MNTG: An Extensible Web-based Traffic Generator

Mohamed F. Mokbel[1], Louai Alarabi[2], Jie Bao[3], Ahmed Eldawy[4], Amr Magdy[5], Mohamed Sarwat[6], Ethan Waytas[7], Steven Yackel[8]

[1,2,3,4,5,6,7]University of Minnesota, Minneapolis, MN 55455, USA
[8]Microsoft
{mokbel[1],louai[2],baojie[3],eldawy[4],amr[5],sarwat[6]}@cs.umn.edu
wayt0012@umn.edu[7] spazard1@live.com[8]

Abstract. Road network traffic datasets have attracted significant attention in the past decade. For instance, in spatio-temporal databases area, researchers harness road network traffic data to evaluate and validate their research. Collecting real traffic datasets is tedious as it usually takes a significant amount of time and effort. Alternatively, many researchers opt to generate synthetic traffic data using existing traffic generation tools, e.g., Brinkhoff and BerlinMOD. Unfortunately, existing road network traffic generators require significant amount of time and effort to install, configure, and run. Moreover, it is not trivial to generate traffic data in arbitrary spatial regions using existing traffic generators. In this paper, we propose Minnesota Traffic Generator (MNTG); an extensible web-based road network traffic generator that overcomes the hurdles of using existing traffic generators. MNTG does not provide a new way to simulate traffic data. Instead, it serves as a wrapper over existing traffic generators, making them easy to use, configure, and run for any arbitrary spatial road region. To generate traffic data, MNTG users just need to use its user-friendly web interface to specify an arbitrary spatial range on the map, select a traffic generator method, and submit the traffic generation request to the server. MNTG dedicated server will receive and process the submitted traffic generation request, and notify the user via email when finished. MNTG users can then download their generated data and/or visualize it on MNTG map interface. MNTG is extensible in two frontiers: (1) It can be easily extended to support various traffic generators. It is already shipped with the two most common traffic generators, Brinkhoff and BerlinMOD, yet, it also has the interface that can be used to add new traffic generators. (2) It can be easily extended to support various road network sources. It is shipped with U.S. Tiger files and Open Street Map, yet, it also has the interface that can be used to add other sources. MNTG is launched as a web service for public use; a prototype can be accessed via $http://mntg.cs.umn.edu$.

1 Introduction

Road network traffic data consist of a set of spatial locations reported by a set of objects moving over a road network. Traffic data have been already leveraged by researchers in different areas, e.g., spatial-temporal databases, transportation, urban computing and data mining. The process of extracting real traffic data requires installing and configuring many GPS-enabled devices and continuously monitoring the locations of such devices, which is a cumbersome task. For instance, GeoLife project [1] took more than

four years to collect 17,621 trajectories dataset with the involvement of 182 volunteers in Beijing. Alternatively, many researchers opt to generate synthetic road network traffic data. As a consequence, several efforts have been dedicated to develop road network traffic generators, e.g., Brinkhoff [2] and BerlinMOD [3].

Even though existing traffic generators are quite useful, nonetheless, most of them suffer from the following: (1) It may take the user significant amount of effort to install and configure the traffic generation tool. For example, in order to run BerlinMOD, the user needs to first install a moving object database, i.e., SECONDO [4], and then get familiar with the script commands used to install it. After the installation, users still need to understand an extensive list of configuration parameters for each traffic generator. (2) It is not trivial to generate traffic data in arbitrary spatial regions using existing traffic generators. For example, to be able to use Brinkhoff or BerlinMOD generators for a different city than the default shipped one (Oldenburg and Berlin for Brinkhoff and BerlinMOD generators, respectively), the user needs to first obtain the road network information for the city of interest, which is a tedious task by itself. For example, to get the road network information for the city of Munich, a user may need to understand the format of OpenStreetMap [5], and then write a program that extracts the road network of Munich from OpenStreetMap. After obtaining the new road network data, the user will then need to understand how to modify the obtained format to match the required one by either Brinkhoff or BerlinMOD. Such set of tedious operations made it hard for casual users to use these traffic generators for arbitrary spatial areas. As a testimony, one can observe that almost all the literature that have used these generators for traffic data have used it for their default cities.

In this paper, we propose Minnesota Traffic Generator (MNTG); an extensible web-based road network traffic generator that overcomes the hurdles of using existing traffic generators. MNTG is basically a wrapper around existing traffic generators, with the mission of enabling an easy usage of all existing traffic generators, and hence help all researchers worldwide in validating and benchmarking their research techniques against various workloads of movings objects over real road networks.

MNTG has three main features that significantly help in achieving its goals: (1) MNTG is a web service with an easy-to-use user-friendly map interface. Behind the scenes, MNTG carries the burden of configuring and running existing traffic generators. Thus, MNTG users do not need to install or configure anything on their local machines. This is in contrast to the traditional usage of Brinkhoff and BerlinMOD generators that require various installations as mentioned above. (2) MNTG can be used for any arbitrary city or spatial area worldwide. Users can just navigate through the map interface, and mark their area of interest with a rectangular area. Once the traffic generation request is submitted, MNTG is responsible for extracting the road network information for the requested area and generating the traffic on the extracted area using one of the existing traffic generators. This is in contrast to the traditional usage of existing traffic generators that is hard to be tailored for arbitrary cities. (3) MNTG users do not need to worry about the processing time or computing resources, where MNTG has its own dedicated server machine that (a) receives a traffic request from the user, (b) internally processes the request in a multi-core multi-threaded program, and (c) emails the user back when the requested data is generated. The notifying email includes a link

to download the data as well as an option to visualize the generated data. This is in contrast to the traditional usage of existing traffic generators that may take significant portion of the user time and computing resources.

Minnesota Traffic Generator (MNTG) is extensible in two frontiers: (1) It can be easily extended to support various traffic generators through few deterministic functions. Currently, MNTG is shipped with the two most common traffic generators, Brinkhoff and BerlinMOD, yet, it also has the interface that can be used to add new traffic generators. As a proof of concept, we extend it with a random generator which generates some kind of random walks over the road network. (2) It can be easily extended to support various road network sources. It is currently shipped with the support for U.S. Tiger files [6] and OpenStreetMap [5], yet, it also has the interface that can be used to add other sources for road network data.

