# VC-bots: A Vehicular Cloud Computing Testbed with Mobile Robots

Duo Lu[†], Zhichao Li[‡], Dijiang Huang[†], Xianglong Lu[‡], Yuli Deng[†], Ankur Chowdhary[†], Bing Li[†]

[†]School of Computing, Informatics, and Decision Systems, Arizona State University
[‡]School of Electrical, Computer and Energy Engineering, Arizona State University

{duolv, zhichao1, dijiang.huang, xianglo1, ydeng19, achaud16, bingli5}@asu.edu

## ABSTRACT

Smart vehicles with computing, sensing, and communication capabilities are gaining popularity. With various vehicular applications equipped, these smart vehicles not only improve driving safety, but also facilitate data collection and information sharing for traffic optimization, insurance estimation, and infotainment. However, developing and testing such cloud based vehicular application is challenging due to the high cost of running the application on actual cars in various traffic scenarios. For the same reason it is also difficult to understand and model the network protocol behavior among multiple vehicles. In this paper we proposed VC-bots, a vehicular cloud testbed using mobile robot vehicles, which can emulate different types of vehicles for testing vehicular network protocols and vehicular cloud applications in various scenarios, which can be easily reconfigured without any infrastructure assistance. To facilitate software integration, we also developed a message based service framework for applications running on the robot vehicle and in the cloud.

## Keywords

Vehicular Network, Vehicular Cloud Computing (VCC), Network Testbed

## 1. INTRODUCTION

In the future, more and more cars on the road, and even indoor service vehicle will be gradually equipped with computing, sensing, and communication capabilities, which makes Internet and cloud available on the wheel anywhere and anytime. To address the challenges and application requirements of cooperative autonomous vehicles, vehicular ad hoc network (VANET) and Vehicular Cloud Computing (VCC) [10] are gradually becoming a reality.

Current research work in VANET and VCC are mainly carried out using simulation tools, which have a large gap with real world testbed due to simplification and approximation in the model. Although a few real world testbeds were built with real vehicles on the road, due to the high construction and operating costs such testbeds are usually difficult to replicate and not openly available all the time. Some researchers cooperated with public transportation department or local property management authorities to deploy mid-scale network testbeds on buses or other service vehicles in operation without interfering with their normal movement [11, 8, 15, 4, 2]. For example, C-VET [4] is a campus scale wireless network testbed at UCLA with 60 vehicles and an infrastructure based mesh network, whose focus is on validating network models and protocols. Another example is HarborNet [2], a relatively large scale testbed covering a seaport using multiple radio technology including WiFi, DSRC / WAVE, and cellular network. These testbeds provide significant real-world mobility pattern data, however their data comes from a limited class of vehicles and cannot be easily generalize for different traffic pattern. Also they lack mobility control and their scenario is determined by the people who operate the vehicle instead of those who are interested in network protocol and application.

One solution is constructing miniature testbed using wheeled robots to represent real vehicles[6, 5, 7, 9, 13]. For example, there is an indoor intelligent transportation testbed built at Ohio State University for urban traffic scenarios [3]. The testbed runs iRobot Create as emulated vehicle on a statically configured road network marked on floor. The focus of this type of testbed is usually general wireless sensor networks instead of vehicular networks or vehicular cloud applications, and they need a dedicated area (usually not very large) and infrastructure support to operate. Traditionally, due to computation, power and security restrictions, miniature vehicles contain limited functions and deploying applications on them are difficult. Connecting these robots to cloud resources mitigates these drawbacks but the limited bandwidth and long delay over wireless network between the vehicles and remote cloud might reduces the useability of many applications significantly. Meanwhile running every application in cloud servers far away from the vehicle may not represent real world autonomous vehicles very well, because vehicles are usually privately owned, vehicular networks are largely ad hoc, and most decisions on the road are made locally by the on board software with coordination with a few adjacent vehicles. Another issue with current vehicular network is IP, which is designed for end-to-end communication over the Internet. Vehicular application needs more service model, such as group based message



