system-wide radio network, and handles all communication to the MBTA's roughly 1000 busses. This server is assumed to be reliable. A NoSQL Real-time database stores received information about bus location and ridership. Any updates which include images from the bus's security cameras are decrypted to temp files and analyzed using machine-vision algorithms to obtain more accurate estimates on ridership. If the new estimates are still above a threshold ratio of seats to riders, the failure detection system is notified.
- **Secondary**: The secondary server maintains a PostgreSQL §6.1.1 as well as algorithms for reading, writing, and analyzing this data. The secondary also has an algorithm listening for update events on the NoSQL database. The algorithm

decides which updates need to be added to calculations in the historical tables of the SQL DB.

- **Backup:** The backup is configured to act in place of the Secondary server in case of failure. All needed programs and data are replicated onto the Backup, and has access to the NoSQL database.
- **API-Server:** The API-server is connected to the primary server, and has the NoSQL database replicated on it. It runs an open source server-side framework (such as NodeJS) and mediates requests from clients to the database.

At the end of the day, performance data is stored in tables in the SQL DB and the rest of the NoSQL database is reset. Statistics on availability, faults, and ridership are stored in two tables, one for data about stops and one for data about buses. The performance metrics are processed before they are stored. A region of 8 TB of the Secondary server (and therefore the backup) is dedicated to storing this data, as described in §6.1.1. Once the region is full, data is overwritten in least-recently-used order. The remaining 2TB provide ample space for storing any programs or data used in operation.

To increase the durability of the system, we also recommend a cloud-based data backup solution. Periodic backups of all data to the cloud are necessary to remove the risk of data loss after a physical fault to the datacenter such as a fire.

3 System Design.

3.1 Design Goals

3.1.1 Reliability

As stated previously, one of the primary goals of the bus system is reliability of service. In the simplest sense, this means ensuring that busses keep to their published schedules, and do not deviate from their routes unless necessary to maintain steady service. The MBTA has already established one metric of reliability in this regard: they aim to have a bus service each stop of every route at least once every 20 minutes. This is the ultimate goal of reliability under optimal working conditions; obviously we will not always be able to maintain this level of service precision. Rather, we strive to keep our service schedules within the reliability service targets also provided by the MBTA, which state that 75% of the time, buses should check in from their origin, midpoint, or destination locations within three minutes of the scheduled arrival times for those locations. If more than 25% of the routes are congested or otherwise slowed beyond this limit, a point of failure is encountered in the reliability of service, and the service

strategy must adapt accordingly. This will be detailed further in §3.2.4 as part of the **Failure Recovery** mechanism.

3.1.2 Passenger Comfort

Another factor we must address is the relative comfort of passengers. While not as immediately impactful as reliability or failure response, passenger comfort will, among other non-technical items not discussed in this paper, help to ensure continued business for the MBTA. With nearly 450,000 daily bus passengers, maintaining a comfortable standard is not easy. Within this report we define *"passenger comfort"* as the ratio of bus passengers to bus seats and the availability real-time information. We believe this is the simplest way for us to convert the qualitative idea of 'comfort' into quantifiable standards.

For the MBTA bus system, we are given a service target ratio of 1.4 passengers per seat on the bus, essentially capping the maximum number of people we can accommodate on any single bus should we choose to strictly adhere to this target. Further, the MBTA guidelines specify that this ratio should be maintained at least 96% of the time. Should multiple busses exceed this ratio, we can execute Failure Recovery protocols regarding bus overcrowding.

Real-time information has also become important for passenger comfort. With traffic conditions, keeping bus schedules to the minute is not possible. But, providing passengers with precise estimates of when the bus will arrive, and providing a map of where busses are in the routes allows for passengers to plan ahead, and wait less.

3.2 Modules

3.2.1 Communication Network

The trunked radio network is responsible for the reliable delivery of messages between the buses and the warehouse and thus is a core component of our system. Our real-time data collection and failure recovery mechanisms (§3.2.2-§3.2.5) are structured around the specifications of the network, specifically the message formatting. All messages sent between the radios adhere to a ~2.4 kilobyte format presented below:

| src_addr | dst_addr | t_type | msg |

- "Source Address" (src_addr -- 6 bytes): Indicates the origin of the message. Buses have unique addresses determined at the warehouse while the server always maps to the unique address of 0x00…0.
- "Destination Address" (dst_addr -- 6 bytes): Indicates the destination of the message. A bus can only set the destination to the server since it does not carry the bus address mapping. A message to be sent to all buses by the server has a unique destination address of 0xFF…F.

