Massachusetts Real Time Transit Control (MassRTTC)

A Group of 3 MIT Students in 6.033

Massachusetts Real Time Transit Control (MassRTTC)

1 Introduction

The MBTA bus network serves almost half a million people daily and is hence key to the economic functioning of Greater Boston. To provide high quality service to its customers, the MBTA needs to provide frequent, reliable, and comfortable service. However, although bus routes are scheduled to meet even peak demands, unpredictable congestion and fluctuating demands can lead to deteriorated quality of service.

To meet these needs, MassRTTC is an integrated real-time control system designed to meet the following goals:

- **Frequent service.** If routes are not serviced once every 20 minutes per stop, riders can perceive the service as unreliable, discouraging them from using it. Thus, frequent service both meets the needs of citizens and sustains ridership.
- **Passenger comfort.** The passenger to seat ratio on buses should be under 1.4, with less than one passenger standing for every passenger seated.
- **Resilience.** The bus network should keep running smoothly even in case of unexpected failures such as road closures or subway breakdowns.

Buses, however, are in an unstable equilibrium because a late bus must pick up more passengers and thus become even later, which prompts the bus behind to pick up fewer passengers and run ahead of schedule. Eventually, the buses form a pair, doubling waiting times and leaving the front bus overcrowded. This effect is known as bunching and is documented as a source of unreliability by Strathman et al. (2003) and Xuan et al. (2016)[3][4]. Preventing this is key to providing reliable service.

MassRTTC includes automatic vehicle location and passenger counting systems. This is used by a central server to control and, if necessary, reroute buses. The server instructs operators to obey minimum and maximum dwell times to prevent bunching, informs them to new routing, and deploys reserve vehicles to meet unexpected demands. While automation will improve the efficiency of management, for certain decisions, the system defers authority to humans, who better understand the complexity of cities.

2 System Overview

MassRTTC aims to achieve coverage, passenger comfort, reliability, and resilience for the MBTA. Figure 1 shows how MassRTTC achieves this. We maintain high coverage by analyzing mobility patterns and adjusting bus service accordingly. To ensure passenger comfort, we gather feedback through a MBTA Twitter account and prevent crowding. Passenger counts for computing this ratio are obtained using buses beam sensors as well as computer vision algorithms running on security camera frames. For buses to be considered reliable, they should arrive at stops every 20 minutes and reach the start, midpoint, and end of their routes within 3 minutes of their schedule. We aim to reduce headway variability using bunching protocols so that buses can meet these reliability goals. Finally, we make sure that buses are resilient by detecting high demand and using bunching protocols to alleviate that demand.

To do this, MassRTTC is designed to do the following:

- **Detect problematic service.** MassRTTC continuously finds out where the MBTAs fleet is and how many people are on its busses. Buses collect data via GPS, beam sensors, and security cameras, and send them to a central server via a radio network.
- **Decide how to correct this.** MassRTTC aggregates this information and uses it to make decisions to improve service. This calculation takes place at a centralized group of computers collectively referred to as the server, which collects and stores data, computes policies, and communicates these back to buses via the radio network.

