On supporting Multi-hop communication in the AllJoyn framework for Proximity Based Networks

Hatim Lokhandwala

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology

Department of Computer Science and Engineering

June 2016

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Hatim Lokhandwala)

_____

(Roll No.)

Approval Sheet

This Thesis entitled On supporting Multi-hop communication in the AllJoyn framework for Proximity Based Networks by Hatim Lokhandwala is approved for the degree of Master of Technology from IIT Hyderabad

(Dr.) Examiner

(Dr. Antony Franklin) Examiner
Dept. of Computer Science and Engineering
IITH

(Dr. Bheemarjuna Reddy Tamma) Adviser
Dept. of Computer Science and Engineering
IITH

(Dr. Kotaro Kataoka) Chairman
Dept. of Computer Science and Engineering
IITH

Acknowledgements

I am profoundly grateful to my thesis advisor Dr. Bheemarjuna Reddy Tamma for his constant help, motivation, support and technical guidance. I express my sincere appreciation for his encouragement and advice that enabled me to pursue this research study. I extend my special thanks to Srikant Manas Kala, Dr. Satish Srirama, Dr. Chi Chang, Mukesh Kumar Giluka and Rashi Gehani for their inputs and help during the course of my thesis. I would also like to thank Dr. Antony Franklin for providing inputs in design and development of enhancements in the algorithms for ad hoc networking which were of immense help in my work. Most importantly, I am forever grateful to my parents (Mr. Mustafa Lokhandwala and Mrs. Tasneem Lokhandwala), brother (Hussain Lokhandwala) and the Almighty for being my strength and their timely support and motivation during my graduation.

Dedication

I dedicate this thesis to my parents, my brother and the people who lost their families during the Chennai floods in November - December, 2015.

Abstract

Internet of Things (IoT) is poised to empower billions of physical objects and devices with sensing, computing and communication capabilities. An important aspect of an IoT network is the connection of these physical objects to the Internet. Another fundamental aspect is concerned with the communication among devices which are in proximity of each other in typical IoT network applications such as smart homes, smart grids, smart transportation systems etc. This leads to the notion of Proximity Based Network (PBN) which is a dynamic, ad hoc network so formed by the devices in proximity of each other. These PBNs are opportunistic networks that enable devices to identify and communicate with each other with minimal need of a communication infrastructure. PBNs further give rise to the domain of Mobile Social Networks in Proximity (MSNP) which are opportunistic social networks formed by devices in proximity.

This thesis employs and explores AllJoyn framework, an open source framework from Qualcomm which facilitates dynamic and ad hoc discovery between devices that are in proximity of each other and enables communication between them. A study of the functional and operational models of the framework is performed. Further, comparison analysis of the framework with other similar technologies that enable proximity based networking is carried out. The analysis reveals that AllJoyn framework emerges to be a better choice due to its support for a plethora of devices of varying specifications. This renders the framework as a suitable platform for various use cases and applications specifically in the context of proximity based networking and IoT where in the participating devices are of different capabilities. A proximity based Android application prototype, *Min-O-Mee* is designed and developed utilizing the AllJoyn framework. The application harnesses spatial proximity of the devices and facilitates information exchange between them. The application serves as a proof-of-concept and could be enhanced or modified for several of IoT and other applications.

The AllJoyn framework however being in its nascent stage offers only skeletal communication stacks, with enormous possibilities for technological enhancements. The thesis further identifies the extensions and advancements that could be incorporated into the framework. Multi-hop communication mechanism is identified as a potential and significant enhancement to the framework due to its relevance in various applications where AllJoyn framework is employed. Approaches for integration of the AllJoyn framework and the Better Approach Towards Mobile Ad hoc Networking (B.A.T.M.A.N) routing have been devised to realize multi-hop communication between devices using the AllJoyn framework. Integration mechanism facilitate devices to discover other devices in ad hoc mode and enable communication between AllJoyn applications on devices which are single hop or multiple hops away from each other. Three integration mechanisms : Naive approach, Improved approach and Ask / Reply approach have been devised with the goal of reducing control information generated for service advertisement. Finally, the developed integration mechanisms have been tested on a real-time test-bed with different topologies comprising of different types of devices (Raspberry PIs, laptops) with external Wi-Fi adapter attached to them. A performance evaluation of the integration mechanisms is carried out through the experiments in real-time test-bed on metrics : Control overhead, AllJoyn service discovery time and network throughput and the trade offs between these metrics across different approaches is determined. A significant reduction of 49.1% and 30.9% in control overhead is observed in Improved approach and Ask / Reply approach in comparison to Naive approach. However service discovery time in Improved and Ask / Reply approach is observed to be more than the double of the service discovery time in Naive approach.

Contents

Chapter 1

Introduction

Internet of Things (IoT) envisions to interconnect myriad things and entities that exist in our surroundings. These entities resemble tangible, identifiable objects in the physical world that are an integral part of our day today lives. They include but are not limited to vehicles (cars, buses, ships, trains, etc.), electrical appliances (fans, air conditioners, light bulbs, coolers), power-meters, civil structures (roads, bridges, buildings), clothing, wearables, etc. IoT aims to incorporate sensing, computation and communication capabilities in billions of devices and objects. IoT would foster realization of seamless integration of the physical world with the global information network (Internet). This would promote new business and revenue generation opportunities [1] and offer significant solutions in diverse domains such as health care [2], industrial manufacturing, logistics and supply chain, civil infrastructures maintenance, home automation [3], transportation systems [4], agriculture [5], environment monitoring [6], etc.

One of the key challenges in IoT is the unique identification of billions of physical entities to ensure reliable access to devices through accurate addressing and build a robust communication framework of these devices over the Internet. Internet Protocol version 6 (IPv6) : a new version of IP protocol, offers a larger address space with its 128-bit addressing mechanism resulting into 2^{128} or approximately $3.4 * 10^{38}$ addresses [7]. This would resolve the problem of addressing and unique identification for billions of devices that would form a part of an ubiquitous network with the evolution of IoT. Authors in [8] presents the use of IPv6 scheme for scalability requirements of IoT.

Another aspect in IoT networks relates to communication amongst the devices and objects in context of particular applications and use-cases. The key challenge here is the diversity and heterogeneity of devices in a typical IoT network. Different objects and devices would have varying computation capabilities (RAM, CPU, Network Bandwidth, storage memory, etc.). Besides, they may function on different underlying platforms (hardware and operating systems), have varying form factors and employ different communication protocols. In addition, they may be manufactured by different vendors and operated under the control of different administrative units. These intricate complexities require utilities, tools and technologies that could offer interoperability and facilitate robust communication and information exchange between such diverse range of devices.

AllJoyn framework from AllSeen Alliance [9] is designed and developed along similar motivations and objectives. AllSeen Alliance is a cross-industry consortium involving leading industry partners, dedicated to enabling the interoperability of billions of devices, services and applications that comprise the Internet of Things [9]. AllJoyn is an open source software framework that makes it easy

for devices and applications to discover and communicate with each other. Applications can operate with the abstraction of the underlying transport, hardware and OS and thus the applications could run on popular platforms such as Linux, Android, iOS, Windows, and many other lightweight real-time operating systems [9].

Due to its diverse features, AllJoyn framework can be employed for various practical applications in the field of IoT, gaming, education, disaster management, vehicular networks, proximity based communication, etc. In [10], AllJoyn framework integration with Lambda architecture (solution used for Big Data storage and analytics) is presented and examined in a smart home based case study. Likewise in [11], authors present an approach to incorporate strong security in IoT based smart home system which is realized through the AllJoyn framework. In this thesis we employ the AllJoyn framework to realize *Mobile Social Networks in Proximity (MSNP)*. A MSNP can be defined as an ad hoc, dynamic, peer-to-peer (P2P) social network, formed between devices or objects in physical proximity. MSNPs are a subclass of proximity based networks (PBNs) which in turn originate through interconnection between devices and objects in an IoT network but with a proximity constraint. *E.g.* An ad hoc network set up between certain number of vehicles in a smart transportation system can be referred to as a PBN. Here, integration of computing and communication capabilities in vehicles relates to notion of pervasive computing in IoT and the communication network set up between vehicles associates with PBNs). Thus IoT is a broader terminology involving sensing, communication, analytics, data management, decision making, large scale information processing and PBN be a subset of IoT dealing with dynamic communication between heterogeneous devices and objects in proximity. Thus IoT is a broader paradigm which involves sensing, communication, analytics, data management, decision making, large scale information processing and PBN thus becomes a subset of IoT operations dealing with dynamic communication between heterogeneous devices and objects in proximity.

1.1 Related Research Work

The emergence of PBNs can be traced back to the paradigm of IoT, which revolutionized modern wireless communication technologies. It propelled the idea that a variety of things present around us can be made to interact and communicate with each other [12]. These things may resemble electronic devices (Radio Frequency Identification (RFID) tags, hand-held devices, smartphones, wearables), vehicles, civil structures, electrical appliances and several other objects that exist in day today lives. The early practical applications of IoT made use of RFID tags [13] to solve problems of discovery, identification and tracking of goods, vehicles etc. Some commercial and management applications of the RFID technology are loading and tracking containers in the shipping industry, collecting highway tolls (EzPass system), automatic fare collection in public transit systems (SmarTrip), supply chain management, security and personal identification through smart keys [14].

PBNs are opportunistic networks (ONs), which are a subclass of MANETs. ONs make use of device-to-device communication along with multi-hop relaying. The most promising feature of ONs is that they are infrastructure-less networks i.e., they exploit wireless ad hoc networking techniques to establish communication between devices [21]. PBNs leverage the twin features of coverage extension and support extension that opportunistic networking offers. Through coverage extension, a mobile device which is not in the direct range of Radio Access Network (RAN) infrastructure is

connected to it through an intermediate node. Likewise, if a device interface is incompatible with the RAN interface, it can be offered access through an intermediate node which is connected to the RAN.

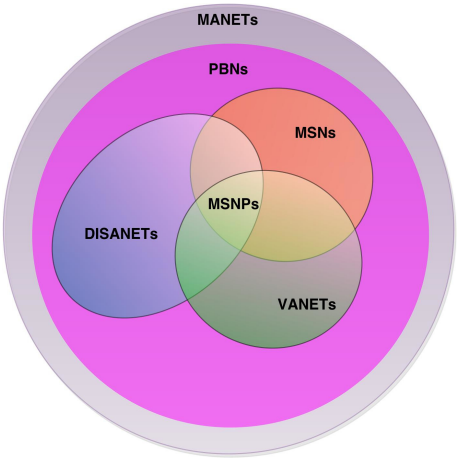


Figure 1.1: Classification of PBNs

PBNs can be further categorized into Disaster Networks (DISANETs), Vehicular Networks (VANETs) and Mobile Social Networks (MSNs) etc., as depicted in Figure 1.1. This classification is based on factors such as application scenario, network infrastructure, device mobility, data sharing mode i.e, real-time or deferred. The overlapping region in the Figure 1.1 between different categories of PBN indicates the common features or characteristics those categories share with each other. MSNP being at the core of the diagram signifies that it has inherited the common features of DISANETs, VANETs and MSN and also that it is not the functional aggregate of the other three categories. PBN architecture forms the basis for MSNP. An MSNP application operational over only Wi-Fi Direct is designed by authors in [15]. A detailed discussion on MSNP architecture and challenges is presented in [16]. Another mobile PBN application is proposed in [16], which focuses on real-time data sharing and human-computer interaction. An MSN based DISANET service for relief and rescue response in urban areas is proposed in [17], which the authors call emergency social networking solution. VANETs which form the backbone of intelligent Vehicular Communication Systems (VCS) require additional infrastructure in the form of stationary road side communication posts, in comparison to MSNs. In [18], authors address the challenge of reliable data transmission in highly dynamic and intermittently connected VANETs through an innovative epidemic protocol. Despite the variation in problem scenarios and the need of specific infrastructure, the different applications of a PBN share a large number of common challenges e.g., discovery and identification of devices, uninterrupted connectivity, device mobility, reliable data transmission etc. Thus, it is necessary to understand the characteristics of PBNs and then harness these features to tackle the prevalent communication issues.

1.1.1 Addressing the Challenges in PBNs

The common challenges faced in a core PBN and its enhanced variants e.g., MSNP, DISANETs, VANETs etc. are listed below. We name all the devices that are a part of a PBN as participating devices (PDs), for ease of reference. Discovery, identification and attachment of PDs and services.

Application	RAT
Serendipity [20]	Bluetooth
Peer2me [21]	Bluetooth
Haggle [22]	Bluetooth, Wi-Fi Direct
ShAir [23]	Bluetooth, Wi-Fi Direct
MSNP P2P App [15]	Wi-Fi Direct
MobiClique [24]	Bluetooth, Wi-Fi Direct

Table 1.1: Underlying RAT used by PBN application

- Addressing mechanisms for PDs.
- Session management.
- Mobility of PDs.
- Ensuring connectivity in a PBN, especially in context of PD mobility.
- Reliable data transfer and packet routing mechanisms.
- Ensuring security in anonymous PBNs.
- Threat to privacy due to multi-hop relaying of data.

PBN applications address these issues by employing existing frameworks and devising application specific mechanisms. For e.g., the MSNP application in [15], utilizes the Wi-Fi Direct framework to implement discovery of new devices. For privacy, the authors propose a three-tier protection mechanism to create an elastic network which is defined to be slightly more permanent than a simple PBN. But how this extended permanence is achieved and the factors on which it is predicated is not clear. In [19], authors propose routing requests through a P2P overlay network built on the geographical proximity of nodes instead of virtual closeness in the network. This again is an example of an application specific routing and path discovery mechanism. In addition to these communication challenges, there are considerations of the Radio Access Technology (RAT) being used to communicate. A PBN is essentially a radio based communication network and the RAT defines its underlying physical connection mechanism. Modern smart-devices support multiple RATs such as Wi-Fi, Wi-Fi Direct, Bluetooth and 2G/3G/4G. In Table 1.1, we list a few proximity based applications which include MSNPs, ONs, mobile peer-to-peer applications and middle-wares, and the respective RATs they communicate on.

It is evident that most PBN applications are not cross platform and communicate only over a limited set of radio technologies. Most of the existing PBN and MSNP applications are operational over limited platforms (software / hardware), set of devices, radio technologies and transports. The next generation wireless systems are envisioned to offer seamless network convergence, i.e., numerous RATs and the wired infrastructure will merge into a single communication delivery platform. But this can be only achieved if the constituent networks such as PBNs are cross platform and possess communication capabilities that span multiple radio technologies and physical layer interfaces. With the pervasive presence of smartphones, gadgets and various embedded devices, there would be a phenomenal increase in the relevance and use of PBN applications. Thus, there is a necessity of a versatile, cross-platform and open source framework which is capable of addressing the current and future challenges of PBNs. This need has been adequately met by the AllJoyn framework developed by Qualcomm.

