An Adaptive Road Traffic Regulation with Simulation and Internet of Things

Shanthini Rajendran, Suresh Rathnaraj Chelladurai, Alex Aravind[*]

Department of Computer Science
University of Northern British Columbia
Prince George, BC, Canada - V2N 4Z9
(rajendr,chellads,csalex)@unbc.ca

ABSTRACT

Traffic congestion is a growing concern in most cities across the world. It is primarily caused by a sudden increase in the number of vehicles in a relatively small number of roads and intersections, while other roads have the capacity to accommodate more traffic. In such situations, distributing traffic to roads in a balanced way could alleviate congestion. With the help of modern technology such as Internet of Things (IoT) and simulation, road users can be encouraged to choose their route on-the-fly, by providing necessary information such as projected travel time on the next leg. In extreme situations, traffic on some critical roads could be adaptively reduced by even introducing levy. A simple solution like providing road traffic information, benefits and penalties, etc., ahead in each intersection would allow travellers to make cognizant choices and therefore could lead to a better, more efficient traffic distribution.

To implement the proposed system, simulation and IoT must be brought together by a suitable communication middleware system so that they can work in synchrony. Implementing an actual IoT infrastructure and then testing the cause and effects of traffic congestion with the system in-place is a daunting task. Simulation would help us to test and validate the IoT system for functionality, performance, and scalability. In this paper, we propose a novel framework for integrating IoT and simulation using a message-oriented middleware in the context of an adaptive traffic regulation system and then demonstrate the framework with the help of a prototype implementation.

Categories and Subject Descriptors

C.4 [**Computer System Organization**]: Performance of Systems; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

[*]Corresponding author

SIGSIM-PADS '16, May 15-18, 2016, Banff, AB, Canada
© 2016 ACM. ISBN 978-1-4503-3742-7/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2901378.2901406

Keywords

Simulation, Road traffic, Internet of things, Dynamic data-driven, Traffic regulation,Traffic congestion

1. INTRODUCTION

Urbanization is an increasing phenomenon worldwide. One of the direct impacts of this phenomenon is the increased traffic in cities. Increased traffic often leads to traffic jams (congestion) when not managed properly, and therefore is a major problem. Traffic congestion has several negative effects such as travel delay, larger number of accidents, increased carbon emission, increased pollution, etc. So, a smart traffic management system is an urgent need across the globe to avoid traffic congestion. In a broader sense, distributing traffic in urban areas contributes significantly to the society's safety and welfare. Fewer congested routes means fewer frustrated drivers and hence a proportionally safer road environment. However, with the increase in the number of vehicles on the road, the problems and challenges on city traffic continue to rise.

1.1 Internet of Things and Road Traffic

Recent times have witnessed IoT explosion ubiquitously. The ability to connect millions of devices to the internet and source data in realtime is a great asset and has been put to thorough use in building traffic solutions that effectively manage congestion. Traditional approaches to IoT solutions follow client-server model, where there is a common point of data aggregation, such as a gateway or a base station. Designing IoT solutions based on a middleware architecture, would allow us to view each sensing device as a service point. All the services can be monitored and accessed from the cloud. We also have access to information from different sensors that may not be part of our system (data sourcing). Interaction of thousands of wireless devices leads to continuous flow of events and massive amounts of data are being generated. The challenge now is "how to deal with this massive flow of online data? "[30]

In the past two decades, many intelligent solutions have been proposed to manage traffic in smart ways. Some of them include placing cameras strategically at intersections for monitoring traffic, sensors and more novel technologies such as Intelligent Transportation Systems (ITS) and Vehicular Ad-hoc Networks (VANETs). Currently, most of the proposed intelligent transport systems are either expensive and therefore cannot be offered to all roads, or not sophisticated enough to be deployed widely. The rise of Internet of Things (IoT) in this decade could allow us to sense, collect, and aggregate information from the most mundane sources possible. Every day, there are new improvements in creating and maintaining compatible communication standards, extending device interfaces to cover as many devices and sensors as possible and much more. Vehicles, roads, intersections provide the capability to house sensors. The development of various communication protocols, multi-hop message transmission protocols allow us to have Vehicle-to-Vehicle and Vehicle-to-Infrastructure communication systems. With the availability of very tiny powerful computers such as Raspberry PI, Intel Edison, the persistent challenge of a "resource-constrained" environment too is lifting off. We are at a crossroad, with the free availability of such different and powerful technologies at our disposal. In this paper, we focus on the amalgamation of IoT, dynamic data processing and analysis, simulation, messaging middleware, and mining repository to build a better, adaptive traffic regulation system.

Since the early 1990's, computer simulation has been used as a vital tool in modeling complex dynamic systems [8, 12]. Traffic systems are quite dynamic and simulation is extensively used to model and study its various aspects and implementation choices. It is easier and more flexible to change parameters and view cause/effect analysis, bottlenecks and various other factors in a simulation system than in the actual system. With the advancement of IoT, simulation can be not only used for modeling and simulation of a system before its implementation but also can be heavily used for prediction of specific aspects of the system during its operation, as illustrated in Figure 1 for a traffic application.

