BRIEF REVIEW ON WIRELESS ROS MASTER- SLAVE COMMUNICATION USING EMBEDDED IoT Device: NodeMCU

Jignesh Patoliya[1], Priyank T. Rajai[2], Jaskiran Kaur S. Saini[3], Karan Khatwani[4]
[1] Assistant Professor, [2,3,4]Student
[1]V. T. Patel Department of Electronics and Communication Engineering,
Charotar University of Science and Technology, Nadiad, India.

Received: February 12, 2019 Accepted: March 26, 2019

ABSTRACT: *As the world is evolving, wired communication system is getting out-dated. We want our systems to look less congested and more manageable. And hence to avoid mess, the world is going wireless now. Wireless communication has gained a lot of importance in the last decade. There are many ways to obtain that but in this paper we are going to discuss about how to obtain wireless- communication using ROS. This can be possible using TCP Communication on NodeMCU.*

Key Words: *ROS, Node-MCU, TCP communication, Wireless mode, Master-Slave Communication.*

I. Introduction

ROS is a Linux–based , open-source, middleware framework for modular use in robot applications. ROS, originally designed by Willow Garage and currently maintained by the Open Source Robotics Foundation, is a powerful tool because it utilizes object-oriented programming, a method of programming organized around data rather than procedures in its interaction with data and communication within a modular graph system. ROS is divided into three conceptual levels: the filesystem level, the computation graph level, and the community level [1].

1. Filesystem Level

The filesystem level is the organization of the ROS framework on a machine. At the heart of the ROS's organization of software is the package. A package may contain ROS runtime execution programs, which are called nodes, a ROS-independent library, datasets, configuration files, third-party software, or any software that should be organized together. The goal of the packages is to provide easy to use functionality in a well-organized manner so that software may be reused for many different projects. This organization, along with object-oriented programming, allows packages to act as modular building blocks, working harmoniously together to accomplish the desired end-state.[2] Packages typically follow a common structure and usually contain the following elements: package manifests, message types, service types, headers, executable scripts, a build file, and runtime processes

2. Computation Graph Level

The computation graph level is where ROS processes data within a peer-to-peer network. The basic elements of ROS's computation graph level are nodes, messages, topics, services, bags, Master, and Parameter Server. Nodes are the small-scale workhorses of ROS, subscribing to topics to receive information, performing computations, controlling sensors and actuators, and publishing data to topics for other nodes to use. The primary method in which nodes pass data to each other is by publishing messages to topics. A message is simply a structuring of data so it is in a useful, standard format for other nodes to use. Standard types, such as integer, floating point, and Boolean, are supported as well as arrays.

Rather than communicating directly with each other, nodes usually communicate through topics. Topics are named hubs in which nodes can publish and subscribe and are the crux of what makes ROS an object-oriented and modular environment. Nodes that generate data are only interested in publishing that data, in the correct message format, to the correct topic. Nodes that require data simply subscribe to the topics of interest to pull the required information. Multiple nodes may publish or subscribe to a single topic as shown in Figure 1. This method of publishing and subscribing to topics decouples the production of information from the consumption of information. It allows nodes within different packages to work harmoniously with each other even though they may have different origins and functions.

In addition to publishing messages to topics, nodes can also exchange a request and response message as part of a ROS service. This is useful if the publish and subscribe (many-to-many) communication method is not appropriate, such as a remote procedure call [3].A ROS node that provides data offers a service under a

string name, and a client node that requires data calls the service by sending the request message and awaiting the response.

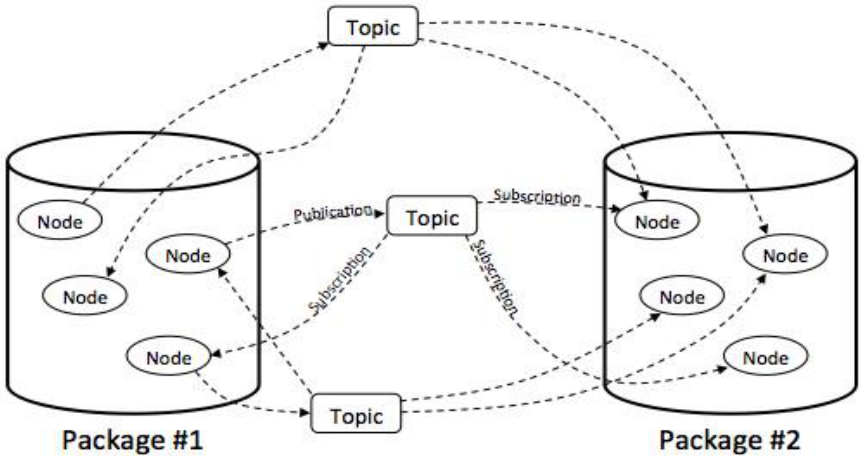

figure 1. Model of how the ROS nodes publish and subscribe to topics[4].

Bags are a method for recording and storing ROS message data. This is a powerful tool that allows users tostore, process, analyze, and visualize the flow of messages. Bags are created utilizing the rosbag tool, which subscribes to one or more ROS topics and stores message data as they are received. This stored data can be replayed in ROS to the same topics, as if the original nodes were sending the messages. This tool is useful for conducting experiments using a controlled set of data streams to test different algorithms, sensors, actuators, and controllers.

