Scaling Real-Time Telematics Applications using Programmable Middleboxes: A Case Study in Traffic Prediction

Annie Chen*§, Navendu Jain†§, Angelo Perinola‡§, Tadeusz Pietraszek*§, Sean Rooney*¶, Paolo Scotton*¶
*IBM Research, Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland
Email: {ach,pie,sro,psc}@zurich.ibm.com
†Email: nav@cs.utexas.edu
‡Email: angelo.perniola@unicatt.it
§This work was done while visiting the IBM Zurich Research Laboratory.
¶Paolo Scotton and Sean Rooney are Research Staff Memebers at the IBM Research.

***Abstract*— Floating Car Data (FCD), i.e. traffic and in-car data collected and transmitted by moving cars, is an emerging telematics application providing a range of networked services to the road users. FCD is an example of a large sensor network, whose complexity increases with the number of session participants. Our work is examining how deploying application-specific functions in the network can help in scaling such systems.**

We have chosen to configure these functions on a dedicated device that is commonly called a middlebox. These middleboxes use programmable network processors in order to attain the required processing and forwarding speeds, while communicating with each other and back-end server using standard middleware components. We present our experience in using such a programmable middlebox in the scaling of a large telematics service, explaining how the middlebox fits into the end-to-end application.

Keywords: telematics, sensor networks, intelligent vehicles, traffic prediction, scalability, middleboxes

I. Introduction

Over the last 10 years, the rapid development of GSM and mobile networks has changed telematics, enabling many new applications. This is particularly significant to the automotive industry, since the cars can get almost uninterrupted access to the global network.

Initially processors were added to cars to monitor and control the cars' electronics. Subsequently, in-car computers were used for processing external information to assist the driver, e.g. the display of current vehicle location on a map. Combining the first two applications with global networking capabilities made it possible for the vehicle to communicate its state to external applications and in return receive information from them.

Urban traffic prediction is an example of a class of telematics applications that are, due to the amount of data and required real-time response, particularly challenging to build. Other applications belonging to this class are dynamic congestion-based charging and assisted urban parking spot identification. These applications may have to handle and process tens of thousands of packets per second.

Our work has been exploring the infrastructure needed to build such real-time telematics applications. We have implemented a realistic prediction model using standard middleware components and tested the number of vehicles it can support. We found that the scalability of the infrastructure could be dramatically increased by the addition of components capable of aggregating information from multiple cars before transmitting it to the back-end servers.

First, we describe the in-vehicle component of the application, then we show how the traffic prediction model was implemented on our back-end servers. Next we demonstrate how the additional aggregation components, implemented on telematic-specific middleboxes and placed between the cars and the back-end servers, allowed the servers to scale. Finally we present measurement results that compare the application behavior with and without these components.

II. Application Overview: Urban Traffic Prediction

The analysis of different urban traffic prediction algorithms is beyond the scope of this paper. For the purposes of our work, we used the model proposed by Peytchev et al. [1]. The model is queue based, considering traffic lights the main factor influencing the traffic. The city is modeled as a set of roads joined by intersections. The prediction algorithm takes the following as input: queue lengths at traffic lights and traffic light frequencies; traffic density[1] in the streets; the probability of which direction a vehicle will go at a junction. This allows the computation the predicted queue lengths and traffic densities an arbitrary number of traffic light cycles into the future. Therefore, assuming that queue lengths and traffic

[1]The traffic density of a street is defined as the ratio between the number of cars in the street and the length of the street.

densities can be measured accurately, the only unknown is the turn probability (i.e. in which direction a vehicle is going to turn) at the junctions.

Todays traffic information is based primarily on sensors built into the roads, such as inductive loops, which can detect whether there is a car above them [2]. These devices allow traffic managers to know whether the queue length is below or above a certain number of discrete thresholds. They are useful for optimizing traffic light behavior, for example allowing long queues to be drained. However, they provide only information about queues, but ignore traffic densities and turn probabilities and therefore are not sufficient to compute an accurate urban traffic prediction.