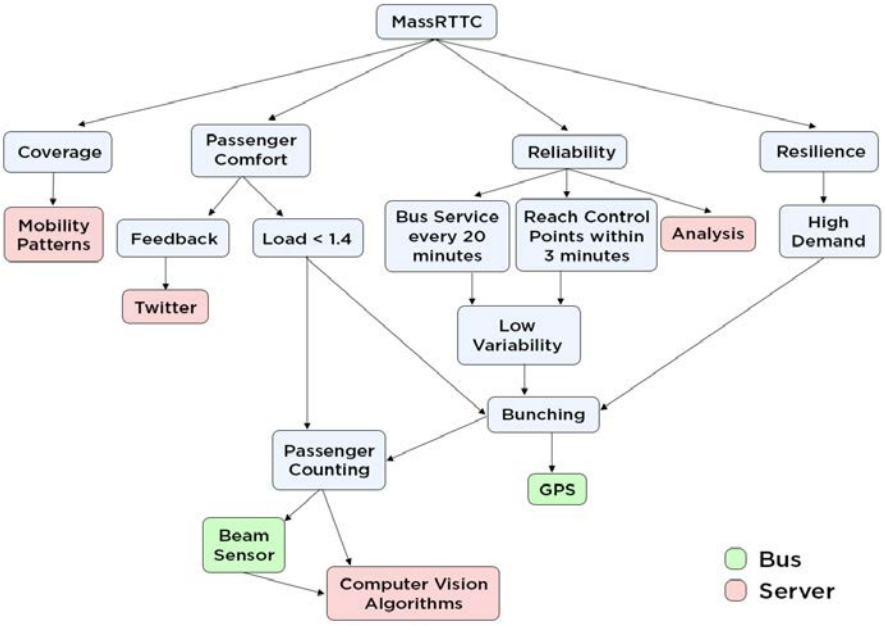

Figure 1: MassRTTC system overview

- **Adjust spacing between buses on a route.** MassRTTC is able to alleviate bunching by sending drivers instructions to stall, improving reliability.
- **Reroute buses.** In the event of emergency or subway breakdown, MassRTTC can redirect buses to use different routes.

We aim to reduce headway variability using the control strategy proposed by Xuan et al. (2011). This will involve, among other techniques, holding buses, that is, stalling them at stops for recommended amounts of time. Sanchez-Martinez et al. (2016) states that this is the most effective and widely used method of preventing bunching. [2]

2.1 Components

MassRTTC works with a number of components, whose interaction is described in Figure 2.

2.1.1 People

The MBTA employs 2000 bus operators who drive the bus fleet. Bus operators may communicate with system administrators using the radio network to check for route changes. They also aid in congestion protocols by telling passengers to not get on the bus and stalling at stops.

There are 10 system administrators in charge of the MBTA warehouse. System administrators have the final say on any route changes, and may make data queries to the master server at the warehouse.

Lastly, there are 446700 daily bus passengers in the MBTA system. We collect both payment and customer feedback from these passengers.

2.1.2 Buses

The MBTA has 1036 buses. Buses are equipped with automatic vehicle location, passenger counting and fare collection systems.

