Towards a Flexible and Scalable Fleet Management Service

Alexandros Efentakis
Research Center "Athena"
Artemidos 6 Marousi 15125, Greece
efentakis@imis.athena-innovation.gr

Sotiris Brakatsoulas
Research Center "Athena"
Artemidos 6 Marousi 15125, Greece
mprakats@ceid.upatras.gr

Nikos Grivas
Research Center "Athena"
Artemidos 6 Marousi 15125, Greece
grivas@imis.athena-innovation.gr

Giorgos Lamprianidis
Research Center "Athena"
Artemidos 6 Marousi 15125, Greece
glampr@imis.athena-innovation.gr

Kostas Patroumpas
School of Electrical and Computer Engineering
National Technical University of Athens, Greece
kpatro@dblab.ece.ntua.gr

Dieter Pfoser*
Department of Geography and GeoInformation Science
George Mason University
4400 University Drive, MS 6C3
Fairfax VA 22030-4444
dpfoser@gmu.edu

ABSTRACT

GPS positioning devices are becoming a commodity sensor platform with the emergence and popularity of smartphones. This abundance of GPS trajectories has fueled significant research around map-matching and related applications such as traffic assessment and prediction. Unfortunately, this research has only been used in costly and complex fleet management solutions. Our latest research endeavor addresses this issue by presenting cost-effective solutions for adapting state-of-the-art research around map-matching and live traffic assessment in the context of fleet management applications. This paper showcases various research results wrapped in a single extensible fleet management platform.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Design

Keywords

Shortest-Path, Map-matching, Isochrones, FCD, Fleet Management

1. INTRODUCTION

GPS positioning devices are becoming a commodity sensor platform with the emergence and popularity of smartphones and portable tablets. This abundance of usually low-sampling-rate (e.g., one point every 1-5 minutes) GPS trajectories have lead to significant increase in research activities around map-matching, the process of aligning a sequence of observed user traces to the underlying road network graph. Still at this moment, practical uses of this research have only been considered in costly and complex fleet management applications. On a quite similar note, novel shortest-path (SP) algorithms (by relying in extensive preprocessing of the road network graph) may answer SP queries in continental networks in few μs. Unfortunately, those algorithms are not efficiently tuned for handling live traffic updates, such as those produced by the aforementioned map-matching algorithms.

Our latest effort in [16] tries to address these shortcomings by creating an efficient infrastructure for low-cost fleet management solutions. The core components of our system (referred hereafter as the *SimpleFleet service*) include (i) a collection mechanism for vehicle tracking data, i.e., Floating Car Data, (ii) a map-matching algorithm that relates the vehicle trajectories to an underlying road network and allows us to derive travel times in relation to the road network, (iii) an efficient data aggregation mechanism to derive speed profiles for the road network, (iv) a shortest-path algorithm that takes live traffic conditions and, hence, actual travel times into account and (v) a visualization platform to interact with the system and visualize traffic conditions based on traffic maps and isochrones.

The outline of this work is as follows. Specific scientific innovations and a description of available system services is presented in Section 2. Section 3 describes the SimpleFleet service architecture and implementation. Section 4 presents the Web interface used to access (a subset) of the implemented fleet management functionalities. Section 5 presents some performance numbers and discussing possible system loads. Finally, Section 6 gives conclusions and directions for future work.

2. SERVICES

Implementing a fleet management infrastructure requires a certain number of services, i.e., data collection and management methods as well as map-matching and shortest-path algorithms. What follows is a description of these services and the respective innovations that were needed for their efficient implementation.

2.1 Data Collection

Essentially, we are dealing with two data sources. One is the actual road network (graph), which in the our case is based on OpenStreetMap data. Since in our system we are dealing with specific geographic areas, we converted OSM data to a routable road network graph, of which we finally used its largest strongly connected component. Strong connectivity is a necessary requirement for the

*On leave from Research Center "Athena", Greece.