A launch file is method of launching multiple ROS nodes, either locally or remotely, as well as establishing parameters on the ROS Parameter Server. It is useful for running large projects, which may have many packages, nodes, libraries, parameters, and even other launch files, which all can be started via one launch file rather than individually running each node separately.

The ROS Master acts as a domain name system server, storing topic's and service's registration information for ROS nodes. ROS Master provides an application program interface (API), a set of routines and protocols, tracking services and publishers and subscribers to topics. A node notifies ROS Master if it wants to publish a message to a topic. When another node notifies the master that it wants to subscribe to the same topic, the master notifies both nodes that the topic is ready for publishing and subscribing. The master also makes callbacks to nodes already online, which allows nodes to dynamically create connections as new nodes are run

3. Community Level

The ROS Community Level consists of ROS distributions, repositories, the ROS Wiki, and ROS Answers, which enable researchers, hobbyists, and industries to exchange software, ideas, and knowledge in order to progress robotics communities worldwide. ROS distributions are similar to the roles that Linux distributions play [5]. They are a collection of versioned ROS stacks, which allow users to utilize different versions of ROS software frameworks. Even while ROS continues to be updated, users can maintain their projects with older more stable versions and can easily switch between versions at any time.

ROS does not maintain a single repository for ROS packages; rather, ROS encourages users and developers to host their own repositories for packages that they have used or created. ROS simply provides an index of packages, allowing developers to maintain ownership and control over their software. Developers can then utilize the ROS Wiki to advertise and create tutorials to demonstrate the use and functionality of their packages. The ROS Wiki is the forum for documenting information about ROS, where researchers and developers contribute documentation, updates, links to their repositories, and tutorials for any open-sourced software they have produced. ROS Answers is a community-oriented site to help answer ROS-related questions that users may have.

II. STRUCTURE

The method we will use to implement wireless communication would be using NodeMCU. We have seen incorporating it as a Wi-fi module or as a microcontroller till now, but taking a step ahead we will use it with our latest technology ROS. The reason being that it is much more powerful and cheaper than AVR Arduinos with more memory and speed and integrating Wi-fi for 10 times less cost [7]. The vision is that every sensor

or actuator node in a ROS-robot can be created using a single ESP8266, with just power supply or power bank battery, avoiding harness and reducing total cost.

We will use Transmission Control protocol (TCP) for peer to peer communication in NodeMCU. It is standard transport layer internet protocol which used in establishing and maintaining communication in between server and client, where every sensor/ slave is connected to a server/master and we have access to control them over the internet. NodeMCU has TCP/IP APIs available using which we can implement TCP Server/Client protocol on it. NodeMCU has Wi-Fi functionality available on board. With this Wi-Fi functionality NodeMCU can connect to any Wi-Fi network as client or it can create a network to which other Wi-Fi enabled devices can connect.

NodeMCU Wi-Fi subsystem is running in background tasks periodically. If any function takes more than 15 milliseconds, it may cause the Wi-Fi subsystem to crash [8]. To handle such functionalities NodeMCU has their APIs through which we can control this subsystem. NodeMCU has four Wi-Fi modes as,

Station (STA) Mode:

In this mode, NodeMCU joins the existing networks. The NodeMCU connects the existing Wi-Fi router. This provides the internet access to the NodeMCU through the Wi-Fi router.

Access Point (AP) Mode:

In this mode, NodeMCU creates its own network and others can join this network. Here it has local IP address with which other devices can connect to it. NodeMCU assigns next available IPs to the other devices.

Station + Access Point (BOTH) Mode:

This is the mode, where it creates its own network while at the same time being joined to another existing network.

Wi-Fi OFF Mode:

In this mode, Wi-Fi remains OFF [8].

In this we will use NodeMCU Wi-Fi which has Access Point (AP) mode through which it can create Wireless LAN to which any Wi-Fi enabled device can connect

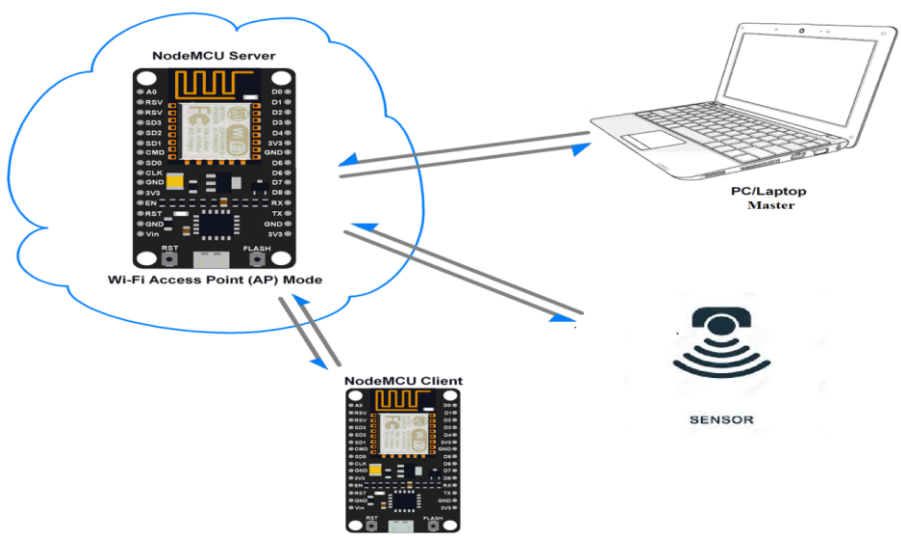

figure 2. Wireless ROS Setup.