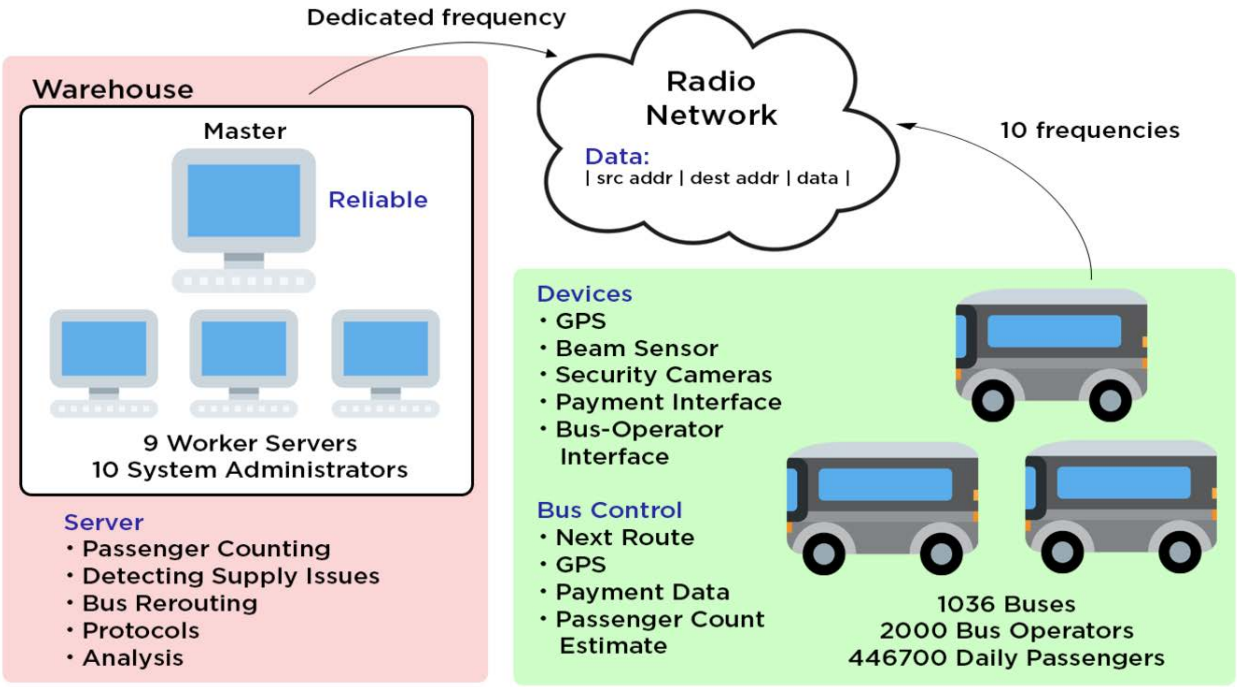

Figure 2: MassRTTC components and their interaction

First, each bus is equipped with a bus control, a small computer which stores the current bus route, as well as GPS data (collected every second) and all the passenger payment data for the day. In addition, there is a bus-operator interface, which displays the next three stops on the current route using route data from the bus control. Any time route data needs to be updated, the data should arrive at the bus control first. The bus control can then communicate directly with the bus-operator interface so that the next three stops can be updated. The bus-operator interface also has an indicator showing whether a bus has been flagged for bunching.

There are other several devices on each of the buses. A GPS sensor collects position data accurate within 5 meters every second, which is stored on the bus control. A beam sensor sends data to the bus control every time a passenger passes through the front door of the bus to 85-90% accuracy. It can distinguish whether a passenger was entering or exiting the bus as well. Finally, there are security cameras on each bus that can collect video feed of the bus.

Each bus comes with a payment interface which accepts payment in CharlieCard, tickets, and cash. All payment transactions are sent to the bus control. Payment data always includes at least a timestamp and a cost. If the passenger chooses to pay with a Charlie Card or ticket, then the payment data will also include the ID for the ticket or card and any transfer information.

2.1.3 Network

The buses, as well as the MBTA warehouse, each have a radio transmitter and receiver. Buses and the server can transmit and receive data simultaneously using a trunked radio network. During the day, buses can transmit data to the server or other buses using one of 10 frequencies, while the server can send data using its own dedicated frequency to either a single bus or all operating buses at once.

2.1.4 Server

The server has a master-worker architecture. The master is the machine that directly communicates with buses via the transmitter. It stores the working data, communicates with buses, and assigns tasks (described in section 3.3) to workers and analyzes the results. The analysis can include protocols to deploy, or data to store or send to buses. It also has an interface for system administrators to manually change system statuses, including deploying protocols for situations the server cannot detect, and querying the database.

The master stores the working data, which includes routes data and any data it gets from buses throughout the day (section 3.3.5).

Worker are susceptible to failing for up to two minutes at a time, but do not lose any data stored. To deal with this, the master assigns splits tasks up evenly and assign them at random to workers, so that in case of a machine failure, less data of each type are lost and the system can recover more easily. The master sends out heartbeat messages that workers respond to to indicate that the are alive. If a worker has not responded within a set timeout, the master assumes that worker has failed, and reassigns its tasks to another machine.

3 System Design

3.1 Bus Data Collection

Data are collected to allow the server to make informed decisions to maintain quality service.

3.1.1 Vehicle location

Buses collect GPS data each second. A bus decides that it has passed a stop when it passes in, and then out, of a 30m radius from a stop. The bus uses this information to display the next 3 stops on the bus interface. This slight overestimate prevents the display from changing before the bus has actually reached the stop.

3.1.2 Passenger counting

Buses rely on both beam sensor and security camera data to count passengers. The bus keeps a local count using its beam sensor, and intentionally overestimates the number of passengers. Each boarding passenger is counted as 1.25 and each alighting passenger is subtracted as 0.85. Furthermore, some passengers alight from the back, where they cannot be detected, so the local count is with very high probability higher than the actual count of passengers. If the passenger-seat ratio estimate exceeds 1.4, the bus sends 5 frames from its security camera to the servers for a more accurate count. Keeping an overestimate and only updating when the count is relevant reduces network and server load to upto 50% (section 4.2.2). Generally, the bus will receive the new count in under 2 seconds. (section 4.4.1).

3.1.3 Data transfer

Buses send security camera frames, GPS and payment data to the server. Each bus stores its GPS reading every 5 seconds and sends data batches to the server every minute. All data are sent with their timestamps. Data on the network are sent infrequently enough that buses generally can send in under 0.1 seconds (section 4.2.3).

