Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A.A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

ENERGY EFFICIENT MIDDLEWARE FOR DYNAMIC DATA DRIVEN APPLICATION SYSTEMS

Aradhya Biswas
Richard Fujimoto

School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

Michael Hunter

School of Civil and Environmental Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

ABSTRACT

Middleware is required to support and interface multi-modal Dynamic Data Driven Application Systems (DDDAS) with back-end and other computing facilities. Middleware is also needed to support distributed simulations and emulations needed in earlier phases of system development. This work describes the Green Runtime Infrastructure (G-RTI), an energy-efficient client server based middleware developed to support distributed DDDAS simulation, emulation and deployment. G-RTI eases and accelerates the development and testing of multi-modal studies, testbeds and DDDAS systems. It serves as a platform for research in energy reduction techniques for middleware services. The services implemented by G-RTI are described and results of benchmarking studies are reported. Its application is demonstrated through a use-case for an end-to-end implementation of a connected vehicle application. G-RTI is open source.

1 INTRODUCTION

The past decade has seen an enormous increase in the development and adoption of sensors in all walks of life. This has been in conjunction with the widespread deployment and adoption of wireless networks and the Internet leading to an explosion of interest in systems composed of sensors or sensory data of various types and modalities. Such multi-modal systems can perceive the physical world better than ever before. Paradigms such as the Internet of Things (IoT), fog computing, cyber-physical systems, and dynamic data driven application systems are examples of some areas born out of these developments. These emerging systems differ substantially from the systems of the past, both in terms of the possibilities they offer as well as their requirements, especially regarding scale and energy efficient operation.

For example, connected vehicle systems include travelers' mobile devices, in-vehicle onboard computing systems, roadside infrastructure embedded in traffic signal controller cabinets and signs, and centralized local and regional traffic management centers coupled through vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. Such systems are transforming urban transportation. A wide variety of applications exist or are under development that will improve safety (e.g., hazard warning or collision avoidance), reduce congestion and traveler delays (e.g., route recommendation, adaptive signal control, event management), and enhance traveler experiences (e.g., entertainment, traveler information systems). Disruptive technologies such as automated vehicles and commercial drones as well as new paradigms such as ride-share systems create new opportunities and challenges for mobile sensor networks in connected vehicle deployments. An amalgamation of these can improve the efficiency or robustness of the system and lead to improved safety and quality of life for citizens.

These systems may utilize predictive simulations and data analytics software to control and manage the system. The simulations will use historical data stored in back-end databases and rely on dynamic real-time data from in-vehicle and road-side sensor networks to derive information about the current and future states

of the transportation network to inform and evaluate decisions. Such an ensemble of technologies and devices enable better informed decisions and allow for more autonomous systems with little or in some cases no human supervision to manage the transportation network.

Dynamic Data Driven Application Systems (DDDAS) (Darema 2004) are systems that can dynamically change their behavior based on the state of the environment in which they operate by incorporating real-time data into computations used for decision making. The evolving and sometimes unpredictable nature of systems such as power grids and vehicular traffic make them well suited to exploit the DDDAS paradigm. Example DDDAS applications utilizing computations embedded within the physical system were described in DDDAS (2018). Tools are needed to support both the development and deployment of systems such as these. The goal of the G-RTI project is to develop a common middleware platform to support research, development and deployment. In the early phases of design, simulation models are used to represent system components. Development of hardware devices and software to be deployed within the system itself requires an emulation capability where real-world devices are intermixed with simulation models and execute in real time. For example, new apps executing on mobile devices must be tested in the laboratory under realistic operating conditions that include aspects such as packet loss in wireless networks. Replacing the simulated elements with operational systems provides a natural pathway to transition the system developed in the laboratory to a system that can be deployed in the field. The system development can be greatly accelerated if a single software environment and tool suite can be used to support all aspects of simulation, emulation, and deployment. Such an interfacing of real and simulated worlds arises in agile system development. As an application is being developed, simulations can be used to consider what if scenarios to inform designers regarding reversions of the system (Fujimoto et al. 2018).

These systems are composed of multiple sub-systems that must be integrated to form a single seamless system. It is not uncommon to find such systems are composed of components ranging from small individual sensors and hand held mobile devices to large stationary back-end infrastructure. Middleware is required to support the wide variety of devices and technologies that are needed. Such middleware should provide a means to quickly and easily interface the components of the system. Such an array of devices utilizing static and real time data makes development and study of such systems difficult. Any study involving such systems would involve multiple simulators, testbeds and data sources. For example, a connected vehicle application might need a mobility (traffic) simulator, a road-side sensor network operating in the laboratory, and a wireless communication network simulator. Hence there is a need to connect and interface these individual components to enable them to interoperate.

Further, many devices, e.g., mobile devices, operate using batteries as their source of power. As such, energy consumption is a major concern. Middleware software and algorithms should be designed to minimize the energy consumption of these devices as well as support techniques for reducing energy use in computation, communications and synchronization. The Green Runtime Infrastructure (G-RTI) middleware was developed to address these challenges. It is intended to support all phases of system development including distributed simulation, emulation, and deployment, while providing a common, simple interface to support a wide variety of devices and services. An implementation of G-RTI is available as open-source software. We envision the use of G-RTI would allow for a growth in number of cross boundary connected application and studies by making it simple to develop and test systems that can derive intelligence, actuation and data from a wide variety of sources and simulations.

