Model-based Design of a Roadside Unit for Emergency and Disaster Management

Nur Hilal
Computer Engineering Department
Boğaziçi University
Istanbul, Turkey
nour.hilal@yahoo.com

Arda Yurdakul
Computer Engineering Department
Boğaziçi University
Istanbul, Turkey
yurdakul@boun.edu.tr

***Abstract*—Every year, a massive number of deaths happen because of traffic accidents. In order to increase the traffic victim's survival rates, it is important to reduce the arrival time of medical and trauma intervention teams to accidents' sites. Automatic incident detection provides faster incident reporting to decrease the delay of arrival time of first responders. In this paper, we propose a roadside unit (RSU) that automatically detects traffic incidents using multiple detection mechanisms, verifies to reduce false alarms and reports to the intelligent traffic management system. The proposed RSU is modeled in Architecture Analysis and Design Language (AADL). Simulation results show that our RSU model is schedulable with low processor utilization factors, and provides incident detection and reporting in under three minutes.**

***Index Terms*—Roadside Units, Incident Detection, Intelligent Transportation System, VANET, Model-Based Engineering, AADL, OSATE.**

I. Introduction

Road traffic accidents are one of the leading causes of death around the world. According to global status report on road safety of World Health Organization (WHO) in 2018, 1.35 million deaths happen each year because of road accidents [1]. To increase the survival rates of trauma victims in road accidents, they need to get rescued, transferred and undergo the appropriate medical procedures within an hour, which is known as the Golden Hour [2]. The first fifteen minutes of Golden Hour is extremely important [3]: the accident detection and reporting should not take more than 5 minutes, and the arrival of medical teams should be satisfied within 15 minutes.

For faster detection of traffic accidents, a new emerging concept known as Intelligent Transportation System (ITS) provides promising studies for incident detection [4]. One of the studies proposes a multi-layer reconfigurable disaster and emergency management architecture [5]. Their architecture consists of three layers: Sensor Processing layer, Intelligent Data Processing layer, and Cloud Processing layer. The Intelligent Data Processing layer consists of small yet computationally powerful units, also known as Roadside Units (RSUs), that are responsible for processing the incoming data from the sensors and provides multiple services such as traffic flow control, local routing, and incident management.

978-1-7281-4973-8/20/$31.00

Automatic Incident Detection (AID) is one of the leading research topics in Intelligent Transportation Systems (ITS) [6]. Some of these studies depend on traffic sensors [7]. In [8], incident detection is done by using data from ultrasonic sensors. In [9] [10], inductive loops that exist on the road are used. There are also existing studies that utilize cameras [11]. Since ILDs are already implanted in the roads for traffic engineering and management purposes, they are more preferable over other road sensors because they cost less and tolerate most weather conditions. Cameras, on the other hand, are very expensive to associate with each RSU for incident detection. To the best of our knowledge, none of these studies uses multiple incident detection techniques and different communications protocols to provide traffic incident detection and reporting services. In this work, we propose a model based engineering method to enhance RSUs for the real-time detection and verification of incidents by fusing multiple detection methods and services that already exist in the literature.

Creating such complex systems takes a significant amount of time and effort so that it can be extensively tested and simulated before it can be implemented. Model-Based Engineering (MBE) provides early identification of requirements' issues, higher system design integrity and early detection of system design errors. Hence, in this paper, we provide the RSU components and internal architecture using a modelling language known as AADL [12]. We provide testing and simulations experiments, such as schedulability and latency of the RSU model and its tasks using different tools.

In this paper, we combine four methods for incident detection. Two of them are incident detection algorithms that enables on-RSU incident detection from incoming data: the first algorithm is developed on Inductive Loop Detectors (ILDs) [9], and the second one makes use of the collected data through Vehicle Ad-hoc Networks (VANET) [13]. For the other two methods, our RSU takes the role of the server side of two different accident detection mechanisms to listen for incidents detected by other components in the ITS: eCall [14] and WreckWatch [15]. We use these four different sources to verify and confirm detected incidents so as to minimize false alarms. To achieve this, we designed a verification and confirmation process that runs in our RSU. Our proposed system is shown in Fig. 1.