Fare and transfer data are stored in the bus control and transferred to the server at night for security. Storing this data does not take up an appreciable amount of the buss available space (section 4.3.2). To prevent an adversary from intercepting data or sending malicious data over the network, packets are encrypted with AES before being sent. Each bus receives a new randomly generated key each day, and the server keeps a table of which buses have which keys each day (section 4.7).

Fare and transfer data are stored in the bus control and only transferred to the server at night for security.

3.2 Network

When a bus wants to transmit data, it will request a frequency from the trunk controller. The controller will then attempt to assign the bus to an available frequency (out of a possible 10). In the event that all 10 frequencies are in use, the bus will continuously request for a frequency again at a random time within an interval of one second until it is successfully assigned a frequency. These requests are random to ensure that requests will not accidentally synchronize with each other. A bus receives a frequency with an average latency of less than 0.1 second (section 4.2.3).

Bus operators may talk to a system administrator at the MBTA using VoIP. Once a bus operator is assigned to a frequency, it will occupy that frequency until the bus operator has finished speaking.

Data sent over the trunked radio network has the following format:
| source address (48 bit) | destination address (48 bit) | header (2 bit) | data |
The packet header will also contain a 2-bit code that specifies what type of data is being sent:

Table 1: Header codes

Code	Meaning
00	GPS
01	CV
10	Fare
11	VoIP

3.3 Server Data Processing

The server does heavier computation to maintain quality functioning of the bus network.

3.3.1 Passenger counting

Every time the server receives a request to count the passengers on a bus, it runs a computer vision algorithm on the video frames provided. It then discards the frames, stores the count, and sends the count to the requesting bus. A single request takes 0.5 seconds to process (4.4.1).

3.3.2 Detecting supply issues

Bunching is the chief cause of crowding and unreliability (section 1). MassRTTC distinguishes between deploys protocols to alleviate different high demand scenarios. For each route, the server checks for pairs of buses whose headway is less than 50% of what is expected and considers three cases:

- **A crowded bus is followed by an uncrowded bus** Separation is needed to balance the load and decrease wait times. The buses are flagged, Protocol 1 is initiated on the leader, and Protocol 2 is initiated on the follower (sections 3.4.1 and 3.4.2).
- **Neither bus is crowded.** Separation is required to meet frequency targets. The follower is flagged and Protocol 2 is initiated (section 3.4.2)
- **Both buses are crowded.** Neither strategy will help. This may indicate that the entire route is crowded. If a number of consecutive buses on the route are too crowded, Protocol 3 is initiated (section 3.4.3).

Buses are considered crowded if their passenger count ratio is greater than 1.4. Buses are unflagged once they are back on schedule. Because the system is an unstable equilibrium, unflagging buses earlier would be useless because the buses would just bunch again.

Since the server counts passengers only on a subset of buses (section 3.1.2), it lacks accurate information about buses with low load. Hence, if a bus has stale passenger count (older than 3 minutes), the server assumes that the bus is uncrowded. If a bus needs to be flagged but its data area stale, the server may send a request for camera frames. In order to deal with changing traffic data staleness, congestion detection is run every minute and usually completes under a second (section 4.4.2).

For all buses flagged under Protocol 1, the server needs to report to the driver of the leader bus how much time is expected until the next bus arrives. This approximation is based on the time it just took for the leader bus to move from where the follower bus is to where it is now, not counting dwell times. This approximation should take into account current traffic conditions and should hence be fairly accurate.

In addition, the server must also compute holding times for the follower buses under protocol 2. These are chosen using the algorithm proposed by Xuan et al. (2011), which are optimized to return buses to their scheduled headways efficiently.

3.3.3 Other Data Analyses

Since these data are not used for real-time control, the analyses are only run at night and take only about 10 minutes (section 4.4.3).

- **Reliability.** Based on location data, the server checks how often buses reach control points and other stops within 3 minutes of scheduled arrival. The server also computes the mean and variance of headways sampled at short intervals, and uses this to estimate average waiting times. Ideal 20 minute service would result in 10 minute waiting times, and if routes chronically fail these targets, adjustments in routing or scheduling may be necessary.
- **Chronic overcrowding.** If routes chronically have high passenger counts under non-bunching conditions, it may simply not be serviced frequently enough. The server can detect this situation and recommend routes for changes in scheduling.
- **Mobility patterns.** Gordon (2012) describes techniques to estimate passenger destinations and transfers even though passengers do not tap out.[1] Combining this with census data can help planners better understand mobility patterns, including popular transfers, the prevalence of transit-dependent passengers, and the coverage of the network. This can inform long-term adjustments to service.