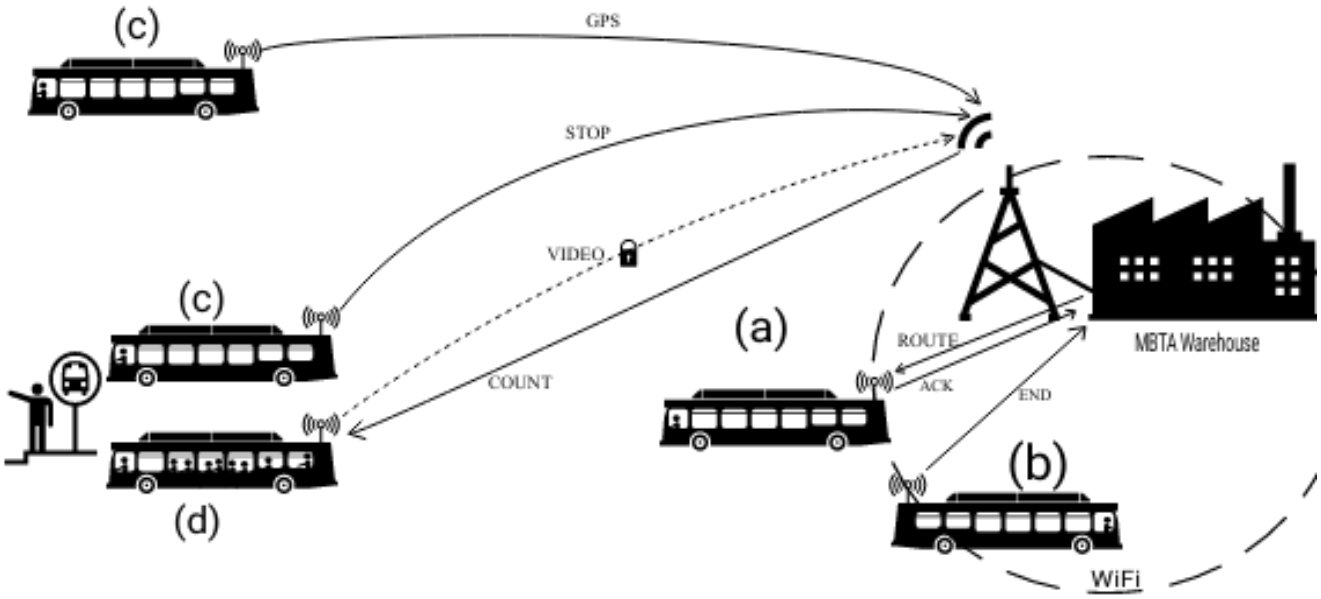
- “Transmission Type” (t_type -- 1 byte): Represents the objective of the transmission. Its values will be indicated by uppercase letters, in similar vein to C/C++ constant declaration standards. We mandate the last bit of t_type to be 0 if a response to the transmission is expected and 1 if not. Although our system doesn’t use 256 transmission types, the size allows more to be implemented if necessary.
- “Message” (msg -- 2.4 kilobytes): Represents information sent which is relevant to the current operation. This could be either real-time data such as GPS coordinates or instructions for failure recovery. The detailed specification depends on the protocol in use and will be described in §3.2.2.

Large messages which exceed the allocated byte size are split and sent in parts through a packet protocol examined in §3.2.2. A 240kB video frame would thus require (240 kB)/(2.4 kB) = 100 messages to send. We do not worry about the overhead from the redundancy of the first 12 bytes since they take relatively little space. The effectiveness of this format is evaluated in §4.3 for the protocols presented in §3.2.2 where we observe minimal issues in response times in the presence of failure-recovery transmissions and many requests on the network.

3.2.2 Messaging Protocols

The primary responsibility of the servers is to send updates to the buses based on information received throughout operating hours. When necessary, the servers can request information from the buses as well. As the operation advances, the buses are expected to report to the servers so that metrics can be updated accordingly. The server periodically analyzes these metrics in case a change needs to be made to the operation. Once an anomaly is detected, the server can contact the appropriate bus with requests or route updates through the radio network; the precise mechanism is presented in §3.2.4. In order to maintain our system guarantees of §3.1, each exchange in the network requires a well designed protocol.

We now describe all the transmission protocols present in our system, split by relevant data and/or operation status, and their implementations by means of the communication format presented in §3.2.1. The explicit message formats are given in §7.2 and are referenced when a message is described. Most classes of transmissions do not require in-depth instructions so only a few number of messages are sent. The other protocols were designed to reduce the frequency of large instructions for the purpose of relieving network congestion and to process these instructions quickly in case they contain information critical to the operation. Overall we observe that these protocols combined with the network enforces modularity and reliability within the system.