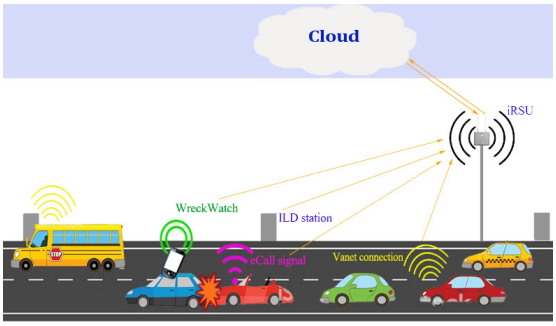

Fig. 1. The proposed system

In this paper, we start with section II by providing a summary about the incident detection methods that we are using in our system. In section III, we explain our methodology and system proposal with insightful details about the internal architecture and components of our RSU. We walk through the experimental work and simulation tests in section IV. In the final section, we conclude our study and talk about future work.

II. Related Work

One of the research papers on AID is based on Inductive Loop Detectors (ILDs) [9]. They proposed an approach of two Fuzzy Inference Systems (FIS) which study traffic data collected by two loop detector stations. The two FIS are assumed to be at the upstream and downstream of the incident location. The proposed system compares normal traffic conditions with current traffic conditions of the same time of the day to find differences. When these differences between normal and observed traffic conditions are "low", then the system is in "normal state", otherwise, it is in "abnormal state". If the abnormal state is detected for a certain amount of time (i.e. two minutes), the algorithm prompts an incident alarm.

Vehicular communication technologies can also help in AID. A study, which provides a solution based on Vehicular Ad-hoc Network (VANET) [13], proposes two AID algorithms that take collected traffic data from passing vehicles to analyze traffic flow and find anomalies related with traffic incidents. These algorithms depend on lane changing information that the passing vehicles share with the RSU, such as vehicles' speed and travelled distance. When changing lanes, these values are significantly impacted in the case of accident existence. The idea behind their algorithms is to aggregate all traffic data related with passing vehicles changing lanes, and constantly monitor the average values of these traffic variables to establish belief measurements of the existence of a traffic incident. An incident alarm will be issued if these belief measurements exceed a pre-established threshold.

In a different approach, there are some other incident detection solutions propose the detection of a traffic accident based on the direct connection to Vehicles' and Smartphones' sensors. One of these solutions is eCall [16]. eCall is an emergency system that provides an automatic incident detection of road accidents and initiates a telecommunication service with the related Public Safety Answering Point (PSAP). eCall system consists of two parts, an In-Vehicle System (IVS) and an eCall data modem installed at all PSAPs. In the event of a traffic accident, the IVS unit detects the accident using some built-in crash sensors within the vehicle, and establish a 112-voice call automatically (or manually by the driver), to the nearest PSAP center through cellular networks. Even if the driver is unable to talk, the IVS will send a Minimum Set of Data (MSD) to the PSAP center to notify authorities with the accident information such as the vehicle location, vehicle identification, number of passengers and other related information [14].

Another study discusses detecting traffic accidents using smartphones [15]. The availability and low cost of smartphones against other means of incident detection and reporting mechanisms makes them a favorable alternative solution. In addition to their wide-spread usage among users, smartphones travel with their owners which provides incident detection whether or not the vehicle is equipped with an IVS unit. In [15], the authors propose a smartphone-based prototype of a client/server application called "WreckWatch". The WreckWatch client is an application installed on the user's smartphone, where it exploits the built-in devices such as GPS, microphone and accelerometer to detect a traffic accident when it happens. The client relays the accident information to the WreckWatch server, which provides data gathering and communication services for first responders, family and friends.

III. AADL Model of RSU for Incident Detection and Verification

In this study we compose incident detection part of our RSU by using some of the methods that are described in Section II. Since our system is complex and has to function in real-time, we chose AADL as the modelling language to present the RSU architecture and study different deployment options.