*IoV-VoI'16, July 05 2016, Paderborn, Germany*



DOI: http://dx.doi.org/10.1145/2938681.2938683

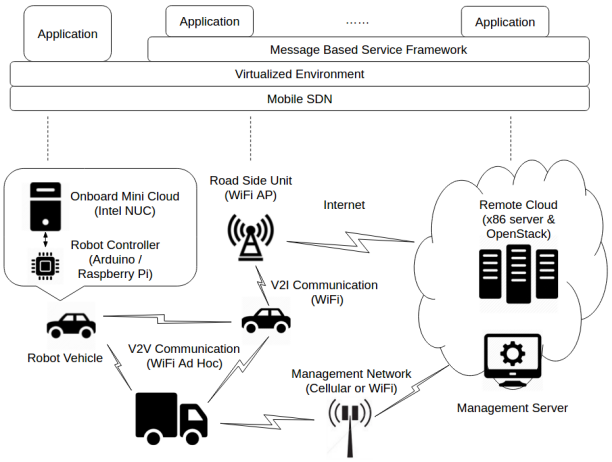


**Figure 1: VC-bots System Architecture**

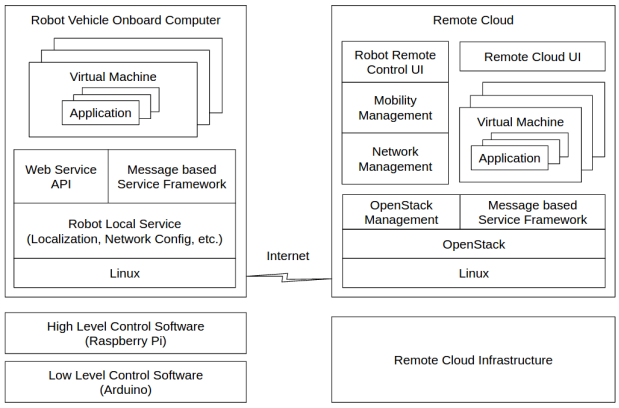


**Figure 2: VC-bots Software Architecture**

dissemination for road hazard avoidance, platooning, cross coordination.

We believe mobility, traffic scenario, protocol, cloud resources, on board application and location awareness are all important factors in VANET and VCC. Both researchers and developers desire an open, convenient, cost-effective, and reconfigurable testing environment. To meet these new requirements, we attempt to build a vehicular cloud testbed - VC-bots, with many different types of mobile robots. VC-bots is designed to provide an open platform for both research experiments and education services on VANET, vehicular VCC infrastructure, and applications developed specifically for future smart vehicles, both on road and indoor. It is an integrated platform for developing and testing applications that consider the vehicles and the cloud as a whole piece, and it offers a controllable and reconfigurable testing environment without any assistance of infrastructure.

## 2. ARCHITECTURE

System architecture of VC-bots is shown in the figure 1. Our testbed contains 5 major hardware components: the robot vehicle, network, robot vehicle on board mini cloud, remote cloud, and management server. Robot software can be further broken down into 3 layer. The highest layer is the on board computer, which contains robot vehicle local service, web service API of control function, message based service framework, and software necessary for on board mini cloud. Below that there are two levels of controller. Software in the remote cloud contains OpenStack, network manager, mobility manager, and system UI. The software architecture is shown in the figure 2.

### 2.1 User Interface and Mobility Management

VC-bots is operated through a Java based mobility management UI. On the management UI, users can see a map showing the configured environment and robot vehicles. The map is both represented as a grid based map as well as a graph based road network. For the graph based road network, edges represent straight roads, and nodes represent either intersections or junctions of two straight roads. Each edge has a width denoting the actual width of the road, and each node has a coordinate for its actual location. Additionally, the map may contain some freely traversable area representing parking lots or open space inside community. Each of such area is a special node connected to the road network. Both the map and road network are virtually configured over a real indoor space, and these properties are represented in real world scale.