(**Figure 2**) Figure 2 shows four different situations in which wireless communication is established between a bus and server. Each situation, (a-d), is explained below. Notice that (a) is just leaving the garage, and (b) is just returning. Also, there is a packetized and encrypted transmission signified by the dotted arrow from (d) which reported a high occupancy estimate.

(a) Start-of-Operation

Before initiating MBTA service for the day, we verify that all the buses are fed the correct route information otherwise our system has already failed. The server will iterate through the bus ID's scheduled for service and send a ROUTE transmission (§7.3.a) to each bus after querying the corresponding route data which consists of stop ID's (8 byte), names, and GPS locations (8 byte) of all stops along the route. We can reasonably assume a stop name consists of no more than 25 characters selected from the alphabet [A-Z] | [0-9] | [" "] | ["/"] | ["-"]. Therefore each stop requires $8+8+(25/8)\log_2 39 <$ 33 bytes to represent so 2.4 kB message is ample space for route information. Alternatively the warehouse WiFi can be used instead; only when a bus needs to change routes while its deployed does the network radio need to be used. Once route data is received, the buses will send an acknowledgement (§7.3.b) to the server. As the server receives acknowledgements, it will update the daily status of the bus on the server to active. If after a wait time of 0.1 sec an acknowledgement isn't received, an administrator is notified. This protocol allows us to identify defective buses before deployment, reducing the probability of disaster mid-operation.

(b) End-of-Operation

The bus control stores fare information collected by the payment interface throughout the route's service, along with transfers from another MBTA service. The

server can send requests for this data, as well as any other data that the bus controller has access to. It is expected that the servers collect payment information daily.

When a bus arrives back at the warehouse to end its service, it sends an END transmission (§7.3.c) containing the day's payment and transfer information to the server. We decided this information should not be processed during operating hours since service reliability takes priority over faults in the payment system and the MBTA network. The message could add congestion to the radio network so the warehouse WiFi is used instead.

(c) Real-Time Location

Every MBTA bus is assumed to have the equipment described in §2.1. The bus control's main function is to send real-time information about the bus's progress on its route back to the MBTA warehouse. The bus control runs a navigation algorithm which obtains the bus's GPS location every 5 seconds. The bus control sends location updates back to the warehouse every 60 seconds which contain only the bus's ID number and its current 8-byte GPS data (§7.3.d).

The bus control also knows the GPS coordinates of all of the stops in the current bus route, and checks if the bus is within 50 meters of its next stop. Since the location is only checked every 5 seconds, this margin allows for tolerance. As soon as it is determined that the bus has arrived at a stop, a location update in the form of stop ID, arrival time, and passenger count is sent (§7.3.e), regardless of the 60 second interval.

(d) Passenger Count Estimates

As soon as the bus leaves the stop, the controller sends information to the warehouse containing an upper bound on the current passenger count deduced from the fare record on the payment interface. If the count surpasses 90% of the bus's total seat count, one frame from each of the bus's two cameras (480Kb total) are sent to the warehouse after being packetized and encrypted in 2.4 kB segments. Two messages signal the beginning and end of the packet transmission (§7.3.f-§7.3.g) and the encrypted frames are sent in between them (§7.3.h). New estimates based on the provided images are returned to the bus (§7.3.i), and overwrite previous estimates. Since passenger count estimates from fare information do not account for exiting passengers, the video data allows the bus control to obtain an estimate on the true count which is crucial for maintaining our standard for comfort. Each frame would require 10 messages to transmit, greatly increasing the amount of requests on the network and thus latency. We chose a 90% threshold for sending video data to keep bandwidth at a minimum as often as possible.

Data send regarding bus location and ridership are not encrypted, as this data is not exploitable and provides no private information. Though if an adversary intercepts video frame and location data, he or she can deduce the location of an MBTA passenger so we encrypt the frame to protect the security of the riders. A reliable service should be trusted by its customers so this level of protection is absolutely necessary. Each bus has a

1.2 kb public RSA key which efficiently encrypt video frame packet in sixteenths. Thus the decryption runtime through the 1.2 kb private RSA key on the server side can be bounded by (200)(16)Decrypt(1.2 kb) = (3200)(40 ms) ~ 2 min with a weak 2 GHz single-core processor.[1] This is not much of a concern since it is reasonable to assume buses take at least two minutes to travel between stops, and a modern processor will be used in implementation.