The set of protocols, services and data formats by which cars transmit information to a server is termed Floating Car Data (FCD) [3]. Each vehicle sends such information as position, current and average velocity, fog light status, etc. Depending on the implementation, transmission can be either periodical or event-triggered e.g. acceleration.

By gathering data from the cars themselves and combining it with data from inductive loop-enabled traffic lights there is sufficient information to predict future queue lengths and traffic densities. The system can also determine the average velocity on a given road from data gathered by vehicles themselves. Given certain traffic densities, this allows predictions to be made about traversal time along a road and consequently enables the shortest time path to be calculated.

III. In-Vehicle Components

Modern cars are equipped with a Controller Area Network (CAN) [4] through which the major vehicle components such as headlights, central lock and on-board computer, communicate. As the communication protocol is standardized, it is possible for third-party devices to connect to the bus and monitor (and possibly generate) messages [5]. We used a prototype of such a device — called TCET — manufactured by IBM for telematics field trials.

The application interface we run on TCET allows a driver to request the shortest path between two given points on the map. This best route is calculated using a time-based weighted algorithm (namely time-based Dijkstra, the most commonly used algorithm in router planners [6]) locally on the TCET using predicted road traversal times calculated on the back-end server which are broadcasted using e.g. Digital Audio Broadcast.

IV. Generating Data

We simulated the behavior of cars moving in a city to test the correctness of our prediction model and the scalability of our infrastructure. As no simulator corresponding to our needs was available, we used an enhanced version of a open source simulator called City Simulator [7] originally designed to test indexing algorithms for location databases.

Our simulation has a diurnal behavior where traffic enters the city during some initial period, remains there during a subsequent period and leaves it during a final period. In order to create local congestions and test the route-finder algorithm we added a notion of attractiveness to roads.

Our model of the behavior of traffic in a city is simple, it considers varying road attractiveness only with time and in isolation from other roads. It ignores traffic light adaptation to congestion, and uses a rather unrealistic model in which cars randomly drive around the city making decisions as to where to turn based on relative attractiveness. That said, it is sufficient for the purposes of testing the scalability of the infrastructure.

In our framework (see Figure 1), the simulator is run as an off-line process writing the data to the file. The packet generator takes the output file produced by the simulation and generates actual UDP packets containing information about the identity, location and velocity of cars and sends them to the server. In addition, information about traffic light state changes is also sent. The format of these packets is similar to that specified in [3].

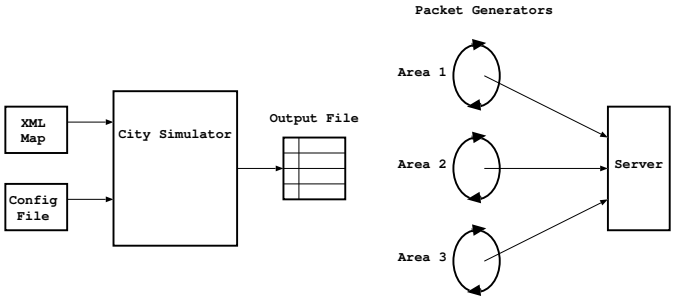

Fig. 1. Data generation.

V. Server

We use a two-tier server model in which a front-end interface receives the packets sent by the packet generators and does some initial processing on the data before making them available to the back-end application in which the actual prediction model is implemented.

Communication between the two server tiers takes place across a tuplespace implementation: TSpaces [8]. A tuple is a vector of typed fields, and a tuplespace is a globally shared memory space in which data is stored and retrieved as a set of tuples using well-defined coordination semantics.

A. Server Front-End

The front-end part of the server performs the following tasks:

- it identifies to which road a given location coordinate corresponds. For an arbitrarily complex map this operation can be computationally intensive. In order to optimize the road identification, we remember the last road on which a given vehicle had been identified and restrict our search to roads in its vicinity.
- it aggregates information as required by the prediction model. In particular it calculates the queue length at traffic lights and the traffic densities within roads, as well as

the average velocity on streets. The data related to a vehicle on a given road is retained until a traffic light state changing packet for that road is received, whereupon the relevant values are calculated and transmitted to the prediction model.
- it calculates statistics about the performance of the server, e.g. following the number of packets sent against the number actually received.
- in addition, some information about individual cars is also passed through, in order to allow the prediction model to determine the accuracy of its prediction and to adapt itself accordingly.