G-RTI is differentiated from other middleware in several respects. First, G-RTI is designed to support a wide array of heterogeneous systems including IOT and fog or edge computing systems. This introduces several considerations. It must provide flexibility to system developers to dynamically choose the amount of resource an end point possess, allowing the system to be heterogeneous and aware. The system must support both pub/sub and pull/push based communications. In addition the server should be able to support on-server applications. As mentioned earlier G-RTI aims to support all phases of the system development including simulation, emulation, and deployment. As such, G-RTI represents a different category of the middleware compared to those listed in Ngu et al. (2017). G-RTI maintains the advantages of the classes listed in this survey while remaining application independent. Finally, G-RTI allows dynamic addition and

removal of clients (simulations or data sources) and objects. This is of importance for interfacing and maintaining simulation models supporting/simulating dynamic systems, which are continuously evolving.

The section that follows presents an overview of G-RTI. This is followed by a description of its architecture and implementation. Results from a benchmarking study are presented both in terms of communication performance and energy efficiency. A case study of a vehicle mounted sensor network emulation using G-RTI is presented. This is followed by a discussion of related work, concluding remarks, and directions for future research.

2 G-RTI OVERVIEW

A runtime infrastructure (RTI) is middleware that implements services to interconnect sensors, databases, simulations and other elements in a distributed computing environment. This term is often used in the context of distributed simulation systems such as those realized using the High Level Architecture for Modeling and Simulation (IEEE standard 1516). Systems such as these are intended to support as-fast-as-possible and real-time distributed simulations that combine simulated and hardware components.

The Green Runtime Infrastructure (G-RTI) middleware was designed to be applicable to different contexts and applications. It supports interconnecting simulations and operational devices and software through a set of services for data exchange, time synchronization, and system management. G-RTI supports interoperability among a wide variety of platforms, operating systems, and programming languages that are interconnected through different mediums, e.g., private as well as public wireless and wired networks. The middleware was designed to be scalable to accommodate many such devices and provide all the functionalities that are expected by distributed simulation and other computation components to execute and exchange data. G-RTI was designed to reduce the effort required to develop a DDDAS, necessitating that application development be straightforward. Specifically, G-RTI utilizes web services and standard protocols to leverage existing software development tools.

Both push and pull based messaging are supported. In push-based messaging the receiver need not be aware of the sources producing messages nor when data will be produced. It is suitable for event based messaging, e.g. a triggered sensor sending data to a service (to save energy) or a simulator listening for events. This kind of messaging is also referred to as publish/subscribe communications. (Cugola et al. 2002; Souto et al. 2004) argue the importance of this type of messaging to support mobile and sensor networks in general. By contrast, in pull based messaging the receiver actively queries the sender for the message. This is suitable for scenarios with on demand message requirements, e.g., a client requesting data from a continuously operating sensor or a database. Use of G-RTI as a middleware for a wireless sensor network controlled by a centralized routing protocol such as Base Station Controlled Dynamic Clustering Protocol (Muruganathan et al. 2005) to conserve energy of the sensors thereby maximizing the lifetime of the sensor network is an example of such a system.

An important aspect of G-RTI that distinguishes it from other RTI implementations is its emphasis on reducing the energy needed to interconnect mobile devices, an obvious requirement for many wireless sensor networks. Such systems are generally composed of sensors which are battery operated and are generally not accessible after they are installed (Schwiebert et al. 2002). The energy constraint determines the operation time of each of these devices. It these scenarios, it is beneficial if the middleware itself considers the energy constrained nature of the system and seamlessly makes the system energy efficient without any specific intervention of the system developer(s). An important consideration is to allow for the use of G-RTI as a platform for development and implementation of such energy conservation strategies. This is discussed in greater detail later.

3 G-RTI SERVICES

G-RTI uses a client-server architecture. In the following it is assumed G-RTI executes at a computational node called the *G-RTI server*. G-RTI is not restricted to requiring a central server, however, this simplifies the discussion. A multi-layered server implementation may be used to meet scalability requirements.

Extending the client server paradigm, each of the computational nodes interacting with G-RTI, be it a data source (e.g. sensor or databases), a distributed simulation entity or other element is referred to as a *client*.

The functionalities implemented by the G-RTI server are referred to as *services*. The functionalities implemented at the client are referred to as *callbacks*. Finally, a client storing a data object is called the *owner* of the data object and all other clients are referred to as *non-owners* with respect to that data object.

As with any RTI there are rules that guides the implementation and use of the RTI. Three rules apply to the current implementation of G-RTI. First, unlike the HLA, data objects can be updated by non-owners. This has two major implications: it allows for an inclusion of clients such as sensors that have limited computation and storage capabilities; the data objects reflecting the state of the physical object sensed by such a sensor can be stored in a remote client. Further, this allows for multiple components of the system to operate on a shared data object. Second, the ownership of data objects can be transferred among the clients. This allows for migration of data among the clients and hence for a change in ownership where the hand-offs are seamless and invisible to other clients. The third design rule is the *thin-as-required* approach. This philosophy is in line with G-RTI's goal to implicitly allow for development and inclusion of clients with different constraints regarding computation and energy. This allows the clients to individually (independent of other clients in the system), choose to be as computationally and storage intensive as required. In other words, a client can choose how "thin" it wishes to be.

There are at present three major classes of services provided by the G-RTI: Management, Data Exchange and Time Management Services. Management Services inform G-RTI about the components of the distributed system. The *Join* service notifies the G-RTI server of the arrival of a new client in the system. The *Leave* service notifies the G-RTI server of the departure of a client. The *Register Object* service allows the client to register their data objects with the G-RTI server. Registration of objects allow other entities in the system to discover and interact with the data object.