1.2 Thesis Organization

The rest of the thesis is organized as follows :

- In Chapter 2, we present an operational overview of the AllJoyn framework, its benefits and the core components within the framework. A comparison analysis of the framework with other technologies for proximity based communications is presented.
- Chapter 3 presents a PBN / MSNP based application prototype designed and developed using the AllJoyn framework. Potential use cases where it could be used are highlighted, followed by functional details of the developed application.
- In Chapter 4, we highlight the bottleneck in the current working model of the framework and propose two technological enhancements that can be incorporated into the framework, describing their potential relevance in various application scenarios.
- Chapter 5 presents comparison study carried out between B.A.T.M.A.N and OLSR to determine the underlying routing protocol to be used for realization of multi-hop communication mechanism in the framework. Description of the B.A.T.M.A.N routing is presented thereafter in the chapter.
- Chapter 6 discusses in detail about the implementation aspects of integrating the AllJoyn framework and the B.A.T.M.A.N routing and the corresponding integration architecture. Various approaches for integration have been discussed elaborately along with the description of how multi-hop communication between AllJoyn services is realised.
- Chapter 7 describes about the topologies on which the integration mechanism are tested, configuration of devices used in testing, wireless network configuration and the logging mechanism to gather logs during the experiments.
- Chapter 8 presents the performance evaluation of integration approaches on certain identified metrics and the trade offs observed between those.
- Finally, Chapter 9 summarizes the conclusion and highlights the future course of research work.

Chapter 2

AllJoyn framework and its comparison with other similar technologies

2.1 AllJoyn framework

AllJoyn is an open source software framework that facilitates dynamic discovery of nearby devices and establish secure communication sessions with them. The framework enables proximity based networking amongst devices in proximity of each other. It is motivated and designed with the vision of Internet of Things (IoT). IoT would accompany myriad devices with varying specifications viz. different OSes, varying hardware architectures, form factors, language bindings etc. This would require a common means or a platform for the devices and eventually the applications running on them to connect and interact with each other. The inherent flexibility of the framework aptly fulfils these requirements. It is designed to be operational on multiple platforms ranging from embedded devices (Adruino, Raspberry PI, etc.) to high end smartphones and gadgets (RTOS, iOS, Windows, Android and MAC). Along with these it has support for following multiple transports as of now: Wi-Fi, Ethernet, Serial, PowerLine, and further support for more transports could be easily integrated.

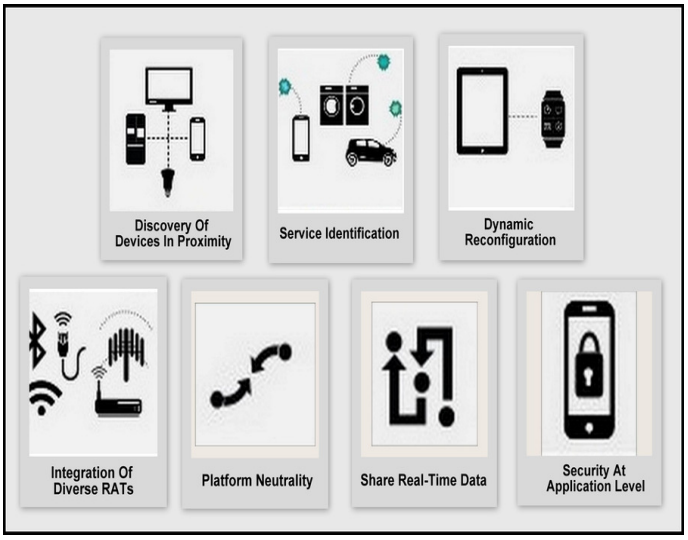


Figure 2.1: Features of The AllJoyn framework

Figure 2.1 depicts salient features of the AllJoyn framework and below we present the potential

benefits of the framework.

- An open source project and hence beneficial for research community as well as for commercial ventures.
- Offers a discovery and advertisement mechanism as an abstraction with support for heterogeneous transport technologies.
- Facilitates ad hoc and dynamic proximity based network configuration.
- AllJoyn implementations can be executed on multiple OSes, making it platform independent.
- Flexibility to the developers as they can pick programming language of their choice e.g., C, C++, Java, C#, Javascript, etc. for application development.
- Greater security by allowing access at the granularity of application-to-application communication. In a nut-shell, the AllJoyn framework provides an easy-to-implement object model and is a comprehensive open source tool for development of network applications which are proximity centric. We now briefly highlight the fundamental details of the AllJoyn network architecture and different functional modules in the AllJoyn framework.

2.1.1 The AllJoyn Network Architecture

Figure 2.2 depicts a sample AllJoyn network comprising of 5 AllJoyn devices (devices with AllJoyn framework installed), namely B1, B2, B3, B4 and B5. Fundamentally, the AllJoyn network architecture consists of two software components viz., AllJoyn Apps and Routers, or simply put, Apps and Routers. Based on the location and the connection between the Apps and the Routers, 3 common configurations exists which are elucidated below :

1. An App uses its own Router. Router in this case is called a Bundled Router as it is integrated with the App. Device titled 'B1 : Bundled Router' in Figure 2.2 falls in this category.
2. Multiple Apps on a certain device uses a single common Router available on that device. In this case a Router is called as a Standalone Router. Device B4 named 'Standalone Router' in Figure 2.2 with 2 Apps and a Router comes under this category where in different Apps utilize a common Router available on the device.
3. App in a certain device makes use of a Router located on another device. Devices B2 and B3 in Figure 2.2 representing some embedded devices corresponds to this category. These devices are typically resource constrained having low computational power, memory and network bandwidth. Device B5 falls into a category of hybrid configuration comprising of a standalone router and a bundled router.

2.1.2 Core Application Layer Components

1. **AllJoyn Bus :** One of the primary component of the AllJoyn framework is the AllJoyn bus. It is a free-way or a software bus available on a device as a module within the AllJoyn router. AllJoyn bus on different devices creates a fast, dedicated path in the distributed network over

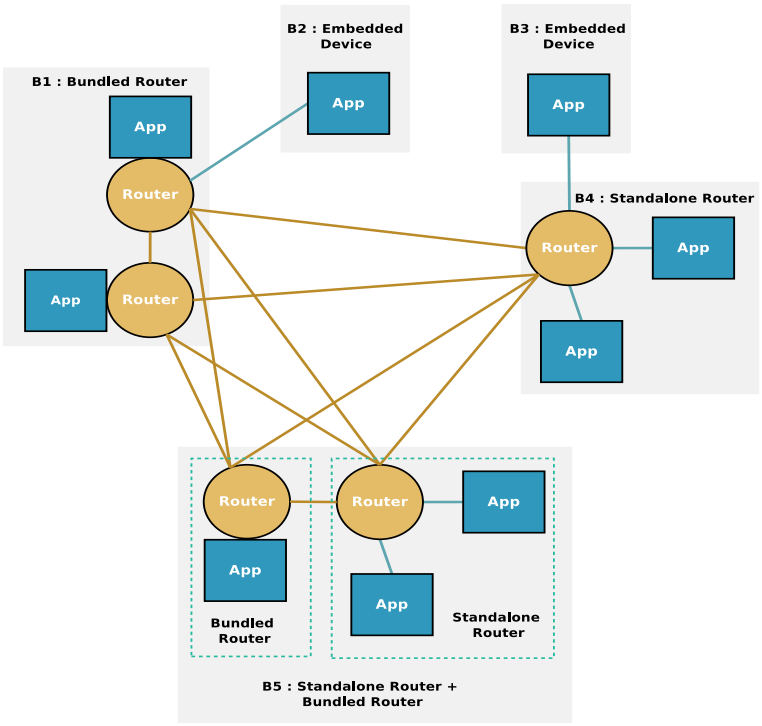


Figure 2.2: AllJoyn network topology and architecture illustration through devices (B1-B5).

which data and signalling messages can be transmitted. Different Apps running on a device connect to the software bus analogous to the way devices connect to the physical bus in an Ethernet network.

2. **AllJoyn Router :** AllJoyn Router is the core module in the framework stack. It handles all the complex tasks viz. service advertisement & discovery, transport abstraction, session handling, security, etc. It has dedicated sub modules for managing different transports, message exchange, AllJoyn software bus, session creation, management of ongoing communications, security mechanisms, marshalling / unmarshalling parameters for Remote Procedure Calls (RPC) etc.

3. **AllJoyn Services :** Each application comprises of one or more services which is a feature or a functionality offered to serve a particular use-case. (E.g: An application on an embedded device may have a service that offers readings of attached sensors to the nearby devices). Services are broadly classified as producer and consumer services.

4. **AllJoyn Bus Objects and Interfaces :** AllJoyn interfaces are interfaces similar to those in the programming languages like Java. AllJoyn interfaces specify methods, variables and signals declaration. AllJoyn bus objects provide implementation/definition for the methods and signals defined in the interfaces. Services offered by an application are realized through one or more bus objects. A single bus object can implement multiple interfaces and a single interface can be implemented by multiple bus objects.

5. **AllJoyn Library :** It is a module that links different applications with the AllJoyn Router. It comprises of multiple endpoints, one of which connects to the AllJoyn Router and the others connect to the applications running on the device.

2.2 Comparison with Other Technologies

Other similar popular technologies that provide solutions for ad hoc service discovery and means for communication in a dynamic proximal network are DLNA (Digital Living Network Alliance), UPnP (Universal Plug and Play) and Bonjour. Table 2.1 provides a comprehensive comparison performed by us of the AllJoyn framework with these technologies. It is evident from the table that AllJoyn framework has significant benefits over other technologies in multiple aspects. Below we present a brief comparison of the AllJoyn framework with other technologies listed in Table 2.1.

Features	**AllJoyn**	**DLNA**	**UPnP**	**Bonjour**
Open Source	Yes	No	Yes	No
Multiple Transports	Yes	No	No	No
Application Centric	Yes	No	No	No
Device Centric	No	Yes	Yes	Yes
Development framework	Yes	No	No	No
Media Streaming	Yes	Yes	No	No
Security	Yes	Yes	No	No
Discovery	Yes	Yes	Yes	Yes
IP Transport	Yes	Yes	Yes	Yes

Table 2.1: AllJoyn Comparison with other technologies

1. **Bonjour :** Bonjour, an Apple implementation for automatic discovery of devices and services offers similar zero-configuration networking as the AllJoyn framework. It offers mechanisms to discover, publish and resolve services within a local network. However Bonjour is operational only on devices with Apples OS X and iOS and comes as an inbuilt software in these devices. Apart from Apple devices it can be installed as an external software on devices running Microsoft Windows. It is an Apples proprietary software and offers support for limited platforms.

2. **UPnP :** UPnP is a standard that permits devices such as personal computers, printers, Wi-Fi access points and mobile devices to discover each other. In order to provide service/device advertisement, communication and data transfer to other devices, UPnP uses XML/HTTP/SOAP over IP. The advantage AllJoyn framework has over UPnP is that application layer data is transmitted as it is, inside TCP / UDP packets. It does not involve overheads of generation and processing of SOAP and HTTP messages, as in UPnP. Besides, UPnP protocol and device implementation lacks any inbuilt authentication mechanisms, which have to be incorporated in the devices through the use of extensions. AllJoyn framework on the other hand has inbuilt support for authentication and encryption. AllJoyn core library leverages PSK (Pre-Shared Key) algorithm, and ECDSA (Elliptic Curve Digital Signature Algorithm) for authentication and AES128 (Advanced Encryption Standard) algorithm for encryption.

3. **DLNA :** DLNA was conceptualised and developed specifically for sharing digital media amongst multimedia devices. It operates primarily on certified devices, products and each device supports different media format profiles. In comparison, AllJoyn can be deployed on any platform and is fully operational across a variety of devices.

However, AllJoyn framework is not the pioneer technology to enable zero configuration device / service discovery and facilitate communication between devices. Despite this, it has gained significant momentum in recent times as it is motivated more towards the vision of IoT (Internet of Things) or rather IoE (Internet of Everything). It fosters communication between applications running on heterogeneous set of devices irrespective of their capabilities (memory and processing power), form factors, OS and hardware stacks. Having compared the various technologies in great detail we now summarize generic advantages that AllJoyn framework has over its peer technologies :

- AllJoyn framework offers support for cloud based connectivity that allows a proximal network to be monitored remotely via Internet through a gateway. This sort of connectivity could also help the devices in the proximal network to utilize cloud based resources through the means of a gateway. This has potential applications in various IoT application scenarios such as smart homes wherein the devices could be monitored and controlled remotely through the cloud.

- AllJoyn framework has different variants for devices with different capabilities. In general, there exists two main variants : one for devices with functionally sufficient communication and processing capabilities (smartphones, tablets, computers, etc.) and other for resource constrained embedded devices.

- The software architecture of the AllJoyn framework is more modular and different modules are loosely coupled with each other. One major advantage of this such architecture is that a device need not have all the modules and may use the modules available to the devices in its vicinity. For *E.g.* embedded devices can have bare bones application and services running on it, leveraging the AllJoyn router from the devices in proximity for advertisement and discovery.

2.3 Conclusion

Based on the above comparison and inherent benefits of the framework, we employ the framework further for development of an MSNP application prototype. Then we present the details of the enhancements that can be incorporated into the framework which the framework currently lacks and realize multi-hop communication mechanism in the framework.

Chapter 3

Min-O-Mee : A Proximity Based Network Application Leveraging the AllJoyn framework

3.1 Introduction

We now present Min-O-Mee, a proximity based application designed to record the minutes-of-meeting (MoM / MOM). The objective of the application (App) is to share the MoM with the participants of a meeting receive in real-time. The AllJoyn framework is ideal to develop Min-O-Mee as all participants of a meeting will be in close proximity. Min-O-Mee is built on top of the AllJoyn framework abstractions, which provide the services such as session management, participant device discovery and attachment, and data transfer. In the following subsections, we describe the potential use case, basic App design, detailed functionalities, smart features and planned enhancements.

3.2 Use Case :

We illustrate the underlying motivation through a realistic example. Let us imagine going out with a group of friends on a weekend trip. All of us have experienced the inconvenience in keeping track of the expenses incurred by the group during the course of the trip. A solution at their disposal is to share a document over the Internet and constantly update it. However, availability of Internet is a constraint that renders this solution less effective as not all members in the group may have Internet access. An innovative idea would be to leverage the fact that they are travelling as a group, close to each other. Thus the closeness or spatial proximity of users and their devices, can be leveraged to create an opportunistic network of the devices which will facilitate real-time data exchange among them. Another benefit of employing a proximity based network is that latency inherent in data shared via Internet will not be experienced in a proximity based network. The above illustration is just a generic example of the application use. The application could be used as a means to set up a mobile social network and facilitate real-time exchange of minutes-of-meeting among the members in an academic or a business meeting based set up, leveraging their proximity. Beside these, it could be leveraged for interaction and communication between people travelling together in an air plane or between people in a crowded football stadium by forming a MSNP / PBN based on their opportunistic arrival in proximity to each other. Likewise many more potential application scenarios can be determined. The key benefit of an PBN / MSNP realization through AllJoyn framework is the support for diverse set of platforms and devices that lacks in the other existing proximity based

communication frameworks. Hence it could be utilized in a variety of scenarios beyond just PBN / MSNP where heterogeneous devices with different capabilities are anticipated to be the part of network.

3.3 Min-O-Mee Application Details

3.3.1 Basic Design

- There will be two types of participants in a meeting.
  - *Scribe :* A single participant who will create the MoM.
  - *Member(s) :* The remaining participants who would view and receive the MoM.
- Scribe initiates the Min-O-Mee session and advertises a unique *session name*.
- The Scribe prepares the MOM which could be shared with the Members in two ways.
  - *Real-Time Sharing :* A *text-pad* on each Member device is instantaneously updated as the Scribe prepares the MoM. Member devices can view the text-pad in the read-only mode.
  - *Deferred Sharing :* The Scribe sends a copy of the MoM as a text / PDF file to all the Member devices immediately after the meeting is over.