B. Server Back-End: Prediction Application

The prediction application, based on our extension of [1], takes the data processed by the front-end as input and produces a set of predicted traversal times for each road for some arbitrary number of traffic light cycle into the future. Due to iterative nature of the model, the results of one iteration can be applied in the next to obtain predictions for an arbitrary number of time cycles into the future — albeit with decreasing accuracy.

C. Discussion

The architecture as described was successively implemented. However, we identified a number of problems. These are, in increasing order of severity, as follows:

- the front-end server is proprietary, so we had to implement the dispatching mechanism ourselves. It would be preferable to use middleware that conforms with the servlet container specification [9], e.g Tomcat.
- the number of server requests is very high (e.g. 20,000 requests/s generated by 100,000 cars sending request every 5 seconds),
- the load the server has to handle increases as a linear function of the number of cars. At some arrival rate at the server the service rate cannot keep up and packets are lost. This causes the behavior of the application to degrade.

The first problem could be solved by using HTTP/TCP communication between vehicle and server, but this imposes significant overhead on the scarce capacity of the wireless hop, and the maintenance of so many TCP connections at the server is impractical. Furthermore, TCP retransmission is not useful in applications where a packet expires after a relatively short time, rendering its retransmission useless.

The second problem is that servers are easily overloaded with a high number of requests even if the equivalent constant-bit traffic is low (e.g. 2Mb/s in the example). We need to change the characteristics of events to use servers more efficiently.

The third problem are particularly severe as the infrastructure built to handle this traffic prediction application should also be usable by other real-time telematics applications. As we do not know the requirements of future applications, it would be preferable to have a solution in which the load at the server increases less than linearly with the number of cars.

These considerations led us to introduce an additional component between the cars and the back-end server.

VI. Server with Middlebox

A middlebox [10] is a device performing a service that requires application logic but which is executed in the network.

The aggregation function performed by the front-end of the server described in Section V-A can be applied to roads independently. In our implementation the aggregation function receives UDP packets, processes them as described above and produces asynchronous XML/RPC messages, which it then sends to the server. XML/RPC is an RPC protocol that uses HTTP as transport protocol and XML for data representation. The front-end server was reimplemented as a Servlet running within the WebSphere Servlet engine. Its function is simplified to receive only XML/RPC messages, process them and write them to tuplespace in a suitable format. The aggregation function combines many UDP packets into a single summary on the state of a road, so the total number of packets received by the server is significantly reduced. Moreover it allows standard components to be used to implement the server front-end.

By adding this aggregation function to a telematics middlebox running within the network and placing those middleboxes at locations close to where the data from the vehicle enters the fixed network, we reduce the distance over which the UDP traffic is carried and the total amount of packets the network is required to support.

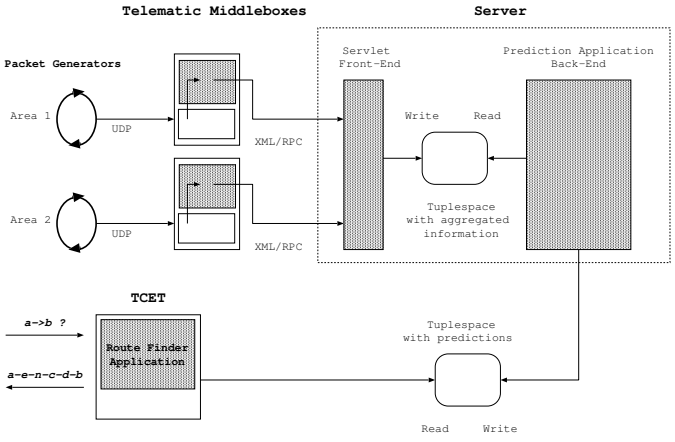

Fig. 2. Overview of the Architecture.