Data exchange services allow for exchange of data among the participating system entities. The *Update Object Value* service allows a client to notify other clients of an updated value for the object. A client can request/read the value of an object owned by some other client using *Query Object Value* service. This falls under the Pull Based data exchange mechanism. G-RTI also allows for a subscription based data exchange mechanism which allows for the client to be notified of any updates to the subscribed data object. This type of data exchange, as mentioned earlier, is referred to as Push based data exchange. The *Subscribe Object Value* service allows clients to subscribe to data objects. As part of the push based data exchange, the *Notify Object* service allows the owner of a data object to notify G-RTI of updates to the object. The *Query Reply* service allows the client to reply to a read object value request.

The time management services allows G-RTI to support synchronization of simulations based on logical time among the distributed simulations and emulations in the system and are similar to the services defined in HLA. The *Time Advance Request* service is intended for time stepped simulations. The *Next Message Request* service is intended for event driven simulations to request for the next available message in logical time. It implicitly synchronizes the client with other simulations in the system. The *Time Advance Grant* callback notifies simulation clients that it is safe for them to advance to the requested logical time.

There are four other call back functions that are implemented by clients. *Read Object Value* callback is initiated by G-RTI as a response to a Query Object Value service invocation. G-RTI expects a value in return (the form of a query reply) which is passed to the client requesting the value. The *Reply* callback is initiated when the client's query has been answered by the object owner. The *Revise Object Value* call back is initiated by G-RTI as a response to an Update Object Value request, to the owner of the object. This callback is required by a client only if it owns a data object. G-RTI initiates the *Reflect* callback for all the subscribers of a data object, whenever there is a notification of an update to the data object. This callback is required by a client only if it is subscribed to the data object.

In keeping with the philosophy of allowing the clients to be as thin-as-required, these call back functions are not required, depending on the functionality of the client. For example a sensor or a simulator which is only sending data to other clients will not need any of these call back functions. On the other hand a data base client might implement all of these callbacks with cases specific to each object it owns or is interested

in receiving. To illustrate typical usage of these services, consider the system depicted in Figure 1. As discussed earlier, G-RTI supports push and pull based messaging. Figure 1 shows a push based messaging scenario. In this scenario, client 1 is a vehicle mounted mobile sensor relaying real-time location and velocity information while client 2 is one of several aerial drones in a swarm responsible for monitoring traffic. Client 3 includes a database that records updates and executes a predictive simulation using historical and online data. Finally *x* denoted an object containing vehicle information that is generated sporadically in an unpredictable fashion that is of interest to clients 2 & 3.

All clients join the system (asynchronously). Client 3 owns the data object and maintains its state. It informs G-RTI of its ownership of object *x* using the Register service. Client 2 is interested in updates to object *x* but does not own the object, so it subscribes to any updates to *x*. Finally client 1 is a source of generation of data object *x*. So whenever client 1 wants to convey a new value for *x*, it uses the Update service to inform the G-RTI of an update. G-RTI then forwards the update to the appropriate client (client 3). In addition, G-RTI also reflects the updated value to any subscriber (client 1).

Another scenario illustrating the push mechanism arises when the owner of the object itself generates an update for a data object. In the example scenario, the predictive simulation might want to fill in for any missing update. Using the previous mechanism that uses the update service of G-RTI would trigger a revise callback, which is redundant and might require extra effort on the application developer of client 3 to remove a possible update loop. G-RTI's notify service handles these cases. Client 3 notifies G-RTI of an update for object *x* and G-RTI reflects the update to all the subscribed clients.

We extend this scenario to illustrate use of pull based messaging. Client 2 would like to know the value of x. It queries G-RTI for this value, which then relays a read request to the owner of the object (client 3). The owner replies to G-RTI using the query_reply service. G-RTI then relays the reply to the requesting client. All these steps are executed asynchronously, which allows flexibility in the application development.

It might be noted that client 1 in this example may be resource constrained. The client only requires resources to sense/gather the required data and make an `http` request. Client 2 may be able to support more extensive computation, but might be constrained by providing only a limited amount of storage. Client 3 might be executing on a server with extensive computation and storage resources. Allowing the clients to be thin-as-required independent of other clients allows them to co-exist and form symbiotic relationships with other clients.

4 IMPLEMENTATION

Design choices in implementing the system were governed by the goals discussed earlier. Web interfaces were used to place minimal requirements and restrictions on clients and to simplify client-side application development. All services can be accessed using a URI, hence allowing for any device capable of connecting to the Internet to become a client. Apache was used because of its proven ability to scale to large deployments. CPPCMS is a C++ based framework which allows a native, high performance interface for inclusion of on-server applications. It was used to provide high performance and scalability. Popular packages and simulation applications used for development and study of networked and distributed systems are based on C++. Finally Apache and CPPCMS are both free and open-source software platforms, simplifying widespread distribution of G-RTI. Apache is released under Apache License 2.0 and CPPCMS with an LGPLv3 license and alternative Commercial License as an alternative for proprietary software development.

Push based messaging is a not a native operation for network communication be it socket or web based. G-RTI implements push based messaging using continuous polling. Simply put, continuous polling is maintaining at least an instance of client request for each client in G-RTI. As can be easily seen, with finite resource there will be a time the server runs out of instants for a client. The state for the client must be reset to polling before any further messages can be pushed to the client. We call this period the resetting period and discuss it further in the benchmarking section.

Another feature of G-RTI is a dynamic web-dashboard. The dashboard allows remote monitoring of the system components and eases system development and debugging. Currently the dash board provides two

views, a client level view that list all the information pertaining to a client, and a data object level view that presents information organized by objects.

5 BENCHMARKING EXPERIMENTS

We next present results from micro-benchmarking experiments to evaluate the performance of the G-RTI services. This study focused on two major performance metrics of the middleware implementation. One is the latencies of the messaging primitives provided by G-RTI. The second concerns the maximum bandwidth of communications for each G-RTI client to update a shared object.