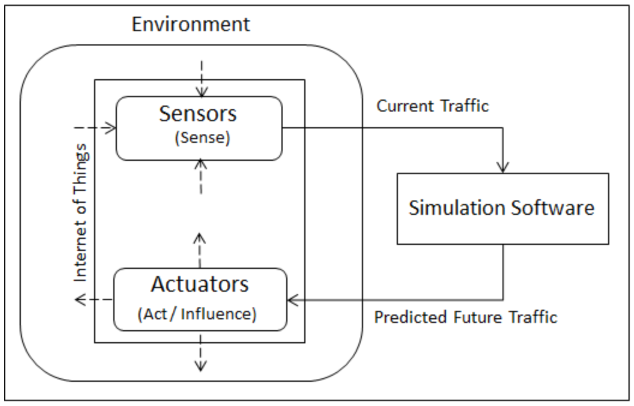

Figure 1: Integration of IoT and Simulation

Here, IoT feeds information of current traffic to the simulator. Then, the simulator can simulate and predict future traffic based on the current state and trend, and that information can then be feed back to influence the traffic. This idea is incorporated into our proposed framework.

In the past, the above mentioned components have been used and well understood mostly in isolation. Bringing them together to solve traffic congestion is what we emphasize in this paper. In such integrated systems, software occupies the major portion of the system including simulation system. Particularly, messaging middleware and mining repository are backbone of the system on which the rest hang. Therefore, messaging middleware and mining repository must be chosen with care based on the mainstream software development standards and trends. To set the context for our selection of middleware components, let us briefly review the software development trend.

1.2 Software Development Trend

Software engineering is a process dominated by intellectual activities focused on developing software systems with immense complexity and numerous unknowns in computing perspective [25]. For the reusability of code and to reduce complexity, software companies along with research organizations, develop and maintain standards and adopt them as best software engineering practices.

Software engineering has evolved and undergone several stages since 1960. In the beginning, organizations used virtually all custom tools, custom processes, and custom components built in primitive languages. This approach was expensive, not scalable, and hence most projects were never completed [18]. From the experience gained in the previous decade, in 80's and 90's, organizations used more repeatable processes, off-the-shelf tools, and about 70% of their components were built in higher level languages [25]. Even then, they were heavily dominated by custom made tools (70%) and only about 30% of the components were commercial products such as operating systems, database management system, networking, and graphical user interface. With this approach, some of the organizations could achieve success only for small and medium size applications. For the applications of higher complexity, the existing tools would not suffice.

Since the early 2000, almost all organizations including industries started using managed and measured processes and integrated automation environments, and more importantly the trend in usage of various software components took a different turn. That is, about 70% are based on off-the-shelf commercial components and only about 30% of the components need to be custom built [25]. So, the trend in software design and development in industries for quite some time has been towards using mostly off-the-shelf commercial components. Simulation integrated software development practices cannot be an exception. That is, if we expect the design and implementation of our simulation and IoT integrated system framework to be adopted widely, it has to be based mostly on off-the-shelf commercial or open source components.

1.3 Messaging Middleware

Messaging middleware is a software layer or a set of sublayers that connects various heterogeneous domains. The messaging middleware is the glue that holds it all together [20]. Research on middleware systems has been gaining momentum over the years. One of the important advantages of a middleware system is its ability to provide seamless interoperability between various components. This allows the programmer to focus on building standard, adaptable, and effective solutions rather than worrying about the finer details of the underlying layers [30]. A complete list of the advantages and disadvantages of using a message oriented middleware is discussed elsewhere [5].

There are various standards and protocols for building message oriented middleware systems. One of the most popular middleware is the Java Messaging Service. JMS provides a standard API for the Java platform as well as many services for interoperablility within and outside the

Java platform. Integration with other languages such as Ruby, Scala etc is possible but very tricky. Therefore, there was this necessity for a messaging standard that will assure interoperability among different platforms and integration services. AMQP emerged out of this need [13, 15, 2]. At the time of writing this paper, AMQP and its various open source implementations are in practice in some of the most critical systems running in the world, specially in the financial industry.

Advanced Message Queuing Protocol (AMQP) is an important protocol heavily used in recent years. It was developed by John O'Hara of JP Morgan Chase Inc., and is a binary wire transmission protocol. AMQP originated in the financial industry as a solution to the problem of seamlessly connecting different processing platforms together. In order to attain this effortless interoperability, AMQP boasts of a well-defined, structured set of rules or behavior for sending and receiving messages. These rules use a combination of techniques including store and forward, publish and subscribe, peer to peer, request/response, clustering, transaction management and security among many because of which the protocol has become valuable for communication across various operating systems, programming platforms, integration services and hardware devices without compromising on performance [2].

RabbitMQ is an open source implementation of the standard AMQP 0-9-1 and is programmed in Erlang. It provides support for all major operating systems and is also available in languages such as Python, Java, Ruby and .Net. RabbitMQ is very extensible and provides a number of plugins to allow communication with other web protocols such as HTTP, XMPP, SMTP and STOMP [2].

1.4 Mining Repository

Completely automated systems for practical use, though possible theoretically, is a long way to go. For most systems including road traffice, human interaction to offer intelligent decisions based on current and predicted future traffic should be an option for several reasons. Such interaction requires a good analytic support based on the data about past, present, and future. In terms of technology, it needs a scalable mining repository where such data can be deposited and necessary analytics could be derived. An IoT application would greatly benefit by the presence of a mining repository that provides on-going, live support for growing, near-real time data. Such repositories are in practice now.