MNTG is equipped with three main components, listed as follows: (1) *Road Network Converter*: that extracts the road network for the area of interest from either US. Tiger files or OpenStreetMap, and converts it to match the format of the underlying traffic generator. (2) *Traffic Processor*: that schedules and executes the received traffic generation requests. The traffic processor processes the incoming requests in parallel using a multi-threading paradigm to increase the overall system throughput. Both the road network converter and traffic processor provide an interface for the system users to incorporate a newly developed traffic generator. To plug-in a newly developed traffic generator, MNTG defines a set of abstract functions that users need to implement. The implemented functions deal with converting/extracting the road network data, executing the traffic generator, and preparing the generated traffic output. Once a traffic generator is plugged-in, users may leverage it to generate traffic data. (3) *System Front-End*: which contains a web interface for users to submit traffic generation requests, an email notifier to send messages or notifications to the user, and a set of tools for the user to download and visualize the generated traffic data.

A preliminary version of MNTG is launched as a web service for public use; a prototype can be accessed via $http://mntg.cs.umn.edu$. The preliminary version supports Brinkhoff, BerlinMOD and the random traffic generators on both U.S TIGER files and OpenStreetMap data. The extensibility interface for adding more generators or other road network sources is currently working internally under our support. Yet, these functionalities will be released to public use in our next version. Since its launch last month, MNTG has received more than 1000 traffic generation requests from researchers world wide. All requests have been efficiently satisfied, and results were sent back to the requesting users. We envision that MNTG will be the de facto standard for generating road network data for researchers in spatial and spatio-temporal databases worldwide.

The rest of this paper is organized as follows: Section 2 highlights related work. Section 3 gives an overview of MNTG. Sections 4 and 5 describe the two main components in the system back-end; (1) road network converter and (2) traffic models, respectively. Section 6 provides the description of the system front-end with detailed usage guide. Finally, Section 7 concludes the paper with pointers to future work.

Environment	Generators
Free Movement	Pfoser and Theodoridis [7], Oporto [8], GSDT [9], G-TERD [10]
Road Network	Brinkhoff [2] , BerlinMOD [3] , ST-ACTS [11] , GAMMA [12], SUMO [13] , Micro Simulators [14]
Multi Environments	MWGen [15]

Table 1. Existing Moving Objects Generators.

2 Related Work

Road network traffic data (i.e., moving objects data) have been widely used by researchers to test and validate their techniques in various spatio-temporal data management problems. This includes objects tracking [16], predictive queries [17], range queries [18], *k*NN queries [19], continuous queries [20], data uncertainty [21], and location privacy [22]. As a consequence, several efforts have focused on creating standard benchmarks for evaluating research on moving objects data [3, 23–27]. As part of creating such benchmarks, generating synthetic moving objects data gained considerable attention in the literature.

Table 1 gives a summary of existing moving objects data generators. Based on the spatial environment where the objects move on, existing moving objects data generators can be classified into the following three main categories:

1. *Free movement* [7–10]. This category assumes that objects can move freely in a two-dimensional Euclidean space. The GSTD generator [9] generates data for either moving points or rectangular regions, where it allows its users to control the lifetime of each moving object. The GSTD generator is later extended to incorporate real-life behaviors like group and obstructed movements [7]. G-TERD [10] has introduced new features to traffic generators, where users can generate arbitrary-shaped objects with tuning parameters that control object speed, color, and rotation over time. Unlike all other moving objects generators, Oporto [8] is particularly concerned with generating the movement of ships, which depends on fishing scenarios where ships head to fish shoals and avoid storms.
2. *Road networks [2, 3, 28, 11–13, 29, 14]*. This category is mainly concerned with generating moving objects data in a road network environment. Constrained by the predefined road network paths, they basically generate traffic data based on real-life trip planning scenarios that simulate the human behavior. Brinkhoff [2], SUMO [13] and micro simulators [14] depend on short-term observations, where representative human behavior is observed for short discrete trips. On the other hand, BerlinMOD [3] and ST-ACTS [11] rely on long-term observations where human behavior is observed for several consecutive days.
3. *Multi-Environments* [15]. This category considers moving objects in multi-environments, e.g., Indoor $\rightarrow$ walk $\rightarrow$ Bus $\rightarrow$ walk. MWGen [15] surpassed the typical functionality of generating data only for a single environment and extends it to support multiple environments, i.e., indoors and outdoors, at the same time.

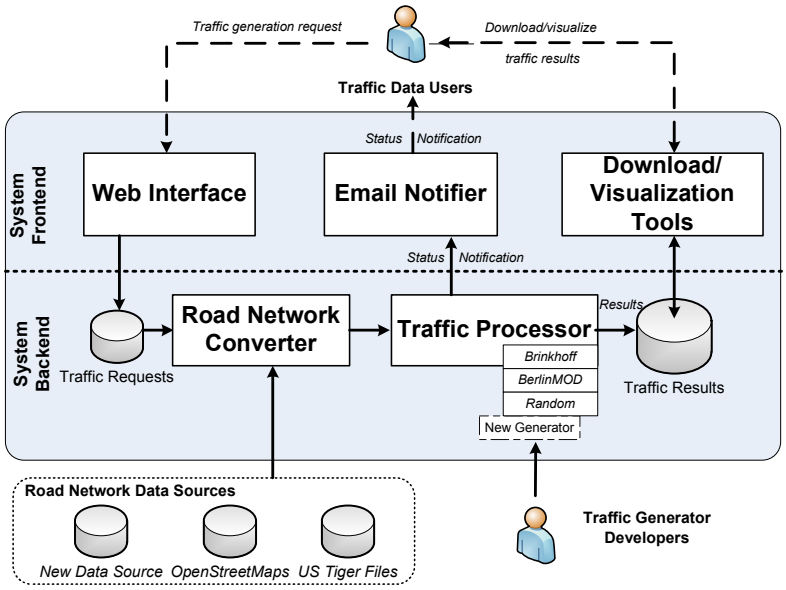

Fig. 1. MNTG System Overview.