The setup for the benchmarking experiments consisted of an implementation of G-RTI executing on a Lenovo ThinkPad T410s laptop with a Ubuntu 14.04 LTS operating system. The setup also included two (three in case of subscription latency) python clients, on the same system. The choice was made to avoid time synchronization issues for computing subscription latency. All other results were replicated on the same platform to maintain uniformity for comparison purposes. All the experimental values presented represent the average of 100 trials.

We define subscription latency as the time between the instants when an update is sent until a subscribed client receives the update. The subscription latency was computed by using three clients in a scenario as shown in Figure 1. The implementation follows the first push based message scenario described earlier. Briefly, client 1 sends an update to an object owned by client 3 and client 2 is subscribed to the object and hence receives a reflect call back. The subscription latency is computed as the difference in the time instant when the update request to the object starts in client 1 and the instant when client 2 successfully completes receiving the reflect callback. As the clients are on the same system the time difference is not affected by any clock synchronization. This is then repeated for varying sizes of update messages.

We define the query round trip latency as the minimum time from when a client sends out a query for an object until the time it receives a reply from the G-RTI server. As noted earlier, query is an asynchronous operation and hence the query latency can vary depending on how the clients are set up. For computing the query latency the experimental setup and the implementation is the same as described in the scenario for pull based messaging discussed earlier. Briefly, client 2 queries G-RTI for the value of the object owned by client 3, which then receives a read callback from G-RTI and replies back to the query. Finally client 2 receives a reply from G-RTI. The latency is computed as the time difference between the instants when client 2 starts sending the query and the instant when it finishes receiving a reply back. Again this is repeated for varying sizes of object values.

We define update round trip latency as the time between the instant when a client sends an update to a peer client and the instant when the client receives an update from the peer client. To compute the latency, client 1 sends an update to an object owned by client 2 and vice-versa. The round-trip includes an instance of the resetting period. The resetting period was empirically determined to be approximately 4 milliseconds. Again the update length was varied.

Figure 2 shows the results of the latency benchmarking experiments. The subscription latency is the least and this can be attributed to the fact that it requires one way communication. Both query and update round-trip times include one resetting period which is a constant irrespective of the message size. Furthermore, the change in size of the message affects only the reply path of the query latency, i.e., the size of the reply changes. The other path is not affected as the size of the query requests stay constant. This explains a lower slope of the line for the query latency with respect to the update round-trip latency and a comparable slope for subscription latency. Finally the difference in the slopes for update-round trip latency and the query latency can be easily accounted for when the resetting period is considered as a constant in both. The constant slopes signify a linear increase in latency as the message size is varied. These results show a very predictable behavior for G-RTI with respect to the latency for the messaging services. Because the subscriptions, objects and clients are managed as hash maps which provide a constant time look up, an increase in the number of any of these is not expected to affect the latencies.

The maximum bandwidth excluding http and lower layer headers was measured to be approximately 3.7 Mbps (and 2.21 Mbps when the resetting period is taken into account). These were computed without

any manipulation of the default settings of the Apache web-server, so should be considered to be conservative values. The bandwidth is currently limited by the size of the request that Apache and FASTCGI allows. Theoretically the request size can be set to any arbitrary number. In that case the limiting factor is the network hardware and software stack. Even so, the bandwidth is comparable with other web-based RTIs (Möller and Dahlin 2006).

The throughput and latency is implementation dependent, making direct comparison difficult. However, widely used micro benchmarks are reported in the literature. E.g. one can compare the results presented here with those presented in Cardoso et al. (2017). G-RTI performs significantly better than the presented middleware implementations on all counts.

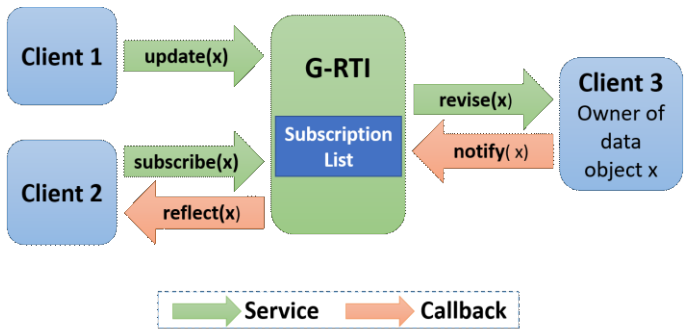

Figure 1: Push based message service usage scenario.

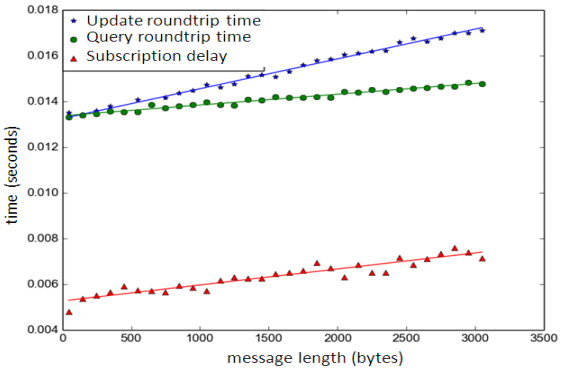

Figure 2: G-RTI messaging primitive latencies.

6 ENERGY CONSUMPTION

Space does not permit extensive discussion of the energy consumption characteristic of G-RTI here, however we cite on-going work in this area. Our previous work (Fujimoto et al. 2017) presented an experimental study to examine the use of message aggregation to reduce energy consumption in clients. In this study a cellular automata based traffic simulation was used (Rickert et al. 1996), that was configured to model several road segments of Peachtree Street in midtown Atlanta. The simulation (executing on a smartphone) sends a stream of vehicle position data via G-RTI over Wi-Fi to a server (executing on a laptop). To study the effect of the aggregating messages on the energy consumption of the client we compute the energy consumed for varying level of aggregation. In each experimental run, the number of updates, and the size of the data in each update remains constant and the only variable is the number of updates aggregated to form a message. Figure 4 shows the amount of energy consumed per byte of transmitted data. As expected, an initial reduction in the energy consumed is observed as message aggregation increases. Thus, from the standpoint of energy consumption, a rule of thumb could be that data aggregation is very effective. It shall be noted that depending on the message aggregation strategy used, it can introduce latency in message delivery.