Our first method utilizes ILD based incident detection algorithm [9]. We consider a 1 km road of two lanes and divided into 10 sections with an ILD at each section per lane. These ILDs will be connected to our RSU through RS485 cables, where they send their traffic variables every one minute time interval. After receiving these data, the RSU will execute the algorithm to detect if there was an incident or not. The second method uses a detection algorithm that collects data from passing vehicles through VANET and process them to find anomalies in the observed traffic variables [13]. The other two detection mechanisms are eCall [14] and WreckWatch [15]. Our RSU will play the server side that listens to accident signal coming from these channels. For eCall, we propose that the PSAP data modem is integrated in the RSU where the IVS will send the eCall MSD directly to the RSU and the RSU will handle the data and pass it automatically to the cloud. We believe that with adding this technique, we provide faster response through automatic reporting and communication. Similary, the server side of the WreckWatch application will

be in our RSU, where it listens to and handles the incoming WreckWatch data, from the client side, processes and passes them automatically to the cloud.

After an incident is detected on the RSU, we propose a verification method to screen false alarms. This method depends on confirming the detected incident by receiving a second incident alarm from a different detection mechanism that matches the same information.

The AADL model of our RSU is decomposed into four units (processes): Input, Detection, Processing, and Output units. The Input unit receives the incoming data and sends it to its related parties, while the Detection unit is responsible for incident detection depending on current traffic variables. The Processing unit takes all the incoming incident alarms and apply the verification technique to confirm them. In the meantime, the Output conveys messages to EVs and passing vehicles in addition to sending confirmed incident information to the cloud for classification and routing plans.

1) ***Input unit****:* Our RSU has to process inputs from four different channels that are expressed with four threads shown in Fig 2: ild_Listener, VanetFilter, WirelessFilter, and eCallPoll.

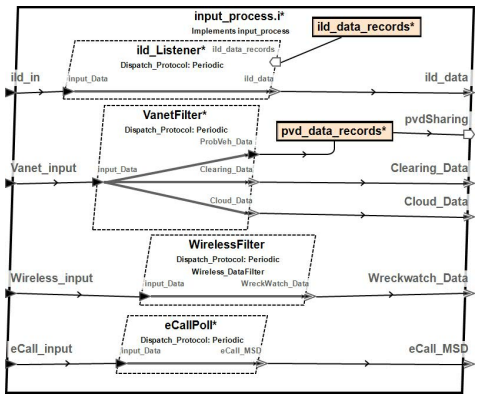

Fig. 2. Input Unit's AADL architecture

The *ild_Listener* is a periodic thread that listens to the ILD devices connected to our RSU. Since the loop detector devices need a whole minute to produce meaningful data, the period of this task should also be one minute, to avoid unnecessary task switching delay. The *ild_Listener* aggregates all the incoming data into one data component, *ild_data_records*. Then, it issues an event and send the aggregated data, through *ild_data* port, to start its detection algorithm.

The *VanetFilter* is a periodic thread that receives and filters incoming cloud data messages and probe vehicles data (PVD) messages through VANET's connection. The thread first has to differentiate between a cloud message and a vehicle message by its header. The cloud message is either a routing message or a cleared incident notification. An event is issued to dispatch the routing thread or the verification thread, respectively, and passes it the message. If the incoming data is a PVD message from a connected vehicle, it will be recorded into a circular queue in the data component *pvd_data_records* until it accumulates 24 records. When new data comes, it replaces the oldest data in the data component.The *eCallPoll* thread is also a periodic thread. It listens to incoming eCall signals. Once an eCall signal arrives, the thread issues an event and sends the data to the eCall handler task in the processing unit. The *WirelessFilter* is also a periodic thread that listens to incoming WreckWatch data coming through the wireless connection. When a WreckWatch message is received, the wirelessFilter issues an event to dispatch the WreckWatch handler, in the processing unit, and passes the message to it. The period of *VanetFilter*, *WirelessFilter* and *eCallPoll* are set as 100 ms.