We can set SSID and Password for AP mode which will be used to authenticate other devices while connecting to it. NodeMCU provides network APIs to use it as TCP server.

III. Methodology

As we know, in ROS we send messages via nodes. Nodes can be of two types, as mentioned earlier, publisher and subscriber. The message is sent via topics from publisher node to subscriber node. Here, we will have NodeMCU as one of node of ROS and accordingly ROS will send data to it. This would act as a publisher node and further send the message/data to other subscribers/sensors using Wi-Fi incorporated on it.

TCP/IP uses the client/server model of communication in which client is provided a service by master computer in the network. Collectively, the TCP/IP suite of protocols is classified as stateless, which means each client request is considered new because it is unrelated to previous requests[9]. Being stateless frees

up network paths so they can be used continuously. The transport layer itself, however, is state full. It transmits a single message, and its connection remains in place until all the packets in a message have been received and reassembled at the destination.

Firstly we add SSID and password of Wi-Fi in the code and also we mention the IP address of ROS master in code. Now this code will be complied and then dump in NodeMCU. In the ROS master we first use the command **roscore** to make the master active to send the data to NodeMCU. On new terminal a command **rosrun rosserial_python serial_node.py tcp**is run. Finally, we can control NodeMCU by sending data/messages via ROS-master and hence forming master slave communication. The entire communication here takes place wireless using Wi-Fi.

Advantages of working with ROS:

- Peer to peer communication - Complex robotic system with multiple links may have multiple on-board computers connected via a network. Running a central master would result in severe congestion in one particular link. Using a peer-to-peer communication would avoid this issue. In ROS, a peer-to-peer architecture coupled to a buffering system and a lookup system (a name service called 'master' in ROS), enables each component to dialogue directly with any other, synchronously or asynchronously as required.
- Free and open-source: Being open-source platform provides reuse of already existing functions provided by the many other ROS users. Their code is supplied in repositories as "stacks". Other people have developed amazing capabilities for robots that have been "open-sourced" and are relatively easy to add incrementally using the ROS development environment.
- Thin: To combat the development of algorithms that are entangled to a lesser or greater degree with the robotics OS and are therefore hard to reuse subsequently, ROSdevelopers intend for drivers and other algorithms to be contained in standalone executable. This ensures maximum reusability and, above all, keeps its size down. This method makes ROS easy to use, the complexity being in the libraries. This arrangement also facilitates unit testing and the developed systems can be completely independent of another system [10].
- Multi-language: ROS is language-neutral, and can be programmed in various languages. The ROS specification works at the messaging layer. Peer-to-peer connections are negotiated in XML-RPC, which exists in a great number of languages. To support a new language, either C++ classes are re-wrapped (which was done for the Octave client, for example) or classes are written enabling messages to be generated. These messages are described in IDL (Interface Definition Language)[11].

IV. CONCLUSION

An attempt is made in this paper to review wireless communication in the modern world using latest technology named ROS. All the study is summarized in this paper step by step .Working with ROS has lot of benefits, which we referred in this paper and there are few drawbacks too, out of which a major one is wired communication. On the basis of entire discussion, it can be concluded that wireless communication is possible using ROS and NodeMCU. Till today, NodeMCU was used as IOT device, but now a step forward is taken to incorporate it with ROS. With help of NodeMCU it can establish TCP communication in network. Moreover, we also discussed the different modes in which TCP can be used and the best suitable mode for it to work with ROS. Therefore we tried to combine both TCP and ROS on NodeMCU to turn the entire perspective into win-win situation by using the plus sides of both.

References

1. Jason M. O'Kane 2013. A Gentle Introduction to ROS
2. Lentin Joseph 2015. Learning robotics using python.
3. Wyatt Newman 2017. A systematic approach to learning robot programming with ROS first edition .
4. Anil Mahtani, Enrique Fernandez, Luis Sanchez 2016. Effective Robotics Programming with ROS
5. Lentin Joseph 2015. Mastering ROS for Robotics Programming
6. R. Patrick Goebel 2014. ROS By Example Volume 2
7. R. Patrick Goebel 2014. ROS By Example Volume 1
8. Joshua S. Lum 2015. Utilizing ROS in Robot Vision and Control
9. Brian Gerkey, William D. Smart, Morgan L. Quigely 2015. Programming Robots with ROS: A Practical Introduction to ROS
10. Antonio Marin-Hernandez 2014. ROS Tutorial: Robotics Operating System (Thesis)
11. Tim De Jager 2013. Robust Object Detection For Service Robotics