Other recent work (Biswas and Fujimoto 2018) examines energy-efficient implementation of the time management services. In particular, a low energy version of the well-known YAWNS synchronization algorithm termed LEY as developed. Preliminary experimental data using benchmark simulations indicate that LEY uses significantly less energy than an energy-oblivious implementation of YAWNS, and requires only a modest amount of energy beyond the minimum required to execute the distributed simulation.

7 CASE STUDY: DDDAS USING G-RTI

Crowd sourcing, mobile computing and a wide variety of public sensor based intelligent/smart systems have made a many varieties of data available for transportation systems. Combined with the emergence of

autonomous vehicles and smart cities (Shueh 2018), intelligent and feedback based systems for transportation system operations and management are gaining in importance. A recent development in the field is the standardization of a message set for the DSRC (Dedicated Short Range Communication) to provide interoperability among DSRC applications. DSRC as defined (Sill 2018) by the United States Department of Transportation (DoT), is a two-way short- to- medium-range wireless communications capability that permits very high data transmission critical in communications-based active safety applications. One of the messages defined in the message set dictionary is the basic safety message (BSM).

Applications such as that proposed in [DSRC app 2018] derive weather information from mobile sensors on the vehicles transmitted as part of the BSM message to assist the management of the roads under adverse weather conditions. Also the vehicle operation information present in Part A of the BSM message can be utilized for user applications such as vehicle emission modeling proposed in Guensler et al. (2017) and energy efficient route prediction. These applications use the vehicle mounted sensors as a network of mobile sensors to generate an understanding of a larger system, which is then used to provide added benefits to the users. Similar DDDAS based traffic systems were studied in (Fujimoto et al. 2006; Hunter et al. 2009).

Development of these applications is not straightforward. Because they rely on the BSM message to capture and predict future states of the system, it is important to understand and consider the uncertainties of mobile wireless communications. Packet drops, e.g., due to network congestion, can greatly affect the services built on top of such networks. Early evaluation and development coupled with the need for reproducible results necessitate the use of simulations operating in controlled environments such as the developer's laboratory. Simulation models of vehicle traffic and wireless communication are needed, but detailed, high resolution models of both are seldom available within the same simulation package, calling for an integrated approach. And to natively test software applications the developers must also interface the application with the set of simulators which can be time consuming. This is further complicated if the application needs to be studied in a real-time environment where an emulation capability is needed. Often these efforts are specific to the study at hand and cannot be reused.

Figure 3 depicts a DDDAS application for transportation system management. In this system a traveler assistance app is depicted that executes on a smartphone. It utilizes updates from a back-end server

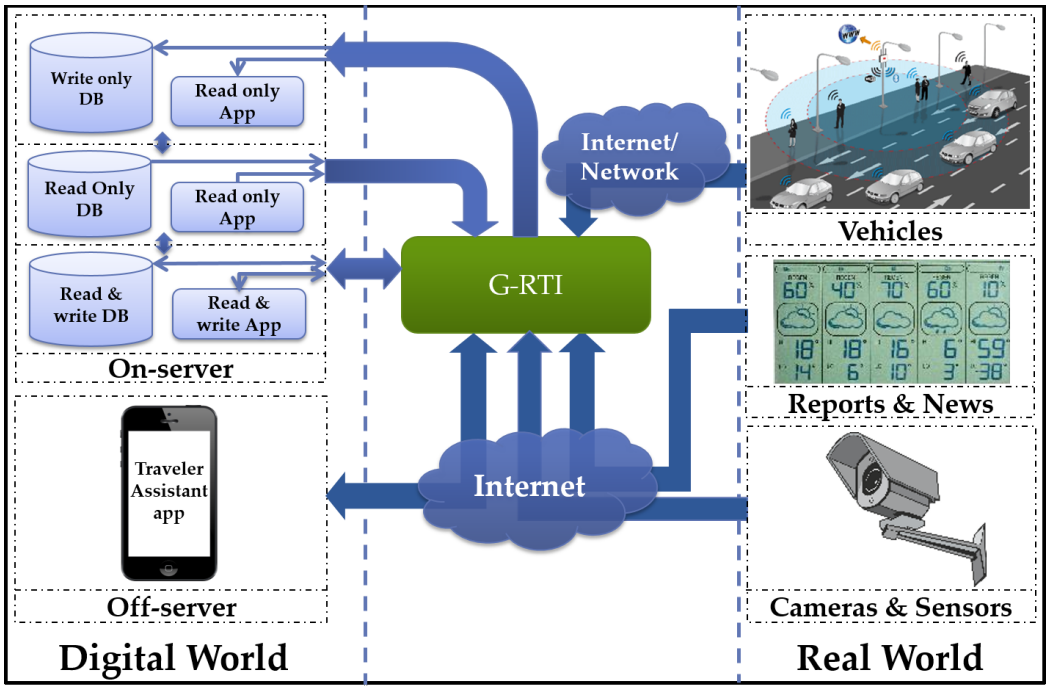

Figure 3: Notional diagram of DDDAS for case study. (V2X image: © Zoloo777) CC BY-SA 4.0 license.

application. The server application relies on BSM data communicated among a network of mobile sensors mounted on the vehicles augmented with other inputs such as weather data to deliver information relevant to the app's user, e.g., data such as travel time and fuel consumption concerning the driver's planned route as well as that for alternate routes.

