Archives of
Transport System Telematics

Volume 10

Issue 1

February 2017

Design and development of a road traffic redirection system

M. BAZAN, T. JANICZEK, K. HALAWA, R. DUDEK, Ł. RUDAWSKI
WROCŁAW UNIVERSITY OF SCIENCE AND TECHNOLOGY, Faculty of Electronics, ul. Janiszewskiego 11/17, 50-372 Wrocław, Poland
EMAIL: marek.bazan@pwr.edu.pl

ABSTRACT

Nowadays, the crucial issue of guidance systems based on a GPS signal is that they are not able to redirect road users, taking into account the current state of traffic (and the predicted state within the time of the travel) in the city. In this paper we present a three layer architecture of a computer system capable of redirecting users of an urban road system via routes with a lighter traffic load in order to reach their declared destination in the city. A basic layer is a multiprocessor calculation server running Dijkstra path search tasks, the middle layer - the one which is visible to the road user - is a replicable proxy server that collects route requests from road users. The third layer is a mobile application. The prototype of such a system was developed by the ArsNumerica Group. The crucial feature of the system is feedback from road users that allows us to adjust the whole Intelligent Transportation System in the city to changes in traffic flow at various road links introduced by the redirection process applied to many users. The performance test strategy to prove the efficiency of the architecture was carried out for the city of Wrocław.

KEYWORDS: traffic redirection, Dijkstra algorithm, fastest path finding algorithms, dynamic rerouting, guidance systems

1. Introduction

In recent years the ArsNumerica Group has implemented and tested three designs of a redirection system that would serve a road user as a guidance system that enables to drive from one point in the city to another via the fastest route (in the shortest time). The fastest of course does not mean the shortest one – it means the route calculated taking into account traffic load in particular places of the city – places that are controlled by an Intelligent Transportation System. The implementation of the prototype of the system that is presented in this paper enabled the establishment of scalability parameters so that the system extended with many calculation nodes becomes a high availability, high responsiveness system.

Figure 1 shows a sketch of the design of the system. It consists of three layers. The one visible to road users is a mobile phone application that enables logging on to the system and to enter a destination point. Afterwards the origin and the destination is sent to the server proxy that checks whether such a query has not been requested by another user recently. The check is done within the local cache. If the same origination - destination has been within the prescribed time it is accessible for users from the cache on the proxy server and no communication with the calculation server – the deepest layer of the system - is necessary. The calculation server obtains a task to process from the proxy server – i.e., to calculate the fastest route between the origin and the destination only if a task is not in a local cache of the proxy server. The calculation server is also responsible for downloading a description of the current traffic situation from an Intelligent Transportation System of the city.

In Poland, systems of a similar functionality are already available. To our knowledge however, all systems available are based on GPS speed estimation [8, 9]. Moreover such centralized systems as [10] are based on GPS data collection from users who possess an Android phone.

The prototype of one calculation node system presented in this paper enables us to establish conditions for a multiple calculation node system (currently being developed by the ArsNumerica Group) to possess a property of high scalability using low cost machines (11]).

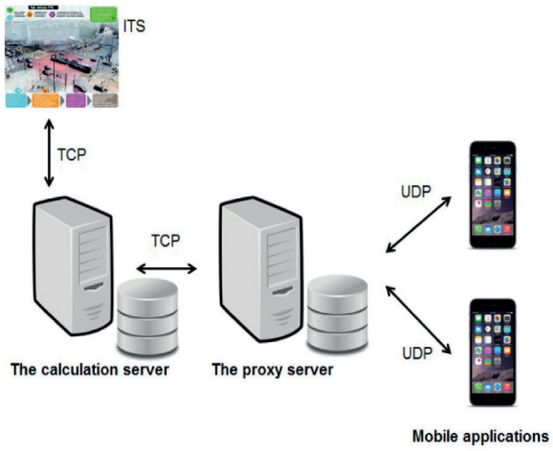

Fig. 1. A 3-layer architecture of the traffic redirection system. For the end user only a mobile application is visible. It communicates with a proxy server using UDP. The proxy server buffers a request from users unless it has not been calculated recently. The proxy server sends uncalculated requests to the calculation server, which performs a bi-directional two threaded Dijkstra algorithm or A* with a prediction of the traffic on the move in successive instances of time by methods described in [21] and [22] on the city map and sends back the results to the proxy server which in turn dispatches the response to an appropriate mobile application user [own study].