IWCTS '13, November 05-08 2013, Orlando, FL, USA
 ACM 978-1-4503-2527-1/13/11 $15.00
http://dx.doi.org/10.1145/2533828.2533835 .

map-matching and routing algorithms used in the following. The second dataset we have to collect is the actual vehicle tracking data. Here, we created an efficient mechanism for collecting and storing considerable amounts of Floating Car Data (FCD) from fleet vehicles. For each urban area covered by our system, we are typically dealing with 2,000-5,000 vehicles producing a data point (GPS position sample) every 60 -180s.

2.2 Travel Time Derivation

Aligning the collected GPS traces to the road network graph requires state-of-the-art map-matching (MM) algorithms. In our framework we use the Fréchet-distance-based curve matching algorithm of [2, 17] and the [11] implementation of the ST-matching algorithm [12]. Still, we had to significantly enhance both implementations to handle FCD streams. In our approach we divide the incoming FCD stream to five minutes intervals, in order to create small trajectories and then performed map-matching on those small subsequences to obtain partial paths and travel time information.

Then we had to aggregate map-matching results per edge for the same interval (5min) to provide live traffic assessment information. Map-matching results were also aggregated on a monthly basis per edge, weekday, hour and quarter-of-an-hour to build historic speed profiles for providing traffic information for areas where no live traffic data is available.

2.3 Shortest-Path Computation

The combination of live traffic and speed profiles is used to provide dynamic shortest-path (SP) computation. We developed an optimized version of the unidirectional "Eager Dynamic" [3] variant of the ALT (A^* + Landmarks + Triangle equality) [9] algorithm. We significantly improved the preprocessing time and SP query performance of the specific algorithm and make it thus suitable for a dynamic navigation scenario, i.e., considering live traffic updates in shortest-path results. We focused our attention on the ALT algorithm, since (i) it is very robust with respect to the metric used [9], (ii) it requires no path unpacking (producing the actual road network path of the shortest route), and (iii) its storage requirements and auxiliary data structure size depend solely on the number of landmarks (and not on the utilized metric). We avoided using hierarchical approaches, such as Contraction Hierarchies [8], because the required shortcut edges need to be re-computed at every batched edge-weight update. Hence creating and dropping shortcuts every five minutes was an additional overhead we needed to avoid. Moreover, the use of shortcuts makes path-unpacking slower, since in our case the full path needs to be returned to the user.

By proposing a novel and efficient landmark selection strategy and expanding several optimization strategies of [6], [7], we managed *to lower the preprocessing time of the ALT algorithm from several minutes* [3] *to a few seconds*. Moreover, we have also improved its *query phase* and *tripled its unidirectional performance* while also *improving bidirectional performance by 44%*. Although we did not alter the actual algorithm, our engineering efforts significantly broadened ALT's entire scope, since (i) its preprocessing is now fast enough for supporting dynamic road networks with frequent traffic updates and (ii) the algorithm is now fast enough to support real-time SP queries for global scale mapping services. For more details on our optimizations for ALT refer to [5].

Although there were some previous shortest-path research works [13] also using OpenStreetMap data for creating the road network graph, they did not have access to live traffic information, as our work does. This is a huge advantage of our approach, since similar to major services like Google and Bing Maps, we are able to suggest the best route according to current live traffic conditions.

2.4 Isochrone Computation

Another important focus of our work was to provide novel and innovative ways to visualize and represent traffic situation in urban areas. A crucial tool for this effort is the concept of isochrones. Isochrones are defined in [1] as the "*set of all points from which a specific point of interest is reachable within a given time span*". Although our service is mainly aimed towards fleet management, isochrones are equally important within other contexts, such as geomarketing (e.g. where a new franchise store should be opened) or urban planning (e.g. where a hospital should be built to accommodate uncovered city areas). Another paper relative to our work was [14], since it was the first to claim that the whole spatial area covered by an isochrone is important and introduced the "Edges' Hull" algorithm which creates a single area composed of the outermost edges of the isochrone network. This approach offers increased accuracy in comparison to previous, typical convex hull approaches.

Although isochrones have been used before in public transport and walking combinations [1], to the best of our knowledge we are the first to combine the state-of-the-art isochrone computation of [14] with live traffic data, in addition to providing this real-time system showcasing our results.