3.3.2 Functional Details

We elaborate upon the functional details, by dividing the App functions into four operational categories *viz.*, Peer Advertisement And Discovery, App Dashboard, Text Editor and Data Sharing.

1. **Peer Advertisement And Discovery :** All participants advertise a unique device ID which comprises of a *social* component and an *identification* component. The device *name* is the social component which is taken as input from the user through the user interface. It is the device advertisement mechanism of Min-O-Mee and the device name should ideally be the Member's name as other Members would easily recognize it. But human names are not unique and are unsuitable to be device IDs. We remedy this problem by concatenating the device name with the subscriber identification module (SIM) number of that particular device. The member only enters the device name and the concatenation happens in the background without her knowledge. In absence of the SIM number a large randomly generated number is used. The purpose of this identification component is to transform the device name into a unique device ID. While the Members only see the social device names, the App running on every device receives the unique device IDs of other Members.

2. **Services and Application Architecture :** In the context of the AllJoyn terminology each AllJoyn application comprises of multiple services. These services could be consumer services or producer services. Consumer services consume services provided by the other applications whereas provider services provides services to the other applications in the proximity. For E.g. consider a motion sensor that determines whether there was movement of people or objects within its range. The motion sensor could be hosting a provider service that intimates the near

by devices whenever a motion was detected. A light bulb in the vicinity having an AllJoyn application with some consumer service would listen to the data provided by the producer service (in this case the motion sensor). Based on the received data the light bulb can turn on/off. Thus here motion sensor acts as a service provider and the light bulb acts as service consumer. An AllJoyn application may have consumer service(s) or producer service(s) or a combination of both.

Min-O-Mee application that we develop has both a producer and a consumer service embedded within it. Figure 3.1 illustrates where the application resides in the protocol stack and modules within the application package. Producer service within the application lets a device advertises about its capabilities through advertisement mechanism of hosting a Min-O-Mee session. Consumer service on the other hand lets a device join a Min-O-Mee session offered by an application on some other device. Advertisement mechanisms lets each device advertise about its producer service with the following details : the *CONTACT_PORT* on which the service is running, name of the service, interface name of the interface service implements and the property details in those. Further details about the IP address at which the service is running is added by the AllJoyn router. Discovery mechanism lets a device discover about the producer services in the vicinity capable of hosting a Min-O-Mee session.

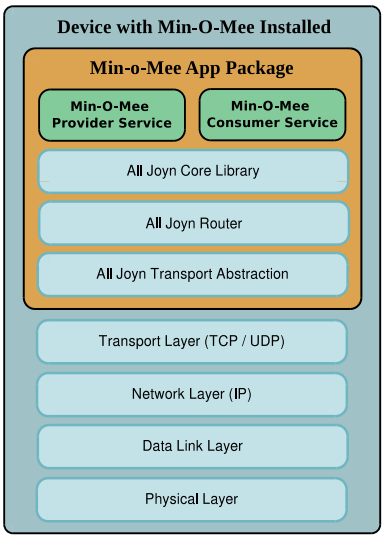


Figure 3.1: AllJoyn Application Architecture

AllJoyn router in the Figure 3.1 is the module that performs the functions of service advertisement and discovery of remote services. AllJoyn library serves as a medium for the application containing producer and consumer services to interact with the AllJoyn router. Below the router resides the AllJoyn transport module which is an abstraction that hides the details of the underlying transport details, Hardware and the OS details from the app users. In conclusion the entire Min-O-Mee application package resides at the application layer of the standard International Standards Organization Open Systems Interconnection (ISO / OSI) model. Below the application resides the traditional Transport, Network, Data Link and Physical Layer.

At a time a device could either act as a scribe and provide the services to other or could act as a participant and consume the services provided by some other scribe. For the sake of simplicity

(a) 'My MoMs' List (b) 'Shared MoMs' List

Figure 3.2: The Min-O-Mee Dashboard

and generality, we assume that members in a meeting or a discussion could decide about who amongst them would act as a scribe and the participants. Accordingly our application provides each user with the list of names of the devices in the vicinity including its own device. Following two cases arise in this context.

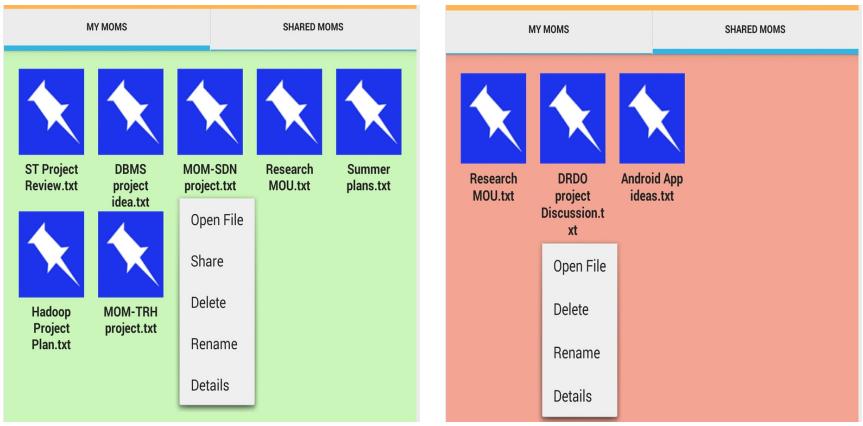
(a) If the user chooses its own name from the list of users/peers in the vicinity then this is considered as in the user wants to act as a scribe. In this case it provides services to the other users, joins the session hosted by itself and let other users join the session.

(b) If the user chooses some other user name from the peer list then it is considered that the user wants to act as a participant in the meeting/discussion. Here the user joins the session hosted by the selected peer.

There can be multiple scribe(s) at a particular point of time but a participant could join session hosted by only single scribe at a time. If the participant opts to join the session hosted by some other scribe, he/she would be automatically disconnected from the session it was previously the part of. However in general there would only be a single scribe decided by the mutual agreement of the users in the meeting

3. **App Dashboard :** The App Dashboard is divided into two sections, *viz.*, 'My MoMs' and 'Shared MoMs'. 'My MoMs' is the list of MoMs which are created by that particular member *i.e.*, she was the Scribe in the meetings whose MoMs are listed. 'Shared MoMs' is the list of MoMs that were shared with the particular member when some other member was the Scribe. When Min-O-Mee is installed in any device two folders are created in the directory where the App is located, one folder for each of the two lists.

   A screenshot of the Min-O-Mee Dashboards of two different devices is exhibited in Figure 3.2. In Figure 3.2 (a), the 'My MoMs' list with a light green background is captured from the 'My MoMs' folder in the Min-O-Mee App directory in the Lenovo Yoga Tablet. Similarly in



(a) 'My MoMs' File Options (b) 'Shared MoMs' File Options

Figure 3.3: Difference In File Options In The Two Lists

Figure 3.2 (b), the 'Shared MoMs' list which has a light red background, is taken from the 'Shared MoMs' folder in the Min-O-Mee App directory in the Samsung Galaxy DUOS I9082. The color coding offers a better user experience. The choice of the colors also highlights the fact that the files in 'My MoMs' list can be edited by the user but she can not modify the content of the files in the 'Shared MoMs' list. However, files in both the lists can be read, renamed and deleted by the respective users. The various file operations that are permissible in the two lists are illustrated in Figure 3.3.

4. **Text Editor :** The text editor is a vital component of Min-O-Mee as the Scribe needs to take the MoM during the course of a meeting. The editor is only available to the Scribe and not to any other Member. However, a Text-pad is available to the Members so that they can either view the MoM being shared in real-time or the MoM file shared by the Scribe after compilation. In either case, a Member is not allowed to edit the MoM file. It is evident from the Figure 3.2 that the file 'ST Project Review.txt' is a part of the 'My MoMs' list of Lenovo Tablet, while the file 'Android App Ideas.txt' belongs to the 'Shared MoMs' list of the Samsung Galaxy I9082. When a user tries to edit the file 'Android App Ideas.txt', it displays a *toast* on the screen with the message 'Only Scribe Can Edit'. In addition, the editor performs a periodic auto-save in the background and when the user hits the back button it again saves all the contents to the 'My MoMs' folder in the application installation directory. Thus, the editor avoids any loss of important MoM data which could happen during the course of compilation of MoM by the Scribe, due to a sudden unexpected behaviour by the App or a device malfunction.

5. **Data Sharing :** The objective of Min-O-Mee is to share the MoM with the Members in real-time. The work is still in progress, and the App is currently in the developmental stage of deferred sharing. This mechanism entails that the Scribe should send a text / PDF file of the compiled MoM with all the Members at the end of the meeting. For example, in Figure 3.2 the file 'Research MOU.txt' lies both, in the My MoMs' list of Lenovo Tablet and in the 'Shared MoMs' list of the Samsung Galaxy smartphone. This implies that the user of the Lenovo tablet was the scribe for this MoM and it shared the file with the user of Samsung smartphone who was a Member in the meeting.

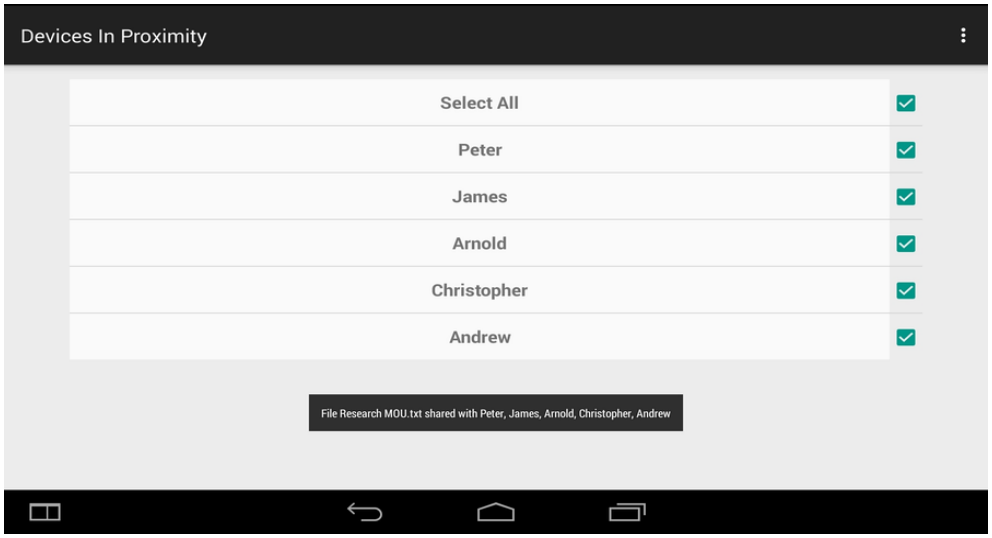


Figure 3.4: Scribe Sharing A MoM

Let us consider Figure 3.4, where the Scribe *BRUCE* operating the Lenovo tablet, creates a MoM named 'Research MOU.txt'. To share the MoM with the rest of the Members, Min-O-Mee creates a list of devices in close proximity which is obtained through the background discovery mechanisms of the AllJoyn framework. The App offers BRUCE a choice to either share the MoM with a few selected Members or the entire Member list. A share button pops up which upon being pressed shares the MoM with the Members selected by BRUCE. The App generates a *toast* notification stating the social names of the Members with whom the MoM has been shared.

A Member which was selected by BRUCE to receive the MoM gets a notification message from Min-O-Mee about the incoming MoM file. The Member may choose to accept or reject the file. However, a Member which receives a MoM is not permitted by Min-O-Mee to share it with anyone else which can be inferred from the absence of *share* option in Figure 3.3 (b).

One important aspect of a shared MoM is the file metadata. Min-O-Mee keeps a record of file attributes, most important among which is *OWNED BY* which declares the owner of the MoM file. The owner is always the Scribe who created the MoM, regardless of its presence in the Scribe's device under the 'My MoMs' list, or in a Member device listed under 'Shared MoMs'. Thus the *OWNED BY* attribute always remains constant. However, the attribute *SHARED WITH* of a MoM file is modified by Min-O-Mee after the Scribe shares it with other Members. This can be noticed in Figure 3.5, which demonstrates that the *SHARED WITH* attribute of a MoM file in the Scribe device is updated with the names of the Members after the file is shared with them. The underlying motivation of this exercise is to keep a track of the Members with whom the MoM is shared.

3.4 Application Deployment And Testing

To test the performance of Min-O-Mee, the '.apk' file generated in the Android Studio IDE for the Min-O-Mee App was installed in 10 android devices. The minimum android version that is supported is 4.0 and the highest compatible version is 5.0. The android devices used in the testing are following

(a) File Details Before Sharing (b) File Details After Sharing

Figure 3.5: MoM File Attributes In Min-O-Mee

: Lenovo Yoga Tablet, Samsung Galaxy DUOS I9082, Google Nexus 6. The Min-O-Mee application was successfully installed in all the devices and each device could discover the others by making use of the AllJoyn discovery mechanism. A Wi-Fi hotspot is created on one of the devices and the others are connected to it. We are able to create and share MoMs by assigning the role of Scribe to one of the devices and keeping the remaining devices as Members. The screenshots of the MoMs created and shared are presented in the previous section.

3.5 Conclusion

This chapter presents Min-O-Mee application which is is a proximity based application or an MSNP application intended to facilitate communication between users in proximity of each other. The application is developed leveraging the AllJoyn framework. The application development provides a hands on experience and acquaintance with the implementation aspects of developing applications using the AllJoyn framework.

Chapter 4

Open Challenges / Extensions to the framework

AllJoyn framework serves the basic purpose of ad hoc discovery of applications running on devices in mutual proximity and establishing communication between them. However the framework is still in its nascent stages of development. Below we elucidate the proposed extensions to the framework, incorporation of new modules and their practical importance and relevance in IoT and other applications.

4.1 Current Working Model

The current working model for devices to communicate with other devices in proximity using the AllJoyn framework requires that Wi-Fi hot spot available on either of the devices be turned on. It is then that the devices under same hot spot would be able to discover and advertise services to each other and communicate further. An alternative set up where in the devices can discover and communicate with other proximal devices is all the devices connect to a same access point. These access points are the ones that exists in various public spots (parks, cafes), campuses, business parks, homes, offices, etc. The devices then connect to those access point and all those devices attached to the same access point could then discover consumer services, advertise their services and communicate with each other. Clearly such set up requires infrastructure (in case of access points) and requires manual intervention in terms of connecting to a hot spot or an access point. It does not permit dynamic creation of a proximity based network or mobile social network in true sense and does not facilitate direct communication between devices when they come in proximity. It requires them to be connected to same access point or hot spot to let them communicate with each other. Next we present two enhancements in the framework that could be of significant in various practical applications of the framework and lead to potential new applications where in the framework could be leveraged.