Each robot vehicle can be instructed to move from one point to another point on the road network by clicking on UI. User can manually select a few nodes on the road network to create a polyline path for the robot vehicle to follow, or provide a given destination and ask the system to plan a polyline based path over the road network. Then the polyline based path is converted to a controller friendly format, which is a collection of line segments and arc segments or a list of coordinates over the grid map. We restrict the planned path to predefined road because it reflects current real transportation system. After the path is received, the on board control software will drive the vehicle to follow the path. Besides path following, the user can also send a sequence of coordinates to each robot vehicle to emulate movement like turning around or parking.

Besides mobility management, another advantage of our testbed is configurable scenario of both traffic and road environment, i.e. build the map and road network. We developed a tool to ease the configuration steps. The map can be obtained either from existing floor plan or by SLAM software on the robot. Then we can draw blocks and drag polylines as roads with the toll, and these shapes are further translated to various community areas and road network automatically. Optionally traffic lights, stop signs, or turn circles can be added on the road network, attached to some node. An example of road map scenario configuration of authors' office building is shown in the figure 3. The map is a virtual layer over the real environment, i.e. no need to decorate the room, but the map must be to scale, so that robot location in the real world can be showed correctly.

### 2.2 Robot Vehicle

Since one major objective of VC-bots is emulating behavior of real vehicles, we built 19 robot vehicles in 4 types, which are 7 VC-truck, 1 VC-van, 1 VC-sedan, and 10 VC-compact. The four different types of robot vehicles varies in payload, mobility, sensing, computing capabilities, cost, and battery life, shown in the figure 4. For VC-truck and VC-van, significant more payload can be carried, so we put

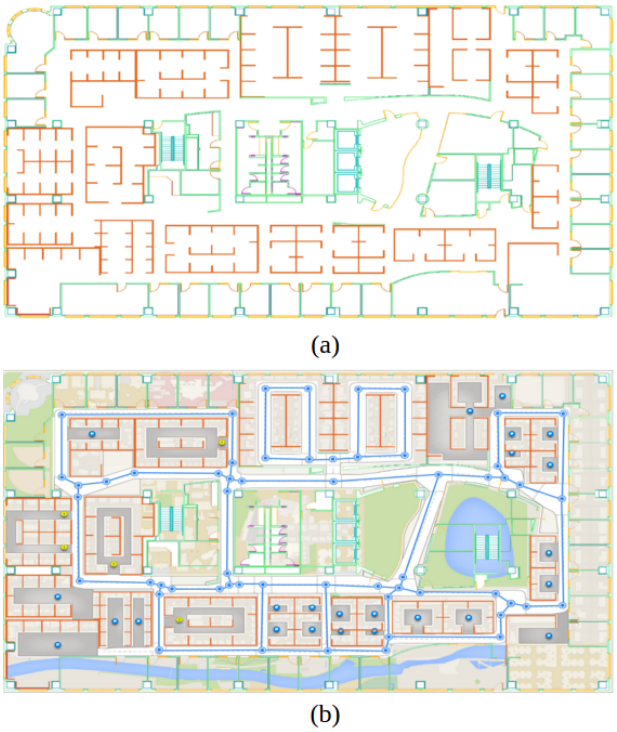

**Figure 3: (a) Original floor plan; (b) Map and road network after configuration**

more sensors and a full fledged computer running Linux with large battery as on board computer for both data intensive processing, robot management, and mini cloud resources. As a trade-off, their maneuverability has more restriction than VC-sedan and VC-compact. Due to form factor and cost restriction, VC-sedan and VC-compact have limited sensor and computation resources, specifically they are not equipped with a powerful on board computer, which is essential for the on board mini cloud. On the other hand, they can accelerate, move and turn relatively fast. Because of these differences, we use VC-truck and VC-van to represent self-driving car and VC-sedan and VC-compact emulate vehicles with less autonomous features, because we believe that both of them will coexist on the road in the near future. The dimension of our robot vehicles is roughly 1/40 to 1/20 of real vehicles, and their mobility performance can be configured to emulate the dynamics of various vehicles. For example, even though it is relatively easy to brake VC-truck on carpet, it is programed to decelerate for a certain amount of time before stop, just as real heavy duty trucks. Major hardware components and specification of our 4 types of robots are listed in table 1.