2) ***Detection Unit****:* The Detection unit is responsible for detecting an incident by studying the data coming from *ild_Listener* and *VanetFilter* as shown in Fig 3.

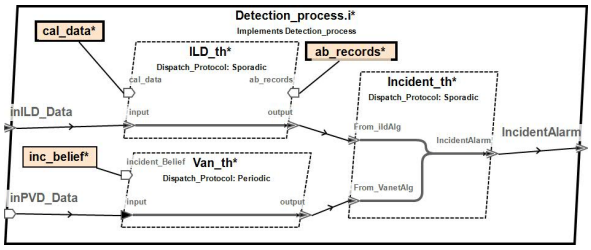

Fig. 3. Detection Unit's AADL architecture

The *Van_th* applies the incident detection algorithm on VANET's traffic parameters [13]. It accesses *pvd_data_records* through *inPVD_Data* port. Once the Van_th thread is dispatched, it reads the accumulated data in the *pvd_data_records* to execute the detection algorithm that increases an incident belief variable, *inc_belief*, in case that it detects an anomaly. If the incident belief exceeds a predefined threshold, the thread issues an event and sends the incident data to *incident_th*.

The *ILD_th* thread is sporadic, and it is dispatched once an event comes on the *inILD_data* connection. It receives the aggregated data coming from the *ild_Listener* in the input unit also through the *inILD_data* connection. This thread applies the incident detection algorithm based on ILD [9]. It uses two data components *ab_records*, and *cal_data*. The latter is the calibration data that was initially defined for specific times in a day of the week when no incident was present. The algorithm uses calibration data to compare with the current traffic variables to detect anomalies. Once an anomaly is detected, the lane and road section data are recorded in the ab_records, which contains a list of all the road sections and lanes that had presented an anomaly in the previous execution only. So, if an anomaly presented itself in the current execution and matched a previous one, it means there is an incident to report. Then, the thread issues an incident alarm event and passes the incident data to the *incident_th* task.

The Incident_th thread is basically the collector of incident alarm information from the preceding two threads. It records the information in a unified representation and passes it with an event to the verification task in the processing unit. The Incident_th thread is also sporadic.

*3) **Processing Unit**:* Fig. 4 shows the processing unit that wakes up when an incident alarm is issued. It handles all the incident detection inputs to verify the detected incidents.

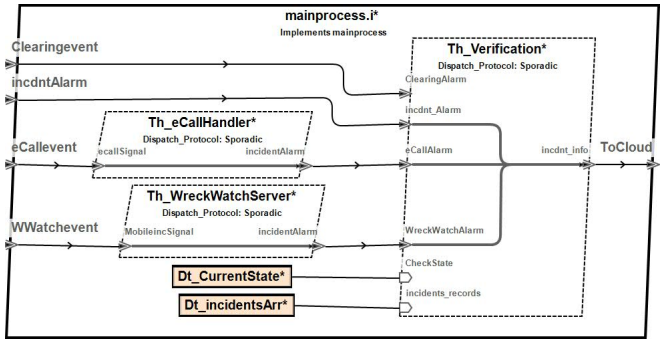

Fig. 4. Processing Unit's AADL architecture

Th_eCallHandler thread is dispatched when the *eCallPoll* in the input unit issues an event. It handles the MSD data coming from the input unit and transfers it in a unified form to the *Th_verification*. *Th_WreckWatchServer* [15] handles the coming data from the *WirelessFilter* and sends the data in a unified form to the *Th_verification* thread. *Th_WreckWatchServer* and *Th_eCallHandler* are sporadic since they only works when WreckWatch or eCall signal, respectively, arrives.