3.2.3 Fleet Monitoring

In order to evaluate the effectiveness of our system, we require the warehouse data to be updated at reasonable times but also need to utilize said data properly. At the end of operation, algorithms on the servers will be run to analyze the data from the most recent operation. With this, we present the following methods to evaluate our system according to MBTA standards. Detailed analysis is provided for metrics related to our design in §4.

- **Frequency of service:** In our protocols, we have timestamps recorded for each bus stop arrival. Therefore the interarrival times can be calculated at each stop and can be analyzed through a variety of graphical methods. To evaluate whether the stop is serviced at least every 20-min over the day, we plot the the distribution of interarrival times and expect to see a right-tailed distribution falling off rapidly after 20-min. To test factors influencing the service frequency, we can also plot frequency versus hour operation and frequency versus estimated passenger count. We expect to meet this standard nearly always under normal hours since no major changes to bus operation is made but more importantly we can observe whether our failure recovery methods are working properly.
- **Coverage/Total Load:** The number of people who do not live within ½ mile away from a bus stop can be easily calculated with census data and route data. This is not updated by our system during service hours and so is left to MBTA data analysts to work with. Our system can assist further analysis of coverage by providing ridership data on specific areas such as low-income neighborhoods. We do not mention such methods here since we prioritize buses reliability servicing the already chosen routes, not the optimality of the routes but it is convenient to know that this can be done if desired.
- **Reliability:** We can perform a similar analysis to frequency of service for each route by focusing on the origin, midpoint, and final stops. This is done for a variety of use cases to evaluate our failure recovery's effectiveness and versatility e.g. delays caused by Red Sox games or closing of roads. This is further specified in §3.2.4.
- **Comfort:** Evaluated with occupancy data provided in §3.2.2, further analyzed in §4.6.2. The data is only guaranteed accurate after 90% seat occupancy which is fine since actionables are taken near 140%.

[1] http://www.javamex.com/tutorials/cryptography/rsa_key_length.shtml

- **Network Value:** Each bus uploads transfer data at the end of the day so we can get a brief idea of how reliable the bus network is and where by plotting locations of transfers. Similarly to coverage, this is not a priority of the system but our data allows for proposed route changes if the MBTA desired to improve its network.

3.2.4 Failure Recovery

As with any design of this scale, there are bound to be events that put strain on the system beyond that of normal operating conditions. To prepare our system for these types of scenarios, we utilize the fleet monitoring tools described in (3.2.3) above as our primary methods of failure detection. These failures can include anything from mechanical problems with buses, to missing service targets such as the passenger comfort metric outlined earlier in this report.

For simplicity, we break down this section into different forms of failure that our proposed system could encounter, as each type of failure elicits a different response type.

Bus experiences mechanical failure on route, and can't drive: A second bus is dispatched from the MBTA's reserve, it first locates and picks up the passengers from the broken-down bus, then continues on-route where the previous bus stopped. Service will then continue per the normal schedule, and MBTA administrators will assist in the recovery of the stranded bus.

Reliability service target not met, due to heavy traffic, road closings, etc.: As noted previously, the MBTA service target for reliability is defined as buses arriving at their origin, midpoint, and destination locations with three minutes of their scheduled time, 75% of the time. We will consider this failure event to be triggered when fewer than 85% of buses meet their three-minute arrival windows. Two buses are simultaneously dispatched along any routes that are meeting their three minute schedule windows. The first bus begins at the origin point and drives the normal route up until the midpoint, after which it moves directly to the final destination point. The second bus, meanwhile, drives directly from the origin to the route midpoint, and then conducts normal service from there to the final destination.

Of course, there may be different causes for reliability strain along bus routes. Some might entail heavy traffic jams spread along the bus's entire route, while others might be simpler, such as a single road being overcrowded due to an accident. In the smaller cases, it doesn't make sense to go through an entire failure recovery process when it would be easier to simply have the bus take a different path between stops. So we leave some flexibility in our system for MBTA bus drivers to use different paths from the established route at their discretion.

Comfort target not met, bus is overcrowded: MBTA service target for comfort is ratio of 1.4 passengers to seats, for 96% of the time. To combat overcrowding, this failure point will trigger for specific routes when their buses report a ratio higher than 1 passenger to seat.

When triggered, bus service to that route will be doubled for the next hour, meaning that we will dispatch a bus every 10 minutes rather than every 20. This will

continue for one hour, at which point we will re-evaluate the route's passenger-to-seat ratio for the most recently dispatched buses. If we still exceed our 1-to-1 ratio at this point, the doubled bus service will continue for another hour block, after which we will once again evaluate as described. If, after the one hour of expanded service, our passenger-to-seat ratio for that route has dropped below our threshold ratio, normal service will resume, with busses being dispatched every 20 minutes.

