

An Interactive Traffic Information System

Moses N. Derkalousdian
Computer Science Department
California State University San Marcos
San Marcos, CA, U.S.A

Dr. John H. Chang
Computer Science Department
California State University San Marcos
San Marcos, CA, U.S.A

Abstract – With real time traffic information, drivers on our highways can adapt and take action when encountering congestion. They can reschedule appointments or choose an alternate route. Existing systems that provide real time traffic information rely on infrastructure and human resources. These techniques are inaccurate, slow and require expensive infrastructure that provide limited coverage. We propose a new and unique form of traffic data collection that provides accurate real time data without relying on any existing techniques. A vehicle containing a global positioning system receiver reports the speed and direction of the traffic for a specific location and time along the highway it is traveling. A central server gathers and processes this data from all such vehicles through an internet connection and reports this information in real time. Through this server, vehicles traveling all highways globally can share traffic information with each other in real time.

Keywords: Traffic Congestion, Real-Time, GPS, Share, Vehicles, Traffic Information

1. Introduction

As people continue to settle in suburbs outside metropolitan or industrial areas and increasingly in neighboring communities, more people are depending on our highways to commute to work. They spend more time and travel further, increasing delays in small cities and rural areas. Construction of new highway capacity has not kept pace. Between 1980 and 2000, highway miles increased by 1.5% while miles traveled by vehicles increased by 76%. As a result, the annual delay for travelers has increased by 293% since 1982 [1]. Traffic congestion is affecting more people with increased severity.

Real time traffic information can help drivers adapt to the worsening traffic conditions. With an accurate estimate of travel time, a driver can take appropriate actions to prepare for or minimize the consequence of unexpected delays. They can reallocate time, adjust their daily schedules, or call ahead to reschedule meetings and appointments. They can choose to take an alternate route if the time difference is beneficial or simply indulge in the fact that they know what's coming up and the severity of it.

Current systems that provide real time traffic information rely on several unreliable or expensive techniques; human observation, infrastructure such as sensors in the pavement and cameras, data processing by humans, etc. These infrastructure based systems are static and cannot respond to changing traffic patterns. Real time traffic data is limited to city streets or highways in major metropolitan areas. A commuter cannot benefit if they spend the majority of their travel time on highways or in rural areas.

The goal of our research is to enhance the methods used to gather traffic information to make data less expensive and widely available. We propose a portable technology to deliver nationwide traffic information to users within milliseconds of any changes in traffic without relying on human interaction or infrastructure. This approach can deliver traffic information at resolutions down to the meter. Improving the accuracy and response time of traffic information allows travelers to make last minute decisions. Users will have all the information necessary to identify an alternate route or reschedule an appointment without having to guess. By using vehicles themselves as traffic sensors, this system can provide inexpensive data that dynamically adapts to changing traffic patterns. Naturally, vehicles exist along all highways where they can be used to report traffic conditions. With traffic congestion, vehicle density increases, increasing the number of sensors available in an area of concern. Those commuters traveling from neighboring communities will benefit equally with commuters in major metropolitan areas. This paper makes the following contributions:

- A client-server architecture that gathers traffic information from traveling vehicles.
- A client algorithm that uses a global positioning system to report the velocity of a traveling vehicle.
- A server algorithm to aggregate velocity reports from vehicles (clients) and report it to clients in real time.

2. Related Work

This section surveys the state of the art as expressed in existing systems. Below is a list of current work done by commercial and government groups that provide real time traffic information. Each entry in the list describes the technology used by the group and at least one unique area for improvement:

1. Automated Traffic Surveillance and Control (ATSAC):
This system takes advantage of underground sensors at signal light intersections to detect congestion and vehicle speeds. The information gathered is only available at signalized intersections. This system can be improved by expanding the coverage area for which traffic information is available.
2. Transportation Management Center – San Diego (TMC):
Real time information is gathered from many sources including underground sensors, ramp meter sensors, and earthquake monitors. Infrastructure is expensive and users interested in traffic information for areas containing little or no infrastructure do not benefit. This system can be improved by gathering data without relying on infrastructure.
3. Traffic.com:
Data is gathered from underground digital sensors and processed by an operations staff to monitor traffic conditions in each city they serve. Techniques that rely on human resources to process information generate inaccurate estimates. This system can be improved by providing traffic information with greater accuracy.
4. Jaytu Technologies (Sigalert.com):
Traffic information available consists of the speed at a limited set of exits along the highway for both directions. This system can be improved by providing an accurate travel time to a user given their start location and destination.

