CHAPTER – I

INTRODUCTION & LITERATURE REVIEW

INTRODUCTION:

Nowadays, the culture of hybrid, all-electronic S.M.A.R.T. and connected autonomous vehicles is on an ever-peaking demand-curve. This also means an extension of the vehicle-security exploitations increment we hear about through daily media, about theft, hijacking or simply vandalism. Such an overwhelming need for an automobiles' security and longevity can only be met by the far-reaching, impactful and tailored technology, suited for the respective scenario. Realizing such endeavors could be only possible owing to the Open Source (F.L.O.S.S.) collective, and thence garnered resources and source codes.

On account of solving this research conquest, such an approach has been applied, such that in order to be able to cater the needs of almost everyone with a direct contact with a vehicle, or any automobile, public, personal or even private can be realized at the minimal costs of upgradation.

We, through the medium of this project, would aspire to address such vehicle optimization adversing security related tradeoffs, and conclusively suggest remedies.

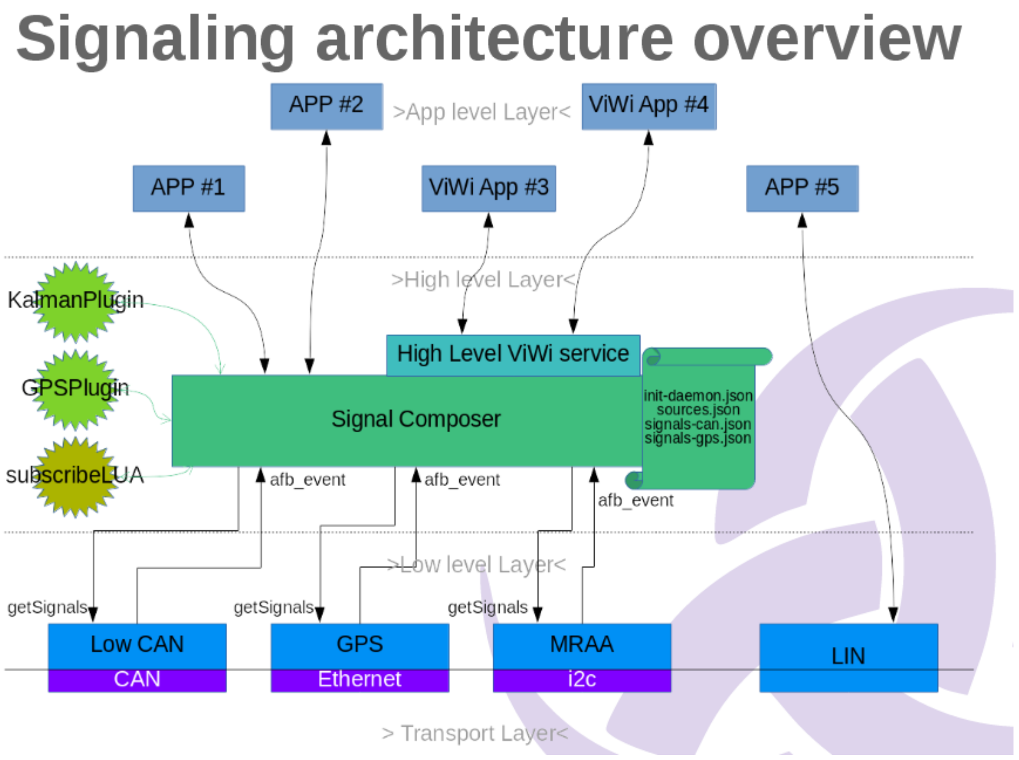
A wide variety of scholarly articles have been referenced to survey the current as well as the previously outdated inter and intra vehicular and network communication and signaling technology. In order to maintain standard universality, all implementations are based using Linux kernel 4.1.

https://mee499.github.io/MEE499R01V007/req002.html - literature-review

	Description	Remarks
List of Sampled Technology Protocols	1: Contoller Area Network (C.A.N.) 2: MAC layer Addressing 3: IEEE 802.11 (b.g.n/a/c)	**https://mee499.github.io/MEE499R01V007/req002.html - list-of-sampled-technology-protocols**
List of Sampled Technology Hardware	1: Adafruit's Arduino UNO 2: Raspberry Pi's 'Model B+' 3: Reannaisance's PorterBoard 2 4: Orange Pi's IoT+ Embedded 5: Stock Daragonboard410c (QEMU Emulated-VM)	**https://mee499.github.io/MEE499R01V007/req003.html - list-of-sampled-technology-hardware**
List of Sampled Software Releases	1: Automotive Grade Linux (A.G.L., Linux Kernel 3.9) 2: Tyzen Operating System (UNIX Kernel 4.11) 3: Qt (For the Graphical Release, Applications) 4: Ubuntu IoT Core (+2.3.26)	**https://mee499.github.io/MEE499R01V007/req004.html - list-of-sampled-software-releases**
List of Sampled Libraries and Modules:	1: AGL 1.1: agl-demo 1.2: agl-appfw-smack 1.3: agl-devel 1.4: agl-netboot 2: UNIX 2.1: gawk 2.2: wget 2.3: git-core 2.4: diffstat 2.5: texinfo 2.6: chrpath 2.7: cpip 2.8: socat	**https://mee499.github.io/MEE499R01V007/req004.html - list-of-sampled-libraries-and-modules**