Moreover, in our recent work of [4] we have already combined the acquired live traffic isochrone computation with demographic data to demonstrate the impact of traffic fluctuations in a geomarketing context. There, one can see in a quantitative way that the influence of live traffic information is considerably important, especially in heavy traffic conditions. Hence, the live traffic isochrones introduced for the first time here, may be extremely useful for many, seemingly unrelated, scientific areas.

3. SYSTEM IMPLEMENTATION

The product of the aforementioned processes/innovations is the *SimpleFleet service* which consists of several components - virtual machines (VMs) that interconnect and cooperate (Fig. 1). The various components are described in the following.

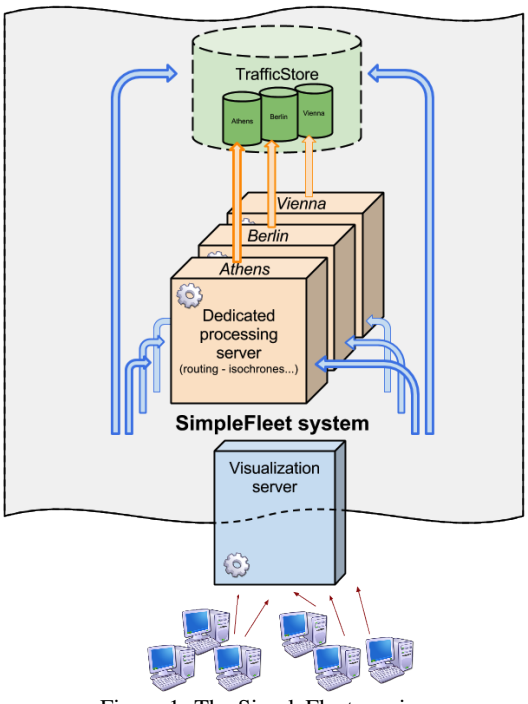

Figure 1: The SimpleFleet service

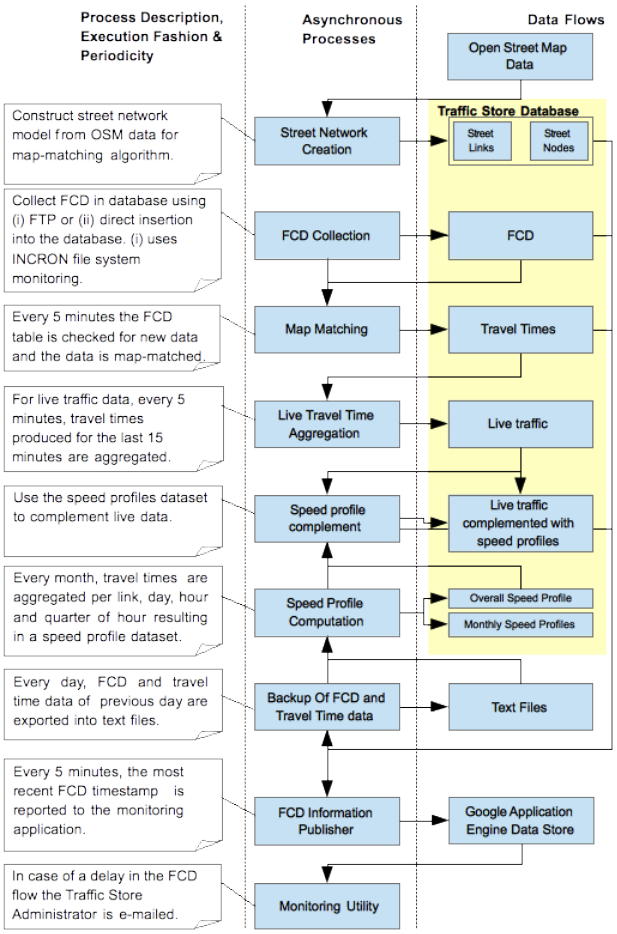

Figure 2: TrafficStore functionality

All SimpleFleet system VMs are hosted in õkeanos [15] IaaS (Infrastructure as a Service) platform of the Greek Research and Technology Network. õkeanos is a cloud service comparable to Amazon Elastic Compute Cloud (EC2) making a potential migration of our entire architecture to such a commercial service simple and seamless.