3.3.4 Storage

The server stores both short-term working data for daily operations and long-term data for analysis.

3.3.5 Working data

The server keeps a table that stores buses in order by route along with location and passenger count to speed up algorithms run during the day (table 2). Data are arranged to make most accesses easy, and only Protocol 1 requires accessing long-term data.

3.3.6 Long-term storage

The data in long-term storage is stored as a log, since data is only added and never overwritten. Each datapoint is stored in the following format:
Datatype | Timestamp | Bus ID | Route ID | Location | Payload
2 | 32 | 48 | 64 | 32 | * = 306 bits = 38.25B
The datatype is a code determining what type of data it is (table 2). The payload section is for passenger count data (where it is the passenger count) and for payment data (where it is the amount paid and the Charlie Card ID, if used).

Each entry also stores a pointer to the entry before it. A metadata table is also stored, which points to the last data point stored and to the location where the last datapoint of the day is stored. This makes

Table 2: Short-term data storage structure

Route (64 bit)	0x0001	0x0001	0x000f	
Bus ID (48 bit)	0x647	0x64c	0x553	
Current Location (64 bit)				
Location timestamp (32 bit)				. . .
Current Passenger Count (32 bit)				
Count timestamp (32 bit)				
Bus AES Key (128 bit)				

it easier to scan the system for data for any particular day. The master stores this data in its long-term storage for the day and pushes it to a backup machine every night. With the total data stored amounting to 130GB/year, this data can be stored for decades.

3.4 Protocols

3.4.1 Protocol 1: Rushing

When a bus is too crowded and behind schedule, and the next bus is a tolerable wait behind, the server may opt to rush the bus. In this case, the bus displays NOT IN SERVICE on its electronic headsign and only allows passengers to alight. The server sends the rushing bus periodic updates of how far behind the next bus is. The bus control shows this information to the driver, who then can tell the passengers waiting at the stop how long they need to wait for the next bus.

Updating the time estimate every minute ensures that the driver always has a fresh estimate to tell the waiting passengers.

3.4.2 Protocol 2: Holding

When buses bunch, the server flags the follower bus, which informs the driver via the interface. The server computes an amount of time that the driver stalls at each stop after allowing passengers to alight and board and periodically sends the driver this amount of time.

3.4.3 Protocol 3: Allocating reserve buses

In situations with unusually high demand, or where a subway breakdown or special event demands shuttling, the system notifies reserve crew to deploy reserve buses from the warehouse, if available. The system prioritizes first scheduled routes, and then special routes with highest demand.

3.4.4 Protocol 4: Route changes

For planned route changes, such as for construction, new schedules are uploaded to the bus at the warehouse at night. Signs are posted at affected stop locations directing passengers to the new stops.

In case of emergency changes, server administrators use the master machines interface to input new routes and send them to affected buses. Passengers on board learn of changes via the bus intercom.

All routing changes are posted on the MBTA website and Twitter page.

3.5 Passenger Feedback

We collect customer feedback using the MBTA Twitter account run by a system administrator. Twitter is preferred over an application because it comes with a ready-to-use interface that is already widespread, whereas an application needs to be deployed. In addition, passengers from low-income households may not

own a smartphone for using an application. On the other hand, Twitter is available on the Web and can be accessed from public computers.

The server stores the bus control points service for decades, allowing system administrators to investigate reliability complaints. The server stores the passenger counts for all buses that report loads close to or exceeding the target, allowing system administrators to investigate passenger comfort complaints.

4 Evaluation

We argue that MassRTTC allows the MBTA to meet its targets, can perform its tasks quickly, and uses a reasonable amount of storage.

4.1 MBTA targets

The dynamic holding strategy proposed in Xuan et al. (2011), in theory, should keep all buses on schedule almost all of the time with high probability, even with the stochasticity of traffic.[4] Because buses are scheduled to meet the fairly inelastic peak demand (the commute), buses will generally not be overcrowded if they follow their schedule. If a bus is chronically overcrowded, its schedule will be adjusted in the long run. Thus in theory, the system should fail to meet targets with very low probability.