3. Design Approach

The design is based on information sharing. If everyone shares their information then everyone can benefit. The system is comprised of a central server and clients. The server is an application that can communicate to clients through a high speed internet connection using the internet protocol (IP). A client is an application that can communicate with the server using IP. A client is considered *dynamic* if it executes on a mobile device containing a global positioning system (GPS) receiver. If all dynamic clients send their position, speed and heading to the server, then the server can estimate traffic conditions from these reports and provide information back to all clients. In other words, if dynamic clients continuously inform the server of traffic patterns as they are traveling, the server will contain real time traffic information. The clients can then request traffic information for their surroundings from the server.

For the purposes of this report, we assume wired or wireless internet connectivity exists to provide clients with continuous internet access.

3.1 Client

The client communicates with the central server over IP and provides users with an interactive interface to view traffic information. The client application is responsible for the following tasks:

- Update central server with position, speed, and heading.
- Read traffic information from the central server and present it to the user.

Both tasks require the client and server to communicate over some application layer protocol. A new protocol was developed to handle the specific type of data that is communicated. This protocol is first introduced before moving on to the two client tasks.

3.1.1 Client – Server Protocol

The communication language used by the client and server is called the *Real Time Traffic Update Protocol* or *RTTUP*. The protocol uses the Transmission Control Protocol (TCP) as its transport mechanism through the internet and defines the syntax and semantics of its commands. RTTUP defines the following specifications:

- Port 4000 as the port the server listens to for connections.
- No pipelining.
- RTTUP uses persistent connections; a TCP connection is left open until a client closes the connection.

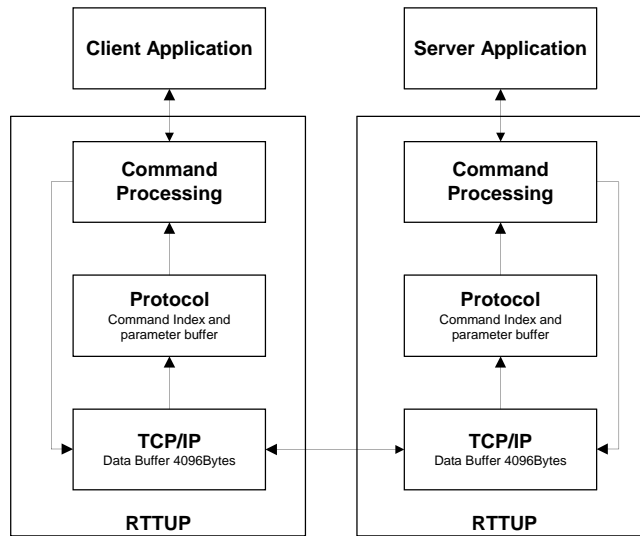


Figure 1: RTTUP Protocol Layers.

The protocol can be broken up into three layers as seen in figure 1. The first layer is responsible for TCP/IP communication through a socket interface. It reads and writes data using the data buffer as its storage medium. The Protocol layer is responsible for parsing data strings from the TCP/IP layer into a command and parameters. It decodes and stores the command and all of its parameters for use by the Command Processing layer. The Command Processing layer is responsible for executing the specific tasks required by a command. Note in figure 1, the Protocol layer is bypassed when sending commands since a string does not need to be parsed.

3.1.2 Updating the Server

In order for this system to work, dynamic clients must exist and be able to share their position, speed, and heading. The client accomplishes this by periodically requesting information from its GPS device and conveying it to the server. In order to conserve communication bandwidth and server processing time the client sends updates at a time interval directly proportional to its speed. A client traveling at slow speeds is not covering ground fast enough to warrant an update, therefore it waits longer. A client traveling at high speeds covers more ground, thus updates occur more frequently. The algorithm used for this is depicted in the flow chart shown in figure 2 below.