The remainder of the paper is organized as follows. In section 2 we describe the architecture of the system more deeply, together with message formats between its parts and the database schema used in the system. In section 3 we present the manner in which we tested the solution with the JMeter tool. Also in this section tests results are shown. Finally in section 4 we draw conclusions showing the scalability range of a multimode system.

2. The architecture of the system

The architecture of the system consists of three layers: a calculation server, a proxy server and multiple client applications.

2.1. The calculation server

The main aim of the calculation server that is the foundation of the whole system is twofold. First of all it provides the quickest route solutions for a given pair of nodes in the graph that represent a map of the urban area. Secondly, it maintains the current state of the weights of the graph based on connection with the intelligent transportation systems. Every 5 minutes it downloads a package with the current state from the city ITS and updates a graph by a procedure of swapping. Here the problem arises with a substantially changed situation on the road. How is the system supposed to react? After downloading the package containing new weights corresponding to the traffic load in the network from the ITS, all the tasks that are currently processed are canceled. The information on cancelation is also sent to the proxy server since some of the queries from the client are not sent from the proxy server to the calculation server but the response is retrieved from the local cache of the proxy server. This operation is performed by the thread that downloads ITS data and is done every 5 minutes.

In case of the ITS state update the cache of the proxy server is cleared. After updating the map graph by a thread on the server the tasks are recalculated. One may think of the UDP message broadcast to all waiting clients that recalculation due to traffic changes is taking place. The protocols used to connect to the ITS can be TCP. The protocol that is used to connect to the proxy server is also TCP. The format of the message that comes from the proxy server is shown in Figure 2, whereas the format of the response is shown in Figure 3. Each request in the calculation server is served in a separate service thread. The aim of a service thread of the calculation server is to effectively calculate the shortest path between start and end nodes in an urban area. The weights on the edges are inversely proportional to the travel speed or the congestion on the corresponding road section. The classical method to calculate the shortest path between nodes in a graph are Dijkstra and A* algorithms [2]. These algorithms do not require any time consuming preprocessing of a graph and thus fit to the architecture of the tested system. In [1] we tested various versions of multithreaded Dijkstra algorithms to find out that the most effective shortest path algorithm from the Dijkstra family is the two thread bi-directional Dijkstra algorithm with Fibbonacci heap used to calculate the closest node to the currently processed one. Each thread starts to search from the start and the end node until the balls around the start and the end node meet and to search for the closest node at each step two instances of a Fibbonacci heap are employed – one per thread. Similar evaluations of road networks can be found e.g., in [5].

Such a version of the Dijkstra algorithm was implemented in our calculation server. The results of our tests are in accordance with the ordering of the algorithms for finding the shortest path with respect to their computational complexity [3]. The bi-directional Dijkstra algorithm is asymptotically the fastest amongst algorithms without any preprocessing.

Algorithms with preprocessing, such as arc flags family algorithms [4] turn out to be problematic since the calculation of arc-flags may last longer than the time between the current road traffic state downloads, therefore their usage in real time may be limited. On the other hand, if we used such an algorithm as the arc-flags algorithm, the separation of a calculation server from a proxy server would be questionable since the calculation server would do only preprocessing common for all client applications. The bi-directional Dijkstra algorithm is used commercially in such software as MapQuest, Yahoo! Maps or Microsoft MapPoint (c.f. [6]).

2.2. The proxy server

The proxy server is a pass through server that collects two types of messages from mobile devices connecting from the city. The first type is a logon message in which the origination and destination nodes are contained. The requests originate from the devices via UDP protocol in the format presented in Fig. 2.

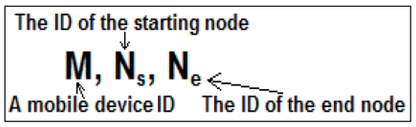

Fig. 2. A format of a message that is received by the calculation server from the proxy server. The first number is a unique identifier of the mobile device that the request has come from – perhaps e.g., the MAC number of the network card. The starting node number is an ID of the node in the map. A database of the GIS locations of nodes should reside on the calculation server and on mobile devices [own study].

The second type of message is a message containing a piece of information on the travel time between the two subsequent nodes that the vehicle containing the mobile device has just achieved. The format of this type of message is shown in Figure 3.