This is complicated by the fact that the holding times proposed cannot be met exactly - because we are only holding vehicles that are already bunched instead of constantly recommending hold times. Furthermore, demands can vary from the Poisson processes arrival at bus stops is often modeled as, although not often, since commutes are fairly inelastic. However, the strategy proposed in Xuan et al. (2011) at least regularizes headways quickly even if a bus is far behind or removed, and so can deal with such variation.[4]

4.2 Data collection and transmission

In these evaluations, we will look at two cases: Business As Usual and High Stress.

4.2.1 Beam sensor

Since beam sensor counts are augmented to be large overestimates, they are useless on buses with high load, since their count would be chronically close to 1.4, and would need to send their security camera data every stop.

But suppose we consider the worst case for an uncrowded bus with extremely high turnover. Suppose a bus seats 40 people. Lets say the bus has 20 people and at every stop 10 people board and 10 people alight, so the true count stays at 20, and suppose one person alights via the front door and the others alight via the back. Then at each stop, the bus can detect 10 passengers boarding and 1 passenger alighting (the actual numbers can vary due to accuracy), but it stores the difference as 10*1.25 - 1*.85 = 11.65. It thus takes three stops for the believed ratio to exceed 1.4. Hence even in this bad case, overestimation causes camera frames to be sent every three stops, and uncrowded buses send data at most this often.

4.2.2 Communication time

Packet headers are 130 bits (48 src + 48 dst + 2 datatype + 32 time).
Network bandwidth = 16Mbit/s = 2MB/s
Latency = 10ms

Bus to server Buses send their GPS data every minute and their camera frames whenever necessary, potentially up to every stop. We assume stops are on average 3 minutes apart.
5 frames = 1.2MB
1 bus can send 5 frames in 1.2 MB/(2 MB/s) + 10ms = 0.61s
GPS Data = 130 header + 64bit GPS = 194 bit = 24.25 B

The bus storing GPS data every 5 sec and transfers once every 1 min (12 points sent at once).
Time to transfer = 12*194 bit/(16 MBit/s) + 10ms ≈ 10ms

Business As Usual

During normal operation, we do not expect many buses to have passenger count ratios close to 1.4. We assume it happens to at most 25% of buses, and overestimating passenger counts leads the remaining buses to send at least every three stops. Thus with the beam sensor estimates, we would expect to have at most 25% + 75%/3 = 50% of buses sending their frames data every 3 minutes, so about 17% of buses (about 170) sending every minute.

Over 10 frequencies 10 buses can send 1 5 frame packet in .61 sec.
170 buses can send 1 5 frame packet in 10.75sec.
10 buses can send their GPS data in 10ms
1000 buses can send their GPS data in 1s

Network usage = $\frac{\text{time to send}}{\text{total time interval}} = \frac{11.75s}{60s} = 20\%$

High Stress

During heavy operation, as many as 75% of buses may be crowded. Thus with the beam sensor estimates, we would expect to have 25%/3 + 75% = 84% of buses sending their frames data every 3 minutes, so about 28% of buses (about 280) every minute.
Over 10 frequencies 280 buses can send 1 5 frame packet in 17.08 sec.
GPS data are sent no more frequently in this case.
Thus network usage = 18.08s/60s = 30%

Server to bus We argue that the server does not use up too much of its outward bandwidth.

The responses to requests for accurate counts are integers, 4B per value, with 16.25B for headers.
Time to transfer = 20.25B/(2MB/s) + 10ms ≈ 10ms
Business As Usual: 170 buses per minute = 170*10ms = 1.7s (out of a minute) to send all the data
High Stress: 280 buses per minute = 2.8s per minute

The server must also transmit information under Protocols 1 and 2 to the buses. These are just integers, 4B + 16.25B.
Time to transfer = 10ms like before.
Business As Usual: 250*10ms = 2.5s per minute
High Stress: 750*10ms = 7.5s per minute

Suppose, in the worst case, 10% of routes must be rerouted at once, or 18 routes. This might happen, if, say, a large molasses flood shut down streets in a large portion of downtown Boston.
The route and alternate route databases are 100MB each. With 177 routes, we assume each route is about 1MB.
Time to send out one new route information = 1MB/2MB/s + 10ms = .51s
Time to send information for 18 new routes = 9.2s

Because rerouting should happen well in advance of one minute, route data does not require low latency, so it can be interleaved with other data. The network loads are thus as follows:
Normal operation: (1.7+2.5)s/60s = 7%
Heavy operation: 10.3/60= 17.2%
Heavy operation with a large one time event prompting sudden rerouting = 19.5/60 = 32.5%

4.2.3 Wait time for frequencies

The wait time for frequencies depends on how synchronized requests for frequencies are. We ran simulations involving all buses (no load reduction due to beam sensors) sending data at different levels of

synchronization (how bunched together requests are). Results suggest that with high probability, we dont expect data to be synchronized, with data being sent roughly uniformly at random. The method for obtaining frequencies also adds randomness, which acts to desynchronize requests. We can generally expect a low latency of .003 - .06s.