The main loop in figure 2 performs three major tasks per iteration: Requesting data from the GPS device, reading the data and finally processing the data. After each task the loop moves on to the next task or performs a retry. Retries are not infinite; the loop fails after a fixed number of reattempts. Some retries force a wait state between attempts. This frees processing time while waiting for data to be sent or received. The update loop begins by requesting data from the GPS device. If the request fails the loop reattempts without any wait states. Wait states are not necessary because data is being sent not received. If successful, the loop moves on to read a response from the GPS device. If the data has not been received or the read fails a retry occurs after a wait state. Again the client loop does not wait forever to read data; it will eventually fail indicating something is wrong. The next step in the loop is to process the information that was received. The client sends an *Update* command using the Command Processing layer to communicate the traffic information to the central server. Before the loop reiterates, it waits some time based on its speed. An update occurs every 10 seconds if the speed is zero. The minimum update period is 1 second at 160 kilometers per hour or above.

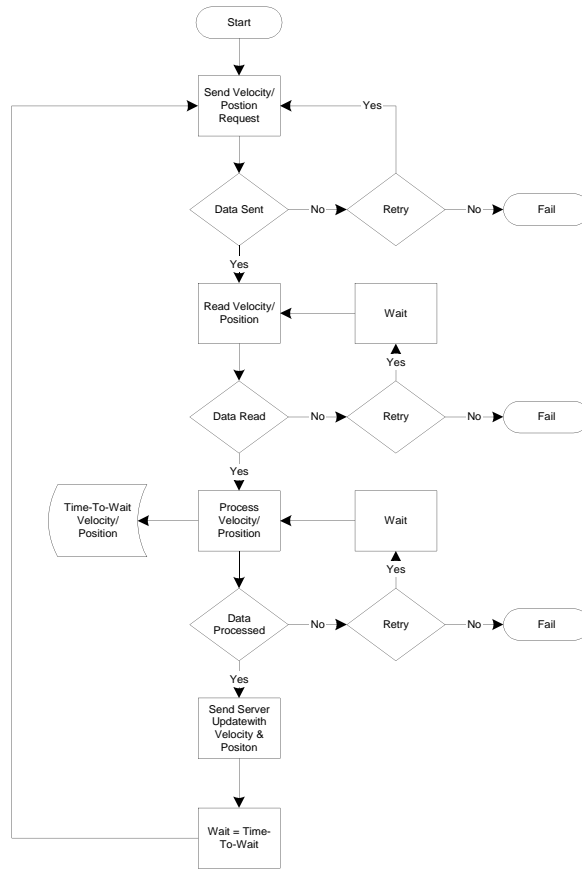


Figure 2: Server Update Flow Chart.

3.1.3 Read data from Server

Acquiring traffic information from the central server and presenting it to the user is the second major task performed by the client. RTTUP defines a *Get* command the client uses to request traffic information. The command requires the center, length, and width of a rectangular area within which traffic information is desired. Heading is optional to limit traffic information to a specific direction. If the client is traveling along the north bound direction it may not be interested in south bound traffic. The server responds with the *Data* command containing all the traffic information available in the requested area. The traffic information returned is broken up into multiple *Traffic Units*. Each unit is approximately 55 meters or 0.0005 degrees squared. It contains the speed, heading, and bounding rectangle of its location. Those Traffic Units with their center inside the rectangle defined in the *Get* command are returned. The *Data* command returns a single string encoded with all the Traffic Units that fit.

3.2 Server

The server is concurrent, connection-oriented, and stateless. It uses the RTTUP protocol defined in the client section. When launched it enters an infinite loop waiting for a client to connect on port 4000. A new thread is created to concurrently handle every client connection. Once the client closes its connection, no information about that client is kept and the thread terminates.

Client threads enter an infinite loop and wait for a command. The TCP/IP layer receives the command, passing it up to the Protocol layer which will parse it and send the command and parameters to the Command Processing layer. The Command Processing layer creates a response for the client and sends it directly through the TCP/IP layer (See figure 1).

The server's main task is to maintain a shared database of traffic information sent by dynamic clients. The server processes arriving Update commands by storing the traffic data into a central database. The server processes Get commands by retrieving the corresponding Traffic Units from the central database. The Traffic Units are formatted into a *Data* command and sent to the client. The remainder of this section describes the implementation of the database.