	3: libdll 3.1: libsdl.2-dev 3.2: gcc-multilib 3.3: libhvac 3.4 libssh-dev	
List of Sampled Prototyping (cMake) Softwares:	1: Reading/Writing a PCB (.gerber format) 2: C, PyPi (Writing Functional Code Snippets) 3: make, build (C-lang) 4: bash (UNIX Scripting)	**https://mee499.github.io/MEE499R01V007/ req006.html - list-of-sampled-authoring-softwares**

Most of the projects in IV, V2V and V2I use the standard IEEE 802.11 protocol for communication. But also GSM, UMTS, GPRS protocols are used in some of these projects. Generally WSNs (Wireless Sensor Networks) are deployed ineffectively and thus "platoonong" is inefficient, since the convoluted network is not 'big' enough in terms of the 'no. of nodes' present in the V2V (Vehicle-to-Vehicle) or V2I(Vehicle-to-Infrastructure) network.



Traditionally, IEE standards like the Basic layers have been employed tor the information transmission. Routing inside a low power area network (LoWPAN) might be considered a challenge, as the RPL has to work over lossy radio links, with battery-powered nodes, multih-op mesh topologies and frecuent topology changes.

To give a solution several working groups are giving support to the RFC's for this protocol.

One of them is the routing over lowpan and lossy networks (ROLL) who is in charge of routing tasks. Meanwhile the "6LoWPAN" is trying to bring the new IPv6 addressing system to these resource-constrained devices.

We try an approach to link the various specific nodes by utilizing the local loopback (lo) to define the real-time attributes of the vehicle by getting the metadata locally, thus reducing latency and speeding up reactance and appreciating the autonomous vehicles susceptibility to its stimuli.

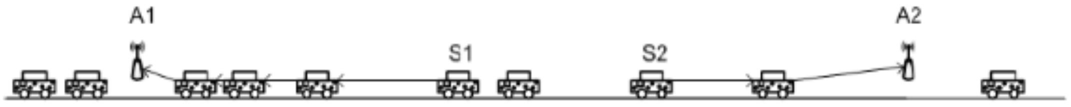
In our design, we have tried to predict the points of failure in traditionally applied approaches of such technology from a signal strength penetration and security (encryption or sniff-proof))testing point of view for application in areal world scenarios and conclude with instantly applicable remedies via pull requests to the FOSS code repositories.

Radio waves and infrared have been studied to give medium support to IVCs. The radio waves include micro, millimeter and VHF waves. The communication with millimeter waves and infrared are usually directional, while VHF is used for broadcast.

The typical radio bandwidth used in IVC is 5.9 GHz in US, 5.8 GHz in Japan and 5.8 GHz in Europe. The FleetNet project chose ULTRA TDD due to the availability of the unlicensed frequency band 2010-2020 MHz in Europe. Most projects, however, have adopted the use of infrared (CarTALK, COOPER, JSK, PA, etc.).

Utilizing a decentralized and thus peer-to-peer node approach, we may link any two nodes in vicinity of each other in a lot quicker manner, than to rely on repetitively transmitted feed data. Our approach helps improve precision in our model considerably and thus also contributes to the lowering down of the time complexity to adjourn the real-time attributes of a neighboring vehicle.

There are two approaches in developing MAC for IVCs. One is using IEEE 802.11 as a radio interface, while the other consists on extended 3G technology, such as CDMA for distributed access.



(Fig. 3) : Vehicle to Infrastructure Communication (schematic)

Both of them have to be modified and adapted to provide an efficient solution for IVCs. The only advantage of using IEEE 802.11 is the inherited support for distributed coordination in ad-hoc mode. On the other hand, 3G extensions present high granularity for data transmission

Further investigations leads to some specifications already presents like the one from Jaguar Land Rover [VISS], for Vehicule Information Service Specification and another from Volkwagen AG named [ViWi], stand for Volkwagen Infotainment Web Interface. Each ones has their differences and provides different approach serving the same goal:

VISS	ViWi
Filtering on node (not possible on several nodes or branches)	Describe a protocol
Access restrictions to signals	Ability to specify custom signals
Use high level development languages	RESTful HTTP calls
One big Server that handle requests	Stateless
Filtering	Filtering, sorting
Static signals tree not extensible [VSS]	Use JSON objects to communicate
Use of AMB ?	Identification of resources may be a bit heavy going using UUID
Use of Websocket	API

About **[VISS]** specification, the major problem comes from the fact that signals are specified under the [VSS], **Vehicle Signal Specification** to make a full inventory of all signals existing for each car is more important, each evolution in signals must be mostly don't comply with the [VSS]. VISS doesn't seems to be an valuable way to handle car's signals, a big component that responds requests, use of **Automotive Message Broker** that use DBus is a performance problem. Fujitsu Ten recent study[1] highlights that processor can't handle an heavy load on CAN bus and that Low level binding adopted for AGL is about 10 times[2] less impact on performance.