4.2 Support for short range technologies such as Bluetooth and Wi-Fi Direct

AllJoyn framework currently has support for technologies such as Ethernet, Wi-Fi and Powerline. AllJoyn web references and documentation though indicates that it has support for Bluetooth as a transport, however in its actual implementation [25] it lacks such support. Implementation source code for AllJoyn has options to incorporate Bluetooth support in different source files and has no

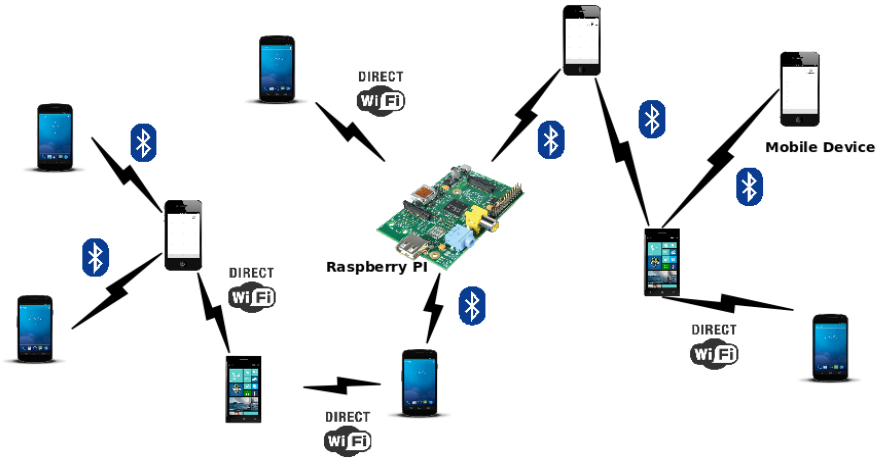


Figure 4.1: Dynamic PBN set up through the support of Bluetooth and Wi-Fi Direct Technology

source code as of now to realize it. However with the inbuilt support for short range technologies such as Bluetooth and Wi-Fi Direct available in devices these days, extension of the framework to incorporate these technologies is of prime importance. These extensions would be useful and relevant in a wide variety of applications such as disaster networks, vehicular networks, multi-user games, proximity based social networks etc. Besides, support for these technologies would render the framework useful in a wide variety of IoT applications such as smart home networks, intelligent transport systems, etc. Objects and devices in an IoT network would be able to discover other such devices in a dynamic manner using the above technologies in a seamless fashion. BLE (Bluetooth Low Energy) as a communication technology would be of high relevance for energy constrained embedded devices in certain IoT applications. Implementation support for technologies such as Bluetooth and Wi-Fi Direct in the framework would foster realization of dynamic PBNs and MSNPs in true sense. Support for technologies such as Bluetooth and WiFi-Direct would mitigate the problems in current working model and lead to creation of dynamic and ad hoc PBNs / MSNPs in real sense. Figure 4.1 represents a dynamic, infrastructure less MSNP that could be set up through if support for technologies such as Bluetooth and Wi-Fi Direct were available. Bluetooth and Wi-Fi Direct interfaces on these devices could communicate with similar interfaces available with proximal devices in an ad hoc and random fashion. Besides this, support for Bluetooth and Wi-Fi Direct technology would also help to extend the network outreach from mere Wi-Fi only network as illustrated in Figure 4.2. The circle in the figure represents the Wi-Fi range of the device whose Wi-Fi hot spot is turned on (Raspberry PI in the figure). Other devices are connected to the Wi-Fi hot spot via Raspberry PI. The devices within the range could then communicate with each other. However with the Bluetooth support the network outreach could be extended and the device could perform service advertisement, discovery and communication using the Bluetooth interface as well, as illustrated in Figure 4.2. A similar analogy would apply to Wi-Fi Direct technology too. Realization of these extensions would require addition of modules for Bluetooth transport and Wi-Fi Direct transport in the AllJoyn router. Router abstraction layer could then determine how to propagate messages further using TCP, UDP, Bluetooth or Wi-Fi Direct, based on the user requirements and the need at the application layer above the router abstraction layer.

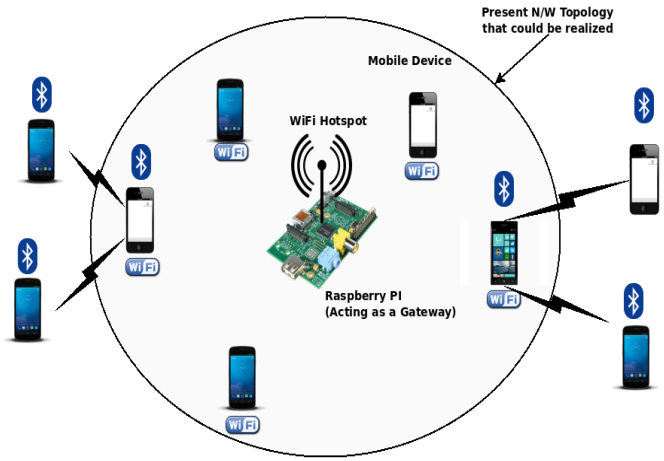
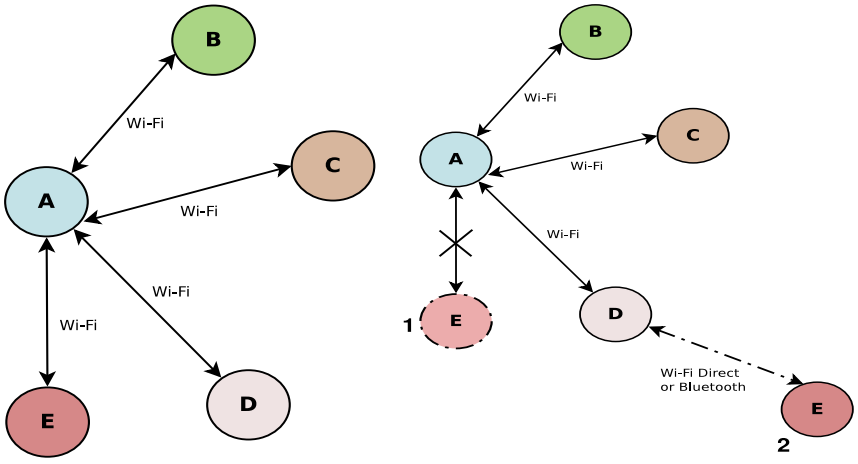


Figure 4.2: Dynamic PBN set up through the support of Bluetooth and Wi-Fi Direct Technology

4.3 Multi-hop Communication support

AllJoyn performs some crucial functions viz., application discovery and advertisement, remote bus attachments, session management and data transfer between devices. However current implementation only supports communication between devices which are one hop away from each other. This is due to inherent drawback in the existing working model of the framework. The existing model as explained previously requires devices to be connected to the same access point or a hot spot. Communication between any two devices occurs via the access point or hot spot only. Below we present the problems that could arise in the existing working model:

- Figure 4.3 (a) represents a topology between 5 devices A B C D and E at time t = t1. Device A has been set up as the Wi-Fi hotspot and devices B, C, D and E are connected to the hotspot on A. All the devices which are in the range of Wi-Fi hotspot are able to connect to hotspot and discover services on other devices connected to the same hotspot. Consider that there are ongoing communication sessions between device pairs B and E, B and D. Due to the mobility of devices, topological changes might arise. This may cause some of the devices to fall outside the range of the device acting as Wi-Fi hotspot. For the sake of generality, let us consider that device E is mobile and has moved out of the coverage area of the Wi-Fi hotspot on A. Figure 4.3 (b) represents the new topology and the subsequent position of the device E. The ongoing communication between devices E and B would be affected due to this topological change. This would result in the termination of session between the two devices and loss of all ongoing packet transmissions. This is due to the operation of Wi-Fi on devices in infrastructure mode and since E moves out of the Wi-Fi range of A it can no longer further communicate with other devices. This is where multi-hop communication mechanism can play a significant role to resolve the problems caused to mobility of devices. However it requires Wi-Fi on devices to be operational in ad hoc mode as in infrastructure modes every communication would go through the access point or hot spot and multi-hop communication cannot be realized. In order to provision a multi-hop communication mechanism it is required that Wi-Fi interfaces be operational in ad hoc mode. Operation in ad hoc mode would also enable realization of proximity based networks in true sense as devices can communicate with those in its range in an ad hoc manner without connecting to access point or a hot spot.



(a) Topology at time t = t1 (b) Topology at time t = t2

Figure 4.3: PBN topology between 5 devices A-E at different instants of time

Multi-hop communication could mitigate the problem caused due to mobility of E by routing the data in the ongoing communication between B and E through D. However the framework currently lacks support for multi-hop service advertisement and discovery. Presence of multi-hop communication mechanism would enable advertisement and discovery of services between devices which are multiple hops away from each other. Besides this it would also enable communication between devices when they are not in their direct communication range either due to mobility or there are positions are such that they fall outside communication range of each other.

Below we elucidate potential applications and scenarios that may benefit from the realization of multi-hop communication in the framework:

- **Vehicular Communication System :** A VCS [26, 27] is a dynamic and distributed network, which consists of mobile vehicles and stationary roadside units as the communicating nodes. The primary objective of a VCS is to deliver an Intelligent Transport System (ITS) [28] that has a plethora of smart features with respect to road safety, traffic management, law enforcement, improved driver assistance etc. AllJoyn framework can be employed in VCS by design and development of AllJoyn applications according to the requirements of VCS. As framework has inherent characteristics of operational across heterogeneous types of devices and inter connect them, it can be used in VCS too where devices on which the AllJoyn applications would be running are vehicles and stationary roadside units. However the current working model of attaching to a Wi-Fi hot spot or access point may not be feasible in VCS. It requires the operation of Wi-Fi interfaces attached to vehicles in VCS in ad hoc mode. However that alone would not suffice due to mobility of vehicles. An enhanced AllJoyn framework with multi-hop routing mechanism can create an extended PBN (ePBN) between mobile vehicles, and relaying of critical traffic related information from a roadside post to auto mobiles ahead will be somewhat simplified. Of course, a VCS which leverages this enhanced AllJoyn routing mechanism will also not be without some challenges e.g., hand off's of vehicles between stationary roadside posts, path discovery to a vehicle on the fringe of the ePBN in high-speed mobile environment etc. Nevertheless, the framework would make successful inroads in solving these challenges.

Disaster Networks: Disaster networks [29] are a very important application of PBNs or MSNs, especially in the light of recent natural calamities such as the Uttrakhand floods (2013), Flash floods in northern India (2013) and Nepal earthquake (2015), in which tens of thousands of lives were lost. Available wired and wireless communication infrastructure is completely disrupted in such scenarios. Disaster mitigation and rescue efforts are extremely time sensitive [30], as loss of lives increases exponentially with delays in providing relief to the victims. PBNs would aid communication in emergency situations when the existing dedicated communication infrastructure is completely disrupted or unavailable. Applications leveraging the AllJoyn framework can be developed that could enable proximity based communication in disaster situations to let people convey their situation to each other utilizing the gadgets at their disposal. Multi-hop communication mechanism integrated into the framework would be of notable benefit in this scenario as the affected civilians could be in different conditions and terrains in the aftermath of disaster. They may not necessarily fall into direct communication range of each other. However through multi-hop communication they can convey their situations to each other or also to the rescue team and seek for help.

4.4 Conclusion

This chapter highlights the support for technologies and mechanisms the AllJoyn framework currently lacks. Multi-hop communication mechanism is identified as a significant and crucial enhancement to the framework due to its relevance in various scenarios. The rest of the thesis focusses on realization of multi-hop communication mechanism in the AllJoyn framework. Two popular routing algorithms in the domain of ad hoc networking have been identified : OLSR and B.A.T.M.A.N. The next chapter gives an comparison overview of the two based on which B.A.T.M.A.N is used as the underlying routing protocol for multi-hop communication. Further chapters describe about the integration mechanism of B.A.T.M.A.N and AllJoyn framework to realize multi-hop communication in AllJoyn framework.

Chapter 5

Better Approach Towards Mobile Ad hoc Networking (B.A.T.M.A.N)

5.1 Introduction

B.A.T.M.A.N is a routing protocol developed for multi-hop communication in ad-hoc networks developed for mesh networks but could be utilized in a variety of other scenarios and networks [31]. B.A.T.M.A.N is under constant development by the Freifunk community [32]. One of the key distinct feature of the B.A.T.M.A.N routing protocol is the decentralization of the network and the topology information. Nodes in the B.A.T.M.A.N network do not maintain the entire topology information and compute the shortest routes to the other nodes in the network. Rather each node just maintains the potential next hop towards other nodes in the network. This chapter is an overview about the B.A.T.M.A.N routing. First we present the comparison analysis of B.A.T.M.A.N routing with the Optimized Link State Routing (OLSR) protocol which is a widely used routing protocol for dynamic and ad hoc networks. Comparison is presented in terms of their operation and their performance evaluation on real-time test-bed by the authors in [33]. From the comparison it is concluded that B.A.T.M.A.N routing is more efficient in terms of CPU overhead, memory requirements, routing packet length, routing flaps, throughput specifically when the number of nodes in the network increases. Hence we highlight the basic operation and the functioning of the B.A.T.M.A.N routing protocol, content of the messages in the B.A.T.M.A.N routing according to the RFC [34], mechanism for computation of the best next hop, flooding mechanism and data structures maintained at the nodes during the operation of the routing protocol. The objective is to have a light weight routing protocol in order to realize the multi-hop communication in the AllJoyn framework. AllJoyn framework primarily is and would be used in varying scenarios and applications that could be of static topologies, slightly dynamic or highly dynamic topologies. The choice of the protocol would have then impact on the multi-hop communication across AllJoyn applications running across different devices.

5.2 Comparison Analysis of OLSR and B.A.T.M.A.N routing protocol

OLSR Here we present a brief overview of B.A.T.M.A.N and OLSR. We do not delve deep into their functioning as plethora of sources exists in the literature that describes these two routing algorithms. However we present an in general comparison of the two in terms of their operation. Based on

the performance evaluation of the two on real-time test-beds by [33] and [35], B.A.T.M.A.N is concluded to be more efficient in comparison to OLSR We thereafter present the operational details of B.A.T.M.A.N routing protocol only as we use it as the underlying routing protocol for conceiving multi-hop communication mechanism in the AllJoyn framework.

Although there exists many routing algorithms in the domain of ad hoc, dynamic and mesh networks, we primarily choose two algorithms B.A.T.M.A.N and OLSR. These algorithms are widely popular, thoroughly studied by the networking community and extensive research on their performance in real-time test-bed on varying types of topologies (static / dynamic), network load etc. have been carried [33], [35], [36] and their potential to be utilised in real applications have been explored too [37], [38].

OLSR is a proactive routing protocol designed for ad hoc networks. OLSR uses HELLO messages and Topology Control (TC) messages for its operation. Each node periodically broadcast HELLO messages comprising the information of its neighbours and the link status to them. This are not rebroadcast by all the neighbour nodes but only by the neighbours chosen as Multi point relays (MPRs). MPRs of a node X is a set of nodes among its neighbours such that each two hop neighbour of X is a one hop neighbour of atleast one multi point relay of X. All nodes in the network select and maintain their own MPRs. A node describes its selection of MPR by a certain node through TC messages. TC messages enable diffusion of topology information and helps in performing route calculations to determine route towards other nodes. Selection of MPRs in OLSR limits the flooding overhead which is an optimization of the classical link state routing algorithm to meet the requirement of ad hoc networks. Figure 5.1 illustrate the difference in flooding mechanism in traditional Link State Routing (LSR) and OLSR. B.A.T.M.A.N on the other hand uses Originator messages for its operation. These are broadcast by a node in periodic intervals which are rebroadcast by its neighbours and further by its neighbours and so on through a controlled flooding mechanism. Nodes however does not maintain the entire route to a destination.