The goal of the database is to maintain traffic information for any part of the world. The traffic database is made up of *Traffic Cells*; the information contained in these cells is a superset of the Traffic Units described under the client section. Traffic Cells are based on position, speed, and heading updates received from dynamic clients. The set of data from these updates are defined as *Data Points*. The database creates and maintains cells dynamically as data points arrive. The surface of the earth is not segmented into cells ahead of time. Because data points are sent by dynamic clients traveling along highways they automatically map all highways on earth. By using the position of the data points to locate cells dynamically, the cells automatically align themselves over highways. Cells are added to the end of the database as they are created. There is no particular order in which they are stored. Traffic cells contain three bounding rectangles to support this functionality. The *Cell Bounds* rectangle specifies the actual cell's borders, *Data Point Bounds* specifies the collective border of all the data points inside the cell, and *Available Bounds* specifies the available area a cell can shift into. Figure 3 diagrams a cell and all of its bounding rectangles. When a new cell is created it is centered over the data point that initiated its creation. This cell can then be shifted in any direction as long as the data point remains inside the cell's bounding rectangle. The Available Bounds specifies the area into which a cell can be shifted to include a new data point. A new data point is added to a cell if it is inside Available Bounds. When a data point is added to a traffic cell the Data Point Bounds are updated to include the new data point. The Traffic Cell and therefore the Cell Bounds are shifted so it is centered over Data Point Bounds, and Available Bounds is recalculated. Available Bounds and Cell Bounds are calculated using Data Point Bounds as follows:

- $d_0 = (\text{CELL SIZE} - \text{Data Point Bounds Height}) / 2$.
- $d_1 = (\text{CELL SIZE} - \text{Data Point Bounds Width}) / 2$.
- Cell Bounds Min Longitude = Data Point Bounds Min Longitude - d_1 .
- Cell Bounds Max Longitude = Data Point Bounds Max Longitude + d_1 .
- Cell Bounds Min Latitude = Data Point Bounds Min Latitude - d_0 .
- Cell Bounds Max Latitude = Data Point Bounds Max Latitude + d_0 .
- Available Bounds Min Longitude = Cell Bounds Min Longitude - d_1 .
- Available Bounds Max Longitude = Cell Bounds Max Longitude + d_1 .
- Available Bounds Min Latitude = Cell Bounds Min Latitude - d_0 .
- Available Bounds Max Latitude = Cell Bounds Max Latitude + d_0 .

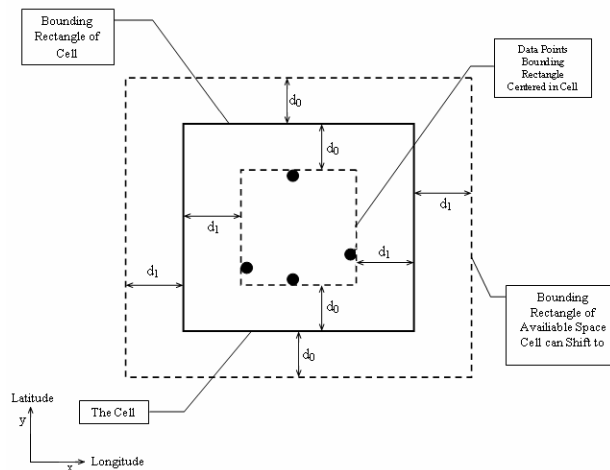


Figure 3: Traffic Cell.

A Traffic Cell defines the speed for some heading, at some position on Earth. Therefore a single position on Earth can have multiple cells if data exists in more than one heading. This can be true for highways containing multiple overpasses. The heading of a cell is equal to the heading of the data point that first created the cell. New data points are added to a cell only if their heading is ± 10 degrees of the cells heading.

A Traffic Cell contains a first in first out (FIFO) queue of data points. This queue has a limit of N total data points possible at any given time. If there are N data points in a cell and a new data point arrives, the oldest data point is deleted from the cell to make room for the new one. If there are less than N data points when a new one arrives, none are removed.

Every data point includes the time it entered the database. This time is used by the database to delete Traffic Cells when the youngest data point of a cell has existed for more than T milliseconds. If a traffic cell does not acquire a new data point every T milliseconds it expires and is removed from the database. This ensures the traffic database does not store old or outdated traffic information.