The vehicle continues to send UDP packets to the server, but these packets are intercepted by the middlebox based on the destination address. Our initial prototype simply used a Linux PC (kernel version 2.4.18) with the *iptables* kernel module installed to prototype the middlebox. In order to enhance performance we have instrumented the same function on a network processor [11]. Figure 2 shows the entire end-to-end architecture.

The middlebox is placed at the boundary between the wireless and fixed networks. For example, if GPRS is used then the middlebox is placed close to the device supporting the GPRS Gateway Support Node (GGSN) function or integrated into that device.

Whereas for the urban traffic prediction application the middlebox implements an aggregation function, for other telematic applications it can instrument other server load reducing functions, e.g. filtering, caching, application-specific forwarding. The support for different applications can coexist on the same middlebox simultaneously.

VII. Measurement Results

The architecture described in previous sections was tested in the following configuration. The city used in the simulation had 39 intersections with traffic lights and 74 bidirectional roads and 1000 moving cars, which corresponds to the central part of a moderately sized city. For the experiments we used configurations with two data generators.

The server ran a servlet environment (Tomcat and WebSphere Application Server) as well as a TSpaces server. The prediction software also ran on the server and exchanged data with the servlet engine using the tuplespace. In addition to application data, some statistics were also generated and stored in the tuplespace.

Two Linux routers were enhanced with the middlebox aggregation function. Our test network was arranged such that each of the data generators was connected to the server through one of these routers. The routers had Pentium II 233 MHz processors and 64 MB of RAM. The server also ran Linux but with a 1500 MHz processor and 1024 MB of RAM. The network used to interconnect the devices was a 100 Mbit Ethernet LAN.

A. Experiments and Results

In order to measure the accuracy of the prediction, we ran the packet generator at a speed such that no packets were lost at the server. We calculate the difference between the predicted queue length some time t in the future and the actual measured queue length at time t and express it as a relative error to the queue length. This is averaged for all the queues in the simulation in order to obtain the average queue length error for each prediction cycle.

If the attractivness of the roads does not vary for a large number of traffic light cycles then we achieve a 15-20% average queue error for 15 traffic light cycles into the future. The residual error is due to the fact that the measurements are discrete, probability-rounding errors, i.e. a fraction of cars cannot turn at a junction, as well as the fact that cars entering at boundaries cannot be anticipated by the model.

We measured the effect of the middlebox on the volume of traffic going into the server and observed a reduction by a factor of 10 in the number of packets sent to the server. However, there was hardly any reduction in terms of the total volume of data measured in bytes due to the rather large overhead of XML/RPC.

Our final measurement was the number of packets per second the server could handle with and without a middlebox. We simulated three scenarios: one without middlebox and the other two with different application servers: WebSphere Application Server v3.5 and Tomcat 4.1.

For each scenario, the simulation was run several times with different runtime parameters, changing the frequency of packet generation and the packet size. The goal was to determine the correlation between packet size and number of packets that can be handled.

We found that the size of the packet had little effect on the number of packets that can be handled. Obviously, the effect would be greater for large packets sent at a high ratio, but as in our case packet-processing time was much longer than network/kernel transmit time, this was negligible. We also note that both servlet engines had approximately the same performance, Tomcat being slightly faster.

Note that our conclusions about packet sizes also apply to requests sent to the server. Using an encoding more efficient than XML/RPC would have little effect on the performance as, in most cases, the number of server requests and number packets would remain the same. Considering these and the advantages of using open standards, we decided to continue using XML/RPC.