Th_Verification task is sporadic and is dispatched every time an incident alarm event is issued by either *Incident_th, Th_eCallHandler, Th_WrechWatchHandler*, or by a clearing event issued by the *VanetFilter* in the Input unit. Algorithm 1 shows the verification task execution steps. It is used for confirming a possible traffic incident by receiving another incident alarm from another detection technology that matched the possible incident. This thread depends on recording incoming incident data and change the system state according to current situation. When no incidents are detected, the system is in normal state. When an incident alarm is issued, the system switches to investigation state if the incident alarm is coming from the Detection unit. When all recorded incidents are confirmed, the system state switches to clearing and awaits for a confirmation from the cloud that the incidents has been cleared to return to the normal state. The system state is recorded in the data component *Dt_currentState* which contains the current state and the number of incidents. It is safe to say that both eCall and WreckWatch are considered confirmed traffic incidents already, since they depend on direct sensors to detect a crash. But, when a detection algorithm depends on traffic variables that are error prone and have some ambiguity, it is important to provide a verification step to insure not to exhaust the authorities with frequent false alarms. So, when an eCall or WreckWatch signal arrives, they are considered confirmed, they are recorded in the data component *Dt_incidentArr*, and then reported to the cloud. However, when one of the incident detection algorithms in the Detection unit sends a possible incident alarm, the verification thread first check if there is a match in the previously recorded incidents, if there is a match and it was detected by another detection technology, then the incident is confirmed and reported. If not, then the incident will be added to the list and awaits confirmation.

Algorithm 1 Incident verification

read *currentstate*, *numberofincidents*, *existing incidents list*.
check input.
if $inputsource = cloud$ **then**
  check *existing incidents list*
  if $matched = true$ **then**
    remove incident from *existing incidents list*
    numberofincident - 1
    if $numberofincidents = 0$ **then**
      state ← normal
else ▷ input is a new incident
  if $currentstate = normal$ **then**
    add new incident to *existing incidents list*
    numberofincidents + 1
    if $new\_incident\_confirmed = true$ **then**
      state ← clearing
      send incident data to cloud
    else state ← investigating
  else if $currentstate = clearing$ **then**
   check *existing incidents list*
   if $incident\_matched = false$ **then**
    add new incident to *existing incidents list*
    numberofincidents + 1
    if $new\_incident\_confirmed = true$ **then**
      send incident data to cloud
    else state ← investigating
   else if $currentstate = investigating$ **then**
   check *existing incidents list*
   if $incident\_matched = true$ **then**
    if $same\_detection\_source = false$ **then**
      confirm existing incident
      add detection source to incident data
      send incident data to cloud
     else
    add new incident to *existing incidents list*
    number of incidents + 1
    if $new\_incident\_confirmed = true$ **then**
      send incident data to cloud
if $state <> normal$ **then**
  if $confirmed\_incidents = numberofincidents$ **then**
    state ← clearing
write *currentstate*, *numberofincidents*

*4) **Output Unit**:* Output unit consists of two tasks: *Routing_Th* and *ToCloud_Th*. The *Routing_Th* thread receives instructions from the cloud to either broadcast a message for the passing vehicles to evacuate a certain lane for a coming EV, or to relay the routing messages to the EV passing through the RSU communication range. *ToCloud_Th* thread is responsible for reporting the confirmed incident data to the cloud for incident classification.

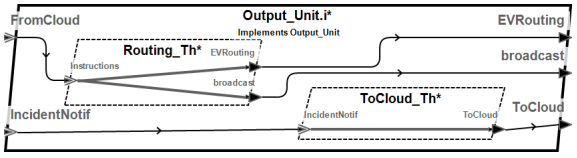

Fig. 5. Output Unit's AADL architecture

*5) **Devices**:* In order to get the data we need to process, the RSU depends on multiple communication devices. Since our RSU uses the ILDs connected over RS485, the connection will be established through UART which will be connected to PCIe 3.0 system bus through UART-to-PCIe

bridge. VANET transceiver, *Vanet_modem*, is responsible for sending and receiving VANET messages from/to the passing vehicles and the cloud through V2I and I2I communication standards [17] [18]. Wireless modem (Wi-Fi), which also exist as a hotspot for smartphones in today's commercial RSUs [19], is responsible for getting the WreckWatch signal and data when a smartphone owner, whose smartphone is successfully connected to our RSU, is involved in an incident.

IV. Experimantal Results

We chose AADL as the modelling language to present the RSU architectures. We used OSATE [20] to model our RSU and produce two analysis tests: schedulability and flow latency analysis. In addition, we used AADL Inspector [21] to provide static and dynamic schedluing simulations on our RSU model.