The velocity of a Traffic Cell is calculated by averaging the velocity of every data point in a cell. The average is taken by summing the velocity of every data point and dividing that by the total number of data points in the cell. Because the data points are stored in a fixed sized FIFO queue, the velocity becomes a sliding window average. The following algorithm describes how a new data point is added to the traffic database:

```

AddPointToDatabase(dataPoint)
1 Longitude ← Longitude of dataPoint
2 Latitude ← Latitude of dataPoint
3 Speed ← Speed of dataPoint
4 Heading ← Heading of dataPoint
5 for all existing cells
6   if Longitude AND Latitude are inside Available Bounds of cell
7     if Heading matches cell heading ±10 degrees
8       if number of data points in cell > N
9         Remove oldest data point
10        Add dataPoint to cell
11        UpdateCellBounds(Longitude, Latitude)
12        UpdateCellSpeed(Speed)
13      return
14 Add new cell to Database
15 UpdateCellBounds()
16 UpdateCellSpeed()
17 return

```

4. Results

The entire system was implemented using C++ on Microsoft Windows platform for testing and estimating performance results. Dynamic clients were provided using a simple traffic simulator executing on a Windows machine. The simulator creates a set of vehicles with a known percentage of them containing dynamic clients. All vehicles travel along a fixed stretch of highway following one another while avoiding collisions. When requested, the simulator induces traffic congestion by decreasing the speed of a set of vehicles forcing all consecutive vehicles to also go slower. The congestion provides traffic data that varies with time. Tests were run to measure the response and performance of the entire system during a change in traffic.

To quantify the accuracy of the entire system the average speed for a fixed segment and heading of the highway is recorded. A recording samples speed continuously on both the traffic simulator and a client reading traffic data from the server, executing on a separate machine. Every time the speed is sampled the time is also recorded. The actual data, $A(t)$, from the traffic simulator and the reported data, $R(t)$, from the client can then be graphed against time for comparison. The result is an error function versus time, $E(t)$, showing the absolute difference between the reported and actual data. The average absolute error, AAE, is used to determine overall performance. The following formulas demonstrate this:

- $E(t) = |R(t) - A(t)|$
- $AAE = \frac{1}{n} \cdot \sum_{i=0}^n E(t_i)$

Only speeds recorded at the same time can be compared. It does not make sense to compare the speed for some location at different times. To synchronize data, both the traffic simulator and client reading traffic data must execute on the same machine. This guarantees the two applications record time read from the same hardware clock. The data collected must also be synchronized. The applications do not sample velocity at the same rate or in synchronization. A separate application called *Create Data* synchronizes the data from both applications. Create Data combines the data from both applications into a single file, ensuring the data starts and ends at the same time, and interpolates the data for times that do not exist. The resulting file contains the actual and reported speeds for the same times with 100ms increments. Table 1 below is the AAE for all dynamic client percentages.

Table 1: System Results.

% Dynamic Clients	AAE
10	35.08753
20	34.99677
30	25.42886
40	22.97508
50	21.72759
60	17.71168
70	14.16118
80	14.54125
90	13.18609
100	12.14986

5. Conclusion – Further Study

This paper presents a new technology for obtaining real time traffic information that was tested and shown to work. With only 20% of vehicles equipped with a client device, real time traffic information is available with an average error of less than 35 kilometers per hour. As more people equip their vehicles, the accuracy grows. As the accuracy grows the system increases in popularity. This fuels more people to equip their vehicles in turn increasing the accuracy. Although many improvements to this system are possible, this project is a step in the right direction. The following is a list of improvements, new features, and further study required.

1. Security is a high priority item that must be implemented in the system. Currently, anyone informed with the server’s address and port number can reverse engineer the RTTUP protocol and send false data to the server. This will corrupt the database in the server, making it useless. Encrypting the communication and forcing users to login would be one solution.
2. Traffic Cells in the server’s database require further study. One possible improvement is to dynamically split traffic cells. If a cell is reading two distinct speeds for enough time it can split to reflect the difference in velocity instead of averaging it out. For example, a traffic cell can be covering a car-pool lane along with regular lanes of a highway. Vehicles in a carpool lane can be traveling significantly faster than those in the regular lanes. This should be reported to users.
3. The sliding window average function of traffic cells requires further study. Dynamically changing the window size in response to the update frequency might be an improvement. For updates that occur frequently, the data could be filtered more often to more closely approximate real time. In this case the window should be made smaller. Updates that occur less frequently could be stored for a longer time because old data is better than no data. The window’s size should be made larger.
4. The Traffic Cells should be organized inside the database for efficient searching. When a client requests data or a new Data Point arrives, the algorithm must search through every cell to choose those that match. With the cells sorted, the searching could be more efficient.

6. List of References

[1] Schrank, D., “*The 2005 Urban Mobility Report*”, Texas Transportation Institute, 2005.