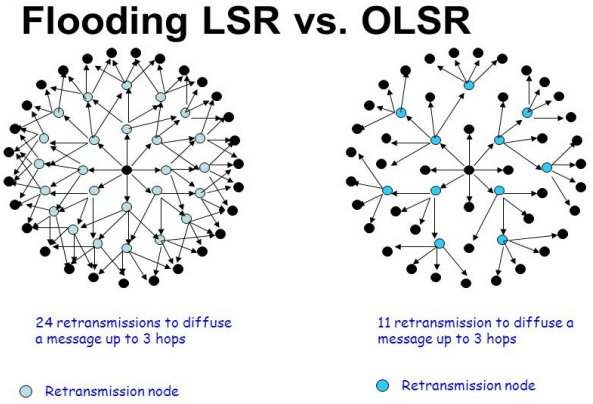


Figure 5.1: Flooding in LSR v/s OLSR.

Performance Comparison on real-time test-bed. Below we present some of the key observations obtained by testing B.A.T.M.A.N and OLSR routing on a real-time test-bed by authors in [33]. The experiments were conducted on 7 * 7 grid of WiFi nodes operation at channel 6 and Wi-Fi standard, 802.11b. Further details of experiments set up and the node configurations can be obtained from here [33].

- **Routing Overhead :** Routing messages exchanged in the protocol for the establishment of routers determine the routing overhead. For B.A.T.M.A.N routing messages are Originator messages (OGM) and for OLSR these are HELLO messages and Topology Control (TC) messages. Figure 5.2 shows the increase in routing packet length for 2 protocols as the number of nodes increases. Routing overhead is determined in terms of bytes/sec, by multiplying the length of packets with the number of packets per second leaving a node. For B.A.T.M.A.N this overhead was of 675 bytes /sec and 6375 bytes /sec for the OLSR for the full 49 node grid in the experiment. Thus clearly routing overhead is 10 times in the OLSR to that in B.A.T.M.A.N for the 49 node network.In OLSR, entire route topology are send in Topology Control messages which leads to a linear increase in the packet length with increase in number of nodes.

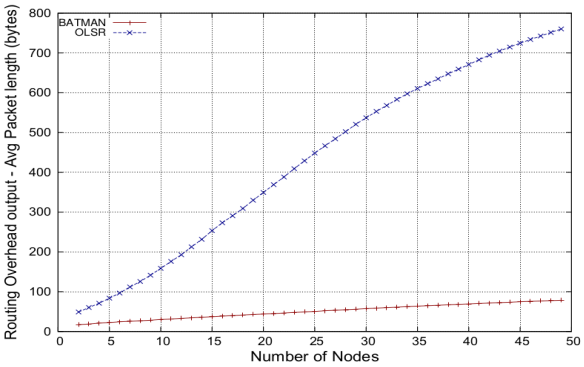


Figure 5.2: Average routing packet length with increase in number of nodes.

- **Throughput measurements** Throughput, delay, packet loss were carried for the two routing protocols on the 7 * 7 grid. For throughput measurements, *iPerf* was used with 8KB read and write buffer size using TCP for 10 seconds. The network was put under load by generating flows across all 2352 (49 * 48) pairs. These measurements were also done with varying topologies obtained by increasing the inter node distance in the experimental set up. Figure 5.3 represents the throughput observed for the two routing protocols with varying distance between the nodes. From the Figure 5.3 it can be observed that B.A.T.M.A.N has approximately 15% better throughput than OLSR at all inter node distances and the network being put under load. OLSR was exhibited to have a 1.68% of packet loss while 2.63% packet loss was observed in the case of B.A.T.M.A.N. Average delay however was less in B.A.T.M.A.N of 7.61 ms as compared to 17.39 ms delay in OLSR. Besides these route flapping test were performed for the two algorithms where route flapping denotes the number of route changes over a duration of time. This was determined using the ping packets which had an option to record the route taken by the packet. Ping packets were send for a duration of 10 secs across the horizontal, vertical diagonal and the 2 cross diagonals (from top-left to bottom-right and from bottom-left to top-right) of the 7 * 7 grid and also between every 2352 (49 * 48) pair. OLSR was observed to have a high degree of route flapping of 12 seconds per node where as B.A.T.M.A.N had route flapping of 25 seconds per node.

- **CPU usage :** CPU usage of the routing algorithms is directly related to the number of packets

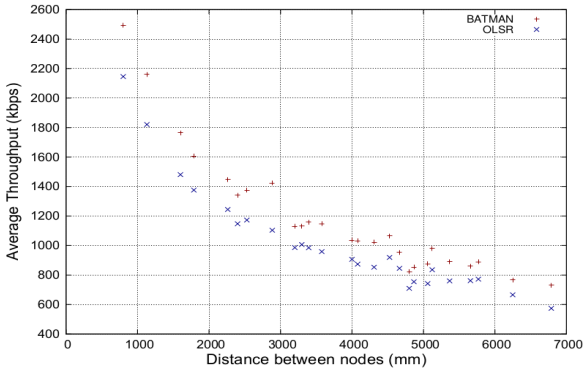
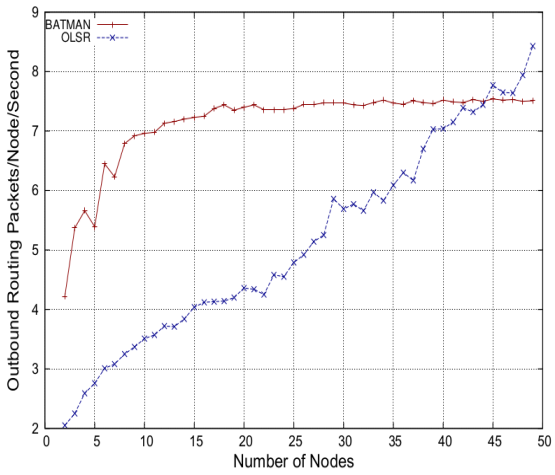
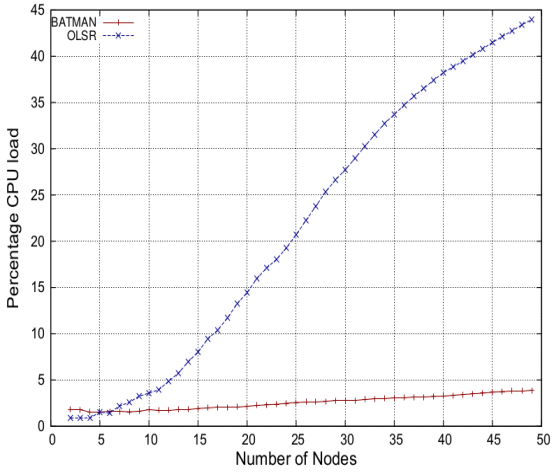


Figure 5.3: Throughput versus distance plot for the 7 * 7 wireless grid.

routing algorithm process. Along with this it is also dependent on the algorithm complexity to compute the optimal routes to build routing tables. B.A.T.M.A.N was observed to have greater number of outbound routing packets below the network size of 45 nodes as shown in Figure 5.4 (a). However B.A.T.M.A.N had far less CPU consumption in comparison to OLSR for network size greater than 6, which is illustrated in Figure 5.4 (b). When the number of nodes are increased to 49 (7 * 7 grid), OLSR had CPU consumption of 44%, significantly higher than the 4% CPU consumption by B.A.T.M.A.N. Higher CPU consumption in OLSR is due to increase in length of packets with increase in number of nodes and hence more CPU consumption to process the embedded topology information in the packets.



(a) Outbound routing packets per node per sec with increase in number of nodes.



(b) Percentage CPU usage with increase in number of nodes.

Figure 5.4: Outbound routing packets and Percentage CPU usage

- **Memory Consumption :** Memory consumption is associated with the memory required to store the routing tables. The nodes in the experimental set up of 7 * 7 grid had 128 MB of memory (RAM). Memory consumption of the two routing algorithms, B.A.T.M.A.N and OLSR with increase in number of nodes is shown in Figure 5.5. OLSR memory requirements increases with number of nodes as it needs to store the entire touring tables where as B.A.T.M.A.N

maintains just the next hop neighbours to reach other destination nodes. From the Figure 5.5, it can be observed that OLSR memory requirements increase sharply beyond 30 nodes.

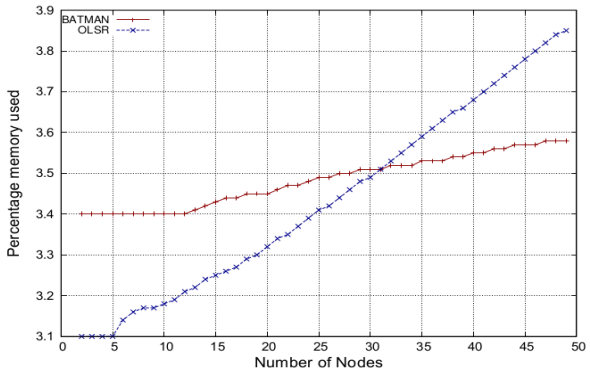


Figure 5.5: Percentage memory consumption with increase in number of nodes.

Choice of routing algorithm for Multi-hop Realization in AllJoyn framework : Multi-hop communication in AllJoyn framework could be provisioned through the use of either of B.A.T.M.A.N or OLSR routing. However the choice of routing protocol would have direct impact on the performance of AllJoyn applications in the domains where AllJoyn framework is leveraged. This domains include and are not limited to, Internet of Things, proximity based network, ad hoc networks, vehicular networks, disaster networks, gaming, proximity based social networking, etc. As observed from the practical evaluation of the two routing protocols by [33], B.A.T.M.A.N is observed to be better in comparison to OLSR, specifically with large number of nodes on parameters such as throughput, delay, route flapping, memory consumption, CPU usage and routing overheads. Based on that we consider B.A.T.M.A.N as more proficient routing algorithm for provisioning of multi-hop communication in AllJoyn framework.

Besides these AllJoyn being a platform agnostic framework, in its real-time uses, AllJoyn applications would be operational across different types of devices (embedded devices, laptops, mobile phones, tablets, etc.) with varying configurations (CPU, memory, hardware and software specifications). Usage of OLSR as the routing protocol for multi-hop realization would result in heavy energy consumption, CPU and memory usage and this would be a bottleneck when AllJoyn applications are running on low powered embedded devices. Even if AllJoyn applications are running on mobile phones, tablets, laptops, etc. and OLSR is used for multi-hop realization, its higher CPU, and memory requirements with larger number of nodes would have impact on the battery consumption which is a crucial resource for these devices. On the other hand B.A.T.M.A.N is observed to have less overheads on the various mentioned parameters, thus it would prove to be more generic and useful for multi-hop realization. Whatsoever be the devices (embedded devices , laptops, mobile phones, etc.) on which AllJoyn applications be running, using it for multi-hop realization would not impact much on CPU, memory due to its less resource utilization. We therefore pick up the B.A.T.M.A.N routing protocol as the underlying routing protocol for realization of multi-hop communication between AllJoyn applications running across devices multi-hop to each other.

5.3 B.A.T.M.A.N routing

5.3.1 Description

Nodes in the B.A.T.M.A.N protocol does not maintain the entire network topology as maintained in various routing algorithms for static networks (Distance Vector Routing, Link State routing etc.) and MANETs. It falls under the category of proactive routing protocols which before hand computes the route towards a particular destination. In contrast to this exists, reactive routing protocols which compute the route towards a particular destination when some data has to be forwarded to it. The fundamental concept behind the B.A.T.M.A.N operation is that nodes determines a best next hop neighbour among its single-hop neighbour(s) for multi-hop destination. Each node maintains the presence of all other nodes in the network which can be either single-hop or multiple hop(s) away. However the nodes do not compute the entire route (shortest route) towards other nodes in the network, neither are they aware of the network topology. Nodes store the best next hop towards other destination nodes in the network which may change over time with changing topology. Whenever a node has to send data to a certain destination node it would forward the data to the available best next hop for that particular destination node. The best next hop neighbour node would repeat the process until the data is delivered to the destination node. Maintenance of just the potential next hop towards other nodes makes the protocol light weight in comparison to those protocols which store the entire network topology, determine the shortest path and then the next hop in that shortest path to whom data is forwarded. Also storage of entire network topology is of no relevance when the network is changing dynamically as in the case of ad hoc networks, opportunistic networks, etc. The topology of the network may change by the time route towards a particular destination is determined. B.A.T.M.A.N approach thus proves useful in such scenarios and is adaptive towards constantly changing topologies. Decentralization of the network information and the route computation are the fundamental characteristics of the B.A.T.M.A.N routing protocol.

5.3.2 Operational Overview

Each node in the B.A.T.M.A.N network is termed as originator node and the interfaces on the originator node on which B.A.T.M.A.N is enabled are termed as originators. Every originator periodically emits an originator message (OGM) to inform originators on other nodes in the network about its presence. This periodic interval is called as Originator Interval (Orig_Int). The originator messages are broadcast to the single-hop neighbours which further rebroadcast to their single-hop neighbours and so on. Eventually every originator at different originator node comes to know about the presence of other originators through the originator messages. Originator messages serve as a means for a node to let other originators know about its presence as well determine the presence of other originators through originator messages from them. There can be multiple interfaces on which the B.A.T.M.A.N routing is enabled and hence multiple originators on an originator node. The originator receiving the originator messages from the originators on the same originator node is not aware of the fact that those originators belong to the same node. The originator message carry the identity of the originator who generated the OGM (which according to the RFC [34] is the IP address of the interface). This identity is preserved during the rebroadcast to let originators running on originator nodes to know of the presence of the other originators. Rebroadcasting of the originator messages in B.A.T.M.A.N is not blind flooding where the received data by a node is

rebroadcast as it is without any checks on the received content or the path / link through which the content was received. Rather the rebroadcast mechanism is a controlled flooding mechanism depending upon the link quality, fields in the originator message and certain other filters applied to the received originator message. The rebroadcasting mechanism is explained in the next subsection.

Determination of Next Best Hop Neighbour : The number of originator messages from a certain originator on an originator node received via each of the single-hop neighbours determines the quality of single-hop / multi-hop route towards that originator. Originator message emitted from an originator might reach originators on other nodes through multiple paths depending upon the network topology. The originator on a node receiving an originator message maintains a list of neighbours through which it receives originator messages from a certain originator. The count of originator messages received via each of the neighbours for that particular originator is logged. The neighbour through which the maximum number of originator messages are received within a sliding window qualify as the potential best next hop towards that particular originator. Sequence numbers embedded in originator messages helps in identification of new and the duplicate originator messages. It also ensures that every originator message gets counted only once in determination of best next hop neighbour towards a destination originator. An originator picks up a random sequence number to start with and then the sequence number are incremented by one with every transmitted OGM. In actual implementations, besides the count of originator messages, transmission and reception quality of the link through which the message is received is also taken into consideration for computation of next best hop.

