Planning for Priority with Pings

Using High Resolution Transit Geolocation Data to Analyze Performance and Identify Intersections that Contribute to Transit Delay

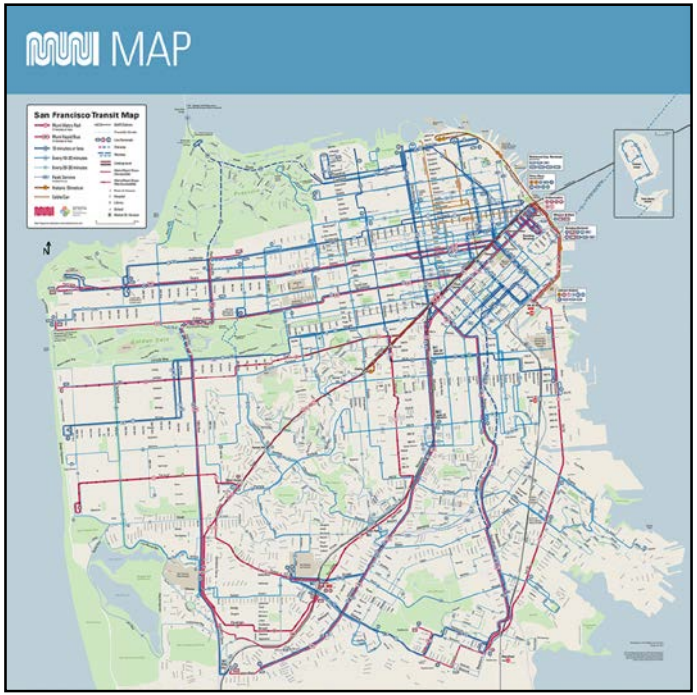
Tal Green, P.E.

SFMTA (Muni), Transit Planner

San Francisco, CA

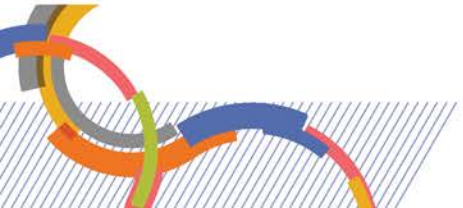
Presentation Outline

- Title breakdown!
- Current transit metrics
- Need for high res data analysis
- Data collection and methodology overview
- Metric #1 – transit signal delay
- Metric #2 - % of buses stopped at intersection
- Next steps



Title Breakdown

Using High Resolution Transit Geolocation Data to Analyze Performance and Identify Intersections that Contribute to Transit Delay



Title Breakdown

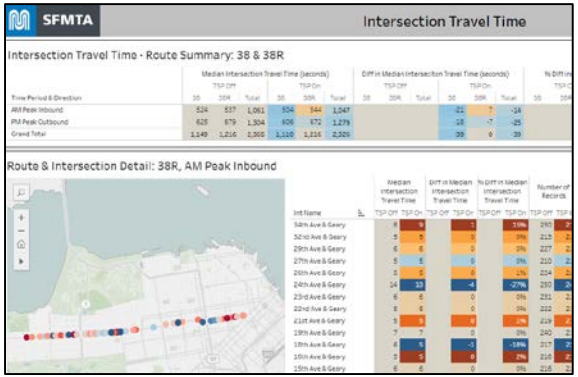
Using **High Resolution** Transit Geolocation Data to Analyze Performance and Identify Intersections that Contribute to Transit Delay

Title Breakdown

Using High Resolution **Transit Geolocation Data**
to Analyze Performance and Identify
Intersections that Contribute to Transit Delay

Title Breakdown

Using High Resolution Transit Geolocation Data to **Analyze Performance** and Identify Intersections that Contribute to Transit Delay

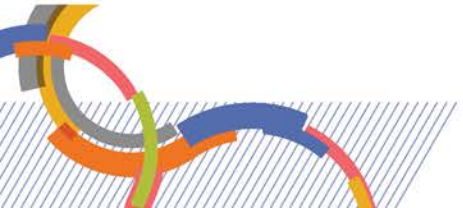


Title Breakdown

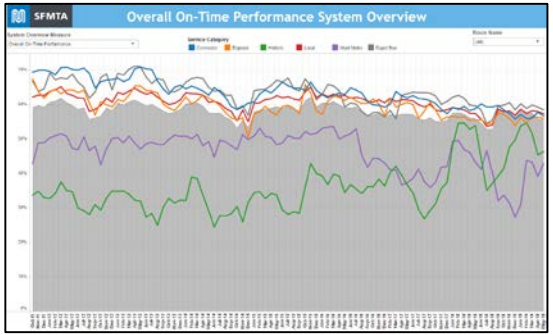
Using High Resolution Transit Geolocation Data to Analyze Performance and **Identify Intersections that Contribute to Transit Delay**