PROGRAM OBJECTIVES:

To come up with a neat-networking protocol schema to address inefficient hop-on /ad-hoc communication propogation delays, possibly trying to implement in a decentralized contract.

To be able to successfully reproduce the hardware based real-time implementation of the AGL release on an ARM based development board.

To provide an future roadmap for Non-hybrid cum Hybrid on-road network integration in a cheap (feasible), environment-friendly (sustainanble) and energy-efficient (if not, utilitarian) by means of a snap-on dashboard powered by a simple smartphones' sensors, transmitters and transducers.

To demonstrate a successful implementation of AGL (improvised fork) during the final review.

Others

Design Elements included:

Engineering Standards	Prototype and Fabrication
Design Analysis	Experimentation
Modeling and Simulation	Software Development

Realistic Constraints to be addressed :

Economic	Ethical
Environmental	Safety
Social	Health
	Manufacturability
Political	Sustainability

METHODOLOGY AND EXPERIMENTAL PROCEDURE

2.1 Methodology

The AGL is first ported to a VM with a usual DebianOS base kernel.

Next, the images are downloaded and flashed onto a SDHCeMMC Memory Card.

Third, the auxiliary input and output peripherals are serially connected to the used Raspberry Pi, or UNO module.

2.2 Experimental Procedure

Hardware Requirements

Dragonboard410c

96Boards Compliant Power Supply

Linksprite 96Boards Touch Screen

Sensors Mezzanine

Audio Mezzanine(Required if using External Arduino)

Arduino Uno(Optional)

DC motor with Propellers

L298 Motor Driver

5mm LED's

330 ohm resistors

Connecting (patch) wires

Arduino

Controlling Fan Speed and LED intensity are handled by the Arduino. Sensors Mezzanine has an ATMega328 microcontroller compatible with Arduino Uno. We use that or any external Arduino Uno for PWM control.

In case of using Sensors Mezzanine, the sketch can be uploaded by using Dragonboard410c itself..

If using Sensors Mezzanine, please follow the below steps on Dragonboard410c running Debian otherwise use Arduino IDE on the host system for programming.

```
$ cd ~/Documents

$ git clone https://github.com/96boards-projects/agl-demo.git

$ cd agl-demo/arduino/hvac
```

Now open the hvac.ino using Arduino IDE and flash it onto the Sensors Mezzanine or Arduino Uno.

Dragonboard410c

Execution environment: Host PC

Software Dependencies:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \

   build-essential chrpath socat libsdl1.2-dev xterm cpio curl
```

Loading AGL Source Code

AGL uses repo tool for maintaining repositories. We need to download the source on the host machine and cross compile it for Dragonboard410c.

```
$ export AGL_TOP=$HOME/workspace_agl

$ mkdir -p $AGL_TOP

$ mkdir -p ~/bin

$ export PATH=~/bin:$PATH

$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo

$ chmod a+x ~/bin/repo
```

Next, checkout the stable branch of AGL.

```
$ cd $AGL_TOP

$ repo init -b dab -m dab_4.0.2.xml -u https://gerrit.automotivelinux.org/gerrit/AGL/AGL-repo

$ repo sync
```

Building AGL

Now, to build the agl-demo-platform for Dragonboard410c.

```
$ source meta-agl/scripts/aglsetup.sh -m dragonboard-410c agl-demo  agl-appfw-smack  agl-devel  agl-netboot
```

Now to move to the directory:

```
$ cd agl-demo
```

Copying the custom HVAC recipie to AGL source:

```
$ cp hvac_git.bb $(AGL_TOP)/meta-agl-demo/recipes-demo-hmi/hvac/hvac_git.bb
```

Executing bitbake command by moving to the build directory of AGL source.

```
$ cd $(AGL_TOP)/build
```

```
$ bitbake agl-demo-platform
```

Flashing AGL onto Dragonboard410c

Once the build has been completed, we have to flash the boot and rootfs images onto Dragonboard410c. Now, boot Dragonboard into fastboot mode by following the instructions here. Then follow the below instructions to flash AGL onto Dragonboard410c.