E.g. Consider the network shown in the Figure 5.6 and that on each originator node in the network there is one originator i.e. each originator node has a single interface (say wlan0) over which the B.A.T.M.A.N routing is operational. Now originator on node B would receive OGMs from originator on node F via multiple paths. Each OGM generated from the originator on node F would carry a unique sequence number. The receiving node B would maintain a sliding window corresponding to each single-hop neighbour (A, E, D and C) through which OGMs from originator on node F is received. It records the count of unique OGMs (identified through sequence numbers) from node F received from the neighbours within the sliding window of that neighbour. The neighbour through which maximum number of OGMs have been recorded within the sliding window is concluded as the next best hop. Assume that in the sliding window maintained for neighbour D maximum number of OGMs from originator on node F are received compared to sliding window maintained for other neighbours A, E and C. Then the originator on node D is concluded as the next best hop towards the originator on node F by originator on B. The sliding window may vary based on the received sequence numbers and hence different neighbours may be concluded as best next hop over time depending upon the link quality and the received OGM count.
Likewise B would receive OGMs from originator on node D from the neighbours A, C and E. B would maintain different sliding windows corresponding to originator on node D for each neighbour A, C and E. Based on the count of the received OGMs within the sliding window of A, C and E for originator on D, sliding window in which maximum OGMs have been recorded is concluded as the best next hop towards D. Each originator (say X) corresponding to every other originator (say Y) maintains sliding windows for each neighbour through which the OGMs from Y are received and then best next hop is determined. Thus the originators on nodes does not determine the entire topology and compute the shortest paths rather just computes the potential next hop towards a

destination.

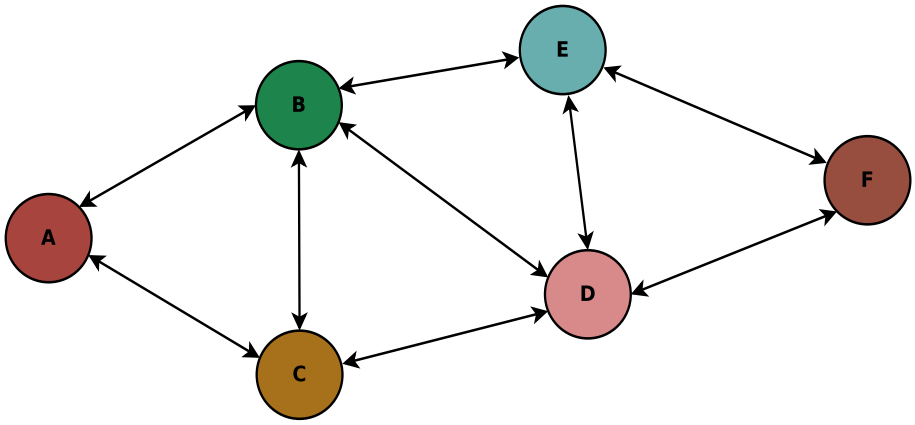


Figure 5.6: An ad hoc network between a group of originator nodes with B.A.T.M.A.N routing enabled on a single interface on those originator nodes.

Bidirectional Link Check / Estimation : Bidirectional link check is the mechanism through which a originator on an originator node determines that the link to its neighbour is working in both directions. Let us explain the mechanism through the originators on originator nodes A and B as shown in Figure 5.7. Consider at time t, originator on originator node A emits an OGM with sequence number 'x'. One of the flags in the OGM is 'direct-link' flag and when originator on A sends the OGM the flag is set to false. Originator on node B on the receival of the OGM message from the originator on originator node A determines that the OGM is from its one neighbour through the match between originator address in the OGM and the source address of the packet. Originator on B sets the 'direct-link' flag as true and rebroadcast the OGM. The rebroadcast OGM would be received by originator on A and would conclude that the link to single-hop neighbour, originator on C is bidirectional. Besides just the check for direct link flag originator on A would also check if sufficient number of OGM are received via originator on B and only then the link to B would be qualified as bidirectional.

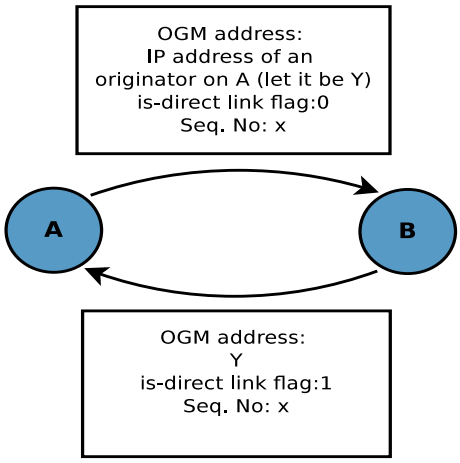


Figure 5.7: OGM rebroadcast from originator on B for bidirectional link determination.

5.3.3 Flooding Mechanism of OGMs :

Flooding mechanism is the means through which originator messages from an originator node are circulated across the network. Originator messages received at a certain node is not rebroadcast if either of the following conditions does not fulfil :

- The TTL of the originator message becomes zero.
- The originator messages is received via a link or through the single-hop neighbour for which bi-directional link check is not validated yet. Consider Figure 5.6, where originator on B received OGM from originator on F via originator on E and that the link between originator on B and E is not the bidirectional link then the OGM from F is not further rebroadcast by originator on node B.
- The transmission quality value in the originator message equals to zero.
- The originator message from an originator is not received via the best next hop towards that particular originator is not further rebroadcast. This condition is used only once the best next hop towards different originators are identified. In context of Figure 5.6, assume that B has concluded originator on E as the next best hop towards originator on node F. If originator on B receives originator from F via originator on E then originator on B would rebroadcast it. However if originator on B receives originators from F via originator on A, C and D then it would not be rebroadcast. They would however be used by originator on B to compute the best next hop towards originator on F.
- The previous sender address in the originator message is equal to the originators IP address, in this case the originator message is echoed from the neighbouring nodes and need not to be further rebroadcast. In Figure 5.6, consider D broadcast an OGM of some other originator. Now on broadcast E would also receive the OGM and assume that E again rebroadcast the OGM which would be received by originator on D again. Originator on D would not process that rebroadcast OGM as it was previously broadcast by itself. However if D broadcast its own OGM and it on rebroadcast from E would be received by D again, then this is used for bidirectional link estimation towards originator on node E.

5.3.4 Originator Message Content :

According to the RFC [34] the originator messages have fixed size of 12 bytes. Each B.A.T.M.A.N packet is encapsulated as an UDP packet and the port 4305 has been assigned for the operation of B.A.T.M.A.N. Originator message can contain optional extension messages each of fixed size 5 bytes appended at the end of the originator message of 12 bytes. The size of the field in the originator message and the order in which they appear in the originator message is described below :

1. **Version (Size - 1 byte) :** This contains the implementation version of the B.A.T.M.A.N. A node operating on a certain version on receipt of an originator message from other node operating on different implementation version would discard the corresponding originator message.
2. **Flags (Size - 1 byte) :** There are 2 flags of one bit each and the remaining 6 bits are reserved for future use. The 2 flags are as follows :

(a) **is-direct-link flag :** This field indicates that whether a node is a single-hop neighbour or not.

(b) **Unidirectional flag :** This indicates that the link to a neighbouring node is bidirectional link or not.

3. **Time To Live / TTL (Size - 1 byte) :** This indicates the number of hops the originator message can be transmitted.

4. **Gateway Flags (Size - 1 byte) :** A node in the B.A.T.M.A.N might be connected to the Internet. It announces its capabilities to act as a gateway for the other nodes to connect to the Internet through the gateway flags. If it does not provide access to the Internet the gateway flags are set to 0.

5. **Sequence Number (Size - 2 bytes) :** A unique number that is send with every originator message. An originator picks up a random number in the beginning at the time routing is enabled on it and increments the number by one with every originator message it sends. This field helps in identification of duplicate messages that can be received by other originators in the network.

6. **GW Port (Size - 2 bytes) :** If the node has the gateway capabilities to act as a gateway to the Internet for the other nodes, it specifies the port on which the other nodes can connect to the advertising node. This port is destination UDP port for the client node connecting to the gateway node.

7. **Originator Address (Size - 4 bytes) :** The IPv4 address of the interface on behalf of which the originator message has been generated.

5.3.5 Data Structures used

Originator Table Each node in the network maintains a table called originator table that contains one entry corresponding to every other originator from which an originator message has been received within the purge time-out interval. A node may have multiple originators while the receiving node is not aware that this originators belong to the same node. Corresponding to the every originator it maintains the list of neighbours from which the originator message from that particular originator have been received. Following points illustrate some crucial details among the others that are maintained corresponding to the every entry in the originator table :

1. **Originator identity :** According to the RFC [34] this identity is the IPv4 address of the originator and it is also included in every originator message generated by an originator.

2. **Neighbour Information List :** Following information is maintained per neighbour through which an originator message of the originator corresponding to which this neighbour list is maintained is received.

   - Sliding Window : A range of sequence numbers with a lower bound and a upper bound determine the sliding window of certain. All the sequence numbers in the received originator messages that fall within the sliding window are termed as In-Window sequence numbers and the rest are termed as Out-Of-Range sequence numbers.

- Packet Count : Packet Count is the count of sequence numbers recorded within the sliding window and this is used as a metric to compute the best next hop towards other originators.
- Last Valid Time : The timestamp at which the last valid OGM was received via this neighbour.
- Last TTL : The TTL of the last OGM which was received via this neighbour.

3. **Last Aware Time :** The timestamp value that is updated with the receipt of an OGM from or via the given originator.

5.3.6 Purging the entries in the Originator Table

If there are no OGMs received from a known originator over PURGE_TIMEOUT interval then the entry corresponding to that originator is removed from the originator list, associated information of the neighbouring nodes to that originator and the best next hop towards that originator is removed from the routing table. The recommended value for the PURGE_TIMEOUT = 10 * WINDOW_SIZE * Orig_Int.

Chapter 6

Realization of Multi-hop Communication in the AllJoyn framework using the B.A.T.M.A.N protocol

6.1 Introduction

This chapter discusses about the integration mechanism of the AllJoyn framework and the B.A.T.M.A.N routing protocol. The objective is to realize that nodes which are multiple hops away to each other be able to discover and advertise the AllJoyn services that are running on them to each other. The integration mechanism to realize the same in the source code of the B.A.T.M.A.N and the AllJoyn framework are described. The beginning of the chapter gives a brief overview about the B.A.T.M.A.N kernel module, how it is compiled and the B.A.T.M.A.N routing is enabled on a certain node followed by illustration of the traditional service advertisement and discovery mechanism in AllJoyn. A Naive approach for the integration is then presented with the implementation details of the realization of multi-hop service discovery and advertisement. Drawbacks in the naive approach are highlighted following which further two approaches: Improved Approach and Ask / Reply approach are realized and their implementation details are described. Another goal is to facilitate the communication between AllJoyn services running on nodes which are multiple hops away to each other. This involves implementation to route the application data from the AllJoyn services using the B.A.T.M.A.N routing tables generated at the node. Implementation details of the same are discussed. Certain generic terminologies are then presented which have been inferred from the various integration mechanisms for multi-hop service discovery and mechanism to route the application data over multiple hops. The details of new packets being generated in different approaches are mentioned at the end of the chapter.

6.2 B.A.T.M.A.N Kernel Module

Loading the module and enabling B.A.T.M.A.N routing : Implementation of the B.A.T.M.A.N routing is obtained from the following source [39]. The implementation is termed as B.A.T.M.A.N Advanced / batman-adv and is available as a kernel level implementation. Kernel module can be generated from the source code through *make* and *make install* procedure and the obtained kernel module can be inserted into the kernel through the *insmod* command.

Enabling B.A.T.M.A.N routing : Insertion of the module just loads the module into the kernel and the routing is not operational yet. For the B.A.T.M.A.N routing to be operational

it has to be specified a particular interface and then the B.A.T.M.A.N routing module would be operational on that particular interface. Insertion of the module into the kernel results into the creation of a folder named "batman_adv" at the following path "/sys/class/net/Y" where Y denotes the supported B.A.T.M.A.N interfaces which are ethX, wlanX, athX and Bluetooth interfaces [40]. Inside the batman_adv folder are two files named mesh_iface and iface_status. When the module is just loaded in the kernel values in the files mesh_iface and show_iface_status are "none" and "disabled" respectively. Now in order to enable B.A.T.M.A.N routing on an interface (say wlan0) the value in the file mesh_iface at the path "/sys/class/net/wlan0/batman_adv" has to be changed from "none" to "bat0" and accordingly the value in the file iface_status at the same path would be changed to "enabled" by the B.A.T.M.A.N routing module. Similar change can be performed to enable B.A.T.M.A.N routing on eth0 interface by changing the value in the mesh_iface file at the following path "/sys/class/net/eth0/batman_adv". Disabling the routing on that interface is done by changing the value back to none in the mesh_iface file. "bat0" is a virtual interface that is created by the kernel module and it acts as bridge between all the interfaces on which B.A.T.M.A.N routing is enabled on a certain node. "bat0" virtual network interface is created on the node once and enabling of routing at other interfaces later connects those interfaces to the created bat0 virtual network interface. Virtual interface is destroyed when B.A.T.M.A.N routing is disabled across all the interfaces on which it was operational before. Corresponding to the bat0 virtual interface a private driver space is allocated to it and it maintains all the physical interfaces linked to it. For the routing messages to be transmitted from a physical interface (wlan0, eth0 etc.) it has to be linked to a virtual interface. As in the implementation it checks whether the physical interface is linked to virtual interface and only then the OGMs are being transmitted periodically. If instead of "bat0" some other value (say bat1) is entered into the mesh_iface file for an interface (say wlan0) then a virtual interface with the name bat1 would be created and the interface wlan0 would be linked to that virtual interface bat1. For the sake of generality and simplicity of explanation in all further sections, assume that "bat0" is written into the mesh_iface files corresponding to all the interfaces on which the B.A.T.M.A.N routing has to be made operational. Thus all the interfaces on which B.A.T.M.A.N routing is operational are linked to the virtual interface "bat0".

batman-adv : The obtained B.A.T.M.A.N implementation (batman-adv) operates at layer 2 of the ISO/OSI model [40] unlike other routing implementations that operate at layer 3. Hence in contrast with the RFC [34] the identity in the originator message in implementation is the MAC address and not the IP address of the originator. Accordingly the maintained originator tables at the nodes contains MAC addresses of the originators form which originator messages have been received. Originator messages in implementation are not transmitted as UDP packets (according to the RFC [34]) rather as the layer 2 packets with the originator message appended at the end of 14 bytes of Ethernet Header. The structure corresponding to the originator message in implementation has following fields and sizes as shown in table 6.1. Thus the originator message size in implementation is of 24 bytes. However it is appended with TVLV information / containers of size 28 bytes. TVLV stands for **T**ype **V**ersion **L**ength **V**alue which is a technique used in data communication protocols for encoding information as type-length-value. In B.A.T.M.A.N it is used as a means to preserve backward compatibility with the previous versions. Hence TVLV information is appended as TVLV container at the end of originator message. Thus the size of the entire originator message including the TLLV container size and the Ethernet header becomes 66 bytes with the actual origi-

Field	Size
Packet Type	1 byte
Version	1 byte
Time to live	1 byte
Flags	1 byte
Sequence Number	4 bytes
Originator Address (MAC address)	6 bytes
Previous Sender Address (MAC address)	6 bytes
Reserved	1 byte
Transmission quality	1 byte
TVLV Length	2 bytes

Table 6.1: Fields in originator message and their sizes in batman-adv implementation.