For all of these tests and simulations, we have to specify each task's execution time and deadline. We imitated the algorithms and applications, to the best of our capabilities, depending on the explanation provided in their papers. We coded each task (thread) using C++ language. We confirmed correctness of each code with synthetic input test data. To find each task's execution time, we used gem5 simulator [22]. Inspired by commercial RSUs [19] [23], we proposed five different execution platforms based on CPU micro-architecture, CPU speed, and memory options. The first option [23], includes an out-of-order ARM processor at 1 GHz speed, with 1 GB DDR4 memory, a 64 KB iCache and a 32 KB dCache, in addition to 1 MB L2 cache. The second option Siemens [19] includes 4 different variations with both in-order and out-of-order ARM and X86 Processors at 800 MHz, with with 1 GB DDR4 memory, a 64 KB iCache and a 32 KB dCache. All options are simulated as one core single thread CPU. We applied se.py configuration script under the proposed configuration options. For each task, we found that the X86 environment provided the longest execution times between 1 - 5 *msec*, while the ARM environment, gave results between 60 - 356 μsec. We make use of these execution times in three types of analyses:

(1) RSU Model's schedulability analysis in OSATE.
(2) RSU Model's flow latency analysis in OSATE.
(3) Scheduling simulations in AADL Inspector

A. RSU Model's schedulability analysis in OSATE

In addition to systems' modeling, OSATE provides multiple analyses, tests and reports at each level of abstraction. In the system model of our RSU, we defined the components and specified scheduling properties along with flow components. For the schedulability tests, we used Rate Monotonic Scheduling (RMS) as the scheduling protocol. We carried out the schedulability test on the selected platform options. Results show that the threads can be safely scheduled in all of the selected platforms. Testing results show that the highest utilization factors are for the X86 environment in-order and out-of-order processors at 68% and 27.1% respectively. The processor utilizations in the ARM environment are at 3.8% and 1.7% for in-order and out-of-order CPUs respectively. These results show that our incident detection model can be included as an added functionality to an existing RSU.

B. RSU Model's flow latency analysis in OSATE

The latency analysis depends on the timing properties of the tasks and hardware components but also on the scheduling policy as well. Latency analysis accumulates all latency contributors starting from the flow source, going through the buses and to where the data are finally transferred. For each end-to-end flow, there is a minimum and a maximum actual latency. The minimum actual latency is the best case where OSATE assumes that all data are ready, and each thread is dispatched immediately without delay. The maximum actual latency is the worst case where OSATE assumes that each thread has to wait for the data to be ready and will be dispatched at the end of the period. As shown in top table of Table I, we notice that the difference between each end-to-end flow of each deployment variation is marginal, because the only difference among the latency contributors are the tasks' execution times and task periods are ignored. However, bottom table in Table I, shows a big difference in the latency of each flow, where the periodic behavior of the tasks affect system response. We see that the ILD flow has the longest latency due to the one minute period of the *ild_Listener*. ILD algorithm has to run at least twice to detect an incident. Hence, we can easily show that our system response time -in case of an accident- is less than three minutes.

TABLE I
Actual latency of all end-to-end flows under all platform variations- top: Minimum, bottom: Maximum

Deployment Variations	ILD flow latency	Vanet flow latency	eCall flow latency	Wreckwatch flow latency
X86 opt.2.1	34.3 ms	20.5 ms	16 ms	15.9 ms
X86 opt.2.2	21.8 ms	8.17 ms	6.4 ms	6.37 ms
ARM opt.1	14.2 ms	0.748 ms	0.535 ms	0.518 ms
ARM opt.2.1	15.3 ms	1.59 ms	1.06 ms	1.02 ms
ARM opt.2.2	14.3 ms	0.735 ms	0.515 ms	0.500 ms