In this study we simulate/emulate a road network with connected vehicles, communicating BSM messages. In addition to showing an end-to-end implementation using GRTI, this study also provides a close to real world example of how G-RTI can be used to integrate different simulations. The traffic simulator models the mobility of the vehicles. A communications network simulator models information exchanged between vehicles and roadside base stations. Here, we utilize two widely used simulators, SUMO (SUMO 2018) a traffic simulator suite and NS3 (NS3 2018) a network simulator. The road network (Sumo Network 2018) has a total edge length of 17.84 Km and a total lane length of 49.53 Km. The network has four origins, four destinations and two intersections. Each road for outbound traffic has three lanes. U-turns are prohibited at all intersections. A higher priority has been given to eastbound and westbound traffic. Vehicle data is output in real-time in floating car data format. The format includes the location of all the vehicles in the simulation in addition to other vehicle data. The simulation using NS3 models moving nodes (vehicles) on a "dynamically specified" path, communicating BSM messages over WAVE. This can be broken down to two major parts, first the mobility simulation for the nodes in NS3 and second the BSM communication simulation.

The mobility for the nodes in NS3 was implemented using NS3's waypoint mobility model augmented with support for dynamic inclusion and removal of vehicle during the simulation. The waypoint mobility model requires the position of the vehicle and the time-stamp at which the vehicle's position was recorded (in this case provided by traffic simulation using SUMO). The waypoint mobility model can be thought of as a list of vehicle position and timestamp data. Hence it requires at least one time-step into the future. This was achieved by simulating SUMO a time-step ahead of NS3 (Eichler et al. 2005). Another constrain of NS3 is that it does not allow dynamic creation of nodes. The dynamic inclusion and removal was made possible by recycling of NS3 nodes and assuming the number of vehicles in the simulation at an instant has an upper bound. This was facilitated by the *max-num-vehicles* argument of SUMO. Another part of the implementation was that the requirement of dynamic paths for the nodes required the vehicle trace to stream to NS3 in real-time from SUMO. This was achieved by using TAP device, which is exposed on one end as a kernel net device and on the other end as a file descriptor in user space. The user space file descriptor can then be passed on to `FdNetDevice` of NS3.

The next part is to model BSM communication for the nodes. The BSM communication was simulated with a packet size of 200 bytes (Sung et al. 2013) with a frequency of 1 Hz with an expected range of 1000m (Cronin 2018) without channel switching and with a max random delay of 10ms before transmitting over WAVE. The simulation was implemented using the `WaveBSMhelper` application class of NS3. Figure 5 shows the effect of increasing the number of nodes in NS3, the vertical axis presents the wall clock time taken by NS3 to simulate one simulation second. The quadratic behavior of the curve can be explained by the nature of the broadcast. In this scenario the limiting factor for achieving real-time performance is the execution time of NS3. This system was developed and tested on Ubuntu 14.04 LTS. It must be noted that the simulations can be executing on other Operating systems as well e.g. VISSIM a proprietary windows based simulator could be used to replace SUMO in this study. A view of the simulated road network is shown in Figure 6. Traces for the vehicles on the emulated road network were generated by SUMO. They are parsed by a python script and passed on to another python script through G-RTI which then relays it to NS3 through the FD-TAP interface. Figure 7 illustrates one frame in an animation of the mobility simulation and network simulation outputs. Although the simulations are run in real-time and concurrently, the NS3 visualization was captured offline due to the lack of a good online visualization program for NS3.

8 RELATED WORK

Many middleware approaches have been proposed in the past. Some support specific types of scenarios and some are more generic (Aberer et al. 2006; Yu et al. 2004). Middleware approaches such as Aberer et al.

(2006) allow reconfiguration of nodes in a live system but require an XML interface to interface the nodes, restricting dynamic changes in the computational resources of the nodes. These middleware approaches assume sensors are very resource constrained; the approach proposed here assumes a mix of devices with varying degrees of resources available to them. It is up to the system developer to assign computational tasks depending on resource availability, which can be static or vary dynamically.

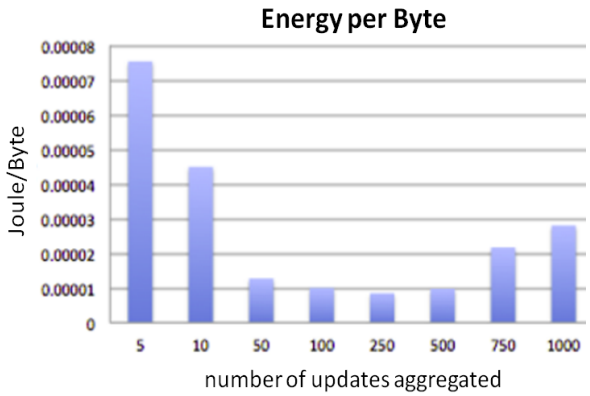

Figure 4: Effect of energy consumed by a client as the number of messages aggregated is altered.

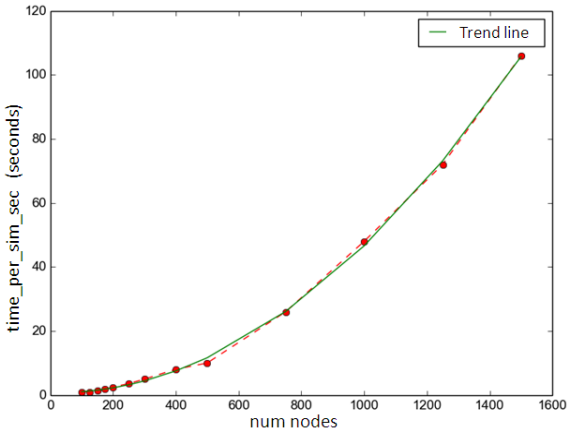

Figure 5: Effect of changing number of Nodes in NS3 on wallclock time versus simulation time.