```
$ cd $AGL_TOP/build/tmp/deploy/images/dragonboard-410c

$ sudo fastboot flash boot boot-dragonboard-410c.img

$ sudo fastboot flash rootfs agl-demo-platform-dragonboard-410c.ext4
```

Hardware Setup (Schematic Protocol Componential to execute HVAC).

Tthe Dragonboard410c is powered off

Connected DC motor and LEDs to Arduino as per above schematic

Connected LCD to Dragonboard410c via HDMI cable for display and Micro USB cable for touch input

Powered on 96Boards CE with compatible power supply

Dragonboard410c should now boot into AGL and homescreen should be visible.

HVAC & Utilities

Execution environment: Dragonboard410c

Navigated to the HVAC application from the Homescreen.

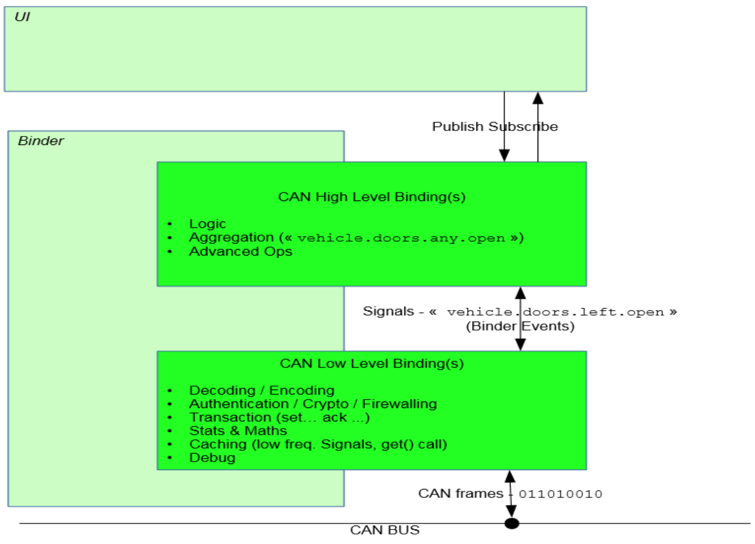
To control the Fan speed, change the position of slider at top.

To control the LED intensities, change the values of L/R temperatures by dragging up the LO box.Turned off power by using the following:

```
$ sudo cd ..
```

```
$ poweroff --no-latch
```

Fig



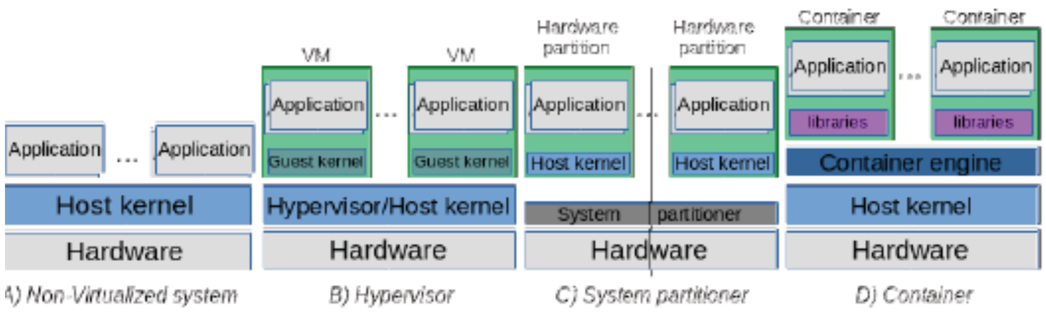
UI
Publish Subscribe
Binder
CAN High Level Binding(s)
• Logic
• Aggregation (« vehicle.doors.any.open »)
• Advanced Ops
Signals - « vehicle.doors.left.open »
(Binder Events)
CAN Low Level Binding(s)
• Decoding / Encoding
• Authentication / Crypto / Firewalling
• Transaction (set... ack ...)
• Stats & Maths
• Caching (low freq. Signals, get() call)
• Debug
CAN frames - 011010010
CAN BUS

. *3* An Example in cloud run: Vehicle Occupancy Flag

Protocol Parameter Development

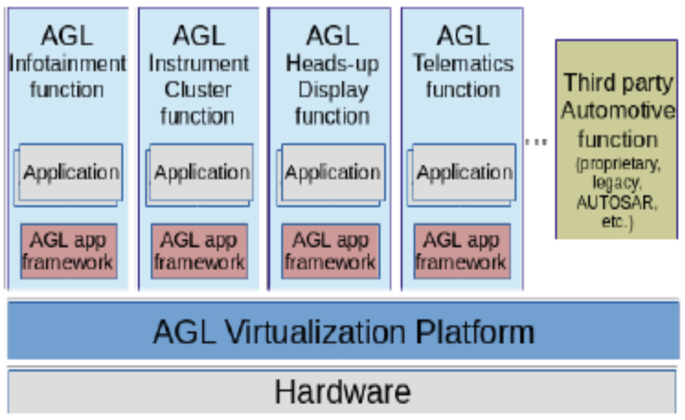
Note*: About Network Layer based communication:

* Realistic Constraints to be addressed.



Almost all routing protocols used by the different IVC projects are position-based. The components of such architecture are:

Communication buses : Consolidated EEs (and their applications) need to interact and communicate with each other. Two types of communication bus are supported by the architecture:

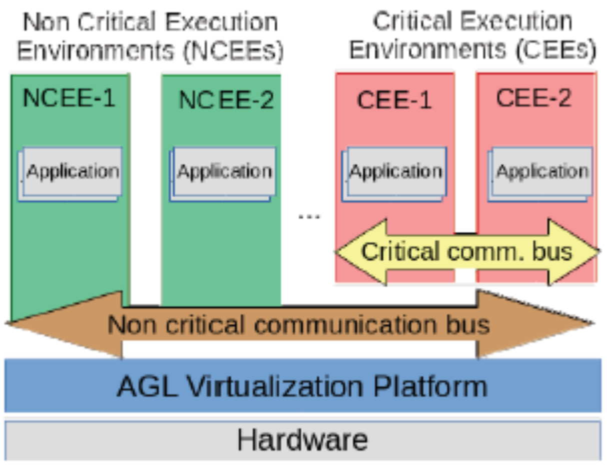

AGL Infotainment function
AGL Instrument Cluster function
AGL Heads-up Display function
AGL Telematics function
Application
AGL app framework
...
Third party Automotive function (proprietary, legacy, AUTOSAR, etc.)
AGL Virtualization Platform
Hardware

.

Critical communication bus: it is restricted to Critical Execution Environments (CEEs) only. The communication here has to address important requirements of safety and security.

Non critical communication bus: it is open to all the EEs available in the system.It has to address high performance and security requirements. In addition, existing MAC ad hoc protocols could be directly applied. But if an optimal performance is desired taking into account the linear nature of the networks seen in section III, modification of the existing routing protocols must be performed.

In addition, the features most of vehicles offer nowadays makes possible to get position information via GPS or GIS, very useful for routing.


Non Critical Execution Environments (NCEEs)
Critical Execution Environments (CEEs)
NCEE-1
NCEE-2
CEE-1
CEE-2
Application
Application
Application
Application
...
Critical comm. bus
Non critical communication bus
AGL Virtualization Platform
Hardware

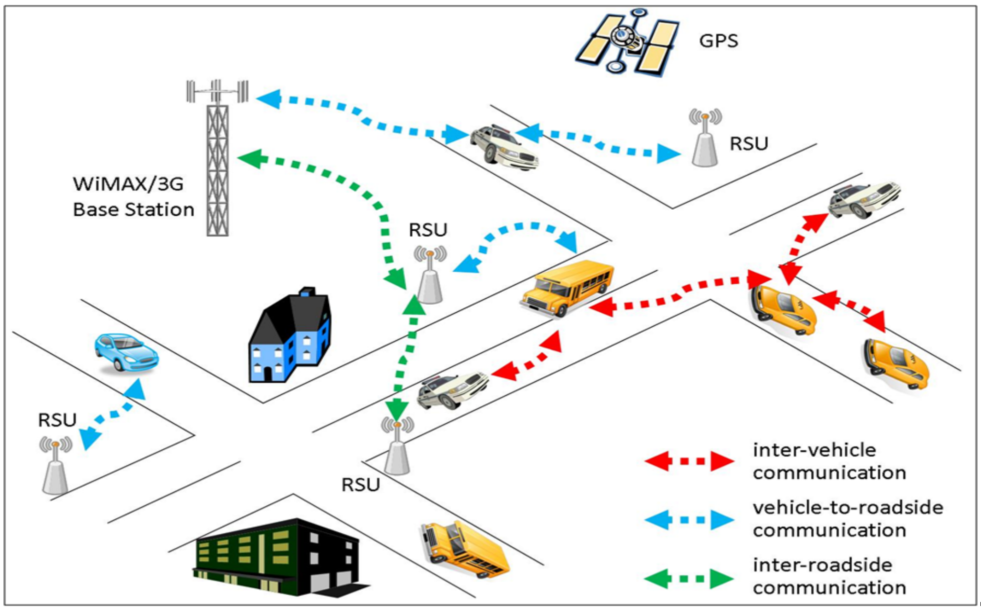
The protocol uses a forwarding scheme which avoids beacons for impactful transmission and effective sensing.

Execution Environments (EEs):

These are silos that run concurrently on the system and enable the execution of different applications. Different types of EE are supported: certified, non certified, trusted, non trusted, open source, proprietary. Some of them are classified as Critical Execution Environments (CEEs), because their applications have an impact of the safety and security of the system. The others are considered Non Critical Execution Environments (NCEEs). They could be implemented as binary applications, combination of a set of libraries with the related application, or full featured operating systems, etc.

Applied Ad-hoc Approach to Network

We employ a Virtual Network layer based on-board computation cum signaling:


GPS
WiMAX/3G Base Station
RSU
RSU
RSU
RSU
inter-vehicle communication
vehicle-to-roadside communication
inter-roadside communication

(Fig 4:) Hybrid or Modern Day Infrastructure

We aspire to prototype a modular approach to convert existing infrastructure of sensors and wireless telecommunication devices, and perhaps even provide pointers on an improved protocol fabrication, which could be deployed at scale, feasibly.

CHAPTER –III

RESULTS AND DISCUSSIONS

PHASE I

3.1 Implementational Details:

Successfully studied the architecture of a PCB (printed) board.