3.1 TrafficStore

The *TrafficStore* is the major key component of the SimpleFleet system for storing all available input and output data. Therefore all additional services are built on top of it. TrafficStore is a complete, integrated data management system for the traffic data pool. It is implemented using a PostgreSQL / PostGIS DBMS. The data management functionality includes FCD collection, map-matching, computation of live traffic assessment and speed profiles. A separate TrafficStore instance is set up for each urban area covered by our system. Currently our service covers the cities of Athens, Berlin and Vienna. Figure 2 gives an overview of the processes running in the TrafficStore and their interactions.

3.2 Dedicated processing server

Dedicated processing servers are used to handle computations in relation to user requests. Typically one server (VM) is used per respective area. Such requests currently refer to (i) live traffic shortest-path computation and (ii) calculation of isochrones (areas on the map that can be accessed within a given timespan). As shown in Figure 1, each processing server communicates only with

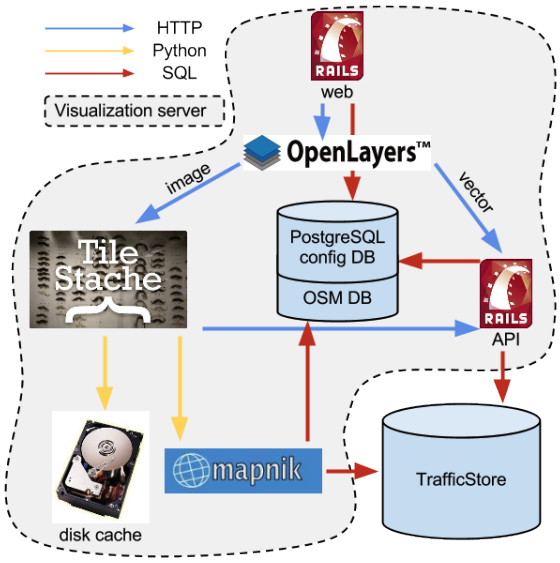

Figure 3: The visualization server's components

the TrafficStore repository of its respective city. This was a deliberate choice, for ensuring maximum efficiency, isolation and scalability. The services are accessed using a RESTful HTTP API utilizing an Apache Tomcat server that can efficiently forward the requests to optimized Java algorithms that typically respond in less than 50ms, even for up to 500 concurrent users (see Sec. 5).

3.3 Visualization server

The *visualization server* supports the interactive Web front-end of the system (Sec. 4). The underlying architecture exposes most of its visualization functionality through APIs, allowing the service to be also used by third-party Web applications.

The online interface is powered by a Ruby-on-Rails (RoR) application and served via Apache through the Phusion Passenger library. This ROR application generates the main page that contains the map, as well as the administration panel that is used to easily monitor and alter configuration settings. All configuration settings (data sources, styles) for supported cities and layers are stored in a local Postgres database along with a collection of OSM resources for generating the base map tiles.

The interactive map interface relies on the OpenLayers JavaScript framework as front-end and the TileStache map tile caching server as the back-end. TileStache is a lightweight web server that utilizes the mapnik framework for converting vector data (stored in PostGIS) to image tiles, adhering to certain style rules (colors, line widths) specified in a CSS-like format. Figure 3 depicts how these visualization server's components interconnect and cooperate.

There is one common visualization server for all cities/urban areas covered by our system. In this way there is only one point of entry to the entire SimpleFleet service, a fact that ensures increased security and easier logging.