Title Breakdown

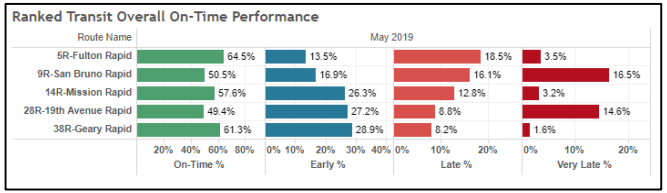
Using High Resolution Transit Geolocation Data to Analyze Performance and Identify Intersections that Contribute to Transit Delay



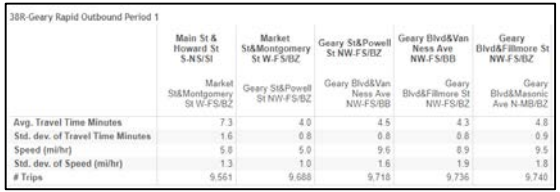
Currently Available Metrics



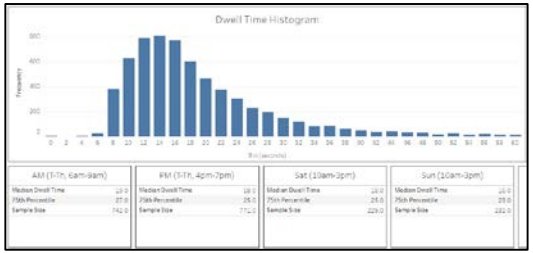
Mode Based On Time Performance



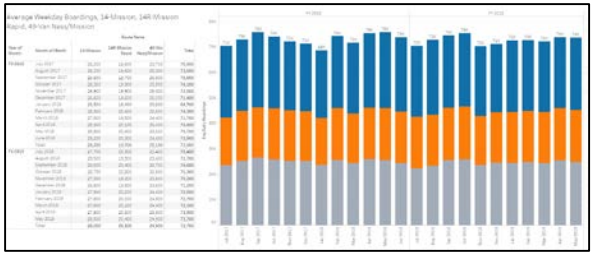
Route Based On Time Performance



Timepoint to Timepoint Travel Time



Stop Level Dwell Times



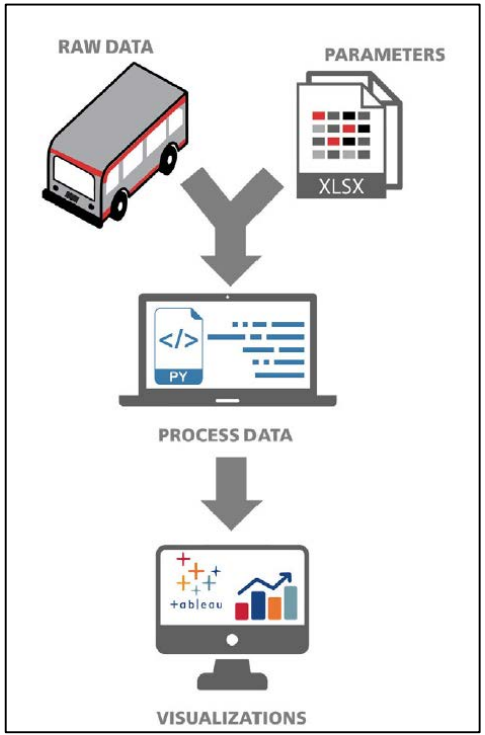
Route Level Ridership

Data Collection Effort

- On board equipment collects data
  - Approximately 500 buses equipped
- Daily data dump in yard over WiFi
- Weekly transmission of all data (~4 GB)
  - Began collecting data in November, 2018
- Data processing occurs weekly
  - Currently on local machine
  - Future: through IT in a production data warehouse

2.7 million GPS points from one day of data

Methodology Overview

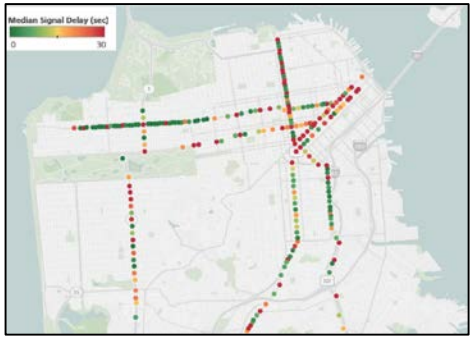


Metric #1 – Transit Signal Delay

- Processing
  - Isolated data to individual trips
  - Matched trip to schedule
  - Determined points of interest, matched to nearest
- Consumable data
  - Pushed to Tableau server
  - Can be pivoted by route, direction, location, time of day, etc.

Processing – target upstream and downstream points

Intersection Name	Median Delay (sec)	Number of Records
25th St & Mission	4	372
24th St & Mission	14	380
23rd St & Mission	11	367
22nd St & Mission	19	365
21st St & Mission	2	350
20th St & Mission	3	347
19th St & Mission	16	343
18th St & Mission	20	346
17th St & Mission	2	343
16th St & Mission	3	339
15th St & Mission	7	329
14th St & Mission	17	332



Analysis – Tabulated and mapped delay information by TOD, direction, route

Metric #2 - % of Buses Stopped at Intersection

- Processing
  - Isolated data to individual trips
  - Matched trip to schedule
  - Draw approach zones for signals
  - Determine if buses traveled <2MPH
- Consumable data
  - Pushed to Tableau server
  - Can be pivoted by route, direction, location, time of day, etc.