GMOBench simulates a scenario like an employee moving in her work building, then walks to the parking lot to drive her car all the way to home, then walks again to enter home and move indoors.

Minnesota Traffic Generator (MNTG) distinguishes itself from all the work mentioned above as it does not provide yet another technique for traffic data generation. Instead, it is an extensible wrapper built around any of the existing traffic generators in the second category mentioned above (road network movement). As an extensible easy-to-use wrapper, MNTG enables a practical use of all road network traffic generators developed over the last two decades. Due to its simplicity, MNTG is expected to give a boost to existing traffic generators by gaining wider user community.

3 System Overview

Figure 1 gives an overview of the MNTG system architecture. A user interacts with MNTG through its system front-end that includes three main components: (1) *Web Interface*, which allows users to submit traffic generation requests by selecting a geographical area on the map and setting the corresponding parameters in a very intuitive way, (2) *Email Notifier*, which retrieves the status updates from the back-end and keeps users posted on their traffic generation request progress, and (3) *Download and Visualization tools*, which allow users to download their generated traffic data as a text file and/or visualize the generated traffic data on the map. The details of the system front-end, with its three components, will be discussed in Section 6.

Internally, the system back-end of MNTG processes incoming traffic generation requests and generates traffic data for the system users. The system back-end consists of the following two main components:

1. *Road Network Converter*, which is responsible for extracting the road network data from the traffic generation request. It receives a rectangular spatial area with two corner coordinates, each represented as a $<$*latitude, longitude*$>$. Then, the *road network converter* exploits its underlying road network data source, US Tiger files or OpenStreetMaps, to extract the information of the selected area, and convert it into an appropriate format understood by the requested traffic generator. The *road network converter* is extensible to support other road network data sources beyond its default ones, US tiger files and OpenStreetMaps. Details of the *road network converter* will be discussed in Section 4.
2. *Traffic Processor*, which takes the road network data from the *road network converter* and feed it to the requested traffic generator, which is currently either Brinkhoff or BerlinMOD. The traffic processor is implemented in a multi-threading paradigm to: (a) allow multiple requests to be served concurrently, and (b) avoid the starving of small traffic generation requests waiting for large requests to finish. The *traffic processor* is highly extensible as it is equipped with modules that allow the traffic model developers to easily plug-in a new traffic generator. Details of the *traffic processor* will be discussed in Section 5.

4 Road Network Converter

Despite the abundance of road network data sources, such as US Tiger files and OpenStreetMap, none of them provides an intuitive way to extract road network paths, i.e., nodes and edges, for an arbitrary geographical area. MNTG, on the other hand, needs to generate road network traffic data for any geographical area selected by the user. To achieve that, MNTG employs a *road network converter* that is responsible for extracting the road network data for each incoming traffic generation request. Moreover, The road network converter is designed to support a wide variety of road network data sources.

In this section, we first discuss the main idea behind the road network converter. Then, we provide two detailed case studies featuring two road network data sources, which are currently implemented in MNTG: US Tiger Files and OpenStreetMap. Finally, we discuss the extensibility of the road network converter to support new road network data sources.

4.1 Main Idea

The road network converter is responsible for extracting road network nodes and edges from different sources and transforming the extracted data into a standard format that can be utilized by different traffic generators. The functionality of the road network converter does *not* depend on the underlying traffic model (e.g., Brinkhoff or BerlinMOD). Instead, it heavily depends on the underlying data source (e.g., US Tiger files or OpenStreetMap). To achieve its goals, the road network converter performs the following two steps for each traffic generation request:

1. *Step 1. Extracting Road Network.* The input to this step is :(a) a rectangular spatial area, defined by two corner <*latitude*, *longitude*> coordinates, and (b) the road network data source (e.g., US Tiger files or OpenStreetMap). The output of this step is the road network information of the selected area, based on the selected road network data source. This is done through an abstract function, called `ExtractRoadNetwork`, that exploits the underlying road network data source to: (a) prune all information that are outside the selected rectangular spatial area, and (b) prune the non road network information from the selected rectangular area. We do so because each road network data source provides data in different formats; for example, US Tiger Files are stored in a binary format with extra information about zip codes, rivers, demographics, etc, while OpenStreetMap stores data in an XML format with extra information about buildings, parks, traffic lights, etc.
2. *Step 2. Preparing Standard Output.* The input to this step is: (a) The road network information of the selected rectangular spatial area, i.e., the output of the `ExtractRoadNetwork` abstract function, and (b) the road network data source. The output of this step is two standard text road network data files: `node.txt` and `edge.txt`, which contain the final set of nodes and edges in the selected spatial area, respectively. This is done through an abstract function, called `PrepareStandardOutput`, that is aware of the data format of the underlying data source and converts it to our standard output format.

We opt to transform the road network information into a standard text format, to make it portable to various traffic generators. Each traffic generator uses its own different input file format and perhaps different spatial coordinate system. For example, Brinkhoff generator uses binary files to store nodes and edges, whereas BerlinMOD expects one text file with two bracketed locations to represent a road segment. Moreover, Brinkhoff generator uses its own spatial coordinates system, where the *latitude* and *longitude* of a location are the offsets instead of absolute values.

An example of the standard format of our generated `node.txt` file is as follows:

```
Node_ID         Lat                 Lng
54956254019183  44.85923581362268   -92.989281234375
19567871005131  45.032414105220745  -93.2028993984375
27380416518383  44.99418225712112   -93.4431044765625
```

`Node_ID` is a unique identifier for the node on the road networks, whereas `Lat` and `Lng` are the latitude and longitude coordinates that represent the geographical location of the node, respectively.