In addition to these strict quantitative rules, we will also allow for MBTA administrators to make discretionary decisions about bus dispatch rates for particular routes and times. This will allow them to plan for expected passenger surges, such as Friday evening rush hours, city events such as the Boston Marathon, or simply a Red Sox game at Fenway.

4 Evaluation.

4.1 Warehouse Data Center

The NoSQL real-time database on the Primary server has 2 main branches, bus and route. The bus branch has roughly 1000 children for the 1000 buses, and the route branch has under 500. Each node holds integers which are overwritten, and would never exceed a few kilobytes each. Therefore, if each node was generously capped at 1MB of data, the database would never exceed 1.5GB. This amount of space is negligible compared to the 10TB of storage available on the main server. All data that is archived is stored in SQL tables on the Secondary server, which has an 8TB partition configured to overwrite the least-recently written data if the partition is filled. Given that each day, 1 row is added to the historical tables per route and bus, there are roughly 1500 rows written per day. Assuming approximately 1 byte per character, we can estimate each row at around 1Kb. This means that around 1.5Mb is written per day, and that it would take thousands of years before data needs to be overwritten in the 8TB partition. Even if the initial estimate was off by orders of magnitude and we needed to write 1GB of data per day, it would still be able to hold 8000 days of data before needing to overwrite space.

Additionally, the use of a backup server to support the secondary server minimizes the downtime of the system incase of a failure. As the back-up server is configured to take the place of the secondary server, it has all needed programs and data available to it, so it can take over quickly. The purpose of the secondary server is to archive data from the NoSQL DB events and analyze incoming security camera frames. These tasks are queued, and a couple of seconds of downtime between switching to or from the backup server would not affect operations.

Overall, the data center provides a robust and fault tolerant system for communicating to the MBTA bus system, and making the data they send quickly available to the entire system and to riders.

4.2 Bus Control

The bus control stores route information acquired through the network at the beginning of its time in service. Compiled binary files of the different location monitoring album are stored on the control, as well as records from the bus's payment interface. Each record contains at most 128 bits. A partition of 16MB is allocated as buffer for messages if the network becomes congested. Assuming an upper bound of 50 stops in a route, 1kb of data per stop, 5 kB for the RSA key pair, and all of the program files use 20MB, there is over 80MB left on the bus control's 128MB hard drive. The remaining 8MB allows for the bus control to store over 4 million records from the payment interface, much more than needed to store a day's ridership records for a single bus.

4.3 Network

4.3.1 Single Packet Latency

Our selection for the message size was based on a loose worst-case analysis on the network latency. One message requires (2.4 kB/(2 MB / sec)) = 0.0012 sec to send. Thus if all the approximately 1000 buses were to synchronously attempt to send a message, only 10 are be accepted at a time leading to an acceptable latency of (1000/10)(0.0012 sec) = 0.12 sec for the last message. Aside from video data, all exchanges in the network involve a small number of messages so we have with high confidence that most data sent can be processed and responded to within a second. This includes location, initial passenger count estimates, and route information during failure recovery which suggests that our communication network remains reliable throughout service.

4.3.2 Packetized Video Frames

A worst-case analysis analogous to §4.3.1 shows the latency for the last packet would be (200)(0.12 sec) = 24 sec. This suggests a more precise analysis since if the data is not sent fast enough, future messages will queue up on the bus control buffer and take longer than the expected 0.12 sec bound. Suppose x_i buses request to send GPS before packet i. The delay from GPS requests for packet i it thus (x_i/10)(0.0012 sec) = (0.00012) x_i sec. The total time for packet i to transmit is thus $(x_1 + x_2 + \ldots + x_i + i)(0.00012)$ sec. Note that the buses which transmit GPS before the first packet will only transmit again before packet i^* once $(x_1 + x_2 + \ldots + x_{i^*-1} + i^* - 1)(0.00012) \geq 5 \Rightarrow x_1 + x_2 + x_3 + \ldots + x_{i^*-1} + i^* \geq 41668$. Clearly this cannot happen with 1000 buses since each bus can only lie in one x_i before

x_{i^*} so $x_1 + x_2 + x_3 + ... + x_{i^*-1} \leq 1000$ which shows that all 200 packets suffer at most (1000+200)(0.00012) = 0.144 sec delay accounting for GPS (or other single message protocols) contention. We now consider the contention from concurrent frame transmissions. Assuming 2 min between bus stops and uniformly random arrival times, we expect 1000 buses / 2 min = 0.45 buses / 0.144 sec. This suggests large amounts of contention during the 0.144 sec required for frame transmission is highly unlikely. Assuming three buses request to send video data within this interval, the maximum amount of delay we expect is (1000+600)(0.00012) = 0.192 sec. Thus we expect all data to be received within an acceptable amount of time.