In a nutshell, we envision that IoT, simulation, mining repository connected by an efficient messaging middleware can play a fundamental role in the advancement of automation integrated future systems.

1.5 Contributions

We first propose a novel framework that integrates simulation and IoT with a message oriented middleware system and mining repository. We then demonstrate the feasibility of the proposed framework by the design and implementation of an adaptive traffic regulation system. Limited simulation experiments were conducted and the results are reported.

The internet of things platform uses sensors to collect data and actuators to influence the system to change its behavior. Discrete event traffic simulation system is used to predict the future traffic behavior based on sensor inputs. Mining repository is used to store sensor data and simulation traces, and provide analysis support to the users and managers of the system. The messaging middleware is responsible for connecting these components and facilitating message transfer in near real-time. It acts like a "postman" system, distributing the right data at the right time to the right storage system and the right subscriber. The proposed framework is completely reproducible and is built from scratch using open source off-the-shelf components.

The rest of the paper is organized as follows. In Section 2, we explain the proposed framework in detail. Section 3 describes the adaptive road traffic simulator. Simulation experiments and the results obtained are outlined in Section 4 and the paper is concluded in Section 5.

2. PROPOSED FRAMEWORK

In this section, we describe in detail the framework to integrate IoT platform and simulation with a suitable middleware. The proposed framework is shown in Figure 2. It has four main components: (i) The application domain where IoT is deployed; (ii) Simulation system; (iii) Mining repository; and (iv) Messaging middleware.

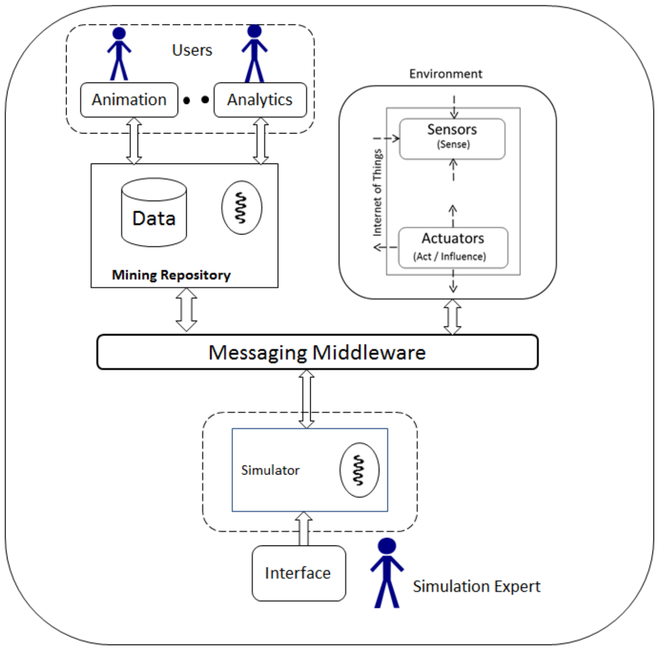

Figure 2: Framework for Smart Regulation System

This framework integrates discrete event simulation and Iot in a simplified way. The application domain where IoT is deployed, may not necessarily be limited to transportation system, rather, this framework can be applied to any practical systems that involve IoT. Next, we will explain the various components of the proposed framework.

2.1 Simulation Component

The simulation component in the framework mainly used to predict the future based on the past and present. It gets it inputs from both IoT (current state) and mining repository (past states). Based on these information, it can simulate and predict the future scenarios, and send that information back to the mining repository. The information stored in the mining repository is then used to derive insights on current state and future states of the system and feed that information back to IoT to influence the current system. Any part of the proposed framework can be modeled and simulated before its actual deployment.

2.2 Internet Of Things

In the proposed system, IoT is primarily used to collect data and influence the system behavior. For our prototype design and implementation, we simulate the presence of an active IoT infrastructure as a part of the traffic simulator.

The IoT infrastructure provided in our simulator, primarily has two components: (i) sensors to collect data; and (ii) actuators to act or influence the system to change. Sensors basically send information such as number of vehicles on the selected road segment. The simulation software, based on the current traffic, in turn predicts future traffic by simulating future scenarios, and sends the relevant information to the actuator component of IoT. The actuator component of IoT then influences the system, and the cycle repeats. The actuator component would be a display board, that will inform the travellers about the present road conditions, time to reach next destination (next intersection, in this case) and the speed limit to be followed.

Simulating an entire IoT system is an effective way to test the design of your system even before developing it. The future is expected to be revolutionized by the integration of IoT and simulation. With simulation, we can test how a target system behaves under different scenarios and road conditions. In summary, to integrate IoT and simulation effectively, we need a middle-ware that is capable of offering message service in near real-time and a scalable mining repository with suitable analytics support. Next, we will discuss about the implementation of such a suitable middleware.

2.3 Messaging Middleware

A messaging middleware helps connecting various heterogeneous devices seamlessly and controls the flow of events extensively by supporting various plugins and interfaces making it very extensible. So, a critical part in realizing the proposed system is using a suitable middleware that could facilitate dynamic data to flow in near real time. Although our framework is generic that any messaging framework can be used, for our prototype implementation we adopt RabbitMQ as our messaging middleware for the following reasons.