Among the four types of robot vehicles, VC-sedan uses rear wheel drive model, and the other three types of robot use differential drive with 2, 4 and 6 wheels respectively. Motion control is conducted by two layers of software, low level controller (LLC, running on Arduino), which interfaces motor and sensor, and high level controller (HLC, running on Raspberry Pi). LLC measures wheel speed with rotary encoder and send the sensed data to HLC. At HLC, motor control signals are generated by a two level proportional-integration-differential (PID) control algorithm based on current motion states and desired states. Then the control signals are handled by LLC, which generates pulse width modulation (PWM) signal to drive the motors. Motion states contain three types, position and orientation (x, y, $\theta$), linear speed and angular speed (v, $\omega$), and linear acceleration (a). Several sensors are used to measure these states. In detail, rotary encoders sense wheel rotation speeds, which further derive (v, $\omega$) through kinematics model and (x, y, $\theta$) by dead reckoning. To compensate accumulated error in dead reckoning, we use a LIDAR sensor with simultaneous localization and mapping (SLAM) software to correct position, and inertial measurement unit (IMU) sensor to correct orientation. IMU sensor can also tell us angular speed and linear acceleration. Based on these sensors, we designed multiple controllers. One is the cruise controller, i.e. (v,$\omega$) controller, which drives the robot with certain linear speed and angular speed. Another is the planar stabilizing controller i.e. (x,y) controller, which drives the robot to a given position regardless of orientation.

**Table 1: Hardware configuration**

| Robot type | VC-truck | VC-van | VC-sedan | VC-compact |
|---|---|---|---|---|
| Size (L*W*H) ($cm^3$) | 49*31*35 | 28*24*25 | 28*16*21 | 20*19*17 |
| Weight (Kg) | 8.7 | 4.6 | 1.33 | 0.83 |
| Payload(Kg) | 5 | 2 | N/A | N/A |
| MaxSpeed (m/sec) | 0.5 | 0.8 | 5 | 2.3 |
| Onboard Computer | Intel NUC | Intel NUC | N/A | N/A |
| Wheel Encoder (CPT) | 3592 | 2797 | 20 | 40 |
| Sensors | Pi Camera, IMU, Ultrasonic Sensor | Pi Camera with pan-tilt, IMU | USB wide-angle Camera with pan-tilt, IMU | Pi Camera, IMU, Ultrasonic Sensor |
| Localization | Odomety, LIDAR, GPS | Odomety, LIDAR, GPS | Odomety, LIDAR | Odomety |
| LIDAR Range, frequency, Resolution | 6 m , 5 Hz, 1° | 4 m , 10 Hz, 0.36° | 3 m , 4 Hz, 1° | N/A |
| Network | WiFi × 4, LTE modem | WiFi × 4, LTE modem | WiFi × 2 | WiFi,× 2 |
| Battery for Motor | 7.2V 20Ah NiMh | 14.8V 7.8Ah Lithium | AA × 6 | AA × 4 |
| Battery for Electronics | 19V 23Ah Lithium | 19V 12Ah Lithium | 5V 6.4Ah Lithium | 5V 6.4Ah Lithium |

When moving on the road, after the robot vehicle gets the path consisting of line segments and arcs, HLC generates desired (v,$\omega$) sequence which determine a practical trajectory satisfying practical hardware constraints such as minimum turning radius and maximum turning speed. Then these sequence are sent to (v,$\omega$) controller to drive the robot. If the robot vehicle gets a collection of (x,y) coordinates, (x,y) controller is directly used. By updating (x,y), step by step the robot can reach the final goal. For freely traversal area like parking lots, the robot vehicle just use (x,y) controller to drive it to the desired position obtained from UI.