3.4 Expanding System Scope

The modularity of the first two components, i.e. the *TrafficStore* and the *dedicated processing server* and the easy configuration of the *visualization server* makes our system very easily extensible to cover additional geographic areas. To do so, the first step would be to prepare the respective TrafficStore for the new city. This essentially means obtaining the OSM data for the new region and preprocess it so that it can be used by the implemented algorithms for

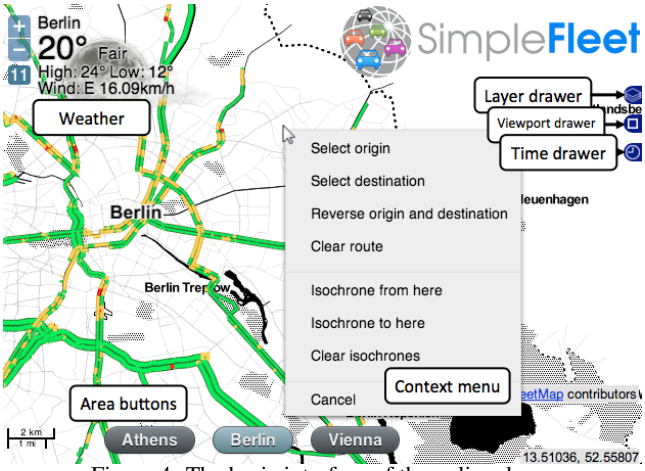

Figure 4: The basic interface of the online demo

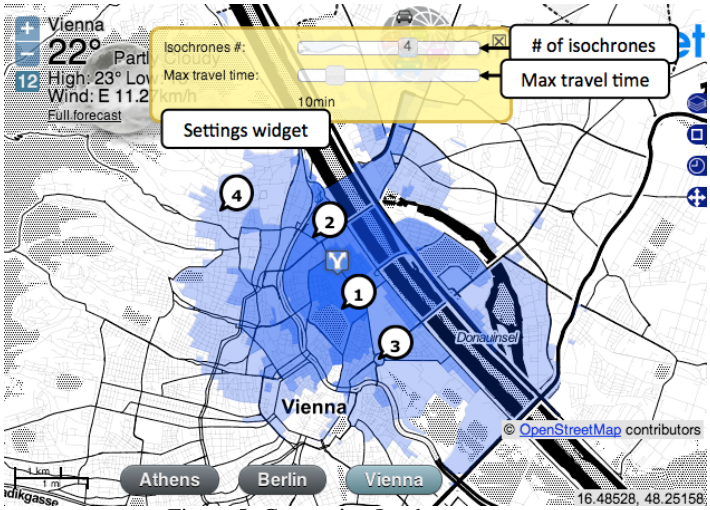

Figure 5: Computing Isochrones

map-matching, shortest-path and isochrone computation. The second step, involves cloning a dedicated processing server and configure it to access data from the new TrafficStore. The third and final step is to add the new area to the *visualization server*'s configuration. This involves two major tasks: Initially we have to prerender the map tiles for the new region, a task similar to the setup of the TrafficStore, in the sense that OSM data is being converted to map tile images and then stored in a cache for faster access. Then the new region may be added to the configuration via the available administration panel; the basic set-up involves setting the URL of the respective dedicated processing server and then choosing which layers would be available for the newly added area.

4. WEB APPLICATION

The Web front-end of our SimpleFleet service features an interactive "slippy map" interface that allows switching between the available geographic areas covered by the system. For each area the following data/services are available.

- Live-traffic map - visualization of traffic conditions, updated every 5*min*
- Speed profiles - visualization of traffic trends to complement live traffic assessment
- Traffic message channel alerts (TMCs available only for Berlin)
- Isochrones - based on live traffic
- Shortest-path routing - based on live traffic

In terms of *data*, the first two layers, i.e., live traffic and speed profiles are available as map tiles (png images), while information about the last three vector layers is available as JSON. From a *service* point of view, the first three layers are directly accessible from the TrafficStore, whereas isochrones and routing features are available from the respective dedicated processing servers (Sec. 3.2).

The three TrafficStore layers (live traffic, speed profiles, TMCs) and the background road-map layer (the choices here are (i) the default black & white theme, Google Maps layer, and OSM layer) may be independently activated by using the *Layer drawer* control located at the right of the map interface. The remaining two vector layers (routing directions and isochrones) may be activated by right-clicking anywhere on the map and selecting the appropriate action from the displayed context menu.