1. As it supports a standard messaging protocol AMQP and it is not confined by any proprietary, client specific messaging protocol.

2. All the messages are collected by the RabbitMQ. This type of message storage pattern is very similar to a push-style data flow. All the messages move from where they are produced to where they are consumed in a fluid manner, without having to periodically pull messages at various end points.

Next we briefly explain how RabbitMQ works.

2.3.1 RabbitMQ

RabbitMQ stores messages in queues and acts as a broker between two types of processes, producers and consumers. There are two core units that form RabbitMQ namely, Queues and Exchanges/Router [26]. In simple terms, every message that is passed through RabbitMQ has to be placed in a queue. The main function of the router is to route the messages from the appropriate producer to the appropriate consumer. Each message consists of a simple header, specifying where it is heading to. The router doesn't read or process the message, it simply delivers the message to the appropriate queues. The consumers on the other hand, can either subscribe to a particular message or keep polling to see if a message is received. The router in RabbitMQ is called as exchanges. Figure 3 shows a simplified architecture of the components involved in the RabbitMQ messaging system [5].

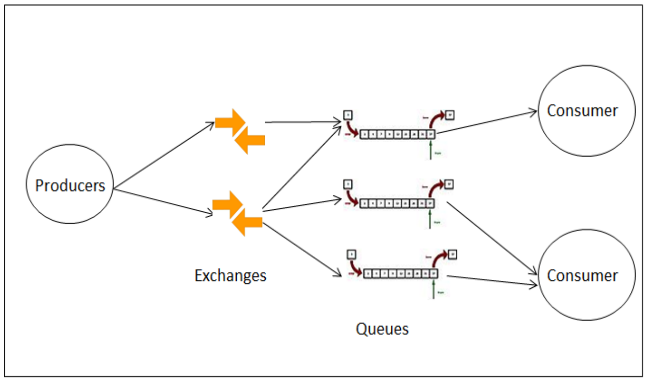

Figure 3: RabbitMQ Architecture

The producers in RabbitMQ generates messages which are then pushed to the exchanges. The exchanges then apply some routing rules on these messages and push each message to the appropriate queues, thus providing a delivery service. The messages can either be directly delivered, or it can be delivered because of an existing subscription system. The routing choices simply depend on the value of the routing key which is available in the header part of the message. This header is constructed by the producer itself. If a particular message is to be sent to more than one queue, then the exchanges take the responsibility of duplicating the message and delivering it to the queues. Consumers always must have a permanent connection with their corresponding exchanges, so that the exchanges may be aware of the exact details of the queues the consumers have subscribed to. In addition to separate queues for consumers, in our prototype system, we also have a common queue that stores all incoming messages to all exchanges in their order of arrival. This common queue is what mining repository is subscribed to. All operations inside RabbitMQ are done in memory and they are transferred to the disk periodically. All the messages in our simulator are time-stamped and their order is maintained consistently throughout the simulation.

2.4 Mining Repository

The most important requirement of a mining repository for the proposed framework is the ability to store and access large data in near real-time and agility for data growth and updates. A near real time search engine with standardized API is its main attraction. Most databases or mining repositories that store large volumes of data, require some sort of sorting, filtering and other such capabilities, to segregate and organise that data so that it can be easy to write queries for searching the data. In this case, offline analysis is the only solution.

Again, any mining repository with above capability can be used in our framework, we adopt Elasticsearch for our prototype implementation. We briefly explain Elasticsearch next.

2.4.1 Elastic Search

Elasticsearch is one of the most popular mining repositories with rich functionalities [19]. We use Elasticsearch in our proposed framework for the following reasons that illustrate its functionality.

1. *Availability and cost:* It is a open source software and that makes it easier to integrate with any application.

2. *Scalability:* When it comes to data storage, we consider scaling from two perspectives. 1) Vertical or scale up - Adding more hardware, powerful servers, 2) Horizontal or scale out - Adding more servers. Elasticsearch is distributed by nature. A running Elastic search server is called a node. Two or more nodes form a cluster. Elasticsearch is heavily supported to run in clusters.

3. *Near real-time Search, Data Analysis, and Visualization:* Elasticsearch stores data in *indexes*. An *index* is analogous to a *database* in a Relational Database Management System (RDBMS). Each document in an Elasticsearch index is a JSON object. All the documents are indexed as soon as they are added to an index, however they are only updated at a pre-defined time interval (1 second). For large scale applications, the Elasticsearch server does not refresh after each update using the default interval because it is costly in terms of disk i/o operations. However, we have the ability to set a custom update interval based on the application.

4. *RESTful API:* Since Elasticsearch is a RESTful server, the most widely used way to communicate with the client is through its REST API. A client normally opens a connection with the Elasticsearch server, posts a JSON Object as a request and receives a JSON object as a response. This is very useful, because there is no restriction on the type of client, the programming languages used. Any client can communicate with the Elasticsearch server with HTTP requests.

5. *Popularity:* It is used by thousands of organizations worldwide including Netflix, Facebook, GitHub, etc.,

In our traffic simulator, we store all the events occurring during the simulation. Each event is associated with a timestamp and is stored on Elasticsearch server in JSON format. By querying the events using the appropriate message, we can get real time results for analysis.