4.3 Storage

We argue that no data is too much for the components of MassRTTC to store.

4.3.1 Server storage

Each machine in the server has 10TB of storage. We assume each working day, which runs from about 0600 to 0200, is about 20 hours, or 1200 minutes.

Permanent data 600 MB Census + 10 MB Bus meta-data + 100 MB Routes +100 MB Alternate routes + 10MB Operator data = 820MB

GPS Each GPS point is 178 bits = 22.25B (section 3.3.5)
Number of data points = 1200 min * (60/5) /bus/min * 1000 buses = 14.4M
Total data size = 14.4M*22.25B = 320.4 MB/day

Passenger count Single count data = 178 (general data) + 32 (value) = 210 bit = 26.25 B
Total data = 280 buses/min * 26.25B/bus * 1200 mins = 8.8 MB/day

Fares Single fare = 178 (general data) + 32 (value) + 64 (charlie card id) = 274bit
446700 people/day * 274bit/person = 15.3 MB/day

Stop servicing frequency 177 routes with at most 100 stops per route = at most 17700 stops
Times per day = 1200/20 = 60
Bus number and time = 48+32 bit = 80bit = 10B
All stops for one day = 10*60*17700 = 10.62MB/day

Headway variation We calculate the mean and variance every 20 minutes. There are 1000 buses, so we have 1000 such numbers, once every 20 min = 60*1000/day
Distance and time data = 64+32 = 96 bits = 12B
Total headway data = 12*60*1000/day = 0.72MB/day

Table 3: Long-term data storage size

Data	Day	Year	5 Years
GPS	320.4 MB	117 GB	585 GB
Passenger Count	8.8 MB	321 MB	18 GB
Fares	15.3 MB	5.6 GB	28 GB
Performance metrics	11.5 MB	4.2 GB	21 GB
Total	356 MB	130 GB	650 GB

With 100 TB of storage, we can store this data for many decades.

4.3.2 Bus storage

Buses have more limited storage, but MassRTTC handles this fine.

Fares Average 446700 people per day per 1000 buses = average 446.7 people per day per bus
Suppose the busiest routes service ten times as many = 4467 people max
Fares require 128 bits per payment, max data to store = 128*4467 bits = 571776 bits = 71472B = 71.5KB

Routing The route data for 177 routes is 100MB, so a reasonable size for a single route is around 1MB.

Since persistent storage is about 100MB, the bus storage is not overloaded.

4.4 Algorithm runtime

We argue that MassRTTC performs its calculations reasonably quickly.

4.4.1 Computer vision

Allowing the computer vision algorithm 0.1s per frame, the server processes one request in 0.5s. Running on one machine:
Business As Usual: 170 requests / 3 minutes = 85s
High Stress: 280 requests / 3 minutes = 140s

Spread out over 9 machines, the server completes 3 minutes of requests in 9.5s (5.3% load) during Business As Usual, and 15.6s (8.7% load) during High Stress.
Data transfer from master to worker for a single request is 1.2MB/(125MB/s) = 0.01s (Within server, bandwidth is 1GBit/sec = 125MB/s).
Delay = time to get frequency + time to send frames + time to move data from master to worker + time to process frames + time to send back count = .06 + .61 + .01 +.5 + .1 = 1.28s
Worst case times to get the data (slow to get a frequency and falling at the end of the processing queues), when requests are moderately synchronized:

Business As Usual: 0.06 + .61 + .01 + 9.5/2 + 4.2/2 = 7.0s
High Stress: .06 + .61 + .01 + 15.6/2 + 10.3/2 = 13.1s
These delays will not be a problem since the time between stops is never this short.