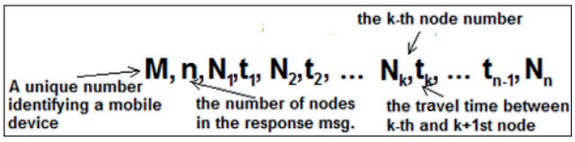

Fig. 3. A format of a message that encodes a response of the calculation server and is sent to the proxy server. The first number is a unique identifier of the mobile device that the request is going to return to. The second is the number of nodes that the optimal route consists of. N_k is an ID number of the k-th node (the vertex in the graph) in the optimal route. Hence, t_k is the travel time required to pass the connection between nodes with IDs N_{k-1} and N_k. It is important that in the graph there is a direct connection between N_{k-1} and N_k for k=2,…,N [own study].

After receiving a request from a road user's mobile device, the first thing the proxy server does is check whether the calculation for the same request has already been done by the calculation server since the last update of the traffic state from ITS or from the proxy server itself – that collects travel times from vehicles already driving via routes sent them from the system. If such a calculation has been performed already it is in the local cache of the proxy server.

The local cache is implemented on the SQL database. The database schema that implements a local cache required to maintain data in order to prevent a considerable number of requests to the calculation server is shown in Figure 5.

The local cache of the proxy server is also used to collect information from mobile devices on the travel time that the vehicle with it needed to pass between two successive nodes. Every prescribed number of passed nodes the mobile device sends a message to the proxy server with its registered travel time (see Fig. 4).

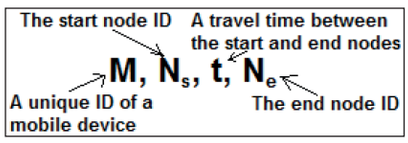

Fig. 4. A format of the message sent from a mobile device to the proxy server to update a current travel time on the link starting from N_s and ending at N_e that the vehicle with the device has just driven [own study].

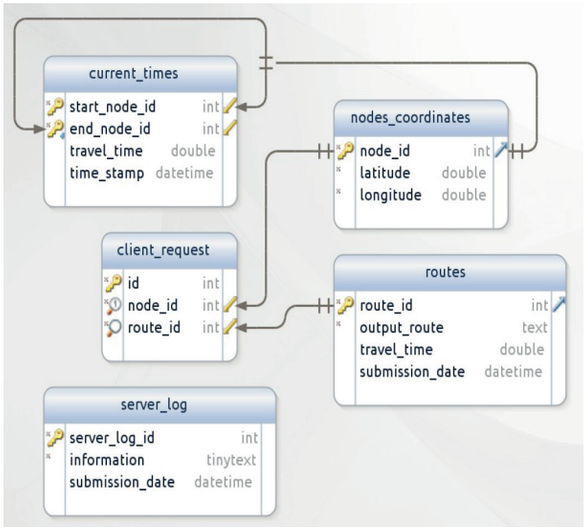

Fig. 5. The database schema of the proxy server. It is composed of four tables. Tables CLIENT_REQUEST, ROUTES, NODES_COORDINATES, SERVER_LOGS are used to handle responses from the calculation server whereas the table CURRENT_TIMES is used to store current travel times received from clients that follow the recommendation of the system. Currently our cache is implemented using MySQL database (see [17]). To speed it up still one can use one of the in-memory databases (e.g. SQLite – see e.g. [18], [19]), which would run for an adequate workload with no need to perform unnecessary I/O disk operations to update or retrieve data [own study].

In such a situation the table CURRENT_TIMES table is updated. Please note that there is no problem with concurrency since only the recent travel times record received is stored for a particular pair of nodes.

The content of the cache is sent to the calculation server every prescribed period of time. It is clear that the travel times contained within the cache are very reliable and thus they can overwrite times from the ITS, those which are not measured using license plate recognition but estimated from the fundamental diagram using only vehicle counts on the intersections (see [12] for a method).

2.3. The road user application

The task of the client application is twofold. First of all it is for sending a request to the proxy server in a format shown in Fig. 2. The N_s is a current spatial position of the device. The request is sent to the proxy server asynchronously with UDP. Once a response is received from the proxy server – which in turn got the response from its local cache or got it from the calculation server – the route is visualized. To do this the application has to have a city map installed. The city map has to be stored in a form of the database table – similar to the table NODES_COORDINATES at the proxy server. It is necessary for the client application to be able to find the closest node from which the route can be tracked with a traffic load. It of course cannot calculate naively the closest

node to the current start position, since the closest node may not give the possibility of reaching a destination point. So the client application has to have the ability of performing the Dijkstra shortest path calculation – of course the weights of the graph would be the length of the edges. As a simple database manager one can use one of the in-memory solutions for mobile systems (e.g., [19]).