Other than the network, decryption is also a possible bottleneck for analyzing video frames and thus important instructions during high demand. Our RSA scheme scheme seems sufficient since accurate counts are sent to the bus before the next stop (§3.2.2.d) and system responses take into effect within an acceptable amount of time as discussed in our analysis of passenger comfort (§4.6). If larger files needed to be sent through the network, a protocol of hybrid encryption, using AES alongside RSA should be used.

4.4 Data Accuracy

Under normal operation, we record real-time data with high accuracy. Specifically, we receive GPS data at least once a minute from each bus. This location data is assumed to accurate despite the urban setting. Passenger counts are also estimated every time a bus arrives at a stop. These estimates are *upper bounds* not actual estimates, since the payment interface only records when passenger enter and not exit the bus. The usage of machine-vision algorithms to get more accurate estimates increases the accuracy of our upper bound to around 97% according to the spec of the algorithm. Additionally, each bus records timestamps of when it arrives at stops, within a 50 meter margin. This margin is small enough to ensure that a bus is at or near a stop, but big enough to allow for the inherent noise of the GPS coordinates, or bus stopping at a slightly different location. These timestamps are recorded for each stop, so arrival times for beginning/midpoint/start should be highly accurate.

4.5 Failure Recovery

One of the largest aspects of our evaluation is whether or not our methods of failure response are effective under conditions that would strain the system. In this section, we present different scenarios that could cause our system to enter a failure mode along a bus route, and demonstrate our method of recovering from each of those circumstances.

4.5.1 Active Bus Failure

Perhaps the simplest form of failure is simply if a bus breaks down while on its route. In this case, we believe that the easiest solution, dispatching another bus as detailed in section 3.2.4, is the most effective. This will cause a relatively minor delay as the replacement bus takes time to pick up where the broken bus left off. However some amount of delay is unavoidable in this situation, and because our communications protocol would pick up on the broken-down bus immediately, we believe our system minimizes effects on passengers and scheduling.

4.6 MBTA Standards

4.6.1 Reliability

We quantify our reliability standards by using the provided MBTA metric: a bus route is considered successfully reliable when buses arrive at their origin, midpoint, and destination locations with three minutes of their scheduled time, 75% of the time. We believe that with proper planning and the insight provided by MBTA administrators, we will be able to maintain that 75% margin along our bus routes under normal operating conditions. Additionally, daily data collected by the busses and stored in the warehouse can be used to further determine usual traffic patterns and common sources of delays during normal operations. This will help the system administrators to adjust routes and timetables moving forward, further increasing our ability to provide consistently reliable service.

The challenge, of course, is maintaining that level of reliability while the system is under stress. By starting our failure recovery procedure when a particular route reaches only 85% reliability, we give our system time to respond to the drop in reliability before crossing the 75% threshold established by the MBTA. At the same time, the 85% threshold is low enough that we have a reduced risk of tripping the system on a route that is only temporarily slowed down.

Because of the structure of our communication system, we know that busses will be sending their passenger and schedule info to the control warehouse after every stop. Thus, we can expect the bus control to detect this 85% reliability threshold within seconds or minutes, limited by the time it takes to receive and decrypt messages relating to scheduled stops. By splitting the route in half immediately following detection of this 85% threshold, we accomplish two items to improve our reliability:

First, we effectively double the number of busses that would normally be present along any part of the route. Under a situation where there is bad traffic along the entirety of a normal route, this will theoretically double the rate at which busses will arrive at stops. In practice this will be smaller than a doubled rate, because of uneven traffic and a variety of other factors. However, even a low-end estimate of a 50% shorter

wait time between busses at a stop will be enough to once again increase our reliability rate.

Second, the route splitting helps to isolate any location-based delays along the bus route. In contrast to the previous point, this is exceptionally useful not when there is poor traffic along an entire route, but rather a single point of congestion. Splitting the route in this situation ensures that at least half of the bus route can function on schedule, even if the second half is facing delays that can't be fixed by the driver taking an alternate path to the next stop. While this second point does not by itself guarantee that we keep to our 75% reliability goal, it provides us with a significant buffer of on-schedule operation without requiring administrators to flood the route with every bus they have on reserve.