To minimize network time, all vector data from either isochrone or routing responses is returned in Google's encoded polyline format [10] that achieves 90% compression. GZIP compression is also

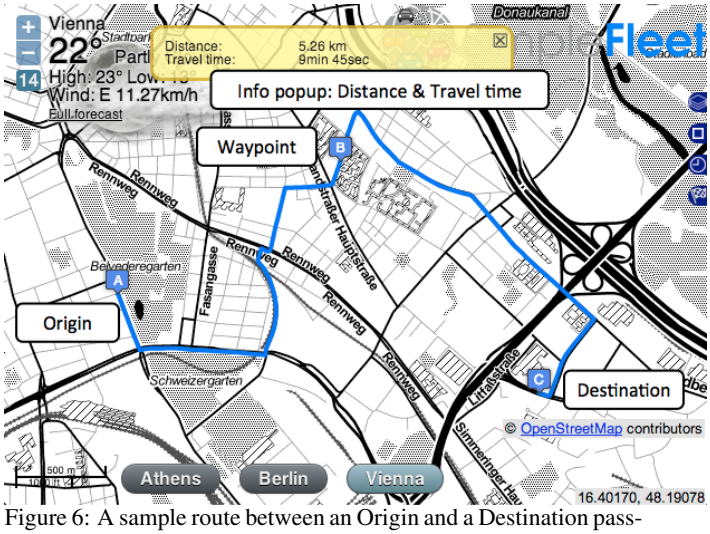

Figure 6: A sample route between an Origin and a Destination passing via a specified Waypoint

enabled, both on the the visualization and the processing servers, to further reduce network latency.

4.1 Isochrones

The Web interface facilitates isochrone computation "from" (and also "to") any location on the map (context menu). The user may either change the total traveled time (up to 30*min*) or the number of isochrone areas returned (up to six). For visualization purposes, each isochrone area has a different opacity with the smaller one being more opaque and the larger one being more transparent. Once the isochrones have been drawn on the map, the starting (or ending) marker may be dragged and dropped to a different location to request new isochrones to be drawn.

In our default setting, six isochrones are returned and the overall maximum travel time is set to 30 minutes. This means that each isochrone's maximum travel time is uniformly distributed in this duration, i.e., the first one covers the area reachable in 5*min*, the second one in 10*min*, etc. Figure 5 shows a tweaked example where the user has requested 4 isochrones and the maximum travel time for the largest one is set to 10*min*.

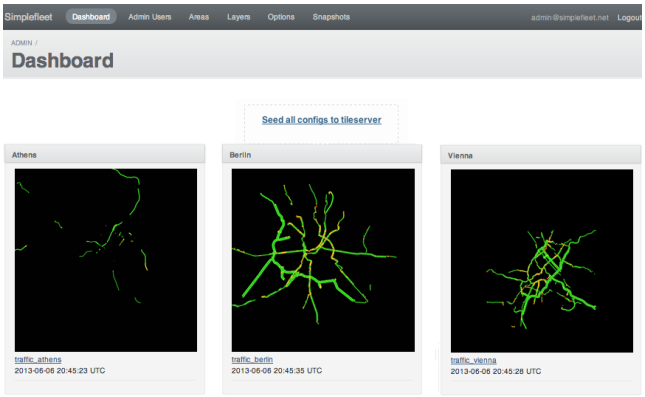

Figure 7: The administration panel

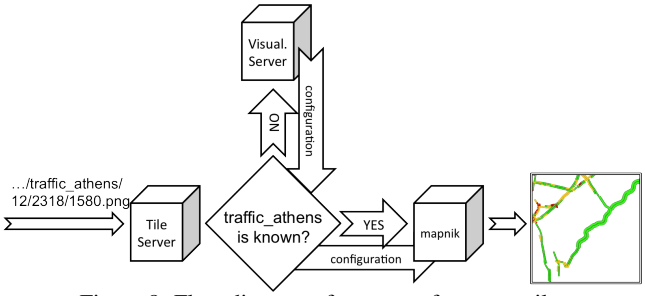

Figure 8: Flow diagram of a request for a map tile.