The second role of the client application is to provide the calculation server – via proxy server - with the update of the current travel time on the edge the vehicle with the device has just passed, while traveling on the route received from the proxy server. This information is sent to the proxy server also via UDP in a format presented in Figure 5 every time a map node is passed while driving.

To provide the multi-platform application the Xamarin package can be used (c.f. [13]). The Xamarin development environment enables us to write a code in c# and to build the executable and deploy it not only on the Windows Phone platform but also on Android and iOS platform.

3. Tests of the system with the Apache JMeter

The implementation of the server side of the solution was done in c++ language. Both the calculation server, as well as the proxy server are multithreaded, enabling to serve multiple users at the same time. The tool that we used to measure how the solution is able to handle a load of many concurrent users is the JMeter created by Apache (see [7]).

The Apache JMeter is a testing tool that allows us to analyze a throughput of internet services and databases via such protocols as TCP/IP or FTP. With this tool one can measure response times, the influence of the volume of the requests handled, the amount of data sent and received. It is written in Java and therefore is fully portable and is an appropriate tool to measure the effectiveness of the implementation of a client-solution deployed on any platform.

The usage of the Apache JMeter in our application for the system testing relies on the creation of the parallel threads which send requests to the tested servers. Then the performance report is created. To generate requests from JMeter threads one has to define their message content and specify a number of messages that are to be sent to the tested server in a prescribed period of time. In this work the JMeter tool is used to test the availability of the solution, scalability and to get the estimation of how to make a load balancing of the system.

3.1. Tests description

To determine the performance of the entire system, each component needed to be tested separately. Based on the result of the tests an overview can be created showing which component is the slowest. Thereby an assessment of the number of processed requests at once is possible as well as a further optimization of the whole system.

All tests presented in this section were carried out on a low cost machine, i.e., a computer with AMD E2-3000M APU processor, with 2-cores, 2GHZ with 4GB RAM SODIMM DDR3. Such a choice was intentionally done to have an insight into the future behavior of low cost machines as part of the calculation cluster implementing the final version of the system (c.f. [1], [2]). During testing each application worked on an individual computer communicating wirelessly via a LAN network, so delays associated with the load of global network were not taken into account. A JMeter tool was tracking response times of each application and their CPU usage, based on the results the charts positioned below were created (Figure 6, Figure 7 and Figure 8).

3.2. Tests of the server proxy only

For the implementation of the proxy server the following scenarios were tested. Responsiveness of the proxy server itself (without the calculation server) was tested on the sequence of tests with the JMeter for

- 100 requests per 1 minute,
- 200 requests per 1 minute,
- 300 requests per 1 minute,
- 400 requests per 1 minute,
- 500 requests per 1 minute.

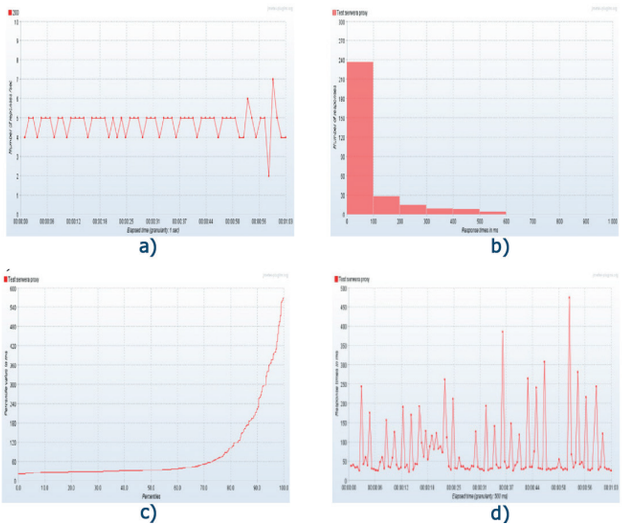

Fig. 6. 300 requests to the proxy server per minute: a) number of seconds needed to serve a request from the client applications, b) a histogram of the service time for requests, c) a percentile plot for responses, d) a scan of a period of 1 minute – a response time on Y coordinate [own study].

The performance of the proxy server was mainly affected by the number of requests received from the client application and size of database content. The greater number of data contained in the database resulted in a longer retrieval time, which caused that the handling of further incoming requests took longer to complete. However, the use of a database on a proxy server, buffering designated routes, is still much more efficient than performing additional calculations on a computing server.