Gained a deep understanding of remote-sensing and GIS in application-layer deployment.

Ported AGL unto raspberryPi and successfully emulated on a HDMI-connected monitor.

Pull Request was successfully merged with the source at git.automotivelinux.com.

Cost and Capacity based market economic analysis.

3.2 WORK OBJECTIVES ACCOMPLISHED:

Came up with a neat-networking protocol schema to address inefficient hop-on /ad-hoc communication propagation delays, possibly trying to implement in a decentralized contract.

Were able to successfully reproduce the hardware based real-time implementation of the AGL release on an ARM based development board.

To provide an future roadmap for Non-hybrid cum Hybrid on-road network integration in a cheap (feasible), environment-friendly (sustainable) and energy-efficient (if not, utilitarian) by means of a snap-on dashboard powered by a simple smartphones' sensors, transmitters and transducers.

To demonstrate a successful implementation of AGL (improvised fork) during the final review.

LAYOUT OF NODES

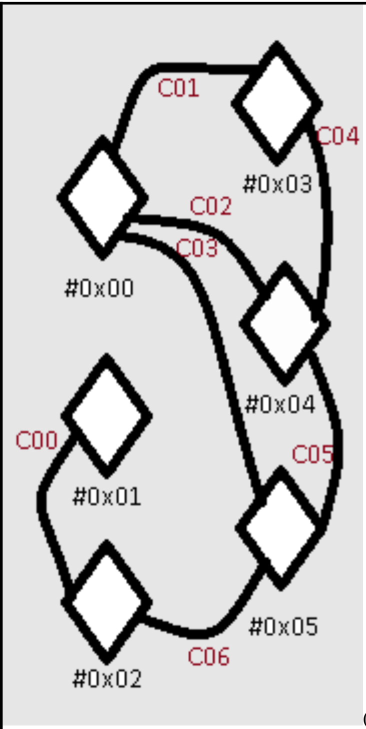
3.1 Networking Approach:

The independent vehicles are coupled within a WLAN region, with each and every client being a separate node.

t=-n
S2
0x01
t=1
t=-1
t=0
0x0n
…
t=n+1.

Fig 5 (Schematic peer-to-peer approach)

The intrinsic chararteristics of the protocol specifically aim to reducy latency within the network at a controller level adminstrative scope.

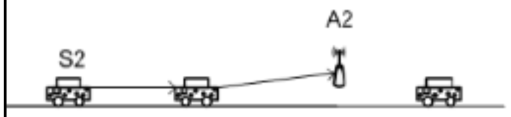
 C01 C04 #0x03 C02 C03 #0x00 #0x04 C00 #0x01 C05 #0x05 C06 #0x02 (*Fig. 6*)	Figure showing networked nodes at a schematic note. Each node is identified with a unique tag, or roll no. Each Autonomous Node Is Indexexed.viz.0x00-0x05. Current Figure Shows A Total Of 6 Openly Connected Communication Channels.

It is serialized in such a fashion that no two nodes could have the same unique identifier. Hence, a specific roll is given to each member node connected to the peer-to-peer network.

PHASE III

CONCLUSIONS

The networking was successfully deployed in real-time on a two-wheeler, and the Raspberry-Pi microcontroller was used to process the network traffic throughput over a locally established WLAN using an ad-hoc approach.

Figure showing traditional approach. Client-to-Server type connection Time Complexity = O(N)	

The benefit of using such an approach diminishes the requirement to have communication channels opened all over the so-called grid network having all clients simultaneously connected at a given point in time.

Using a protocol with ad-hoc approach enables us to avoid the slack (latency) caused due to congestion prevalence. Thus, a time complexity of O(N) is simly transformed into a time complexity of O(ln(N)), due to connection channels being only opened with the other clients in the domain of the end-client's fidelity domain, i.e. the currently (time-dependent) region of local reactance to a received wireless signal.

FORMULATION

For high-speed internetworking, i.e. in our case 36Km/h (10m/s), we found a connection latency of 0.0014 bauds/m-sec, with boundary at roughly 20m-25m.

	Figure showing traditional approach. Client-to-Client type connection Time Complexity = O(ln(N))

(Fig. 8) Client-to-client, one to one approach

Rather than establishing a direct client-to-server connection, our approach uses a peer-to-peer, ad-hoc approach to route the throughput in a region temporally local to the client device.

The NMEA GPS agent listens to a NMEA server that produces GPS data. The GPS data

received are made available to clients of the signaling agent.

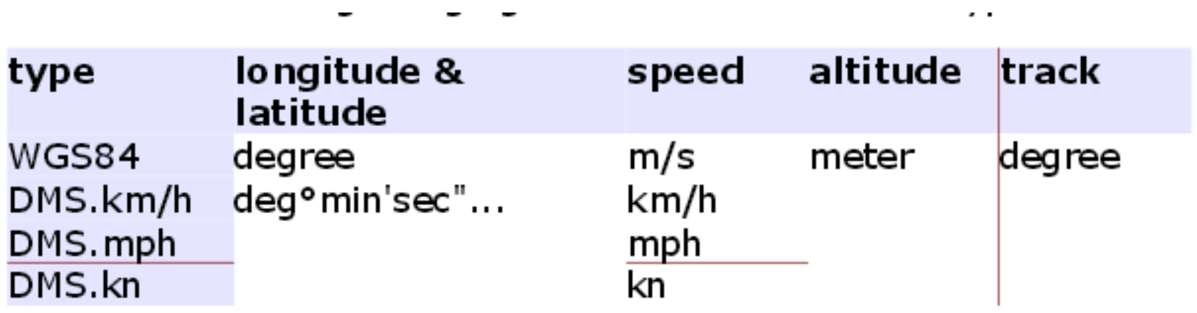
Clients can query the last known GPS position at any time using the "get" method.

Example of answer to the “gps/get” query:

```
{
"response": {
"type":"WGS84",
"time":76994000,
"latitude":47.410213545797532,
"longitude":357.04750632886282,
"speed":6.7083555549760003
},
"jtype":"afb-reply",
"request":{"status":"success"}
}
```

When the client queries the last known position, it can specify the type of the position it requires. The NMEA GPS signaling agent offers four different types:

Note that the agent does not send the data parts of the position that are missing.



type	longitude & latitude	speed	altitude	track
WGS84	degree	m/s	meter	degree
DMS.km/h	deg°min'sec"...	km/h		
DMS.mph		mph		
DMS.kn		kn		

In the returned example, the data for altitude and track (heading) are missing.

A client can also subscribe to be periodically notified of the position. The subscription can specify the type of position expected and the period in milliseconds between two notifications.

Example of answer to the “gps/subscribe” subscription query:

```
{

"response":{"name":"GPS","id":1},

"jtype":"afb-reply",

"request":{"status":"success"}

}
```

This reply includes the event name (“GPS”) and a numeric id that must be used whenunsubscribing.Notifications of the position are events.

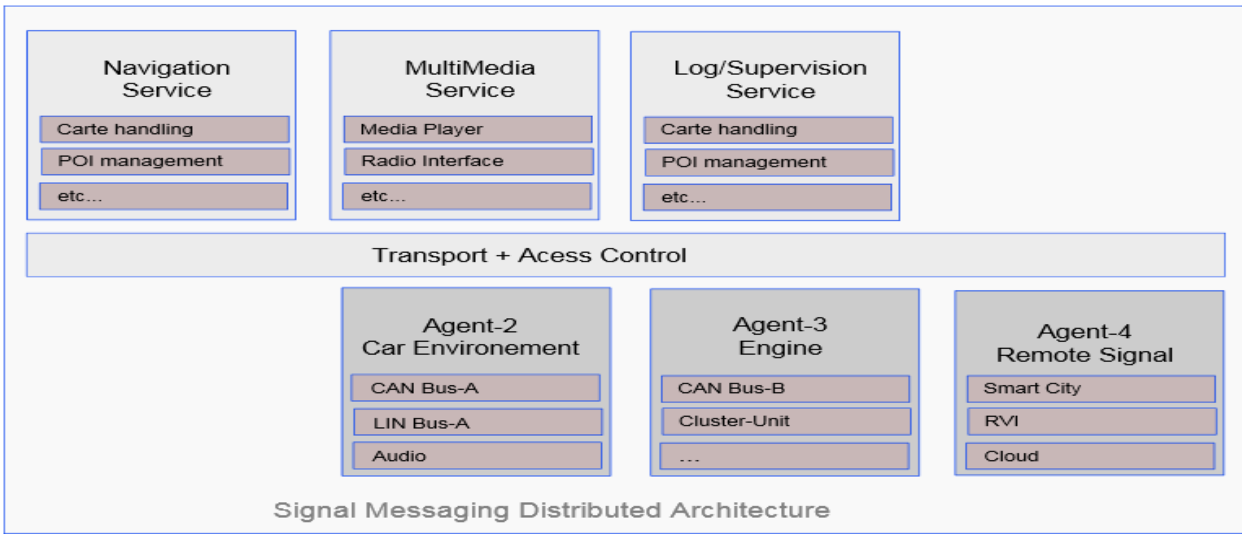
Example of received event:

```
{

"event":"gps/GPS",

"data": {

"type":"WGS84",

"time":78021000,

"latitude":47.361904282981413,
"longitude":357.06160208259365,

"speed":7.119911110496

},

"jtype":"afb-event"

}
```

A client can unsubscribe to an event using the numeric id received during **Subscription**.