For VC-truck and VC-van, we install ROS [12] with Hector SLAM on the on board computer, and use them to obtain location together with LIDAR. Each robot vehicle's initial location and orientation are manually aligned and set, so that using LIDAR we can get its location in the absolute world reference frame. We also encapsulate most controller functions as well as sensor interface to provide them as RESTful web services. On board applications can use this interface to obtain sensor data or operate the vehicle.

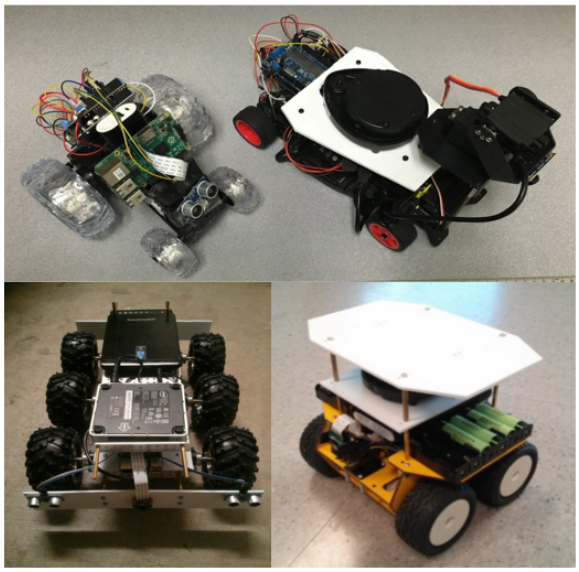

**Figure 4: VC-truck (bottom left), VC-van (bottom right), VC-sedan (top right), VC-compact (top left)**

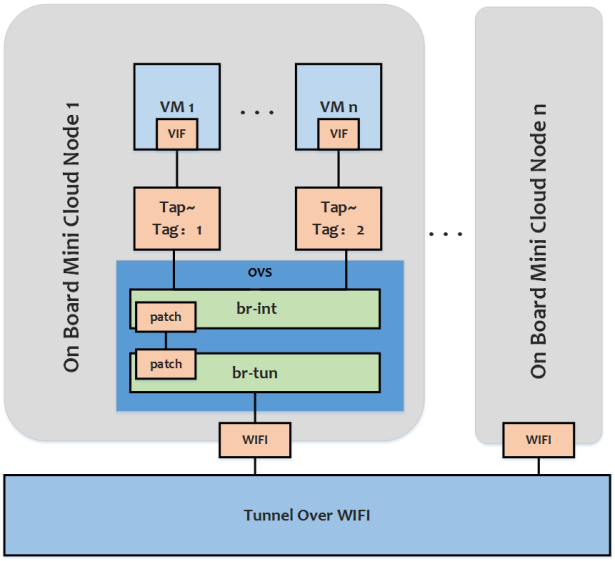


**Figure 5: Network Model**

## 2.3 Network Model

Since VC-bots is designed as a testbed, multiple network access methods are supported. First, we use an independent WiFi network exclusively for robot management and remote control. Additionally, each robot is equipped with an LTE modem in case it is out of the range of management WiFi network. Second, each robot vehicle is equipped with multiple WiFi interface, and several WiFi routers are deployed as road side unit (RSU) which are connected to the Internet. One interface is setup as WiFi access point, and other interfaces can either connect to another robot vehicle for vehicle-to-vehicle (V2V) communication, or connect to an RSU for vehicle-to-infrastructure (V2I) communication. The connection between robot vehicles can be formed by sending commands to robot vehicle over the management network, which is used by the UI to allow user to setup network topology. We call this V2V and V2I network application data network because it is mainly used by actual vehicular applications to communicate with each other. Third, three software defined radio (SDR) cards are mounted on three VC-truck for potential requirements of testing custom MAC layer and physical layer protocols.