4.4.2 Congestion detection

This involves comparing adjacent buses on a route. Because buses are stored in order according to their routes in the working data table (3.3.5), finding adjacent buses involves looking at a few entries before and after around one entry, which is an O(n) operation, where n is the number of buses on the route. We ran a simulation and found that this algorithm was far faster than Computer Vision for all buses, completing under a second, and so its speed is not a concern.

4.4.3 Night Analyses

Based on the analysis above, lets assume a server can work with 1KB in .0005s = 2MB/s So to run the analysis will take 1.2GB/(2MB/s) = 600s = 10 min Since there are 4 hours at night, the analyses can easily done in time, even with far less efficient algorithms.

4.5 Measurement accuracy

While passenger count estimates are not very accurate, any server side count comes from the fairly accurate computer vision algorithm, which seldom miscounts by more than two people and hence does not affect sensitivity to crowding.

The passenger count data can be as much as 3 minutes old, which is about a stop. This means that congestion detection may not flag a bus until a stop after it is overcrowded. However, the bus system can take longer than this to get back on schedule, so this delay is not significant.

The GPS data has an inaccuracy of 5m, but the algorithm for detecting stops already takes this into account. This inaccuracy will hardly not affect whether buses are flagged.

While data about bus locations are fairly precise and can allow for long-term fine tuning in scheduling, data about how many people are in buses at one time, and who boarded and alighted where may be less accurate. These data can only provide an understanding of broad demographic trends in ridership, but these trends are sufficient for informing long-term planning decisions.

4.6 Scalability

The server architecture allows easy recovery in the event of a machine failure. Data is sent over the network sparingly, with about 30 percent used, even under high load. Physical limits on the capacity of a city makes added load on the server less concerning. The amount of data stored is also relatively small. Thus, while our system was not designed to scale, we can easily scale to a city with 2x more buses, with the same server and network capabilities.

An upgrade in the bus control in terms of CPU would increase scalability. A 10-20x compression on the frames data would result in the data being the same size as the GPS data, and would reduce network usage to 2%, thus vastly more scalable. It can also make our system more accurate as the counts can be obtained from the server more often. An upgrade on the network between the buses and the warehouse would have similar effects.

4.7 Security

An outside adversary might want to discover where a person is by examining all the security camera data. However, such an adversary listening to the network cannot gain any information, since all data is encrypted. Furthermore, because packets include what time they are sent, adversaries can neither forge nor replay packets to misdirect buses.

An MBTA insider cannot easily stalk someone over time either, because frames of security camera data are deleted soon after being processed. While the MBTA might, over time, know where passengers tend to get on and off over time because MassRTTC collects CharlieCard data, this loss in privacy is a minor concern compared to the enormous benefits derived from mining this data to improve routing.

5 Conclusion

Bus reliability and passenger comfort are paramount in a customer-driven system like the MBTA bus network. MassRTTC is a integrated real-time control system for the MBTA bus network that is designed to provide frequent service, decrease headway variability, and maintain passenger comfort.

MassRTTC collects data on service frequency, route coverage, passenger load, and network value from MBTA buses, and uses the data to evaluate whether it is meeting MBTAs service targets for reliability, coverage, and comfort. It can handle cases of high demand, route unavailability, and other unexpected events. Our system can scale to 2 times bigger bus networks with the same network and server resources. Such a system is crucial for maintaining high quality bus service.

References

[1] J. B. Gordon, Intermodal Passenger Flows on Londons Public Transport Network: Automated Inference of Full Passenger Journeys Using Fare-Transaction and Vehicle-Location Data, M.S.T. and M.C.P. thesis, Department of Civil and Environmental Engineering and Department of Urban Studies and Planning, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2012.

[2] G. E. Sanchez-Martinez, H. N. Koutsopoulos, and N. H. M. Wilson, Real-time holding control for high-frequency transit with dynamics, Transportation Research Part B: Methodological, vol 83, pp. 1-19, January 2016.

[3] J. G. Strathman, T. J. Kimpel, and S. Callas, Headway Deviation Effects on Bus Passenger Loads: Analysis of Tri-Mets Archived AVL-APC Data, Final Research Report for Tri-Met, 2003.

[4] Y. Xuan, J. Argote, and C. F. Daganzo, A Dynamic Holding Strategy to Improve Bus Schedule Reliability and Commercial Speed, unpublished, 2016.

6.033 Computer System Engineering
Spring 2018