Kibana is the graphical front end for Elasticsearch and provides data visualization and analytic capabilities. Kibana is a browser based interface and provides the capability to search, view and interact directly with the data stored at the Elasticsearch indices. Kibana automatically generates queries for Elasticsearch, the queries are similar to the ones a programmer might write, using the Elasticsearch API. Our kibana dashboard shows us a list of events that are being pushed to the Elasticsearch server at each update.

Kibana allows us to view the events (in our case) logged by the Elasticsearch server, sort and perform search queries on the data presented. Advanced search options such as saving a search, loading a saved search are available and is very handy. Kibana supports visualization tools in the form of various kinds of charts. This allows us to view and compare various scenarios from the live data presented to us and not having to stop and start the simulator again and again. We focus on the business intelligence aspect of Elasticsearch in our simulator, mostly because of ease of use and its plug-in type architecture. There are other scalable search engine solutions that are available like Solr could be used in-place of Elasticsearch.

3. ROAD TRAFFIC SIMULATION

Traffic systems provide a variety of features such as dynamic state change, real time decision making, unexpected congestion etc, that makes it hard to conceive, analyze and test this system. Also, a traffic system consists of many participants with dissimilar interests. Simulation can be a very useful tool to help capture the effect of influencing factors in a real-life traffic system, such as pedestrians, road blockage etc and view the outcomes without disturbing an existing implementation. We briefly review some of the research done in traffic simulation systems. The authors in [12] present an elaborate survey on the state-of-art Intelligent Transportation Systems presently in use across the globe. The authors identify three main research trends on Intelligent Transport Systems (ITS) -

ITS based on Wireless Sensor Networks

An example of an ITS based on WSNs would be Advanced Traveller Information Systems (ATIS) for Indian Cities developed in India in 2014[6]. In this project, over 100 GPS devices and cameras were mounted at intersections. These use wireless communication to transfer information collected on the traffic conditions to the central control center. This information is then converted into data that can be used by travelers, using travel time prediction models and algorithms. The travelers would then be informed about the traffic by the use of strategically placed Variable Message Signs(VMS).

ITS based on Vehicular Sensor Networks

An example of a VSN based traffic control system would be the Mobeyes Project[11]. In this project, all the data sensed by the vehicular nodes are locally processed within the vehicles. The data nodes generate feature-rich data summaries with time and context information. There is no necessity for the road-side infrastructure. Instead of road-side infrastructure, the project has Mobeye data collectors that work more or less like police patrolling agents and collect data from their neighboring vehicular nodes.

Vehicular Ad-hoc Network Simulators

In literature, a road traffic simulator is normally called a VANET simulator. The authors in [3] classify VANET simulation software in three different categories: (a) vehicular mobility generators, (b) network simulators, and (c) VANET simulators. Vehicular mobility generators are useful in creating realistic, accurate patterns of movement of vehicles in various scenarios. *Vehicular mobility generators* fundamentally apply mobility models used in the VANET domain to simulate the movement of vehicles accurately. Output from a vehicular mobility generator will generally be a trace file, containing trip details for each vehicle, spanning across the simulation period. This trace file, can then be used in network simulators such as OMNET++[28], NS3[23] etc for further analysis. *Network simulators* perform in-depth analysis of data packets, packet loss, transmission delays etc. *VANET simulators* allows us to change the behaviour of vehicles based on a given condition within an integrated framework [3]. VANET simulators provide both the ability to generate vehicular mobility as well as network simulation. The VANET simulators specified above, presently, has no support for integration with the Internet of Things[4, 22].

The future is going to be revolutionized by the Internet of Things, specially in the mobility space. Therefore, we see a need for simulators that incorporate necessary support for Internet of Things. In the next section, we describe in detail, our road traffic simulator, that integrates both Internet Of Things and discrete event simulation in this context.

3.1 Simulation System

Traffic simulation models can be classified either as microscopic or macroscopic. In a microscopic model, simulation is centered around an individual vehicle and its performance is evaluated across the entire network. In a macroscopic simulation model, the entire network is modeled and this is used to particularly model large scale systems. In order to retain simplicity, we need a different model that allows us to abstract the required entities and reduce the complexity of the overall system[11]. The various components of a traffic control management system such as vehicle, intersections, road segments, sensors, traffic lights etc are to be modeled independently. Discrete-event simulation supports this type of modeling and provides us with an environment where we can freely integrate the modeled entities. There is a global simulation clock, that controls the entire simulation time-range. Each entity has a list of events associated with it. All events are ordered and executed based on their priority within the event queue. All the events passing through the middle-ware are consistently logged in the mining repository for further data analysis.

The traffic network considered is a simple grid. The nodes in the grid represent intersections and paths connecting the nodes represent the road segments between intersections. The network is connected, meaning there exists a path between two nodes. The IoT part of the system essentially contains a collection of sensors and actuators. In our simulator, there is a sensor-actuator pair at each intersection, in all the directions branching out from that particular intersection. The sensor keeps measuring the number of cars passing the road segment from the chosen intersection to the next intersection, and posting this information continuously to the middle-ware as depicted in Figure 4. The simulator takes the sensor input and uses this input to predict future traffic and the real-time road congestion scenario is passed on to the actuators as messages. The actuator present at each intersection, displays the current congestion statistics, which will then influence the drivers' choice of staying on the same route or taking a different route to their destination. The IoT system can simply send and receive messages from the messaging middleware using AMQP requests.