3.3. Tests of the calculation server only

To test the calculation server only a sequence of tests consisting of considerably fewer requests were carried out. This is due to the fact that considerably fewer requests will come to the calculation server because of the caching on the proxy server. Therefore test scenarios covered

- 25 requests per 1 minute,
- 50 requests per 1 minute,
- 100 requests per 1 minute.

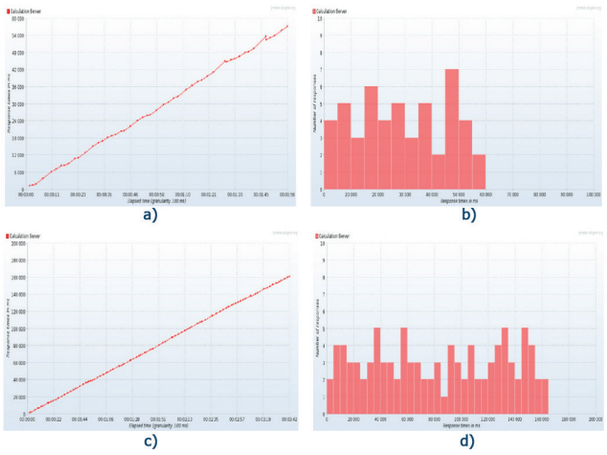

Fig. 7. a), b) 50s request per 1 minute to the calculation server; c), d) 100 requests per 1 minute to the calculation server. As one can see a) and c) show a saturation effect – the time needed to serve requests constantly increases [own study].

Figure 7 shows results for the cases of 50 and 100 requests per 1 minute. One can see that such an overload is too much, because the response time increases during the whole time. The saturation point for the calculation server was about 25 requests per 1 minute.

3.4. Tests of the whole system

The third test of the server side of the implemented system was to measure responsiveness of the calculation server when the request came from the client application through the proxy server. After the calculation of the route is completed the response with appointed route is transmitted to the proxy server, which sends it back to the mobile application. We performed the search for a level of cached requests with respect to the total number of requests served by the proxy server, so that the system manages to handle all requests and is not overloaded. As parameters of the optimization we used

- the percentage of the number of messages *a* that the proxy server handled using its cache,
- the number of requests to the proxy server *b*.

We carried out tests for boundary conditions i.e., sending requests to the computing server via proxy server and awaiting for a response in scenarios where

- 90% requests is served by the proxy server cache,
- 80% requests is served by the proxy server cache.

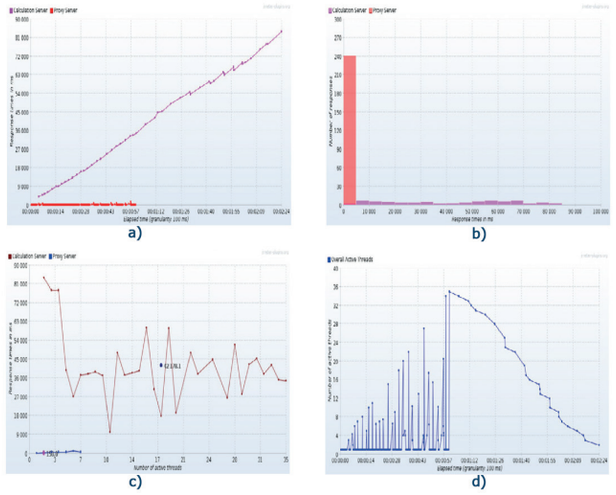

Fig. 8. All plots contain aggregated graphs for the proxy server as well as for the calculation server for 300 requests per minute where the system is slightly overloaded; 80% percent of these requests are served by the proxy server, the remaining requests go to calculation server a) response times over time, b) histogram of the response times c) response times per active request d) active requests over time [own study].

We found that the best performance is obtained for $a = 88\%$ to be served by the proxy server cache and not more that $b = 200$ requests per minute. Figure 8 shows plots with measurements from the JMeter for this configuration.