Processing – define and utilize approach zones

Intersection Name	% Stopped	Number of Records
25th St & Mission	29%	400
24th St & Mission	47%	408
23rd St & Mission	41%	415
22nd St & Mission	69%	379
21st St & Mission	5%	374
20th St & Mission	14%	378
19th St & Mission	55%	396
18th St & Mission	70%	394
17th St & Mission	4%	377
16th St & Mission	33%	375
15th St & Mission	34%	380
14th St & Mission	59%	362

Analysis – Tabulated and mapped bus stopped % information by TOD, direction, route

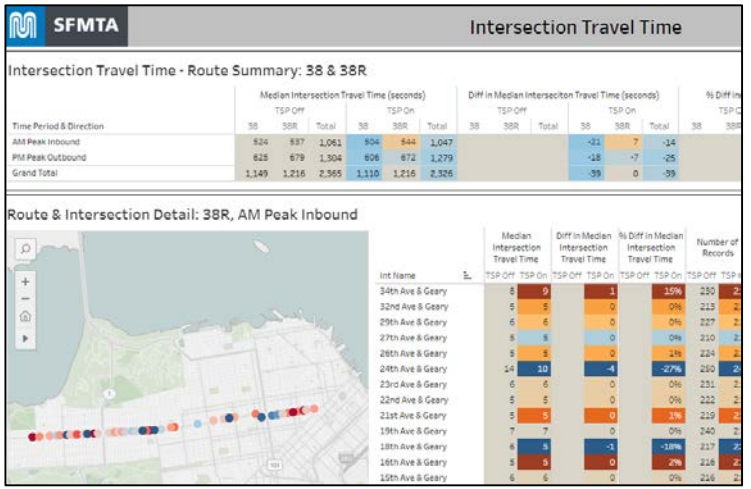
For More Details…

```
#During Daylight Saving (comment out if not) - March to November
allPoints['TimestampPacific'] = allPoints['TimestampNew'] - pd.Timedel

allPoints['tripID'] = allPoints['tripID'].astype(str)
allPoints['date'] = allPoints['TimestampPacific'].dt.strftime('%Y-%m-%
allPoints['trip_Date'] = allPoints['date'] +" "+ allPoints['tripID']

#display(allPoints.tail())


# Loop through each zone to get minimums in each per trip_Date

for z, zone in allZones.iterrows():

    try:
        pointsInZone = allPoints.loc[(allPoints['route'] == zo
                                      (allPoints['DIRECTION'] =
                                      (allPoints['lat'] > zone[
                                      (allPoints['lng'] > zone[
                                      (allPoints['lat'] < zone[
                                      (allPoints['lng'] < zone[
                                     ]

        # Go to next zone if filter yielded nothing
        if pointsInZone.empty:
            continue

        # Determine the minimum value in each zone
        minPoints = pointsInZone.loc[pointsInZone.groupby("tri

        # True if stopped, False if didn't stop
        minPoints['stopped'] = minPoints["speed"]<2

        # Add values from current Zone series to each record
        minPoints['zoneID'] = zone['zoneID']
        minPoints['CNN'] = zone['CNN']
        minPoints['intName'] = zone['intName']
        minPoints['intSequence'] = zone['intSequence']
        minPoints['includesNearsideStop'] = zone['includesNearsideStop

        #LOOK INTO THIS: at some point, the order of the columns chang
```

```
        mergedData["segmentID"] = segment["segmentID"]
        mergedData["CNN"] = segment["CNN"]
        mergedData["intName"] = segment["nAme"]
        mergedData["freeFlowTT"] = segment["freeFlowTT"]
        mergedData["includesNearsideStop"] = segment["includesNearsideStop"]


        # Calculate delay
        mergedData["delay"] = mergedData["travelTime"] - mergedData["freeFlowTT"]


        # Flag extreme outliers (True if > 10 minutes to get through intersection)
        mergedData["extremeOutlier"] = (mergedData["travelTime"]>600)


        # Drop extra data
        mergedData.drop(['PUBLICROUTENAME','TRIPID','LASTUPDATE','TimestampNew'],axis=1,inplace=True

        #resultsDF = pd.concat([resultsDF,mergedData])


        return mergedData

    except:
        print("ERROR!")
        # Add code here to better track errors (create list and add to list)

def progressBar(chunkCounter, totalChunks, currentSegment, totalSegments, progress=0):
    newProgress = int(100 * float(currentSegment)/float(totalSegments))
    if newProgress >= (progress+5):
        clear_output()
        print("Progress: ",newProgress,"% complete (",currentSegment,"of",totalSegments,"trips analy
        progress = newProgress
    return progress
```

Next Steps

- Conduct before/after studies
- Productionalize processing efforts
- Include intersections with nearside stops
- Address poor GPS readings in NE due to Urban canyon effect



Acknowledgement

Ian Martin
Student Intern
Summers 2018, 2019

Thank You!

Tal Green, P.E.
Tal.Green@sfmta.com