Deployment Variations	ILD flow latency	Vanet flow latency	eCall flow latency	Wreckwatch flow latency
X86 opt.2.1	60134 ms	20320.5 ms	216 ms	215.9 ms
X86 opt.2.2	60121.8 ms	20308.2 ms	206.4 ms	206.4 ms
ARM opt.1	60114.2 ms	20300.8 ms	200.6 ms	200.5 ms
ARM opt.2.1	60115.3 ms	20301.6 ms	201.1 ms	201.1 ms
ARM opt.2.2	60114.3 ms	20300.8 ms	200.8 ms	200.5 ms

C. Scheduling simulations in AADL Inspector

In this section, we provide the real-time simulation results to verify that the system meets all deadlines using AADL Inspector. We carried real time static testing with Cheddar simulator on X86 opt 2.1 platform that has the longest execution times and a processor utilization of 68%. Figure 6 shows that all threads are scheduled according to their periods, and all threads finish before their deadlines. Dynamic scheduling simulations are done with Marzhin simulator. Figure 7 shows that when a thread is finished, it dispatches the corresponding next thread according to each thread's scheduling properties.

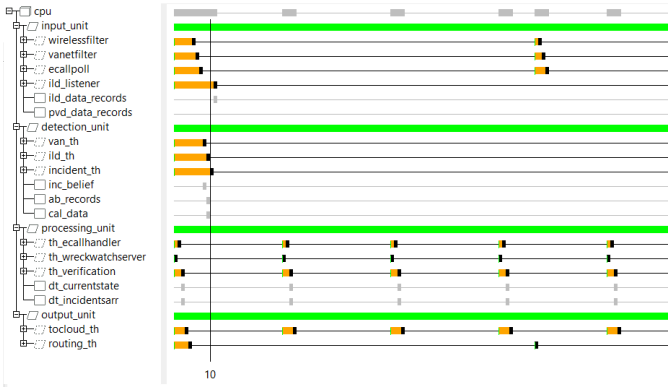

Fig. 6. Static scheduling simulation by AADL Inspector

For example, once *WirelessFilter* thread finishes its execution, *th_wreckwatchserver* is dispatched. After it finishes, *th_verification* thread is dispatched and when it finsihes, *toCloud* is dispatched.

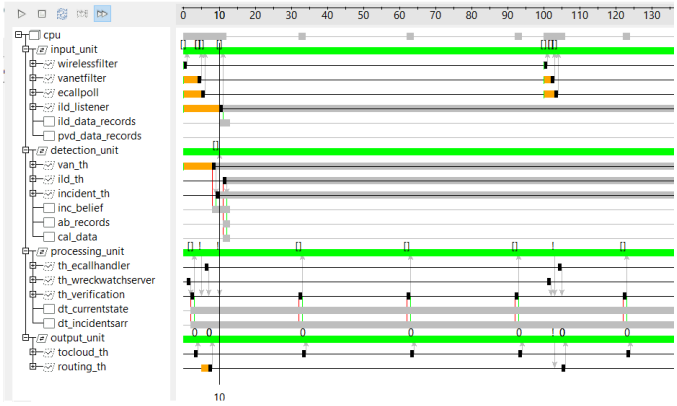

Fig. 7. Dynamic scheduling simulation by AADL Inspector

V. Conclusion & Future Work

In this paper, we proposed a system that consists of a cloud layer and a Roadside Unit (RSU). The RSU provides incident detection using multiple algorithms and solutions, while the cloud layer is responsible for incident classification and creating routing plans. The internal architecture of the proposed RSU is modelled with AADL by describing its software, hardware and execution platform components. In addition, we applied scheduling and flow latency analysis that showed that all response times of our RSU is under the five minutes limit of the Golden Hour. Results also showed that the proposed RSU is schedulable with low processor utilization factors. From that, we come to the conclusion that the system can be added as a functionality to an existing RSU to provide emergency management along its existing functionalities.

In the future, we plan to refine the model where we add behavior and error model extensions, add machine learning techniques, apply safety analysis and produce better simulation results.