The largest concern with our reliability recovery mechanism is any passengers who board on the first half of the stressed route, and want to travel to a destination on the second half of the route. They will be forced to disembark a bus at the route midpoint, and wait for the second bus to pick them up. Because busses are normally dispatched along routes every 20 minutes, these passengers may be waiting up to 20 minutes for the second bus. While this is not an ideal situation, we feel it is an acceptable trade-off to maintain predictable bus service that functions on schedule for the majority of its runtime. We also note that this trade-off has a precedent in the current MBTA subway system. When significant delays or mechanical failures occur along a rail line, it is not uncommon for the MBTA substitute bus service for subway transportation, and requires passengers to disembark the subway at a specific location, to be picked up by bus and carried along the remainder of the usual route.

4.6.2 Comfort

From the MBTA service targets outlined previously, we know that our passenger comfort target can be quantified as maintaining a 1.4 passenger-to-seat ratio on board all busses at least 96% of the time. Because our system architecture expects each bus to update its passenger count with the data warehouse at each stop, one can expect the system to be alerted to our 1-to-1 passenger-to-seat threshold almost immediately. Upon detecting that this threshold has been crossed, the system will automatically shift into recovery mode for passenger comfort.

Once this shift occurs, the dispatch system will send a bus along the overpopulated route every 10 minutes rather than every 20. Thus, the maximum time between a route exceeding our 1-to-1 threshold and additional busses being dispatched is on the order of 10 minutes. For this recovery system to fail, then, a specific bus route would need to experience an unexpected passenger surge of 40% -- to increase the passenger-to-seat ratio from 1.0 to 1.4 -- within a timeframe of less than 30 minutes, which is approximately how long it would take the extra busses to propagate along the route.

As a final safety measure, we've also given MBTA administrators the ability to adjust route timing as needed for expected passenger surges due to rush hour or other

events in the greater Boston area. Thus, we restrict any passenger comfort failures to only occurring when there is an unexpected, unplanned passenger increase of 40% or more. We are confident that this will not occur on more than 4% of bus routes during operating hours, thus we are well within our 96% passenger comfort target outlined by the MBTA.

5 Conclusion.

Our analysis suggests that our proposed system delivers a well-designed solution that MBTA administrators and passengers will be able to rely on. We believe that the design aspects we chose to focus on, namely fault tolerance, reliability, and passenger comfort, are the best design principles to follow in this type of scenario. Fault tolerance is our primary concern, as addressed directly above. The ability to respond to random incidents is crucial to serving customers properly. This connects directly to the ideas of reliability and passenger comfort defined above - neither would be possible without a solid fault-tolerance scheme.

To a lesser degree, security also played a role in our design choices, regarding how we encode data and when it is transmitted from busses to the central servers. We were able to find an excellent compromise between security and transmission speed: we are certain that our data is not vulnerable in-transit, but is still able to be decrypted and processed by the warehouse quickly and without limiting responsiveness. Within the warehouse itself, data on our servers would be vulnerable to unauthorized access or attack by someone with physical access to the machines. However, because of the small number of administrators working in the warehouse and the fact that physical warehouse security was outside the project scope, we chose not to focus on that aspect of data security.

We decided not to heavily incorporate other core design principles such as scalability into the system. This project is constrained to a specific scope for the MBTA, and decisions involving expansion are outside of that purview. It's possible that the basics for our system could be extracted and used to create a much larger system, either for another city or for planning an expansion of the MBTA's service area. We expect that our computational needs will quickly scale upwards as the assigned service area and number of busses is increased. Raw storage capabilities would not be an immediate concern, but the transmission channel would very quickly fill up as the number of busses is increased. At a certain point the complexity of the expanded system would necessitate additional communications protocols, and new servers for processing and relaying data. Fortunately, we are confident that our system performs reliably at the assigned scale, and we are able to at least qualitatively analyze which parts would pose limitations if we were to expand it further.

Finally, we recognize that there are still factors left unaddressed by our proposal. For example, our current proposal does not incorporate any type of passenger feedback

mechanism. Likewise, we do not incorporate historical usage data into our system. We made these choices primarily in the name of focusing on a basic, reliable system. We believe that once the system we propose is running as planned, it would be relatively easy to make slight adaptations to include a feedback option or alter routes according to population data. But once again, in the name of relative simplicity, we chose to disregard those elements for our initial system proposal. We are confident that this design offers reliability and adaptability that is needed for any public transportation system, with the opportunity to expand and add new features once the system has been established and proven to succeed.