Similarly, an example of our generated `edge.txt` file is as follows:

```
Edge_ID  Node_1          Node_2          Tags
0        33352568523324  33481417542144  highway
1        35667555893384  38510824242033  oneway
2        34881576878577  35839354585144
```

`Edge_ID` is a unique identifier for the edge on the road networks, whereas `Node_1` and `Node_2` are node IDs contained in the `node.txt` file. It means that the two nodes (`Node_1` and `Node_2`) are connected by the edge `Edge_ID`. `Tags` attach extra information to road edges which can be used by some generators.

4.2 Case Study 1: US Tiger File

US Topologically Integrated Geographic Encoding and Referencing (Tiger) Files [6] are published by US census bureau on a yearly basis to provide the most recent information about US geographical data, which include city boundaries, road networks, address information, water features, and many more. In MNTG, we focus on extracting the road network information from the `Road` directory of Tiger files.

A very unique feature of US Tiger Files is that the files are partitioned and organized based on US counties. In other words, all roads in a county are packed in a compressed file with a unique file identifier, e.g., `tl_2010_01001_roads.zip`, where `tl` means tiger line, `2010` indicates the publishing year of the data, `01001` is a unique identifier for the county (in this case is *Autauga, Alabama*), and `roads` represents the type of data.

The tricky part of the road network conversion with US tiger files is that a user may select a geographical area that covers multiple counties. This means that the road network converter needs to access road network data that spans multiple files. Hence, the most important step here is to find the corresponding counties covered by the user-selected area. To this end, we extract a minimum bounding box (`UpperLat`, `UpperLng` and `LowerLat`, `LowerLng`) for each US county. Then, we create a database table to store the bounding box corresponding to each county ID.

When a traffic generation request is received, we retrieve the road network data from US Tiger Files by first selecting all counties that overlap with the spatial region selected by the user. Then, we load the road network files for all overlapped counties and filter out the nodes and edges based on the user specified area. Finally, we write the qualified nodes and edges in the standard output format.

4.3 Case Study 2: OpenStreetMap

OpenStreetMap is a project that aims at digitizing geographical data for the whole world by providing geographical data that is free to use, distribute, and manipulate. Since it is maintained by volunteers, data in OpenStreetMap is updated frequently, whereas the data quality may not be as good as the data extracted from other commercial/official data sources. OpenStreetMap maintains a very large file, i.e., `Planet.osm`, to record all spatial objects in the whole world, e.g., road networks, buildings, rivers, etc. Essentially, `planet.osm` is one large XML (Extensible Markup Language) file that consists of the following four primitive data types:

- *Node*, that represents a spatial point by its latitude and longitude coordinates.
- *Way*, that consists of a sequence of *nodes* which, connected together, form a line or a polygon.
- *Relation*, that specifies the relation between *ways* and *nodes*. For example, two *ways* are connected together.
- *Tags*, that provides description for any of the other data types, *node*, *way*, or *relation*, using a key-value pair.

We carry out the extraction of OSM data in three phases, namely, *parsing*, *indexing* and *querying* where the first two phases are offline and the third phase is online.

In the parsing phase, the XML `planet.osm` file is processed to extract node and edge files for the whole world in the format discussed in Section 4.1. All nodes are extracted and stored in one `node` file. Ways are filtered on the fly based on the associated tags and only those associated to the road network are extracted. Each way is stored as a sequence of edges in one `edge` file. The indexing phase preprocesses the `node` and `edge` files (56GB and 98GB, respectively) to speedup range queries that selects a particular area. We initially tried to load them in a PostGIS database with an R-tree index but the loading process did not finish in a reasonable time (we terminated the process when it took more than a week). As an alternative solution, we used Spatial-Hadoop [30], a MapReduce framework for spatial data, to build an R-tree index in a cluster of 20 machines which took around four hours. The first step is to join the node and edge files to project coordinates of both ends of an edge, and then build an index over the edges based on their minimal bounding rectangles (MBRs). Once the R-tree index is constructed, it is extracted out of the cluster in the format of a *master* and *data* files where the master file stores the region occupied by each data file as an MBR while the data files store the data records. The `node-edge` joined file ended in 10,000 data files with an average file size of 12MB. The final phase is the querying phase in which range queries are processed on the R-tree to extract node and edge files in a particular area based on a user traffic request. First, the master file is examined to select the data files that need to be processed. Next, these data files are processed to generate the node and edge files that are then processed by the selected generator.

4.4 Extensibility with Other Road Network data Sources

As was pointed out in Figure 1, MNTG is extensible to support new road network data sources. Thanks to the modular design of the *road network converter*, extending MNTG with another road network data source is as simple as providing the contents of two abstract functions. Assume a service provider that has a new road network data source, termed *MyRoadNetwork*. To include this data into MNTG, we provide a template java file, where the service provider needs to fill the contents of: (a) the `ExtractRoadNetwork` abstract function which will basically select the information of the selected area form *MyRoadNetwork*, as discussed in Section 4.1, and (b) the `PrepareStandardOutput` abstract function that outputs the nodes and edges information of the selected area in the standard output format, as discussed in Section 4.1.

It is important to note that filling the contents of the abstract functions in the template does not really have to be by the service provider of the new data source. Instead, third parties or volunteers can provide this functionality. In other words, crowd sourcing can play a major role here in extending MNTG to support various road network data sources. We have started this by providing the abstraction of US Tiger files and OpenStreetMap, as described above in Sections 4.2 and 4.3, respectively. Yet, we call for the efforts of research community and volunteers to support more data sources within MNTG.

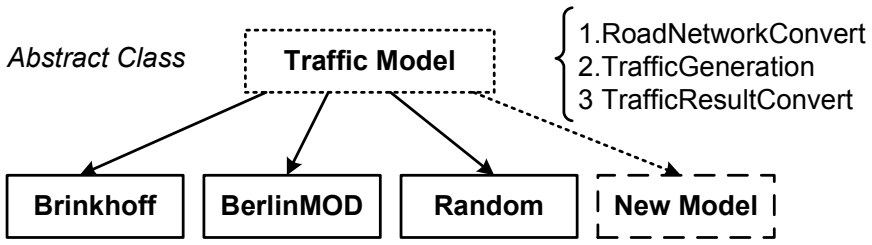

Fig. 2. Traffic Model Class.