4.2 Routing

Similar to isochrone computation, routing requests may be computed between any two locations selected on the map (context menu). The server responds with a Google's encoded polyline representation of the calculated path, along with the travel time computed for this route and the total distance traversed. The travel info appears as a balloon tip on top of the map, while the actual route is drawn. Similar to massive online mapping services, the user may drag the Origin and Destination markers or add/delete intermediate points to a route. The origin marker (labeled A) and the destination marker (labeled B) are both draggable and thus dragging them automatically recomputes the path to match the altered location(s).

4.3 Traffic Messages

The TMC layer shows Traffic Message Channel (TMC) alerts and is only available for Berlin. TMC alerts are short informative messages which appear in the electronic road signs above major roads. The user may click on the TMC icons and retrieve the corresponding message.

4.4 Administration

To help administer the service, i.e., add new cities and map layers, an online administration panel (available only to super users via password authentication) was built (see Figure 7). The main two data components available in this panel are *areas* and *layers*. Each area corresponds to a separate city/region. Layers are used for storing configuration parameters for the different type of data we want to display in the map area of the online interface. The configurations settings are stored in the local PostgreSQL database of the visualization server.

The *admin panel* allows the super user to add new spatial areas or hide/remove existing areas from the list of available ones via the "Areas" section. An administrator may easily specify the URL of the *dedicated processing server* that corresponds to each area, so that all requests for that respective area will be accordingly routed to the proper server. Similarly, one may also specify which layers will be available for each area via the "Layers" configuration page. This page allows a user to add or remove new layers, to enable or disable existing layers or change a layer's behavior such as enabling or disabling caching for it. For image-based layers, the user has the ability to specify the data-source (a specific *TrafficStore* instance) along with the exact SQL query that will be used to fetch the results, as well as the specific styling rules that will be used for displaying the layer on the map. For example, for a traffic layer an example query would be to fetch all the road segments that are currently congested, i.e., have a travel time that indicates that vehicles are moving on it at 25% or less of the normal road speed.

Dynamic Configurations. The main advantage of having all the layer settings configurable via the *admin panel* is the ability to update the layer settings on demand with minimum effort. In a classic setup all the layer configuration such as the data-source to connect to, the exact query to fetch the data and the list of styling rules to apply when drawing the map tiles of that layer are saved in a static XML file on the disk. If a setting changes, one would typically have to log in to the server, edit and save the file, then restart the tile server (*TileStache*). In our setup we use the "Dynamic Layer" module provided by our tile server in order to read each layer's configuration from the *visualization server* the first time that a specific layer is requested. Once a "never-seen-before" layer is requested, an extra request is made to the *visualization server* to look up its configuration and export it to the tile server using a HTTP request. Figure 8 demonstrates the aforementioned scenario. The layer's configuration is then cached to boost performance. Since a typical single page request contains (on average) around 30 image tiles, it would be very inefficient to ask for the same configuration over and over again. A small patch has been applied to this "Dynamic Layer" module in order to allow this configuration to be updated on demand, i.e. expire the cache.

Overall, the user may change any setting of any layer and then use the link provided in the Dashboard section of the admin panel to seed this new configuration to the tile server, forcing it to expire any previously cached configuration and read the new settings from the *visualization server*. This approach is especially helpful when fine-tuning and testing new settings, e.g., trying out new map styles or queries.

Snapshots. For image-based layers (live traffic and historic speed profiles), the administration panel offers a "snapshot" functionality which may be configured to run automatically at predefined intervals or requested to run on demand. The "auto-snapshot" process, which runs every 15 minutes, creates a zoom 11 snapshot of the entire bounding box of the layer. The Snapshot screen lists all the available stored snapshots, tagged with the time they were taken. The user has the ability to filter the displayed snapshots by layer and within a specified time range. The resulting snapshots may additionally be viewed as a slide-show (movie) by clicking the "Slideshow" button on the upper right corner (Fig. 9) creating visualizations of traffic patterns.

5. PERFORMANCE

In order to test the SimpleFleet service's performance, we used the popular regression-test tool Apache JMeter. Apache JMeter is an open-source Java desktop application designed to stress-test functional behavior and simulate server load to analyze overall performance under different load types.