We installed Open vSwitch (OVS) on VC-truck and VC-van's onboard computer to support software defined network (SDN) over the robot vehicles, RSU, and remote cloud. For each pair of vehicles which are connected through direct V2V communication, a generic routing encapsulation (GRE) tunnel can be configured between their OVS. Also for each robot vehicle connected with the RSU, a GRE tunnel is configure between the vehicle and remote cloud. Like topology setup, all tunnel set up can also be done by remote command and UI. As shows in figure 5, within the mini-cloud of robot vehicle, VMs' virtual interface (VIF) are connected to TAP devices on the host. Then, each TAP device would be connected directly to the integration bridge, br-int. Integration bridge performs VLAN patching and un-patching for traffic coming from and to VMs. After that, VLAN-tagged traffic are transferred from the integration bridge to GRE tunnel through the tunnel bridge, br-tun. The translation between VLAN IDs and tunnel IDs is performed by rules installed on br-tun. Also tunnel ID in the GRE header is used to differentiate between different networks. In this way a whole layer 2 broadcast domain, spanning over all connected robot vehicles and servers in remote cloud. The SDN controller is a server installed with OpenDaylight, which communicates to each OVS through OpenFlow over the management network.

## 2.4 Cloud Resource

In the convergence of both cloud computing and mobile computing, a net paradigm called edge computing or mini cloud has been created in the past a few years. Cloud resource in our testbed contains both robot vehicles on board mini cloud and remote cloud in the data center. The remote cloud is in a well established OpenStack cloud system. It consists of a few Linux servers managed by OpenStack to provide a cloud management user interface, centralized control and storage.

For the mini cloud, currently we installed KVM on VC-truck and VC-van to provide basic virtualization environment to isolate different applications from multiple users. Users will be able to create virtual machines (VM) or import existing VM images into the remote cloud or the mini cloud, then deploy their application in the virtual machine. Since each VM will be able to access the local network that the robot's on board system is connected to, users' applications are able to communicate directly with robot controller and sensors through RESTful web service or using the message based service framework mentioned below. We are also planning to allow user to create VM overlays and provisioning VMs (as taking snapshot) and do live migration from a robot vehicle to another.

# 3. MESSAGE BASED SERVICE FRAMEWORK

To facilitate both management and user application, we

propose a convenient and lightweight communication middleware for service composition and data sharing in our testbed. Currently it is built upon RabbitMQ [1] over TCP/IP. In our framework every robot vehicle has a local message broker (i.e. RabbitMQ server). There are two global message brokers in the remote cloud, one for application in the remote cloud and the other dedicated to delivering remote control commands from the mobility management UI. Software components (i.e. RabbitMQ client) which need communication are abstracted as an object and connected to the broker. Instead of address, object names containing multiple segments separated by dot are used. For example, "robot1.mctl" is the object representing motion control component of our 1st robot. All message communication follows advanced message queue protocol (AMQP) version 0.9.1.

Services can be composed with our framework by objects exposing methods, and RPC on the method can be made from another object of network. For example, "robot1.mctl.setSpeed" means a method to allow the remote control from management software. This is implemented by creating a queue with the method name. The motion control module listens to the queue and handles messages. Also an object can advertise an event source as publisher for any other object who is interested. For example, "robot1.mctl.location" means an event source that publishes the current location of the robot. This allows robots to share sensor reading (or synthesized environment perception result) with other vehicles or applications in remote cloud. This is implemented by creating a fan out exchange with the event source name. Any subscriber can create a queue bond to the exchange to receive corresponding events. In our testbed controllers, sensors and other components such as path planning all exposes their interfaces both through robot vehicle local message broker and global message brokers.

We also developed an adapter to bridge ROS [12] message IPC and our framework because these two service models can be mapped to ROS service and topic semantics. We chose not to directly use ROS for several reasons. First, our framework is very lightweight (just a library on the client side), which does not require the build system and other depended components in ROS. Second, VC-sedan and VC-compact does not have a full fledged on board computer to run ROS, while they can still use our framework as long as they can run Java or Python code. Third, our framework can be easily set up with secure communication over SSL between any message broker and any software component connected to the message broker. This is necessary because the network between robot vehicles and remote cloud heavily relies on the public Internet.