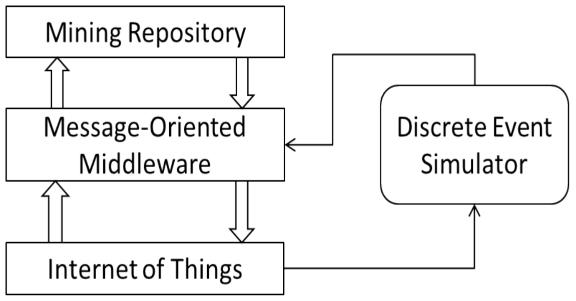

Figure 4: Data Flow within the proposed framework

The simulator can be run on one master computer or on several nodes, with one node acting as the coordinator. It can be implemented in any language, we used JAVA. One key design point of our simulator is that we wanted to make the code as reproducible as possible and that can be achieved only by implementing the simulator using off-the-shelf components. Any traffic simulator which has the capability to send/receive messages following the Advanced Message Queuing Protocol (AMQP) can be used in place. The chosen messaging middle-ware (RabbitMQ) offers us a wide variety of plugins and interfaces for both the IoT environment as well as the mining repository.

We next describe the simulation system components or entities and events of the system.

3.2 System Entities

- Vehicle - represents the model of a vehicle in an actual traffic system. Each vehicle will know its source, destination and will travel from the source to the destination based on the velocity allowed in that road segment. Vehicle has intelligence to take routing decisions based on the available real time sensor information at each intersection.

- Node - represents the model of an intersection in an actual traffic system. A node has four traffic lights, representing a simplified model of a four-way intersection.

- Traffic Lights - represents the model of a traffic signal in an actual traffic system. It controls the flow of vehicles from one intersection to the next.

- Road Segment - represents the model of an actual road connecting two intersections. Each road segment has a maximum capacity that specifies the allowable amount of traffic at a particular time. The speed limit of the road segment varies according to the current load. Current load gives the number of vehicles traversing the road segment at that time. The maximum allowable velocity (V_{max}) is calculated using a *Congestion Formula* given as follows -

$$V_{max} = \text{Max } (10, V_{max}\text{- f(current load)})$$

- Sensor - represents the model of a actual traffic sensor, such as an induction loop or an overhead camera. The sensor reports the number of vehicles passing through the intersection in a real time basis.

3.3 System Events

The simulation basically involves following events.

1. Vehicle Create Event - creates a vehicle with a source, destination and a random velocity in the range (12, 28) m/s. The creation of this event is controlled using a probability distribution.
2. Vehicle Begin Event - denotes the departure of a vehicle from an intersection. Also invokes the Sensor Increment Event.
3. Vehicle Reached Event - denotes the arrival of a vehicle at an intersection. Also invokes the Sensor Decrement Event.
4. Sensor Increment Event - increments the number of vehicles by 1, for that particular road segment attached to the sensor. It also decreases the maximum allowable velocity for that road segment based on the *Congestion Formula*.
5. Sensor Decrement Event - decrements the number of vehicles by 1 when a particular vehicle reaches a node attached to the road segment to which the sensor is connected.
6. Traffic Light Green On Event - enables green light on a particular intersection causing all the vehicles entering the intersection to travel through its associated road segment without stopping at the intersection.
7. Traffic Light Green Off Event - disables green light on a particular intersection and will invoke a Traffic Light Red On Event.
8. Traffic Light Red On Event - enables red light on a particular intersection causing all the vehicles entering the intersection to stop at the intersection and will create a delay before invoking the Vehicle Begin Event.
9. Traffic Light Red Off Event - disables red light on a particular intersection and will invoke a Traffic Light Green On Event.

3.3.1 Simulation Input Parameters

The road network which we have considered as the input for this simulation is stored in a JSON format in the local file system. This JSON file contains the information about the nodes, intersections, road segments and how they are connected to each other. At the beginning of the simulation, the simulator parses through JSON file and converts it to a in-memory graph data-structure. Vehicle Create Event is called for the specified number of vehicles specified in the simulation input parameters and each vehicle gets attributed a random source and a random destination. The shortest path between a given source and destination node is calculated using Dijkstra's shortest path algorithm.

3.3.2 Vehicle Routing

Each vehicle will check the sensor input and chooses it route to the destination accordingly. The vehicle takes into consideration, the maximum allowable speed limit of the K shortest paths to the destination and predicts the time taken for each of the K shortest paths to reach the destination. The vehicle chooses the path which has minimum time to reach the destination and routes to the next node based on this time.

When the current load of a particular road segment increases, the velocity of the cars traveling on that road segment decreases. This decrease is calculated using the formula,

New Velocity = Max (10, Road segment current velocity - (1.3 * current load))

4. SIMULATION EXPERIMENTS

To demonstrate the functionality of the proposed simulator, we present two experiments to compare the performance of traditional traffic regulation system and adaptive traffic regulation system. The first experiment compares the number of vehicles that reach their destination. The second experiment compares the average time taken by the vehicles to reach the destination.