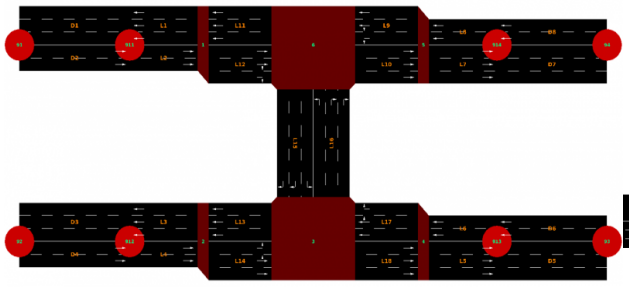

Figure 6: View of simulated road network (not drawn to scale)

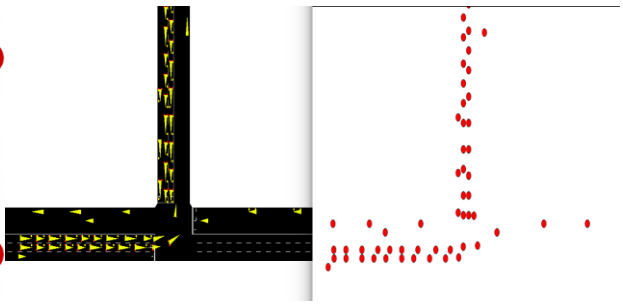

Figure 7: Graphical outputs of SUMO (left) and NS3 (right) at same simulation time.

Approaches such as (Cugola et al. 2002; Soini et al. 2007) present middleware that are based on a publish/subscribe method for communication rather than pull-based messaging. (Heinzelman et al. 2004; Yu et al. 2004) proposed a peer-to-peer middleware approach for sensor networks. Soini et al. (2007) investigates and argues the advantages of a centralized approach over peer-to-peer middleware for sensor networks, citing reasons such as faster synchronization, discovery, routing, and data dissemination. Our experiences thus far with both centralized and peer-to-peer middleware are consistent with these comments. In addition, the centralized approach is better able to support thin-as-required clients compared to peer-to-peer approaches. Another advantage of a centralized middleware is it is more natural in supporting the integration of on-server applications.

9 CONCLUSION AND FUTURE WORK

This work described G-RTI, a middleware to support DDDAS developments consisting of heterogeneous components. G-RTI provides a common platform for research, emulation and deployment of systems. Major design goals include support of a wide variety of devices and application contexts, reduced

development effort, providing a platform for energy efficient deployment and the possibility of thin-as-required clients. The suite of services offered by G-RTI was developed from our experience in the design and implementation of distributed systems ranging from purely analytical system to systems with real world components. We consider G-RTI as toolbox where the system developer has the choice of using the tools that they find most efficient for the specific system development. The services offered by G-RTI and the expected callbacks were described as well as implementation details and usage scenarios. Performance measurements from a benchmarking study characterize the delays required for various messaging services and some energy consumption properties. An important goal for G-RTI is to improve energy efficient transparent to the application. Toward this end strategies to reduce energy consumption and integrate it in the G-RTI platform are under development. Current work is also focusing on developing energy efficient approaches to realizing time management services for distributed simulation. Finally, another area of future work aims at making a testbed based on G-RTI available for researchers to study and develop application that interact with real-world components.

ACKNOWLEDGMENTS

Funding for this research was provided by Air Force Office of Scientific Research Award FA9550-17-1-022 and Air Force Office of Scientific Research / National Science Foundation Award 1462503. V2X image (Fig. 5) adapted from Zoloo777 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=54463824. CCTV camera, phone and weather image from https://pixabay.com (CC0). SUMO road network (Figure 6), source http://sumo.dlr.de/wiki/Tutorials/quick_start, CC BY-SA 3.0.

REFERENCES

Aberer, K., M. Hauswirth, and A. Salehi. 2006. "The Global Sensor Networks Middleware for Efficient and Flexible Deployment and Interconnection of Sensor Networks." Technical report No. LSIR-REPORT-2006-006. Ecole Polytechnique F´ed´erale de Lausanne (EPFL), Lausanne, Switzerland

Biswas, A, R. Fujimoto. 2018. "Zero Energy Synchronization of Distributed Simulations." *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 85–96, New York, NY: ACM.

Cardoso, J., C. Pereira, A. Aguiar, and R. Morla, 2017. "Benchmarking Iot Middleware Platforms." In *IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks*, Macau, China, 1–7.

Cronin, B. 2018. "Vehicle Based Data and Availability." Accessed April 1st, 2018. http://www.its.dot.gov/itspac/october2012/PDF/data_availability.pdf.

Cugola, G., H. Jacobsen. 2002. "Using Publish/Subscribe Middleware for Mobile Systems." *ACM SIGMOBILE Mobile Computing and Communications Review* 6(4):25–33.

Darema, F. 2004. "Dynamic Data Driven Applications Systems: A New Paradigm for Application Simulations and Measurements." *Int. Conf. on Computational Science,* Berlin, Germany, 662–669.

DDDAS. 2018. "Dynamic Data Driven Application Systems." Accessed 1st April, 2018. http://1dddas.org/

DSRC app. 2018. "5.9 Ghz Dedicated Short Range Communication Vehicle-Based Road and Weather Condition Application Messaging Requirements." Accessed April 1st, 2018. http://www.cts.virginia.edu/wp-ontent/uploads/2014/04/PFS_DSRC02_Task1_Messaging_Reqs_007-101-02.pdf.