5 Traffic Processor

The *Traffic Processor* in MNTG is responsible for generating the requested traffic data based on the selected traffic generator. It takes the extracted road network from the *road network converter* component (Section 4) and feeds it to the selected traffic generator. The challenge here is on how to accommodate the various input formats, parameters, and running environments for different traffic generators. To this end, the *Traffic Processor* component in MNTG provides an abstract way to accommodate various traffic generators. It currently includes two famous ones, Brinkhoff and BerlinMOD, however, its abstract design makes it highly extensible to support more traffic generators. In this section, we first discuss the main idea behind the traffic processor. Then, we provide three detailed case studies featuring Brinkhoff, BerlinMOD and random walk traffic generators. Finally, we discuss the extensibility of the traffic processor to support other traffic generators.

5.1 Main Idea

To generate the traffic data based on a particular traffic generator, MNTG basically aims to run the traffic generator as is. However, this is hindered by the fact that different traffic generators: (a) employ different execution methods and (b) require different configuration files and/or parameters. For example, Brinkhoff model is executed with a java jar file, while BerlinMOD runs with a script file. As the purpose of MNTG is to enclose various traffic generators, it builds a wrapper around each traffic generator to make them all look the same when it comes to receiving a traffic generation request and producing the final result.

The main idea is to create an abstract class `Traffic Model` in MNTG, as depicted in Figure 2. Then, all definitions and functions for each traffic generator has to be incorporated inside this abstract class. In general, there are four key data structures that should be inherited by all traffic generators in MNTG:

1. *Traffic Request ID*, as the traffic request identifier, which is automatically generated for each submitted request. It is used to link the input/output traffic data to the

corresponding traffic data requester and to send the traffic result as well as the status notifications to the submitting user.
2. *Traffic Model Name*, which is another identifier to indicate the type of the selected traffic generators, e.g., Brinkhoff or BerlinMOD.
3. *Traffic Generation Area*, which is the user selected rectangular area to generate traffic data in. The area is represented by two corner points of the form <*latitude*, *longitude*>.
4. *Traffic Generation Parameters*, which includes the parameters (e.g., number of moving objects and simulation time), specified by the users, which will be used for the traffic generation. The parameters may be specified differently for different traffic generators.

Additionally, there are three main abstract functions that need to be implemented for each traffic generator to be included in MNTG:

1. `RoadNetworkConvert`: this function converts the standard road network format received from the Road Network Converter (Section 4) to the specific format used by the traffic generator.
2. `TrafficGeneration`: this function produces the traffic data based on the various parameters specified by the user request. MNTG runs the traffic generator with its own scripts or commands.
3. `TrafficResultConvert`: this function converts the output of the traffic generation process into a standard simple output format. The main reason behind this function is that different traffic generators produce different formats of traffic data, while users may want to use the same program to analyze them.

An example of the standard output format of the traffic processor is as follows:

```
OID TS Type      Lat               Lng
0   0  newpoint  44.986362410452   -93.2982044219971
1   0  newpoint  44.998948892253   -93.1812858581543
2   0  newpoint  44.966607085432   -93.2727378845215
0   1  move      45.031348772862   -93.2991374040413
1   1  move      44.953949943361   -93.3676484298706
```

where `OID` is a unique identifier for the moving object. `Lat` and `Lng` are latitude and longitude coordinates that represent the spatial location of the object. `TS` represents the time unit at which object `OID` was at (`Lat`,`Lng`) spatial location. `Type` determines whether the generated point is a new object or an existing object that has just moved to a new location.

5.2 Case Study 1: Brinkhoff Model

Brinkhoff traffic generator is one of most widely used traffic generators [2] (cited 650+ per Google Scholar). The general idea behind Brinkhoff generator is to simulate the object movements from two random locations using the shortest path. To realize Brinkhoff

generator inside MNTG, we have implemented the three abstract functions (introduced in Section 5.1), as follows:

1. `RoadNetworkConvert`. In this function, we convert the output of the *Road Network Converter* into two binary files based on the descriptions in Brinkhoff documentation [1], and rename them as `request_ID.node` and `request_ID.edge`.
2. `TrafficGeneration`. In this function, we prepare Brinkhoff configuration file, i.e., `property.txt`, where we update the corresponding path for the input files (the two generated binary road network files) and the output path. Then, we assemble the command using the parameters specified by the user in this request. Finally, we make the following external call:

   ```
   java -classpath generator.jar generator2.DefaultDataGenerator RequestID
   ```

   where the only modification for the original generator is that it now takes the `RequestID`, and produces the traffic result accordingly.
3. `TrafficResultConvert`. In this function, we convert the traffic data produced by Brinkhoff generator into our standard output format. An example of the Brinkhoff output is as follows:

   ```
   Type     OID Seq Class TS X       Y       Speed  Next_X Next_Y
   newpoint 0   1   0     0  14839.0 10262.0 1093.0 14782  10765
   newpoint 1   1   2     0  26319.0 1430.0  922.9  26317  1260
   newpoint 2   1   0     0  11443.0 10983.0 1093.0 11431  15703
   ```

   where`Type` determines whether the point is a new object or an existing object. `OID` is a unique identifier for the moving object. `Seq` is the sequence number for the moving object, and `Class` determines the type of the moving object. `TS` represents the time unit during the simulation time. `X` and `Y` show the location of the object, as Brinkhoff employs a different coordinating system that uses the offsets to represent the location. `Speed` is the current moving speed of the object. `Next_X` and `Next_Y` are the locations for the node in the road networks, where the moving object will pass for the next movement.
   As a result, we write a program to: (1) extract only the `OID`, `Type`, `TS` from the original output, and (2) convert the `X` and `Y` to be the latitude and longitude coordinates. After that, MNTG is able to generate traffic with any road networks using Brinkhoff model.