Experiment 1: Number of vehicles that reach destinations

The simulation parameters used for experiment 1 is given in Table 1.

Parameter	**Value**
Input Grid Size	50 x 50
Initial Number of Cars	1000
Road Length	[500 to 1600]m
Road Capacity	[50 to 160]
Initial velocity - car	[12 to 28]m/s
Simulation Period	1800 ticks

Table 1: Simulation Parameters for Experiment 1

In this experiment, we show the number of cars that reach the destination with sensor input from the Internet of Things environment in contrast to the scenario wherein the cars are routed without any sensor input. In terms of the number of cars reaching their destination at each simulation clock, we see that the inclusion of sensor data consistently results in increased performance.

The performance gets closer as the cars started decreasing near the end of the simulation.

Experiment 2: Average time to reach destinations

In this experiment, we measure the average time taken by the vehicles in the simulation to reach the destination with and without the presence of the sensor input, while we vary the rate of vehicle generation. The rate of vehicle generation specifies the number of vehicles that will start at the same simulation tick.

The simulation parameters used for experiment 2 is given in Table 2.

In this experiment, we observe that the simulation with sensor input performs fairly well for lower rates of vehicle generation (rate $\leq$ 12). The greater this rate grows, each road segment in the simulation becomes heavily congested. This causes the vehicle to be re-routed to many different paths before reaching the destination. In this scenario, the simulation without sensor input performs comparatively well.

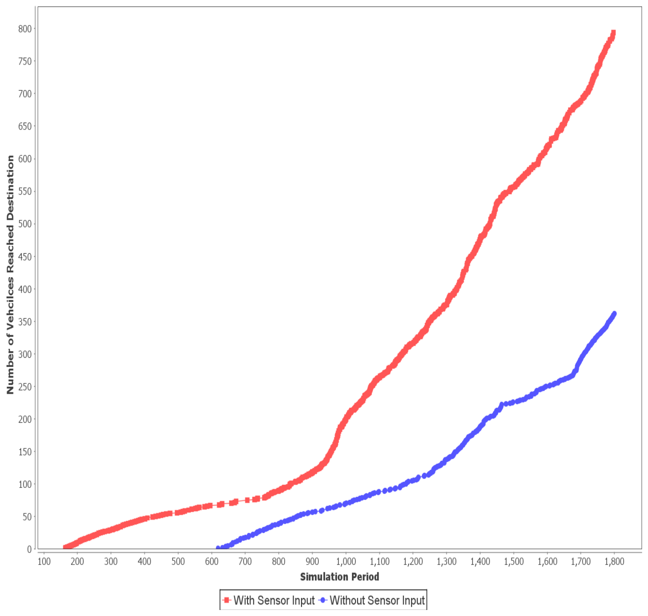

Figure 5: Number of vehicles reached

Parameter	Value
Input Grid Size	20 x 20
Initial Number of Cars	1000
Road Length	[500 to 1600]m
Road Capacity	[50 to 160]
Initial velocity - car	[12 to 28]m/s
Simulation Period	1000 ticks

Table 2: Simulation Parameters for Experiment 2

5. CONCLUDING REMARKS

In this paper, we proposed a novel framework integrating simulation, IoT and mining repository using a message-oriented middleware. The efficacy of the framework is demonstrated in the context of simulating traffic congestion. Simulation is used for prediction. IoT again has two components sensors and actuators. Sensors collect data of the current state of the system and feed to the simulation system. Based on the simulation scenarios and data mining, decisions are made to influence the system through actuators. The messaging middleware glue together these components and facilitate effective communication among them. The middleware could easily be hosted as a distributed system making it a very fast and scalable solution. Using Elasticsearch on the data mining for real-time data analysis give the simulator an extra edge at the same time gracefully solving the scalability problem.

We have conducted a limited simulation experiments to demonstrate the proof of concept. Although these experiments give some interesting results, we need to keep in mind that vehicles with human drivers could be different as we have limited knowledge on how each driver will behave based on additional information about the traffic. The proposed system would be more suitable for driver-less vehicular systems. Manual interaction with the system would result in better traffic regulation for vehicles with human drivers. There are several possibilities for further research on this work including in depth study on road traffic congestion and applying the proposed framework for other applications such as energy system, resource management, etc.

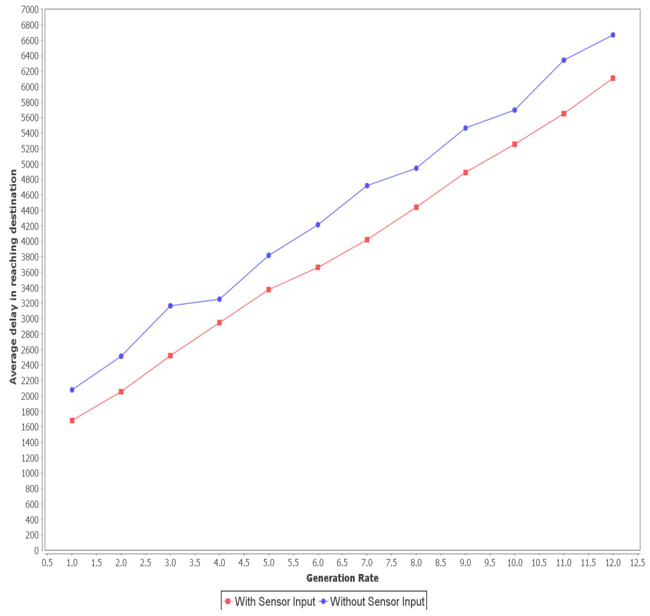

Figure 6: Average Time to Reach Destination

6. REFERENCES

[1] INRIX Report. 2015.