```
/*
* Get the last known position
* parameter of the get are:
* type: string: the type of position expected (defaults to "WGS84" if not present)
* returns the position
* The valid types are:
* +==========+=======================+=======+=========+=======+
* | type | longitude & latitude | speed | altitude | track |
* +==========+=======================+=======+=========+=======+
* | WGS84 | degre | m/s | | |
* +----------+-----------------------+-------+ | |
* | DMS.km/h | | km/h | | |
* +----------+ +-------+ meter | degre |
* | DMS.mph | deg°min'sec"X | mph | | |
* +----------+ +-------+ | |
* | DMS.kn | | kn | | |
* +==========+=======================+=======+=========+=======+
*/
```

Furthermore quality development is still required to network autonomous vehicles going at a greater speed, whereas, for applications on vehicles like electric-bikes, or even simple bicycles is currently technically feasible.The agent implements basic logic for delivering events: when a new position arrives to the NMEA sockets, the clients whose period is expired receive the notification. So theperiod of subscription is the minimal period between two notifications.



(Fig 9) : Signal Passing as bypassed within the subnet (local).

These strengths increased significantly with reduction in vehicle speed, and as a consequence, reduced latency, providing a faster-to-establish communication link, rather than a traditional fast (e.g. Lifi) link, with a small delay, nonetheless a trade off in quality assurance of the success in link establishment.

INFERENCE

Fig.1 Ad-hoc approach.

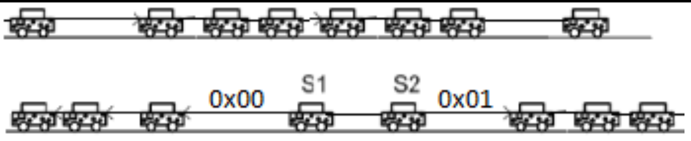


Fig.2 Real-time Routing

O(ln(N)) time complexity

Client-to-Client type connection

(Fig. 10) Real-Time Implementation and Scaling approach

The Automotive Grade Linux system when implemented hands-on with an ad-hoc approach drew better results in terms of connection stability, ease-of-user-setup and higher reactance in slow mobility environments, which could be seen as far more reliable over any GSM or Client/Server networking approach.

Finally we add an configuration file to route public keys and attributes

```
via { global: afbBindingV1*; local: *; };
```

local schematic binding over the ad-hoc type network connection.

This protocol benefits the end-node applications on the client device such as distance-to-drop forecasting, local neighbor detection, faster network resource sharing, improved auto-pilot torque prediction model, among other functions inbuilt to the AGL core.

Gantt (Progression / Commits) chart

Week	1	2	3	4		5	6	7	8		9	10	11	12		13	14	15	16		17	18	19	20		21	22	23	24
Pre-Activity	#	#	#																										
Literature		#	#	#																									
0th Review					#																								
Lit. Survey			#	#	#	#	#	#																					
Sampling					#	#	#	#	#																				
1st Review										#																			
Re-Editting											!	!												!					
Experiments												!	!	!		!	!	!	!		!								
Anlysis														!		!	!	!											
Thesis																		!	!		!	!	!	!					
2 nd Review																									!				
Submission																									!				
Acceptence																										!	!	!	
Result																													#

! := done (submission)

# := done (review)

Note:

Commit history as taken from https://github.com/MEE499/

Code frequency history as taken from https://github.com//14BME0133/

Pull Request was successfully merged with the Automotive Grade Linux repository hosted (public::master@master).

REFERENCES

Books, Whitepapers and Journal Publications referenced through the literature are cited below:

[1] Jawhar, I., Mohamed, N., Zhang, L.: Inter-Vehicular Communication Systems, Protocols and Middleware. pp. 1–3 (2010)

[2] Yang, X., Liu, J., Zhao, F., Vaidya N. H.: A Vehicle-to-Vehicle Communication Protocol for Cooperative Collision Warning. pp 1–14. (2003)

[3] Thangavelu, A., Saravanan, K. Rameshbabu, K.: A Middleware Architectural Framework for Vehicular Safety over VANET (In-VANET).pp 277–282 (2009)

[4] Luo, J., Hubaux, J.: A survey of Inter-Vehicle Communication. pp 1–12. (2004)

[5] Böhm, A.: State-of-the-art in networks aspect for Inter-Vehicle communication. pp 1–25. (2007)

[6] Keskin, U.: In-Vehicle Communication Networks: A literature Survey. pp 14 (2009).

[7] Nekovee., M.: Quantifying Performance Requirements of Vehicle-to-Vehicle Communication Protocols for Rear-end Collision Avoidance. pp. (2008)

[8] Inter-Vehicular Communication Systems, Daniel López García, Danckelmannstrasse 46/47 Berlin.

SOURCES AND CODE SNIPS:

https://MEE499.github.io/

https://14BME0133.github.io/MEE499/

https://git.automotivelinux.com

https://MEE499.github.io/agl-7782-0002-0009/

https://github.com/14BME0133/MEE499/Wiki/doc

http://iot.bzh/download/public/2018/Signaling/AGL-Signaling_Feb18.pdfs

https://github.com/iotbzh/af-gps-binding/blob/master/src/af-gps-binding.c

https://github.com/mee499/a7952_0x02.pptx

https://github.com/mee499/a7952_0x02.pptx