# 4. EXAMPLE APPLICATION AND EVALUATION

As an example of vehicle cooperation application, we demonstrate how to implement longitudinal platooning algorithm [14] with our testbed, i.e. several vehicles driving in the same direction form a line cluster on the road with relatively small separation distance and all vehicles except the platoon leader will drive automatically. First each robot vehicle with our platooning implementation is a cluster whose header is itself. The platooning software component registers a method "robotx.platoon.join", a method "robotx.platoon.follow", and a event source 'robotx.platoon.reportState" on its local message broker. Then other platooning members directly connect to the platooning header through V2V communication, and call the header's join method as well as subscribe the state reporting event source. The return of join method is the name of the robot vehicle at platoon tail. Since platooning algorithm also requires each vehicle to send its acceleration and speed to its successor in the cluster, the new platooning member also connects to the platoon tail and call its predecessor's follow method and subscribe the state reporting event source. After successfully joining the cluster, the platooning algorithm regularly calculates desired speed based on predecessor's states, separation distance as well as the header's state to maintain stability of the whole cluster. The platooning application invokes the controller interface to drive in the calculated desired speed. Platoon leaving is implemented in the reverse order by calling the an unfollow method on the predecessor, a leave method on the head, and unsubscribing the event sources.

**Table 2: VC-compact Mobility Experiment Results**

| | Straight Line Movement(5m) | | X-Y Movement (1.5m,1.5m) | |
|---|---|---|---|---|
| # trail | Longitudinal Error(cm) | Lateral Error(cm) | X Error (cm) | Y Error (cm) |
| 1 | 4.2 | 5.1 | 11.3 | 4.6 |
| 2 | 6.7 | 8.8 | 8.2 | 10.6 |
| 3 | 8.0 | 8.9 | 7.8 | 8.0 |
| Average | 6.3 | 7.6 | 9.1 | 7.7 |
| Percentage | 1.3% | 1.5% | 6.1% | 5.2% |

Since VC-compact has cheapest chassis, motor, and encoder, and it does not equipped with LIDAR sensor, its mobility control would be the most difficult. So we evaluated the controller performance of VC-compact as a baseline through two types of experiments, and 3 trials in each type. First the robot vehicle moves in a straight line of 5m using the trajectory following controller. Second the robot vehicle starts at original point heading toward x-axis and moves to a specific coordinate (1.5m, 1.5m) using planner stabilizing controller. In both experiments difference between its final position and expected position is measured after it stopped. The result is shown in the table 2. Generally line or arc following can achieve about 1.5% error because IMU can be used to adjust orientation, while moving to an arbitrary coordinate can not utilize IMU feedback so the error is higher. However, in both experiments the position difference can exceed several centimeters and it will accumulate as the robot vehicle continue moving, so using exteroceptive sensors such as LIDAR to remove such error is necessary.

We also evaluated the message performance our service framework with two experiments using one VC-van, one RSU, and one server in remote cloud. The robot vehicle was connected to the RSU directly with 802.11g WiFi, and the server was connected to the RSU directly with 1Gb Ethernet. A global message broker is setup in the server. A Java program running on the server emulates application in remote cloud, and another Java program running on the on board computer of the robot vehicle emulates robot application. They both connect to the global message broker. First the robot vehicle makes 1000 RPC back-to-back on a method provided by the server with various payload