In our test scenario we experimented with 500 concurrent users, executing 60 mixed (for all cities) routing requests with a delay

Figure 9: Slideshow functionality of the administration panel

of 10*s* between requests to emulate realistic usage of the service, i.e., each user sends a request and waits for the response before proceeding to another request. Results showed that each of those 30,000 requests is answered in average time of 45*ms*. This clearly shows that *the minimal setting of our prototype architecture can efficiently handle a significant number of concurrent users*. Keep in mind that those recorded times are dominated by network latency, i.e., the time required for the actual response to be sent to the user. The actual time required for calculating a shortest-path is typically less than 5*ms*. After all, we aim at providing an infrastructure to service a limited number of fleet management companies and not on competing with well established global mapping services, such as Google or Bing Maps.

6. CONCLUSION

This work describes a unified fleet management system in terms of its available services, their implementation and an existing Web interface. The latter is used to access certain functionality and showcases example services such as traffic maps, shortest path and isochrone computation based on collected Floating Car Data. Our SimpleFleet system binds several web and server technologies together in order to provide a powerful and integrated platform that provides extensibility and scalability. We have also described its basic usage scenario, its administration panel, as well as its basic performance for a significant number of concurrent users.

Despite its, we believe, strong characteristics, this prototype application is still a work in progress. New services are being added to our infrastructure and the modularity of its architecture allows for the simple (comparatively) addition of new and probably more impressive features. Therefore we believe that it will play a crucial role in demonstrating SimpleFleet system's huge potential.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme "SimpleFleet" (`http://www.simplefleet.eu`, grant agreement No. FP7-ICT-2011-SME-DCL-296423).

7. REFERENCES

[1] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks. In *Proc. 16th ACM SIGSPATIAL GIS conf.*, GIS '08, pages 78:1–78:2, New York, NY, USA, 2008. ACM.

[2] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proc. 31st VLDB Conference*, pages 853–864, 2005.

[3] D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In *Proc. 6th Experimental algorithms conf.*, WEA'07, pages 52–65, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] A. Efentakis, N. Grivas, G. Lamprianidis, G. Magenschab, and D. Pfoser. Isochrones, Traffic and DEMOgraphics. In *Proc. 21st ACM SIGSPATIAL GIS conf.*, 2013. To appear.

[5] A. Efentakis and D. Pfoser. Optimizing Landmark-Based Routing and Preprocessing. In *Proc. 6th ACM SIGSPATIAL Computational Transportation Science Workshop, 2013*. To appear.

[6] A. Efentakis, D. Pfoser, and A. Voisard. Efficient data management in support of shortest-path computation. In *Proc. 4th ACM SIGSPATIAL Computational Transportation Science Workshop*, CTS '11, pages 28–33. ACM, 2011.

[7] A. Efentakis, D. Theodorakis, and D. Pfoser. Crowdsourcing computing resources for shortest-path computation. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 434–437, New York, NY, USA, 2012. ACM.

[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *Proc. 7th Experimental Algorithms conf.*, WEA'08, pages 319–333, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *16th ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, 2004.

[10] Google. Encoded Polyline Algorithm Format [Online]. `https://developers.google.com/maps/documentation/utilities/polylinealgorithm`, 2013.

[11] L. Kabrt. Travel Time Analysis. `http://code.google.com/p/traveltimeanalysis/source/browse`, 2010.

[12] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *Proc. 17th ACM SIGSPATIAL GIS conf.*, GIS '09, pages 352–361, New York, NY, USA, 2009. ACM.

[13] D. Luxen and C. Vetter. Real-time routing with openstreetmap data. In *Proc. 19th ACM SIGSPATIAL GIS conf.*, GIS '11, pages 513–516, New York, NY, USA, 2011. ACM.

[14] S. Marciuska and J. Gamper. Determining objects within isochrones in spatial network databases. In *Proc. 14th ADBIS conf.*, ADBIS'10, pages 392–405, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] Okeanos. ōkeanos IaaS [Online]. `https://okeanos.grnet.gr/home/`, 2013.

[16] SimpleFleet. Democratizing fleet management [online]. `http://www.simplefleet.eu`, 2013.

[17] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proc. 18th SSDBM conf.*, pages 379–388, 2006.