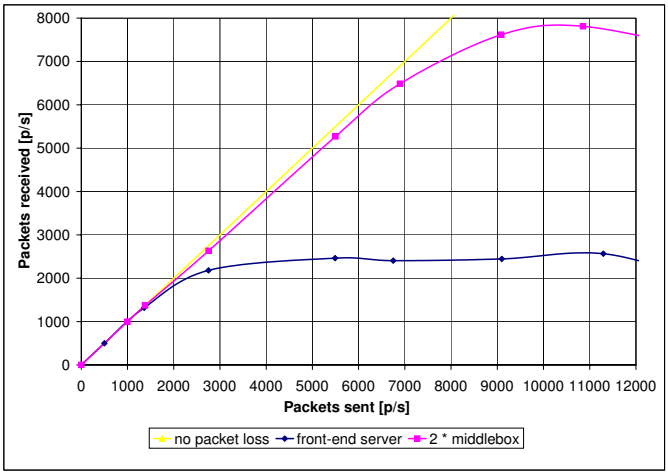

Fig. 3. System performance and packet loss.

Figure 3 shows how a server scales with and without middleboxes. The graph plots the number of UDP packets received as a function of the number of packets sent. The measurement was performed at the server when there is no middlebox and on the middleboxes themselves when they are used.

The experiment without middleboxes started to lose some packets at a load of less than 2,000 p/s and has losses of more than 50% at 5,000 p/s. The experiments with middleboxes show no significant loss until the load is higher than 6,000 p/s, and at 11,000 p/s the loss is about 25%. At the same rate the scenario without middleboxes has nearly 90% loss.

The performance of even a single middlebox with server proved to be better than a server alone. This result is surprising

as the introduction of an additional processing layer should slow the system down. Moreover, the middlebox was running on a much slower computer than the server used in the first scenario. The problem with the original solution was that the tuplespace operations were synchronous, which became the bottleneck of the system. In the middlebox solution, these operations were also synchronous, however the webserver calls were asynchronous. As the webserver has been designed and optimized to process a high number of requests simultaneously, it used many threads to optimize performance. This result could also have been achieved by redesigning the code of the original solution, which was beyond the scope of our project.

B. Discussion

The machines used in the test environment are much less powerful than those that would be used in a real system; therefore the absolute figures for load handled are not truly representative. Of more interest is the gain achieved by using the aggregation function on middleboxes. Note that these middleboxes were much slower than the server. The results showed that in the test environment the use of the middleboxes reduces the amount of packets the server has to handle by a factor of 10. Although the amount of data transferred is the same, better performance is observed as the bottleneck is the number of packets that can be handled, rather than the throughput.

It should also be stated that our prototype application contains no back traffic from the server to the client. For other applications, back traffic may be routed through the middlebox, which again reduces the server load. In addition, the middlebox does protocol transcoding, changing proprietary protocol based on UDP to a standard protocol based on HTTP, which can be handled by ready-to-use webservers and servlet environments. This means that it is possible to build scalable and efficient systems using standard and available components, which may greatly reduce development and testing time.

VIII. Related Work

Currently implemented traffic control systems are based primarily on infrastructures built into the roads. For example, SCOOT [2] developed in The Transport Research Laboratory uses inductive loops for vehicle detection and measuring queues. The central computer tries to optimize cycles and offsets the traffic lights to discharge long queues.

Another approach to distributed telematics applications is to use cars as both information gatherers and routers. An example of ad hoc vehicle-to-car communication based on IEEE 802.11 is Fleetnet [12] and [13]. There are no scalability issues in this approach as cars communicate directly with each other and data is not passed to a central server. The primary advantage of this approach is that the services available locally (i.e. provided by other cars) do not have to be processed by a central server. Urban traffic prediction, on the other hand, requires a global overview and uses historical data. FCD data needs to be coupled with data coming from other sources and therefore ad hoc vehicle-to-car communication is not a complete solution to this problem. There is still a need for middleware components to ensure scalability and support the volume of traffic.

The problems of deploying FCD on a larger scale on highways have been addressed in [14]. There also have been projects on urban FCD such as BERTRAM [15]. Scalability issues have not been clearly stated.

InternetCAR[16], [17] project develops the architecture needed to connect car to the Internet, but does not address the problems of infrastructure to enable the services. Investigating the problems of telematics applications for automotive industry and describing infrastructure needed, our work is complementary to InternetCAR.