5.3 Case Study 2: BerlinMOD

BerlinMOD is another very popular traffic generator [3], where it simulates human movements during the weekdays and weekends. Users can specify their work and home areas in the road networks, then the generator simulates the users movements based on two rules: (1) during the weekdays, a user leaves Home in the morning (at 8 a.m.+ *T1*), drives to Work, stays there until 4 p.m.+ *T2* in the afternoon, and then returns back

[1] http://iapg.jade-hs.de/personen/brinkhoff/generator/FormatNetworkFiles.pdf

Home, (2) during the weekends, a user has an 0.4 probability to do an additional trip which may have 1-3 intermediate stops and ends at home.

To run the BerlinMOD traffic generator, a user would need to set up a SECONDO database [4], and uses a set of script instructions to query it. To realize BerlinMOD generator inside MNTG, we have implemented the three abstract functions (introduced in Section 5.1), as follows:

1. `RoadNetworkConvert`. In this function, we read the standard road network files and transform it to the format used in BerlinMOD. Ultimately, we produce a data file named `street.data` with the following information:

```
(OBJECT streets ()
    (rel (tuple ((Vmax real)(geoData line))))
        ((50.0(
            ( -93.276029  45.035464  -93.275936  45.035877 )
            ( -93.275936  45.035877  -93.275764  45.037752 )
                        ....
```

   As a result, BerlinMOD requires us to represent the road segments with a pair of locations bounded by a set of brackets.
2. `TrafficGeneration`. In this function, we prepare the script based on the generation parameters specified by the user, i.e., `BerlinMOD_DataGenerator_RequestID.SEC`, to query the underlying SECONDO database. In MNTG, we prepare a generic script for BerlinMOD and replace its parameters based on the user's request. Then, we run the following command line to execute the BerlinMOD generator:
   `SecondoTTYNT -i BerlinMOD_DataGenerator_RequestID.SEC`
3. `TrafficResultConvert`. In this function, we build a program that converts the traffic data produced by BerlinMOD into a standard format. An example of the traffic data generated by BerlinMOD is as follows:

```
Mid Tid Tstart              Tend     Xstart   Ystart  Xend     Yend
1   2   2007-05-26 10:34:40 10:34:42 -93.1767 45.0449 -93.1767 45.0448
1   2   2007-05-26 10:34:42 10:34:44 -93.1767 45.0448 -93.1766 45.0446
1   2   2007-05-26 10:34:44 10:34:46 -93.1766 45.0446 -93.1765 45.0444
```

   where `Mid` is the unique identifer of the moving object, `Tid` is the trip identifer, `Tstart` and `Tend` represent the start and end timestamps for the record, while `Xstart, Ystart, Xend`, and `Yend` are the corresponding locations when the object starts and ends during that time period.
   As a result, we write a program to: (1) extract only the `Mid`, `Tid`, `Tstart` from the original output to identify the moving objects, and (2) convert the `Xstart` and `Ystart` to be the latitude and longitude. Then, MNTG is able to generate traffic with any road networks using BerlinMOD.

5.4 Cast Study 3: Random Generator

As a proof of the concept of generation model extensibility, we implement a random generator that generates random walks over the road network. The simplicity of the

model used in this generator allows it to handle requests with large areas and hundreds of thousands of objects in a reasonable time. In addition to the road map of the selected area, the random generator takes as input two user parameters, number of moving objects and total simulation time. The generator starts by assigning an initial position for each object by selecting a random node in the road network. At each time step, each object advances one step by selecting a random edge from the edges adjacent to current node. To avoid going back and forth between two nodes, the last visited node is stored and is removed from possible choices of next nodes. If an object cannot find a possible next node (i.e., the only next node is the last visited node) or if the next node falls off the grid, the object is removed from the map and a new object is placed in a new random position. This simulates the event of a vehicle ending its trip and a new vehicle starting a new trip. This also ensures that the total number of objects in the map is fixed at the user defined parameter. Although this generation model does not accurately simulate real life, it is very useful for generating huge traffic data is a very short time which allows end users to test the scalability of their systems.

5.5 Extensibility with Other Traffic Generators

As was pointed out in Figure 1, MNTG is extensible to support various traffic generators. Extending MNTG with another traffic generators is as simple as providing the contents of the three abstract functions, defined in Section 5.1. Assume that a traffic generator developer has invented a new traffic generator, termed *RandomGenerator*. To include the *RandomGenerator* into MNTG, we provide a template java file, where the traffic generator developer needs to fill the contents of the three abstract functions: `RoadNetworkConvert`, `TrafficGeneration`, and `TrafficResultConvert`, as described in Section 5.1.

Similar to extending MNTG for new data sources, filling the contents of the abstract functions of a new traffic generator may be done by third parties or volunteers. Again, crowd sourcing can play a major role here in extending MNTG to support new traffic generators. We envision that MNTG will act as a vehicle that gives existing and forthcoming traffic generators a boost to gain wide users community. Thus, it is to the benefit of the traffic generator developers and to the research community in large to incorporate new data generation tools within MNTG.

6 System Front-End

The system front-end provides a set of tools for users to generate and visualize their requested traffic data. As MNTG is deployed as a web service, the system front-end represents a web interface that users can access over the internet. The web interface is designed for simplicity where users may generate, download, and visualize traffic data with few interactive, rather intuitive, steps.