References

[1] (2018) Global status report on road safety 2018. [Online]. Available: https://www.who.int/violence-injury-prevention/road-safety-status /2018/en/

[2] E. B. Lerner and R. M. Moscati, "The golden hour: scientific fact or medical "urban legend"?" *Academic Emergency Medicine*, vol. 8, no. 7, pp. 758–760, 2001.

[3] F. J. Martinez, C.-K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni, "Emergency services in future intelligent transportation systems based on vehicular communication networks," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 2, pp. 6–20, 2010.

[4] Intelligent transportation systems. [Online]. Available: https://www.isbak.istanbul/intelligent-transportation-systems/

[5] S. Niar, A. Yurdakul, O. Unsal, T. Tugcu, and A. Yuceturk, "A dynamically reconfigurable architecture for emergency and disaster management in its," in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 2014, pp. 479–484.

[6] E. Parkany and C. Xie, "A complete review of incident detection algorithms and their deployment: what works and what doesn't. transportation center, university of massachusetts," Technical Report, NETÇR3 7, Tech. Rep., 2005.

[7] R. Weil, J. Wootton, and A. Garcia-Ortiz, "Traffic incident detection: Sensors and algorithms," *Mathematical and computer modelling*, vol. 27, no. 9-11, pp. 257–291, 1998.

[8] U. Khalil, A. Nasir, S. M. Khan, T. Javid, S. A. Raza, and A. Siddiqui, "Automatic road accident detection using ultrasonic sensor," in *2018 IEEE 21st International Multi-Topic Conference (INMIC)*, 2018.

[9] R. Rossi, M. Gastaldi, G. Gecchele, and V. Barbaro, "Fuzzy logic-based incident detection system using loop detectors data," *Transportation Research Procedia*, vol. 10, pp. 266–275, 2015.

[10] T. Cherrett, B. Waterson, and M. Mcdonald, "Remote automatic incident detection using inductive loops," *Proceedings of The Institution of Civil Engineers-transport - PROC INST CIVIL ENG-TRANSPORT*, vol. 158, pp. 149–155, 01 2005.

[11] P. Chakraborty, A. Sharma, and C. Hegde, "Freeway traffic incident detection from cameras: A semi-supervised learning approach," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov 2018.

[12] P. H. Feiler and D. P. Gluch, *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*. Addison-Wesley, 2012.

[13] O. Popescu, S. Sha-Mohammad, H. Abdel-Wahab, D. C. Popescu, and S. El-Tawab, "Automatic incident detection in intelligent transportation systems using aggregation of traffic parameters collected through v2i communications," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 2, pp. 64–75, 2017.

[14] R. Oorni and A. Goulart, "In-vehicle emergency call services: ecall and beyond," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 159–165, 2017.

[15] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "Wreckwatch: Automatic traffic accident detection and notification with smartphones," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 285–303, 2011.

[16] R. Filjar, K. Vidović, P. Britvić, and M. Rimac, "ecall: Automatic notification of a road traffic accident," in *2011 Proceedings of the 34th International Convention MIPRO*. IEEE, 2011, pp. 600–605.

[17] V. Jindal and P. Bedi, "Vehicular ad-hoc networks: introduction, standards, routing protocols and challenges," *International Journal of Computer Science Issues (IJCSI)*, vol. 13, no. 2, p. 44, 2016.

[18] C. Campolo and A. Molinaro, "On vehicle-to-roadside communications in 802.11 p/wave vanets," in *2011 IEEE wireless communications and networking conference*. IEEE, 2011, pp. 1010–1015.

[19] Siemens connected vehicle roadside unit (rsu). [Online]. Available: https://www.mobotrex.com/product/siemens-connected-vehicle-roadside- unit/

[20] Osate. [Online]. Available: https://osate.org/

[21] Aadl inspector. [Online]. Available: https://www.ellidiss.com/products/aadl-inspector/

[22] Gem5 simulator. [Online]. Available: http://gem5.org/Main_Page

[23] Nxp, intelligent roadside unit. [Online]. Available: https://www.nxp.com/applications/solutions/automotive/connectivity/intelligent-roadside-unit:INTELLIGENTRSU