[2] J.O'Hara. Toward a commodity enterprise middleware. *Queue*. 5(4):48-55, 2007.

[3] F.J. Martinez, C.K. Toh, J.C. Cano, C.T. Calafate, and P. Manzoni. A survey and comparative study of simulators for vehicular ad hoc networks (VANETs). *Wireless Communications and Mobile Computing*. 11(7): 813-828, 2011.

[4] M. Piorkowski, M. Raya, A.L. Lugo, P. Papadimitratos, M. Grossglauser, and J.P. Hubaux. TraNS: realistic joint traffic and network simulator for VANETs. *ACM SIGMOBILE mobile computing and communications review*. 12(1): 31-33, 2008.

[5] Nannoni, N. Message-oriented Middleware for Scalable Data Analytics Architectures. 2015.

[6] N. Sivanandam, V. Lelitha Devi, V. Ravi, S. Krishna Kumar. Advanced Traveler Information Systems(ATIS) for Indian Cities. April 2014.

[7] L. Bononi, M. Di Felice, G. D'Angelo, M. Bracuto and L. Donatiello. MOVES: a framework for parallel and distributed simulation of wireless vehicular ad hoc networks. *Computer Networks*. 52(1): 155-179, 2008.

[8] R. M. Fujimoto. Parallel discrete event simulation: Will the field survive? *ORSA J. Computing*. 5(3), 1993.

[9] NS-3 - Discrete Event Network Simulator, accessible at *http://www.nsnam.org*.

[10] E. Baccelli, O. Hahm, M. WÃd'hlisch, M. GuÌĹnes, and T. Schmidt. RIOT: One OS to Rule Them All in the IoT. 2012.

[11] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P.

Bellavista, and A. Corradi. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *IEEE Wireless Communications*. 13(5): 52-57, October 2006.

[12] N. Kapileswar and H. Gerhard. A Survey on Urban Traffic Management System Using Wireless Sensor Networks. *Sensors*. 16(2): 1-25, 2016.

[13] M. Albano, L. Ferreira, and A. Alkhawaja. Message Oriented Middleware for smart grids. *Computer Standards & Interfaces*. 38: 133-143, 2014.

[14] A. Dunkels , B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. 455-462, November 16-18, 2004.

[15] S. Appel, K. Sachs, and A. Buchmann. Towards benchmarking of AMQP. *In Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. 99-100, 2010.

[16] H. Arbabi, and M.C. Weigle. Using DTMon to monitor transient flow traffic. *In Vehicular Networking Conference (VNC), IEEE*. 110-117, 2010.

[17] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli. A simulation platform for large-scale internet of things scenarios in urban environments. *In Proceeding of the of the First International Conference on IoT in Urban Space (Urb-IoT 2014)*. 50-55, 2014.

[18] Brooks Jr, and P. Frederick. "The mythical man-month (anniversary ed.)". (1995).

[19] O. Kononenko, O. Baysal, R. Holmes, and M.W. Godfrey. Mining modern repositories with elasticsearch. *In Proceedings of the 11th Working Conference on Mining Software Repositories*. 328-331, May, 2014.

[20] E. Curry. Message-oriented middleware. *Middleware for communications*. 1-28, 2004.

[21] W.H. Lam. Development of intelligent transport systems in Hong Kong. *In Intelligent Transportation Systems Proceedings*. 1000-1005, 2001.

[22] R. Mangharam, D. Weller, R. Rajkumar, P. Mudalige, and F. Bai. Groovenet: A hybrid simulator for vehicle-to-vehicle networks. *In Third Annual International Conference on Mobile and Ubiquitous Systems: Networking & Services*. 1-8, July 2006.

[23] C. Gustavo. NS-3: Network Simulator 3. UTM Lab Meeting. 20, April 2010.

[24] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. *In Local computer networks, proceedings 31st IEEE conference*. 641-648, 2006.

[25] W. Royce, K. Bittner, and M. Perrow. The economics of iterative software development: Steering toward better business results. Pearson Education, 2009.

[26] S. Warren and S. Smallen. Building an Information System for a Distributed Testbed. *Proceedings of the ACM Annual Conference on Extreme Science and Engineering Discovery Environment*. 2014.

[27] ITU Internet Reports 2005: The Internet of Things. 7th edition, 2005.

[28] V. AndrÃąs and R. Hornig. An overview of the OMNeT++ simulation environment. *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. 2008.

[29] M.A. Chaqfeh and N. Mohamed. Challenges in middleware solutions for the internet of things. *In Collaboration Technologies and Systems (CTS),IEEE*. 21-26, May 2012.

[30] K. Paridel, E. Bainomugisha , Y. Vanrompay, Y. Berbers, and W. De Meuter. Middleware for the internet of things, design goals and challenges. *Electronic Communications of the EASST*. 28, Jun 2010.