IX. Acknowledgments

The authors gratefully acknowledge the help of Chris Giblin and Daniel Bauer in developing the infrastructure described here.

X. Conclusion

Urban traffic prediction with floating car data is one example of an emerging telematics applications, taking advantage of global networking capabilities. Improving the scheduling of a city's resources by exploiting vehicles' computational capabilities would potentially bring considerable social benefits, for example less time wasted in traffic jams and less pollution.

However, building the infrastructure to enable these applications is far from trivial, due to the large amount of data needed and short processing times required. We have described a prototype of an infrastructure to realize one such application. We have shown how intermediate telematic specific middleboxes can help in scaling this application as some of the data can be aggregated before being transmitted to the server, reducing both the amount of the data that has to be sent to the server and the amount of processing the server is required to perform.

References

[1] E. Peytchev, A. Bargiela, and R. Gessing, "A Predictive Macroscopic City Traffic Flows Simulation Model," in *In Proceedings of European Simulation Symposium ESS'96, Genoa, 2, pp. 38-42.*, 1996.

[2] P. Hunt, D. I. Robertson, R. Bretherton, and R. Winton, "Scoot - a traffic responsive method of co-ordinating signals , ttrl report 1014," Transport and Road Research Laboratory, Crowthrone, Berkshire U.K., Tech. Rep., 1991.

[3] ISO, "Transport information and control systems reference model architecture(s) for the tics sector, iso/tr 14813-1:1999," ISO, Standards of TC 204, Tech. Rep., 1999.

[4] Bosch, "Control area network specification, version 2.0," Robet Bosch GmbH, D-70422 Stuttgart, Tech. Rep., 1991.

[5] G. Leen and D. Hefferman, "Expanding Automotive Electronic Systems," *IEEE Computer*, vol. 35, no. 1, pp. 88–93, Jan 2002.

[6] K. Nagel, J. Esser, and M. Rickert, "Large-scale traffic simulations for transportation planning," *In Annual Reviews of Computational Physics VII*, pp. 151–202, 2000.

[7] "City Simulator IBM alphaworks open source," http://www.alphaworks.ibm.com/tech/citysimulator, 2001.

[8] IBM, "IBM TSpaces Programmer's Guide," *http://www.almaden.ibm.com/cs/TSpaces*, 2002.

[9] Sun Microsystems, "Java Servlet Specification version 2.4," *Java Community Specification JSR-000154*, August 2002.

[10] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan, "Middlebox Communication Architecture and Framework," Internet Engineering Task Force," Internet Draft, Dec. 2001. [Online]. Available: file:///afs/z/g/netapp/biblio/internet-drafts/draft-ietf-midcom-framework-06.txt
[11] "IBM PowerNP, NP4GS3," http://www-3.ibm.com/chips/products/wired/products/np4gs3.html.
[12] NEC Europe Ltd., "Ad hoc routing. Fleetnet Project," http://www.ccrle.nec.de/adhoc.html, 2002.
[13] J. K. Nils Gura, Albert Held, "Proactive Services in a Distributed Traffic Telematics Application," *Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit*, pp. 585–592, 2001.
[14] D. U. Fastenrath, "Floating Car Data on a Larger Scale," ITS-World Congress, DDG Gesellschaft fr Verkehrsdaten mbH, Tech. Rep., 1997.
[15] H. Kwella B., Lehmann, "Floating Car Data Analysis of Urban Road Networks," *Lecture Notes in Computer Science 1798 Springer 2000*, pp. 357–267, 2000.
[16] T. Ernst, K. Uehara, and K. Mitsuya, "Network mobility from the InternetCAR perspective," in *17 th International Conference on Advanced Information Networking and Applications (AINA'03)*, 2003, pp. 19–26.
[17] InternetCAR Project, "InternetCAR webpage at WIDE," http://www.sfc.wide.ad.jp/InternetCAR, 2003.