Ethernet Header	14 bytes
Originator message	24 bytes
TVLV information	28 bytes

Table 6.2: Content and size of different segments in the Default OGM packet.

nator message content in it of size 24 bytes. We term the entire 66 bytes of information as Default OGM messages and the corresponding packets that would carry this information as Default OGM packets. The table 6.2 represents the content and the size of different segments in the Default OGM packet. The actual originator message content in the entire 66 bytes is that of 24 bytes only.

All the additions and incorporation that we perform for AllJoyn integration with B.A.T.M.A.N are done in the obtained kernel module source code [39]. In my thesis the original batman-adv implementation without additions is termed as Default approach (Version I) and the originator messages in it are termed as Default OGM packets.

6.3 Traditional Service Advertisement and Discovery Mechanism in AllJoyn

AllJoyn Applications and services and a basic overview of how they communicate : A node (laptop / embedded devices /mobile phones etc.) can have one or more AllJoyn applications running on it. Each application can have one or more services in it and that each service has a unique name. The services connects to the software Bus in the AllJoyn router via Bus Attachments and requests for a unique name. If the request is satisfied the unique name is allocated to that service else the router specified unique name is allocated to the service. The unique name allocated to the service is advertised by the AllJoyn router to other nodes which eventually come to know about the service running on the advertising node. Now the AllJoyn router can be a bundled router where in each application has its own router, standalone router where multiple applications use the same router and a remote router where in the AllJoyn applications uses the AllJoyn router on a an another device in proximity. Figure 6.1 gives an overview of of the architecture how an AllJoyn application, services in it are connected via Bus Attachments to the AllJoyn router.

Every AllJoyn service implements certain interfaces and in the service advertisement, the interface the service implements are also advertised through the AllJoyn router. The realization of interfaces in a service are done through Bus Objects which provide definitions to the methods declared in the interfaces. Thus a service comprises of one or more Bus Objects those provide definitions to the interface the service implements and that each service is binned to a port. Thus the details that

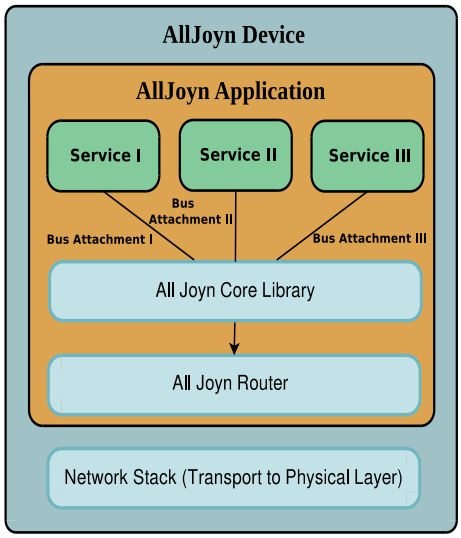


Figure 6.1: An AllJoyn application architecture.

are included in the advertisement of the service comprises of the name of the service, the interfaces the service implements, with the details of the methods and properties in those interfaces and the port on which the service is running. Theses details are advertised by the AllJoyn router for each service connected to it via the Bus Attachments along with the device identification (IP address). Device identification ensures uniqueness among multiple services with same name running across devices. Remote services (say X) on the other hand specifies to the router the types of services it is looking for by specifying the name prefixes to the AllJoyn router. AllJoyn router notifies the service X when it discovers services with name whose prefix matches with the specified prefix by the service X. Remote services on discovery of the other services and the interfaces it implements, requests for the connection on the specified port, a session is created and the remote service creates proxy Bus Objects to communicate with the Bus Objects of the service it is connected to. In this context one service acts as the consumer service and the other as provider service and the corresponding applications in which the services are running acts as consumer application and provider application. *E.g.* Consider a motion sensor and a light bulb and that each of them have an AllJoyn application running on it. The service name on the motion sensor is a.b.Sensor and that on the bulb is a.b.Bulb. Assume that these services are connected to the AllJoyn router on their respective devices via the Bus Attachments and that the router is advertising the details of these services. Further, the service a.b.Sensor implements interface which has methods that returns the sensor readings the motion sensor has sensed and the service is running on a particular port. Service on light bulb has specified to its router that it is looking for services with prefix 'a.b'. Router on discovery about the service "a.b.Sensor" would notify the service "a.b.Bulb" that it received advertisement about the service whose prefix matches with the specified prefix. The service a.b.Bulb would then initiate the session establishment mechanism to connect to the device and the port on which service a.b.Sensor is running. It would then create proxy Bus objects to communicate with the service on sensor and invoke methods that service implements. Based on the obtained value from the sensor light bulb can be turned on / off.

Advertisement Mechanism : Following two mechanisms are employed for services advertisement and discovery over AllJoyn proximal network [41].

IPv4 Multicast group address	224.0.0.251
IPv6 Multicast group address	FF02::FB
Multicast port number	5353

Table 6.3: IANA assigned addresses and ports for NGNS mechanism in AllJoyn

- Name Based Discovery.
- Announcement Based Discovery.

We focus on name based discovery mechanism as the AllJoyn source code which we use for integration with B.A.T.M.A.N uses name based discovery mechanism. AllJoyn source code "standard core v15.04" is obtained from the following source [25] and all the modifications required for integration are performed in it. Discovery mechanism employed in all the source code releases later to version 14.06 is termed as Next Generation Name Service (NGNS) which is a name based discovery mechanism with performance enhancements on it. The name based discovery mechanism employs IS-AT and WHO-HAS messages for service discovery and advertisement.

IS-AT message : The IS-AT messages advertises the services using the well known name of the service, interfaces it supports, details of those interfaces and the port at which the service is running. AllJoyn router sends out IS-AT messages periodically advertising about all the services connected to it via the Bus Attachments. Each device which has a service, IS-AT messages corresponding to those are send out by the router and the device can be termed as producer device. The device discovering the service is called consumer device. Eventually every device acts as a producer device for the services it advertises and is a consumer device for the services it discovers. **WHO-HAS message :** The WHO-HAS messages discovers the services based on their names. A service on a device could specify the router a name prefix that states that the service is looking for other services with the specified prefix. The AllJoyn router would then generate WHO-HAS messages containing the requested name prefix. All those devices which have the services whose name matches with prefix in WHO-HAS message, AllJoyn router on those devices would respond with the IS-AT messages providing the details of the service. The IS-AT messages can be generated in respond to WHO-HAS messages. The generation of WHO-HAS messages is optional and that the services on a device can come to know of the service on other devices through IS-AT messages. The NGNS mechanism sends out IS-AT messages periodically over the IANA (Internet Assigned Numbers Authority) assigned multicast IP address and port numbers [41] shown in the table 6.3.

6.4 Naive Approach of B.A.T.M.A.N and AllJoyn integration

Modifications are performed on the B.A.T.M.A.N kernel module obtained from [39] and AllJoyn source code obtained from this [25]. AllJoyn applications comprising of services, AllJoyn router and the AllJoyn library all operate at the user space and the B.A.T.M.A.N routing module operates in the kernel space. The applications in the kernel and the user spaces cannot communicate with each other directly. There exists following ways in which the applications in user space and the kernel space can communicate with each other :

- Sysfs
- Debugfs

- System Calls
- Netlink sockets

We employ the **debugfs** mechanisms to let the AllJoyn framework communicate with the B.A.T.M.A.N routing module. Debugfs is an in kernel file system that is primarily used for debugging purposes that provides means for an user space application to read data from / write data to files in debugfs file system. Modules in the kernel which are monitoring those files detect the write operation by a user space application and it can obtain the written data by the user space application. Similarly the kernel module can write to those files and the application in the user space can fetch data from those files. There are proper APIs in linux system to perform all the debugfs file system related operations and we do not go into the details of those APIs. We present how we used the debugfs to let the AllJoyn framework communicate with B.A.T.M.A.N and the actions performed by each of them when the write operations are performed by the other.

Why debugfs? When the B.A.T.M.A.N module is loaded into the kernel a folder named batman_adv is created at the root of the debugfs file system (/sys/kernel/debug/) besides the batman_adv folder created at the following path "/sys/class/X/" mentioned in the section 6.2. There exists a folder for all the B.A.T.M.A.N supported interfaces inside the batman_adv folder at the debugfs root. Now when the B.A.T.M.A.N module is enabled on a certain interface on the device, virtual interface named "bat0" is created as explained in the section 6.2 and a folder corresponding to the that virtual interface is created as well inside the batman_adv folder at the debugfs root. Inside the "bat0" folder B.A.T.M.A.N creates various files such as neighbours, originators, gateways, socket, etc. These files exist in the debugfs file system and the B.A.T.M.A.N routing module writes data to these files during its operation. User space applications can fetch data from these files if needed. *E.g.* A user application can fetch content from the originators file to display the list of originators from which the routing module has received OGMs. Sysfs is an another virtual file system that lets information about hardware devices and the associated drivers to be exposed to user space applications. However the sysfs file system is maintained corresponding to different hardware devices and hence different interfaces would have different sysfs directories. We wanted that services information about the AllJoyn services be broadcast through all the interfaces on which B.A.T.M.A.N routing is enabled. Debugfs file system provides a common place to write service advertisement related information and that routing module could fetch the data from there and advertise service information across all the interfaces on which B.A.T.M.A.N routing is enabled. If sysfs file system is used then corresponding to different interfaces there would be different sysfs directories at the path "/sys/class/net/X" where X denotes the name of an interface. Then the B.A.T.M.A.N routing module have to read /write across all sysfs directories corresponding to interfaces on which B.A.T.M.A.N routing is enabled. Thus we choose debugfs file system which provides a generic location from where the B.A.T.M.A¿N routing module could read data and send across different interfaces as well write data received across different interfaces irrespective of number of interfaces on which B.A.T.M.A.N is enabled.

Modifications in batman_adv implementation are performed to create a file named 'serviceInfo' inside the "bat0" folder at the following path : "/sys/kernel/debug/batman_adv". This file is created in the debugfs virtual filesystem and it is created when the B.A.T.M.A.N module is enabled on a certain interface as "bat0" folder at the above path is created when B.A.T.M.A.N module is made operational on an interface. Appropriate open, read, write and close callbacks are implemented for

the file 'serviceInfo'. This callbacks are triggered when the an user space application performs a file operation on the 'serviceInfo' debugfs file. In our case the user level application is the AllJoyn router. AllJoyn router typically multicasts the service advertisement of all the services attached to it. In our implementation we stop the service advertisement related multicast from the AllJoyn router over the IANA assigned multicast address. Rather the service details which the router would be advertising are written into the 'serviceInfo' file at the path "/sys/kernel/debug/batman_adv". B.A.T.M.A.N module would detect the write operation on the 'serviceInfo' file from the AllJoyn router through the debugfs callback APIs. The module would then read the details from the 'serviceInfo' and form a message corresponding to the service advertisement.

Detailed Description : AllJoyn router periodically would advertise the details about the services attached to it. We write the details about the services attached to it in the 'serviceInfo' file in the debugfs file system. In our implementation these details include the name of the service, name of the interfaces it implements and the details of those interfaces. We assume that each service implements a single interface and accordingly the mechanisms are designed. It however could be extended if a service implements multiple interfaces. The details about the interface the service implements would comprise of all the methods (with their input parameters, types, output type) and the variables the interface has. B.A.T.M.A.N module would obtain the written data from the 'serviceInfo' file which it would come to know through the write call back. The module would form an appropriate message comprising the details of the service and advertise these details through B.A.T.M.A.N enabled interface. The details corresponding to every service those are advertised in our implementation with the size of each field in the details are listed below :

- **Name (Size - 20 characters / 20 bytes) :** This represents the name of the service and the name is acquired by the service through the usual mechanism in AllJoyn. The name is written by the router in the 'serviceInfo' file.
- **Interface Name (Size - 20 characters / 20 bytes) :** This represents the name of the interface the service implements. This is also written by the router in our implementation into the 'serviceInfo' file.
- **sourceIP (Size - 4 bytes) :** IP address of the interface through which the service is being advertised. This information is included by the routing module depending upon the interface through which service is being advertised.
- **sourceMAC (Size - 6 bytes) :** MAC address of the interface through which the service is being advertised. This information is also included by the routing module depending upon the interface through which service is being advertised.
- **propCount (Size - 1 byte) :** Count of the properties present in the interface. Properties refer to methods and variables in the interface. This detail is included by the router and is written into the 'serviceInfo' file.
- **Port (Size - 2 bytes) :** This corresponds to the port on which the service is running and is also included by the router.
- **Protocol (Size - 1 byte) :** The protocol corresponds to TCP / UDP and it specifies that whether the service can be connected and that the methods in the service be invoked through

TCP or UDP. In our implementation TCP is assigned as value 0 for the protocol field and UDP as 1.

The maximum length of the service name and the name of the interfaces in AllJoyn implementation is 256 characters / 256 bytes. However in our implementation we restrict their maximum length to 20 characters. Restriction for the length is not made at the AllJoyn router level but at the kernel module level. So when the module reads a service name and it is found to be of length greater than 20 characters, module would not advertise the information about that service. The reason behind limiting the maximum length is explained in the further description. The size of all the fields above in the implementation sum up to a total of 56 bytes.

Next we present the details about the methods and variables in the interface that are advertised with the service advertisement. The fields and the size of those are mentioned below :

- **isMethod (Size - 4 bytes) :** This field represents that property is a method or a variable. If it is a variable the value is set to 0 else 1.
- **name (Size - 20 characters / 20 bytes) :** The name of the property.
- **details (Size - 30 characters / 20 bytes) :** This field is useful when the property is a method. Details represent the input and the output parameters of the method maintained through following format : "o:x,i:x,i:x ...". o corresponds to the output value of the method, x towards its type which can take value i for integer, s for short, c for char, l for long. Similarly i corresponds to the input value and the corresponding x value separated by a colon towards its type. This format basically conveys the details about the input and the output parameters of the method along with their types.

The size of all the above fields sum to a total of 56 bytes in the implementation due to structure padding. Corresponding to the every property of the interface the following details are maintained and advertised with the service advertisement. For the sake of generality, we assume that each interface has one method and that the service implements the method in the interface.

Relevance of Interface details An AllJoyn service implements certain interfaces the definitions of the methods in those are provided by the Bus Objects. Advertisement about the details of the methods and the variable in the interface is done in the regular AllJoyn implementation as well and it lets a remote service know about the details of those. The remote service could then invoke the methods, the advertising service implements as it knows this through the advertisement of the interface details of the interface the service implements. This invoking is basically through RPC mechanism where the remote service creates a Proxy Bus Object and that call to the method is encapsulated by the proxy bus object as an message. This message is then communicated to the advertising service through TCP / UDP as specified by the advertising service at the port on which the advertising service is running. The response to the method is then encapsulated back as a message and transmitted over TCP or UDP which so ever the service is operating on.

How service information is advertised : B.A.T.M.A.N routing mechanism involves transmission of originator messages at every Originator interval / Orig_Int. The default originator message is of 66 bytes including the Ethernet header and TVLV container as discussed previously. We append the originator message with the AllJoyn service information to advertise the service details to other originators. Now the basic service details we advertise about the AllJoyn services is of 56 bytes and

that each property detail is of 56 bytes. If there are X services attached to the router and that the interface these services implements, each of them has Y properties, the number of bytes required for the advertisement of the X services would be **X * (56 + Y * 56)** bytes (assuming that each service implements a single interface). Different devices may have different number of services and also the value Y may change based on the number of properties in the interface those service implements. In order to account for the uniform size of information for the service advertisement by each device we consider that each device has a single service and the interface that service implements has a single method. This consideration is taken so that same size of information is appended for the advertisement of AllJoyn service information by all devices. The mechanism would work with the existence of multiple services and that each of them have varying number of properties. However here the objective is to provision a basic mechanism for multi-hop service discovery and for that we consider each device has a single service and the interface service implements have a single method.