Eichler, S., B. Ostermaier, C. Schroth, and T. Kosch. 2005. "Simulation of Car-to-Car Messaging: Analyzing the Impact on Road Traffic." In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Atlanta, USA, 507–510.

Fujimoto, R. M., R. Guensler, M. Hunter, H.-K. Kim, J. Lee, J. Leonard III, M. Palekar, K. Schwan, and B. Seshasayee. 2006. "Dynamic Data Driven Application Simulation of Surface Transportation Systems." *Workshop on Dynamic Data Driven Application Simulations*, Berlin, Germany, 425–432.

Fujimoto, R. M., M. Hunter, A. Biswas, M. Jackson, and S. Neal. 2017. "Power Efficient Distributed Simulation." *ACM Conf. on Principles of Advanced Discrete Simulation*, Singapore, 77–88.

Fujimoto, R. M, Barjis, J, Blasch, E, Cai, W, Jin, D, Lee, S and Son, Y. 2018. "Dynamic Data Driven Application Systems: Research Challenges and Opportunities." In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe et al.. Piscataway, New Jersey: IEEE.

Guensler, R., H. Liu, Y. Xu, A. Akanser, D. Kim, M. P. Hunter, and M. O. Rodgers. 2017. "Energy Consumption and Emissions Modeling of Individual Vehicles." *Transportation Research Record: Journal of the Transportation Research Board* (2627):93–102.

Heinzelman, W. B., A. L. Murphy, H. S. Carvalho, and M. A. Perillo. 2004. "Middleware to Support Sensor Network Applications." *IEEE Network* 18(1):6–14.

Hunter, M., H.-K. Kim, W. Suh, R. M. Fujimoto, J. Sirichoke, and M. Palekar. 2009. "Ad Hoc Distributed Dynamic Data-Driven Simulations of Surface Transportation Systems." *Transactions of the Society for Modeling and Simulation Intl.* 85(4):243–255.

Möller, B., and C. Dahlin. 2006. "A First Look at the Hla Evolved Web Service API." *Proceedings of 2006 Euro Simulation Interoperability Workshop*.

Muruganathan, S. D., D. C. F. Ma, R. I. Bhasin, and A. O. Fapojuwo. 2005. "A Centralized Energy-Efficient Routing Protocol for Wireless Sensor Networks." *IEEE Communications Magazine* 43(3):S8–13.

Ngu, A. H., Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Quan Z. Sheng. 2017. "Iot Middleware: A Survey on Issues and Enabling Technologies." *IEEE Internet of Things Journal* 4(1):1–20.

NS3. 2018. "Ns3." Accessed April 1st, 2018. https://www.nsnam.org/.

Rickert, M., K. Nagel, M. Schreckenberg, and A. Latour. 1996. "Two Lane Traffic Simulations Using Cellular Automata." *Physica A: Statistical Mechanics and its Applications* 231(4):534–550.

SAE J2735. 2018. "Dedicated Short Range Communications (Dsrc) Message Set Dictionary." Accessed April 1st, 2018. https://www.standards.its.dot.gov/Factsheets/Factsheet/71.

Schwiebert, L., S. K. S. Gupta, P. S. G. Auner, G. Abrams, R. Iezzi, and P. McAllister. 2002. "A Biomedical Smart Sensor for the Visually Impaired." *Proceedings of IEEE Sensors*.

Shueh, J. 2018. "Atlanta Smart Corridor Evolves into a Springboard for Autonomous Vehicles, Iot and More." Accessed April 1st, 2018. http://smartatl.atlantaga.gov/index.php/blog-post/atlanta-smart-corridor-evolves-into-a-springboard-for-autonomous-vehicles-iot-and-more/.

Sill, S. 2018. "Dsrc: The Future of Safer Driving." Accessed April 1st, 2018. https://www.its.dot.gov/factsheets/dsrc_factsheet.htm.

Soini, M. N. K., J. Van Greunen, J. M. Rabaey, and L. T. Sydanheimo. 2007. "Beyond Sensor Networks: Zuma Middleware." *Wireless Communications and Networking Conf*. Kowloon, China, 4318–4323.

Souto, E., G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. 2004. "A Message-Oriented Middleware for Sensor Networks." *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad-hoc Computing*. Toronto, Canada, 127-134.

SUMO. 2018. "Sumo." Accessed 1st April, 2018. http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/.

Sung, S., D.-g. Noh, and J. Park. 2013. "Development of Reliable V2v System Based on Wave." *23rd International Technical Conference on the Enhanced Safety of Vehicles (ESV)*, (No. 13-0474).

Yu, Y., B. Krishnamachari, and V. K. Prasanna. 2004. "Issues in Designing Middleware for Wireless Sensor Networks." *IEEE Network* 18(1):15–21.

AUTHOR BIOGRAPHIES

ARADHYA BISWAS is a Doctoral student at College of Computing at Georgia Institute of Technology. He received his undergraduate degree in Computer Science and Engineering from the Indian Institute of Technology Hyderabad. His email address aradhya.biswas@gatech.edu.

MICHAEL HUNTER is an Associate Professor at the School of Civil and Environmental Engineering at the Georgia Institute of Technology. He received his Ph.D. in Civil Engineering from The University of Texas at Austin. His email address is michael.hunter@ce.gatech.edu.

RICHARD FUJIMOTO is a Regents' Professor in the School of Computational Science and Engineering at the Georgia Institute of Technology. He received a Ph.D. in Computer Science & Electrical Engineering from the University of California-Berkeley in 1983. His e-mail address is fujimoto@cc.gatech.edu.