The system front-end consists of three main modules: (1) *Web interface*, which allows users to easily interact with MNTG in terms of submitting traffic generation requests (Section 6.1), (2) *Email Notifier*, which acknowledges the receipt of the traffic

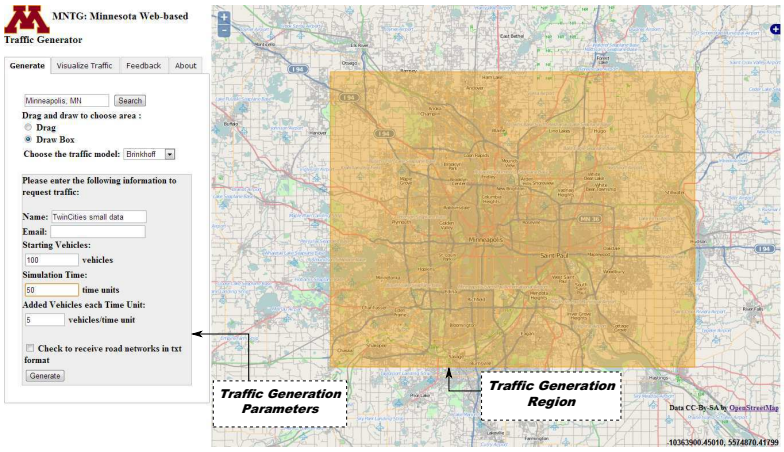

Fig. 3. MNTG Web GUI: Traffic Generation

request as well as notifies the user back when the request is finished with links to download and visualize the generated data (Section 6.2), and (3) *Download & Visualization tools*, which allows the user to download its traffic data in a plain text format and/or visualize the generated data in an OpenLayers map interface (Section 6.3).

6.1 Web interface

Figure 3 depicts MNTG web interface. To generate road network traffic data, a user would perform the following four easy steps:

1. Either drag/zoom the map or write an address in the search field to get the surroundings of the geographical area of interest.
2. Draw a rectangle around the area that you want to generate traffic within. This is done by two left mouse clicks for rectangle corners.
3. From the drop down menu, select the traffic generator that you want to use as either *Brinkhoff* or *BerlinMOD* traffic generators.
4. Click on the *Generate* button, and enter the traffic simulation parameters.

6.2 Email Notifier

MNTG may take a while to process a traffic generation request for two main reasons: (1) Depending on the size of the submitted traffic generation request (e.g., large number of moving objects or large simulation time), the underlying traffic generator (e.g., Brinkhoff or BerlinMOD) may spend significant time in simulating the requested traffic parameters, (2) Even though MNTG employs a multi-threading paradigm where several traffic requests can be processed concurrently, the system may be overloaded when the number of concurrent requests is more than the number of available threads. In that

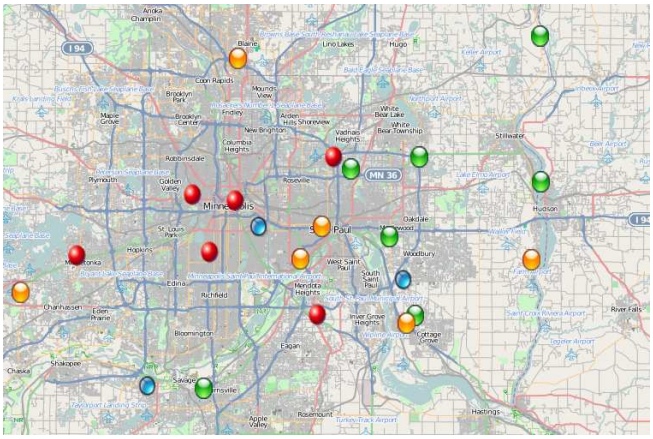

Fig. 4. MNTG Traffic Visualization

case, MNTG employs a waiting queue, where incoming requests have to be enqueued waiting for a system thread to be available.

To this end, MNTG *email notifier* has two functionalities: (1) when the user submits a traffic generation request, the email notifier sends an email back to the user acknowledging the receipt of the request, and (2) Once MNTG finishes processing the user's traffic generation request, the email notifier sends a notification message that contains two links; the first one is where the user can download the generated traffic data as a text file while the second one is where the user can visualize the generated traffic data on the map.

6.3 Download & Visualization Tools

As mentioned earlier, MNTG produces its output generated data in a uniform text format. Users may download the generated traffic data, including object ids, timestamps, latitude, and longitude coordinates and/or visualize the generated traffic data on the map using MNTG Map interface. MNTG stores the generated traffic data in the unified format mentioned above inside a MySQL database. Traffic visualization in OpenStreetMap is performed using OpenLayers v2.12 API for displaying overlays in HTML. The data is loaded via Javascript into the web page which then creates an overlay for each time stamp of the traffic results. Overlays are an OpenLayers concept and can consist of many different types of data, as shown in Figure 4. In this case, document fragments are created for each object at a time stamp, which is then added to the overlay for that time stamp. When the data is being animated, it simply consists of displaying the corresponding overlay to the time stamp and hiding the remaining overlays. Overlays are used instead of traditional markers because of the speed at which they can be loaded in comparison to the maps built-in markers.

7 Conclusion and Future Work

This paper has proposed Minnesota Traffic Generator (MNTG); an extensible web-based road network traffic generator. MNTG is basically a wrapper that can be built around existing traffic generators to make them easy-to-use, configure, and run for any arbitrary spatial road region. To generate traffic data, MNTG users just need to use its user-friendly web interface to specify an arbitrary spatial area on the map, select a traffic generator method as one of the two most highly used traffic generators, Brinkhoff and BerlinMOD, and submit the traffic generation request to the server. MNTG dedicated server receives and processes the submitted request, and emails the user back once the request is fulfilled. Users can then download their generated data and/or visualize it on MNTG map interface. MNTG is composed of three main components: (1) *Road Network Converter* that extracts the road network information of the spatial area of interest from either US Tiger files or OpenStreetMap, (2) *Traffic Processor* that executes the submitted request using the selected traffic generator on the extracted road network, and (3) *System Front-End*, that includes the web interface, email notifier, and download/visualasion tools for the traffic result. MNTG is highly extensible in two frontiers: (1) It can be easily extended to support various traffic generators, beyond Brinkhoff and BerlinMOD, by defining three abstract functions for each new generator, and (2) It can be easily extended to support various road network sources, beyond US Tiger files and OpenStreetMap, by defining two abstract functions for each new data source.