3. Conclusion

In this paper we presented a prototype of traffic redirection system that allows us to redirect vehicles via the fastest routes through the city. The system consists of the calculation server responsible for the shortest path calculations based on the current state of the traffic, the proxy server responsible for sending requests to the calculation server, or retrieve previously calculated results from its local cache and from mobile applications installed in vehicles of the end users. The two threaded bi-directional Dijkstra is used for finding the fastest route. We presented the results of the tests for responsiveness of the prototype carried out using the Apache JMeter. The prototype intentionally was tested on a low-cost machine. The tests show that the optimal load of the system is about 200 requests per minute to the proxy server and about 12% of all messages arriving to the proxy server is sent further to the calculation server to calculate a new route. The majority of the responses for requests i.e., 88% is retrieved from the local cache. These results show that if the calculation server consists of 20 replicated inexpensive nodes then assuming around 90% of repeated requests (that may be served by the cache of the proxy server) about 4000 vehicles (about 7% of the number of vehicles at one time during morning rush hour in Wrocław – about 56,000 – the value calculated using methods in [15] based on data published in [16]) can effectively use the system within the period of one

minute. It means that for a period of 10 minutes approximately 70% of all vehicles on roads in Wrocław in the morning rush hour could be handled. Another conclusion from these results is that we can now propose a load balancing strategy based on FIFO queues so that not all requests at once, from those that need to go, are forwarded to the calculation server but are queued on the proxy server until there is no calculation node with fewer than 3 parallel tasks running at a time (see Figure 8). The load balancing strategy however is a subject for further study.

Acknowledgements

The work in this paper was partially financed from grant 0401/0230/16.

Bibliography

[1] BAZAN M., et al.: Multithreaded enhancements of the Dijkstra algorithm for route optimization in urban networks, Archives on Transport Systems Telematics, Vol. 9, Issue 2, 2016, pp. 3-7.
[2] CORMEN, T. H., et. al.: Introduction to Algorithms. MIT Press, 2nd edition, 2001.
[3] SCHRIJVER, A.: Combinatorial Optimization — Polyhedra and Efficiency. Algorithms and Combinatorics 24. Springer. ISBN 3-540-20456-3, vol. A, sect.7.5b, p.103, 2004.
[4] BAUER, R.: On the Complexity of Partitioning Graphs for Arc-Flags, Journal of Graph Algorithms and Applications, vol. 17, no. 3, 2013, pp. 265–299.
[5] ZHAN, F.B., NOON, C.E.: Shortest Path Algorithms: An Evaluation using Real Road Networks. Transp. Sci., 32, 1998, pp. 65–73.
[6] KOZYNTSEV, A.N.: www14.informatik.tu-muenchen.de/lehre/2010SS/sarntal/07_kozyncev_slides.pdf, Facultat fur Informatic, TU Munchen, 2010, [date of access: 26.02.2016].
[7] ERINLE, B.: JMeter Cookbook, Packt, 2014.
[8] https://www.tomtom.com/ [date of access: 27.02.2017].
[9] http://www.automapa.pl/pl/start/ [date of access: 27.02.2017].
[10] http://maps.google.com [date of access: 27.02.2017].
[11] MAGED, M., et. al.: Scale-up x Scale-out: A Case Study using Nutch/Lucene. 2007 IEEE International Parallel and Distributed Processing Symposium. p. 1. doi:10.1109/IPDPS.2007.370631 [date of access: 26.03.2007].
[12] HELBING, D.: Derivation of a fundamental diagram for urban traffic flow, The European Physical Journal B, July 2009, Volume 70, Issue 2, pp 229-241.
[13] PETZOLD, Ch.: Creating Mobile Apps with Xamarin.Forms, Microsoft 2015.
[14] SZYMAŃSKI, A., et al.: Two methods of calculation of the origination destination matrix of an urban area, Raport W04/P-007/15, Wrocław University of Technology, 2015.
[15] Biuro inżynierii transportu, Pentor Research International, Kompleksowe Badania Ruchu - Wrocław 2010, (in Polish).
[16] WHITE, T.: Hadoop: The Definitive Guide: Storage and Analysis at Internet Scale, 4th Edition, O'Reilly, 2015.
[17] DUBOIS, P.: MySQL (5th Edition) (Developer's Library), Addison-Wesley, 2013.
[18] OWENS, M., ALEN, G.: The Definitive Guide to SQLite, Apress, 2010.
[19] DAS, S.: SQLite for Mobile Apps Simplified, Amazon, 2014.
[20] HALAWA, K., et al.: Road traffic predictions across major city intersections using multilayer perceptrons and data from multiple intersections located in various places, IET Intelligent Transport Systems 10 (7), 469-475, 2016.
[21] CISKOWSKI P., et al.: Estimation of travel time in the city based on intelligent transportation system traffic data with the use of neural networks, Dependability Engineering and Complex Systems, 85-95, 2016.