6 Appendix.

6.1 Database Schemas

6.1.1 SQL Databases

The Secondary server, as discussed in §2.2 holds nine tables, totalling about 1GB in total data. The two 'historical' tables have data from operations appended each day. A row is added for each bus and each stop, so around 1500 rows are written per day. Roughly estimating, this is 1.5MB per day. Given that the historical tables are allocated an 8TB partition on the secondary server's hard drive, there is 8,000,000MB available. This means over 5,000,000 days can be written to the database before data needs to be overwritten.

- census - [name, location, income, age] **(600MB)**
    - Name: resident's name
    - Location: approximate coordinates of the resident's residence
    - Income: approximate income bracket
    - Age: approximate age of the resident
- bus_meta_data - [bus_uid, operator_uid, route_id, META] **(10MB)**
    - Bus_uid: the id of the bus (key)
    - Operator_uid: id of the bus's operator, see operator table
    - route_id : id of the route the bus is assigned to
    - META: extra information about the bus. Make, model, etc
- route_schedule - [route_id, stops, timetables] **(100MB)**
    - Route_id - unique identifier for the route
    - stops - a list of stop ids (see stops table)
    - Timetables - timetables for scheduled stops on that route
- stops - [stop_uid, name, coords, description] **(100MB)**
    - stop_uid - the unique identifier for the stop
    - name - the english name of the stop (i.e. '77 Massachusetts Ave')
    - coords - GPS coordinates of the stop's location
    - description - text description of the stop
- alternate_stops - [stop_uid, name, coords, description] **(100MB)**
    - See stops table
- Fare - [bus_uid, total_fare, period]
    - Bus_uid - bus from which fare data came
    - Total_fare - fare collected in the period
    - Period - time period reflected in the total_fare
- operators - [operator_uid, in_service, current route, hist_routes] **(10MB)**
    - operator_uid - a unique identifier for the bus operator
    - in_service - true if the operator is currently driving
    - current_route - Name of the route currently operating on
    - Hist_routes - a list of routes the operator has driven historically over the last month
- historical_bus - [bus_uid, operator_uid, overcrowd, splits] **(dynamic)**
    - overcrowd - percentage of time operating with more than established maximum ratio of seats to riders
    - splits - tuples containing times for (first stop, midpoint, last stop)
- Historical_stop - [stop_uid, services, ridership_estimate] **(dynamic)**
    - services - tuple containing (bus_uid, time of arrival at stop)
    - Ridership_estimate - estimate of riders on the bus after leaving this stop

6.1.2 NoSQL Database

A NoSQL system was chosen for the Primary server's real time database. As this data needs to be high performance (only read and write), and doesn't typically need relational data to be queried, such as in SQL. Here, the nature of the data is more hierarchical. Below, we can see the hierarchy.

- Root (/):
    - Bus
        - {bus_uid}
            - Operator_uid (int)
            - Route_id (int)
            - Location (int)
            - Last_stop (int)
    - route
        - {route_id}
            - busses (array)
            - In_service (bool)
            - Active_stops (array)

6.2 Bus Control

Static variables:

- pub_rsa_key

Operation-dependent variables:

- route_info
- passenger_count
- payment_data
- transfer_data
- current_gps
- next_three_stops

Functions that are available to the bus control include:

- bus_is_near_stop(stops, current_loc)
- gps.get_coords()
- cameras.front.getFrame()
- cameras.back.getFrame()
- ir_counter.get_count()
- operator.get_current_route()
- operator.send_new_route(route)
- message.packetize(message)

- gpg.encrypt(message)

6.3 Message Formats

Below is a table of message formats used in the system.

Transmission	src_addr	dst_addr	t_type	msg
a) ROUTE	0x00..0	bus_id	ROUTE = 0x00	stop_1 stop_2 ...
b) ACK	bus_id	0x00..0	ACK = 0x01	NULL
c) END	bus_id	0x00..0	END = 0xFF	payment_transfer_data
d) GPS	bus_id	0x00..0	GPS = 0x03	gps_data
e) STOP	bus_id	0x00..0	STOP = 0x05	stop_id timestamp
f) BEGIN_PKT	bus_id	0x00..0	B_PKT = 0x0F	NULL
g) END_PKT	bus_id	0x00..0	E_PKT = 0xF0	NULL
h) VIDEO	bus_id	0x00..0	VID = 0x11	video_packet
i) COUNT	0x00..0	bus_id	COUNT = 0x10	passenger_count

6.4 Citations

All diagrams in this document were created by Stephen White. Some icons included in the figures were obtained under the Creative Commons License from https://thenounproject.com/. Specifically:

- Factory by Lil Squid from the Noun Project
- Bus Stop by Luis Prado from the Noun Project
- Lock by DewDrops from the Noun Project
- Database by Yo! Baba from the Noun Project