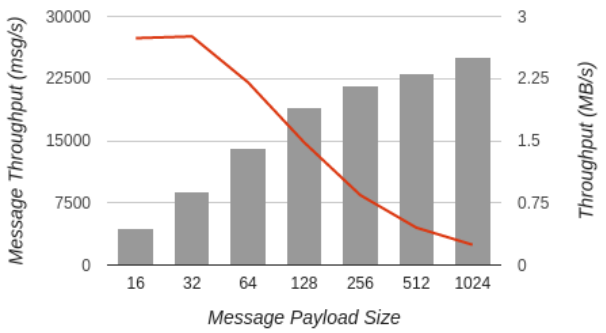


**Figure 6: Message Communication Throughput**

size form 16 to 1024 bytes. Meanwhile we also ping the server from the robot vehicle 50 times with the same payload size. The average RPC round trip time (RTT) is 3.1 ms. Compared with the 2.7 ms ping RTT, delivery delay in the message broker is not significant. Next the robot vehicle creates an event source, which is subscribed by the server. Then the robot vehicle publishes 100,000 events. The experiment was done in 7 times with different event payload size. As shown in the figure 6, we measured the payload throughput in MB/s (the grey bars) and the message throughput (the red line). Compared with the theoretical 54Mbps bandwidth of WiFi, the publisher / subscriber payload throughput achieved at most 37% of the total wireless bandwidth. Although message overhead is one cause for performance degradation when each message has a small payload, we believe message level acknowledgement between client and broker in AMQP is also a reason.

## 5. CONCLUSIONS AND FUTURE WORKS

VC-bots is an evolving platform for testing vehicular network and vehicular cloud applications, and construction of VC-bots presents a significant engineering challenge of incorporating and applying knowledge of multiple disciplines. Currently we only built a proof-of-concept prototype and further development of of VC-bots is still work-in-progress. In the future, we are also planning to develop an SDN controller and topology manager that can set up connection and flow table utilizing location information. Moreover, a UDP based implementation of message protocol is under development to substitute RabbitMQ for the service framework, since current AMQP implementation mainly use TCP, which is not the best choice for mobile network environment.

## 6. ACKNOWLEDGEMENT

This research is sponsored by ARO grant W911NF-11-1-0191, NATO grant SPS984425, and NSF SaTC 1528099.

## 7. REFERENCES

[1] Rabbitmq. https://www.rabbitmq.com/. Accessed: 2016-04-13.

[2] C. Ameixieira, A. Cardote, F. Neves, R. Meireles, S. Sargento, L. Coelho, J. Afonso, B. Areias, E. Mota, R. Costa, et al. Harbornet: a real-world testbed for vehicular networks. *Communications Magazine, IEEE*, 52(9):108–114, 2014.

[3] S. Biddlestone, A. Kurt, M. Vernier, K. Redmill, and Ü. Özgüner. An indoor intelligent transportation testbed for urban traffic scenarios. In *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*, pages 1–6. IEEE, 2009.

[4] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau. C-vet the ucla campus vehicular testbed: Integration of vanet and mesh networks. In *Wireless Conference (EW), 2010 European*, pages 689–695. IEEE, 2010.

[5] T. A. Dahlberg, A. Nasipuri, and C. Taylor. Explorebots: a mobile network experimentation testbed. In *Proceedings of the 2005 ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis*, pages 76–81. ACM, 2005.

[6] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G. S. Sukhatme. Robomote: enabling mobility in sensor networks. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 55. IEEE Press, 2005.

[7] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. A. Syed, S. Sharma, and T.-c. Chiueh. Mint-m: an autonomous mobile wireless experimentation platform. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 124–137. ACM, 2006.

[8] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: vehicular content delivery using wifi. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 199–210. ACM, 2008.

[9] R. Fish, M. Flickinger, and J. Lepreau. Mobile emulab: A robotic wireless and sensor network testbed. In *IEEE INFOCOM*, 2006.

[10] M. Gerla. Vehicular cloud computing. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2012 The 11th Annual Mediterranean*, pages 152–155. IEEE, 2012.

[11] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. Cartel: a distributed mobile sensor computing system. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138. ACM, 2006.

[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[13] J. Reich, V. Misra, and D. Rubenstein. Roomba madnet: a mobile ad-hoc delay tolerant network testbed. *ACM SIGMOBILE Mobile Computing and Communications Review*, 12(1):68–70, 2008.

[14] S. Sheikholeslam and C. A. Desoer. Longitudinal control of a platoon of vehicles. In *American Control Conference, 1990*, pages 291–296. IEEE, 1990.

[15] H. Soroush, N. Banerjee, A. Balasubramanian, M. D. Corner, B. N. Levine, and B. Lynn. Dome: a diverse outdoor mobile testbed. In *Proceedings of the 1st ACM International Workshop on Hot Topics of Planet-Scale Mobility Measurements*, page 2. ACM, 2009.