MNTG is still an undergoing project in data management lab at the University of Minnesota. Its first release is already available for a public use at $http : //mntg.cs.umn.edu$, where it has received and fulfilled over 1000 traffic generation requests since its release. Future work of MNTG includes: (a) supporting more traffic generators beyond the two we have for now, Brinkhoff and BerlinMOD, and (b) supporting more new data sources beyond US Tiger files and OpenStreetMap. A distinguishing feature in MNTG is that its future plans can be fulfilled via crowd sourcing, where interested developers and researchers world wide can enrich the infrastructure of MNTG by their contributions of new traffic generators and data sources. Plug-in functions are available for that purpose. With the increase of volume for traffic generation requests, we plan to move our server to a more powerful server machine with GPU cards to support large-volume traffic visualization.

References

1. Y. Zheng, Y. Chen, X. Xie, and W.-Y. Ma, "GeoLife2.0: A Location-Based Social Networking Service," in *MDM*, 2009, pp. 357–358.
2. T. Brinkhoff, "A Framework for Generating Network-based Moving Objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.
3. C. Düntgen, T. Behr, and R. H. Güting, "BerlinMOD: a Benchmark for Moving Object Databases," *VLDB Journal*, vol. 18, no. 6, pp. 1335–1368, 2009.
4. R. H. Güting, T. Behr, and C. Düntgen, "Secondo: A platform for moving objects database research and for publishing and integrating research implementations," *IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 56–63, 2010.
5. "OpenStreetMaps," http://www.openstreetmap.org/.

6. "US TIGER LINES," http://www.census.gov/geo/maps-data/data/tiger-line.html.
7. D. Pfoser and Y. Theodoridis, "Generating Semantics-based Trajectories of Moving Objects," *Computers, Environment and Urban Systems*, vol. 27, no. 3, pp. 243–263, 2003.
8. J.-M. Saglio and J. Moreira, "Oporto: A realistic scenario generator for moving objects," *GeoInformatica*, vol. 5, no. 1, pp. 71–93, 2001.
9. Y. Theodoridis, J. R. O. Silva, and M. A. Nascimento, "On the Generation of Spatiotemporal Datasets," in *Proceedings of the International Symposium on Advances in Spatial Databases, SSD*. Springer, 1999, pp. 147–164.
10. T. Tzouramanis, M. Vassilakopoulos, and Y. Manolopoulos, "On the Generation of Time-Evolving Regional Data," *GeoInformatica*, vol. 6, no. 3, pp. 207–231, 2002.
11. G. Gidófalvi and T. B. Pedersen, "ST-ACTS: A Spatio-temporal Activity Simulator," in *GIS*, 2006, pp. 155–162.
12. H. Hu and D. L. Lee, "GAMMA: A Framework for Moving Object Simulation," in *SSTD*, 2005, pp. 37–54.
13. D. Krajzewicz, G. Hertkorn, C. Rössel, and P. Wagner, "SUMO (Simulation of Urban MObility): An Open-Source Traffic Simulation," in *Proceedings of the 4th Middle East Symposium on Simulation and Modelling*, 2002, pp. 183–187.
14. "SMARTEST: Simulation Modelling Applied to Road Transport European Scheme Tests," http://www.its.leeds.ac.uk/projects/smartest/.
15. J. Xu and R. H. Güting, "MWGen: A Mini World Generator," in *MDM*, 2012, pp. 258–267.
16. H.-P. Tsai, D.-N. Yang, and M.-S. Chen, "Mining Group Movement Patterns for Tracking Moving Objects Efficiently," *IEEE TKDE*, vol. 23, no. 2, pp. 266–281, 2011.
17. H. Jeung, Q. Liu, H. T. Shen, and X. Zhou, "A Hybrid Prediction Model for Moving Objects," in *ICDE*, 2008, pp. 70–79.
18. M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases," in *SIGMOD*, 2004, pp. 623–634.
19. W. Wu, W. Guo, and K.-L. Tan, "Distributed Processing of Moving K-Nearest-Neighbor Query on Moving Objects," in *ICDE*, 2007, pp. 1116–1125.
20. M. F. Mokbel and W. G. Aref, "SOLE: Scalable On-line Execution of Continuous Queries on Spatio-temporal Data Sreams," *VLDB Journal*, vol. 17, no. 5, pp. 971–995, 2008.
21. B. S. E. Chung, W.-C. Lee, and A. L. P. Chen, "Processing Probabilistic Spatio-temporal Range Queries Over Moving Objects with Uncertainty," in *EDBT*, 2009, pp. 60–71.
22. H. Hu, J. Xu, and D. L. Lee, "PAM: An Efficient and Privacy-Aware Monitoring Framework for Continuously Moving Objects," *IEEE TKDE*, vol. 22, no. 3, pp. 404–419, 2010.
23. S. Chen, C. S. Jensen, and D. Lin, "A Benchmark for Evaluating Moving Object Indexes," *VLDB Journal*, vol. 1, no. 2, pp. 1574–1585, 2008.
24. A. H. F. Laender, K. A. V. Borges, J. C. P. Carvalho, C. B. Medeiros, A. S. da Silva, and C. A. Davis, "Integrating Web Data and Geographic Knowledge into Spatial Databases," in *Spatial Databases*, 2005, pp. 23–47.
25. C. Shen, Y. Huang, and J. W. Powell, "The Design of a Benchmark for Geo-stream Management Systems," in *GIS*, 2011, pp. 409–412.
26. T. Tzouramanis, "Benchmarking and Data Generation in Moving Objects Databases," in *Encyclopedia of Database Technologies and Applications*, 2005, pp. 23–28.
27. J. Xu and R. H. Güting, "GMOBench: A Benchmark for Generic Moving Objects," in *GIS*, 2012, pp. 410–413.
28. R. H. Güting, V. T. de Almeida, and Z. Ding, "Modeling and Querying Moving Objects in Networks," *VLDB Journal*, vol. 15, no. 2, pp. 165–190, 2006.
29. M. Vazirgiannis and O. Wolfson, "A Spatiotemporal Model and Language for Moving Objects on Road Networks," in *SSTD*, 2001, pp. 20–35.
30. A. Eldayw and M. F. Mokbel, "A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data," in *VLDB*, 2013.