The fields in the originator message in the default implementation are explained in the table 6.1 and that total size of those is 24 bytes. We append one more field of 4 bytes called as "serviceLen". This field contains the value corresponding to the number of bytes occupied by the service and the interface details that follows in the message. Based on the number of services that are advertised the value of this field may change. However as we consider that each device has a single service and interface the service implements has a single method the total size of the service information being advertised becomes 112 bytes. Accordingly the value of "serviceLen" field is set to 112 bytes. In general the value of serviceLen could be X * (56 + Y * 56) where X denotes the number of services and Y the number of properties in the interface, the service implements.

Total size of originator message : With the incorporation of an additional field of 4 bytes in the originator message content of 24 bytes, the total originator message size becomes 28 bytes. However as in the default implementation every originator message is appended with TVLV information of 24 bytes we append those 24 bytes in our implementation as well so as not to disrupt the normal operation of B.A.T.M.A.N routing. In addition to that we append 112 bytes of information corresponding to the AllJoyn service running on the device. Effectively the size of originator message becomes 182 bytes including the Ethernet header. We term this approach to incorporate the AllJoyn service information into the B.A.T.M.A.N originator message as the Naive Approach for integration (Version II). The 182 byte sized originator message so generated in the Naive approach is termed as the OGM-AllJoyn packet with the actual originator message content in it of 28 bytes. Table 6.4 gives the content and size of different segments in the OGM-AllJoyn packet. Likewise multiple services information can be accumulated in the originator message with the 'serviceLen' variable holding the amount of information corresponding to different services being appended in the originator message. However this would lead to packets of different sizes, and thus in order to have a packet of single type we make the assumption of a single service and a single property in the interface the service implements. Also the reason behind the size of service name field and the interface name field to be restricted to 20 bytes from 256 bytes is to conserve the amount of information propagated for the service advertisement. Length of 20 bytes would provide a sufficient long name according to the AllJoyn syntax of the form a.b.c.. for the service and the interface names.

Now in the Default approach of B.A.T.M.A.N routing, Default OGM packets are broadcast by the originators in every originator interval. Similarly the OGM-AllJoyn packets are broadcast in the Naive approach every originator interval The originator message content in the OGM-AllJoyn

Ethernet Header	14 bytes
Originator message	28 bytes
TVLV information	28 bytes
AllJoyn service information	112 bytes

Table 6.4: Size of different segments in the OGM-AllJoyn packet.

packets enables the operation of the B.A.T.M.A.N routing as the way it operates. However the serviceLen field in OGM-AllJoyn packet, lets a receiving node determine the presence of the service information in the packet. The receiving node would fetch the details about the service and would come to know of the service name, interface it implements, MAC address, IP address and the port on which it is running. Thus the single-hop neighbours would come to know of the services existent on the device corresponding to the advertising originator. The service information so retrieved is written to the 'serviceInfo' file in the debugfs file system and that AllJoyn router would read the information about the received service advertisement. It can then notify the services attached to it if the service name prefix in the received advertisement matches with the requested prefix by the attached services. The OGM-AllJoyn packets are rebroadcast as it is by the single-hop neighbours, further by two neighbours and so on. Eventually every originator could come to know of the AllJoyn services running on device corresponding to other originators. Likewise every originator appends the AllJoyn service information at the end of the originator message. These information is broadcast by the originator and rebroadcast by other receiving originators according to rebroadcast rules of B.A.T.M.A.N only.

Modifications in the Originator Table : Nodes maintained an entry for each other originator from which an OGM was received in the default approach. Similarly the originator tables are maintained in the naive approach too with all the details as it is with an additional detail about the AllJoyn services. Since the OGM-AllJoyn packet carries the originator message the usual details in the originator table can be populated from it. Similarly the calculation of next best hop would work in the usual manner through the originator message content in the OGM-AllJoyn packet. Originator table corresponding to every originator entry now have the fields to store the service information it received from that particular originator. The service information from the received OGM-AllJoyn packet is captured and stored into these fields. Purging of the service information is done with the purging of the originator entry and a separate time-out interval is not maintained for purging the service information.

Integration mechanism explained through an example : Consider the network shown in the Figure 6.2. Nodes in the network have been annotated with alphabets, here it is from A-E. Links between the nodes denote that the nodes are reachable or single-hop neighbour to each other. Nodes have B.A.T.M.A.N routing operational at the kernel level and there exists AllJoyn service and router at the application level. The two are integrated using the debugfs file system on each node and the service information from the router is written to the 'serviceInfo' file in debugfs file system and fetched by the routing module. Similarly the information returned by the routing module is fetched by the router and hence the bidirectional link between the two in Figure 6.2. Now on each node let there be one interface (say wlan0) on which the B.A.T.M.A.N routing is enabled, thus there exists one originator on each node. Also there exists an AllJoyn application

comprising of a service and the service implementing an interface which has a single method on each node. Different services exist on different nodes. AllJoyn router on each device would receive the request for a name from the service and the service is allocated the requested name if the name is not existent with any other service attached to the router. The AllJoyn router on each node would write the service details, comprising the service name, name of the interface it implements, method details of those, propCount of the number of properties in the interface, port and protocol on which it would accept incoming connections into the 'serviceInfo' file in the debugfs file system. The method details are converted into the format specified previously (o:x,i:x,i:x...) by the AllJoyn router itself in our implementation. The B.A.T.M.A.N module operating in the kernel space would detect the write operation by the AllJoyn router and accordingly would fetch the content from the 'serviceInfo' file. The information corresponding to the service, its name, interface name of the interface it implements, port and the protocol are available in the data fetched from the 'serviceInfo' file. However the details such as sourceMAC address and the sourceIP address are generated by the routing module. Corresponding to the interface through which the originator message would be send the following fields would have the MAC address and the IP address of that interface. Now as we assumed that B.A.T.M.A.N routing is enabled on the interface wlan0 across all nodes. Routing modules in each node would incorporate the sourceIP address and the sourceMAC address of their respective wlan0 interfaces. This basically conveys other originators who would receive the service advertisement appended at the originator message, the IP address and the MAC address of the originator on which the service is existent. The name of the service, interface name of the interface it implements, sourceIP address, sourceMAC address, propCount, port and the protocol constitute the basic service details about a service and the size of these details is 56 bytes. The property details likewise are each of 56 bytes. However as we consider that their is a single property (a method) in the interface the service implements the total size of property details come up to be of 56 bytes. If there would have been X services then there would be X * 56 bytes for representing the basic details of those and similarly if each of them had Y properties then total X * Y * 56 bytes would be required for the advertisement of those Y properties in the interface X services implements.

In our case total information for the advertisement of the service is 112 bytes, 56 bytes of the basic service details and 56 bytes of the method details of the method in the interface service implements. Routing module on each node A-E would obtain the respective service information about the allJoyn service existent on it. It would generate 112 bytes of the information corresponding to the service information it read from the 'serviceInfo' file. The details such as sourceIP address and the sourceMAC address are populated by the routing module. serviceLen field in the originator message content is updated according to the size of the service information appended at the end. 112 bytes of service information is appended at the end of the originator message content of 28 bytes and 28 bytes of TVLV information and accordingly the serviceLen field would have the value as 112. Thus an OGM-AllJoyn packet of size 182 bytes including the 14 bytes of Ethernet Header is formed. Traditionally each originator would broadcast 66 bytes of information as the Default OGM packet. Here in the Naive approach of integration, each originator would broadcast 182 bytes of OGM-AllJoyn packet. The originator message content would enable and facilitate the operation of regular B.A.T.M.A.N routing and that serviceLen would convey that there is service information about an AllJoyn service in the packet. The rules for rebroadcast are not altered and are according to the default routing mechanism. In figure 6.2, originator on node A would form an OGM-AllJoyn packet

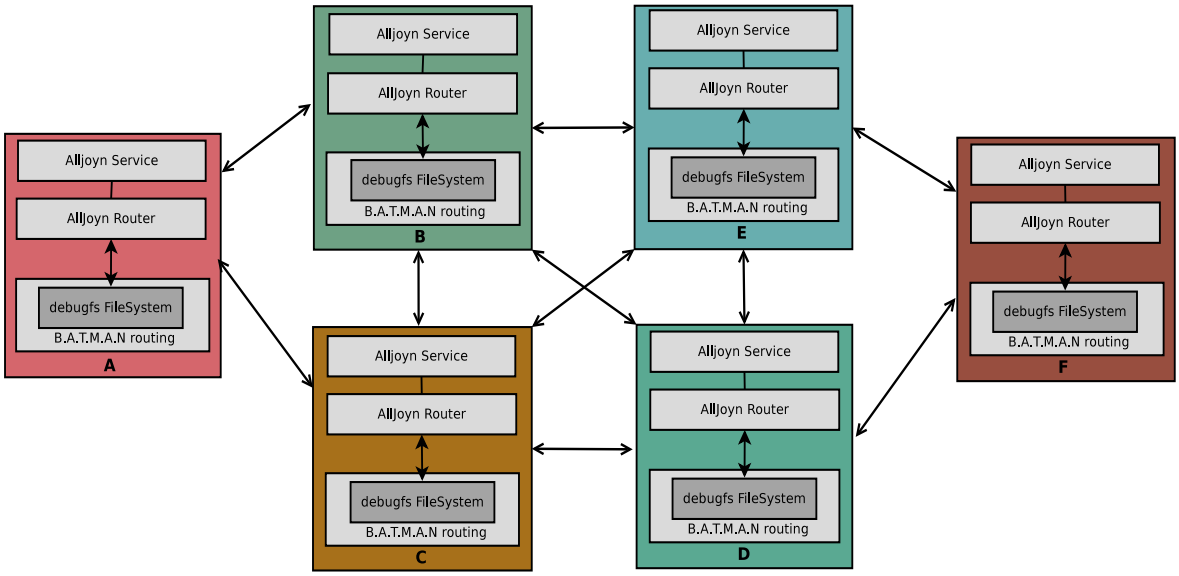


Figure 6.2: A network of nodes with B.A.T.M.A.N routing and AllJoyn framework integration on nodes.

appending the existent AllJoyn service information it obtained from the AllJoyn router on node A. The OGM-AllJoyn packet would be broadcast by A and the corresponding neighbours (B and C) will receive the packet. Neighbour would fetch the content related to the service information from the OGM-AllJoyn packet received from originator on A. The routing module at B and C would update the originator table with the service information from the OGM-AllJoyn packet corresponding to the entry for originator on node A. The routing module would write the service information to the 'serviceInfo' file as well, with all the service details and the method details of the method in the interface service implements. The AllJoyn router on receiving nodes would fetch the written information about the received service advertisement and hence would come to know of the service on originator A along with details : the name of the service, interface name of the interface it implements, sourceIP address, sourceMAC address, port protocol on which it is running and the method details. Receiving originators on node B and C would rebroadcast the OGM-Alljoyn packet from A to their neighbours, which would further rebroadcast to their neighbours and thus every originator would come to know of the service existent on node A. Similar mechanisms would be employed by all other originators on node B, C, D, E and F and thus every originator would come to know about the services on other nodes (single-hop or multi-hop) through the OGM-AllJoyn packet in the Naive approach. Now an originator may receive OGM-Alljoyn packet from an another originator via multiple paths, however the service information in all those OGM-AllJoyn packets would be same. The receiving node would update the service information in its originator table when it receives the service information from an originator first time. Later it would update the service information corresponding to an originator in the originator table only when it received an OGM-AllJoyn packet that has some new information compared to the existing service information corresponding to the originator in the originator table.

Architecture Figure 6.3 represents a consolidated view of the architecture or the network stack in node with AllJoyn and B.A.T.M.A.N integration. AllJoyn operates at the application layer comprising of the AllJoyn services, AllJoyn core library, AllJoyn router and its sub-modules : session

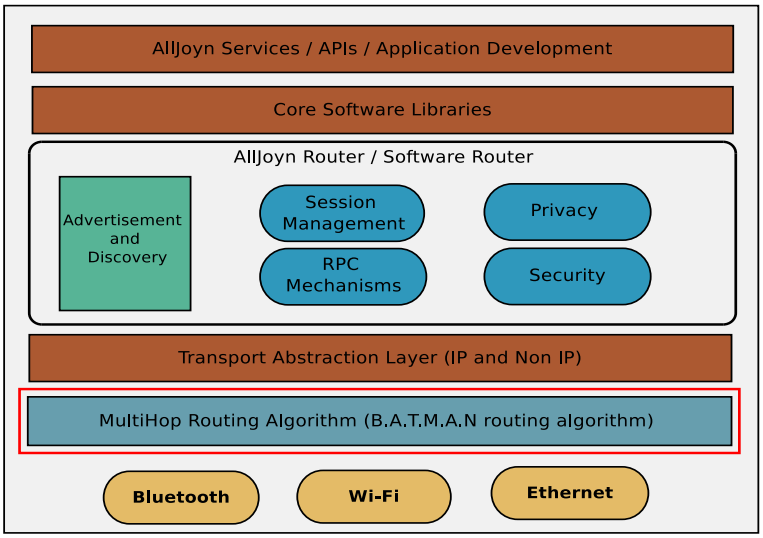


Figure 6.3: Architecture Representing B.A.T.M.A.N AllJoyn Integration on a Node.

management, privacy, security, advertisement and discovery. Below in the stack exists the routing algorithm - B.A.T.M.A.N routing operating in the kernel space. Since our objective was to provision multi-hop service discovery and advertisement communication between the services which are existing on nodes which are multiple hops away from each other. We focus only on advertisement and discovery module in the router and hence route the entire service advertisement and discovery messages to the B.A.T.M.A.N routing module. Apart from the service advertisement and discovery the data between the services is also routed through the B.A.T.M.A.N routing. For communication between two services, consumer service and the producer services, data between the two services is routed through the B.A.T.M.A.N routing. All the session establishment mechanism is not followed as it is for data exchange. The details of the communication between AllJoyn services and how data is routed is explained in the further section. Integration of other modules with the B.A.T.M.A.N routing is considered as the future goal and in the naive approach we provision multi-hop service discovery and advertisement and data routing only.

Figure 6.4 illustrates propagation of service advertisement to multi-hops in context of the nodes in the Figure 6.2. This is explained in taking the Nodes A, B and E into the consideration and the OGM-AllJoyn packet from A. The same analogy could be extended for the OGM-AllJoyn packet from B and E as well for other single-hop and multi-hop paths. The services on node A, B and E are different services and are termed as service A,B and E respectively. The sequence of events as shown in the Figure 6.4 are presented below :

- Each service A, B and E connect to the AllJoyn router on their respective nodes via the AllJoyn library. This connection is made through the use of Bus Attachment.
- On successful connection to the AllJoyn library and the AllJoyn router, the service on each node requests for a unique service name from their respective router. The names could be provided by the service and if the router finds that no other service connected to it has that name, the service is allocated the requested name. If the request name is there with some other service on the router the service is allocated the router provided name.
- Service then requests router to advertise about its existence to other services.

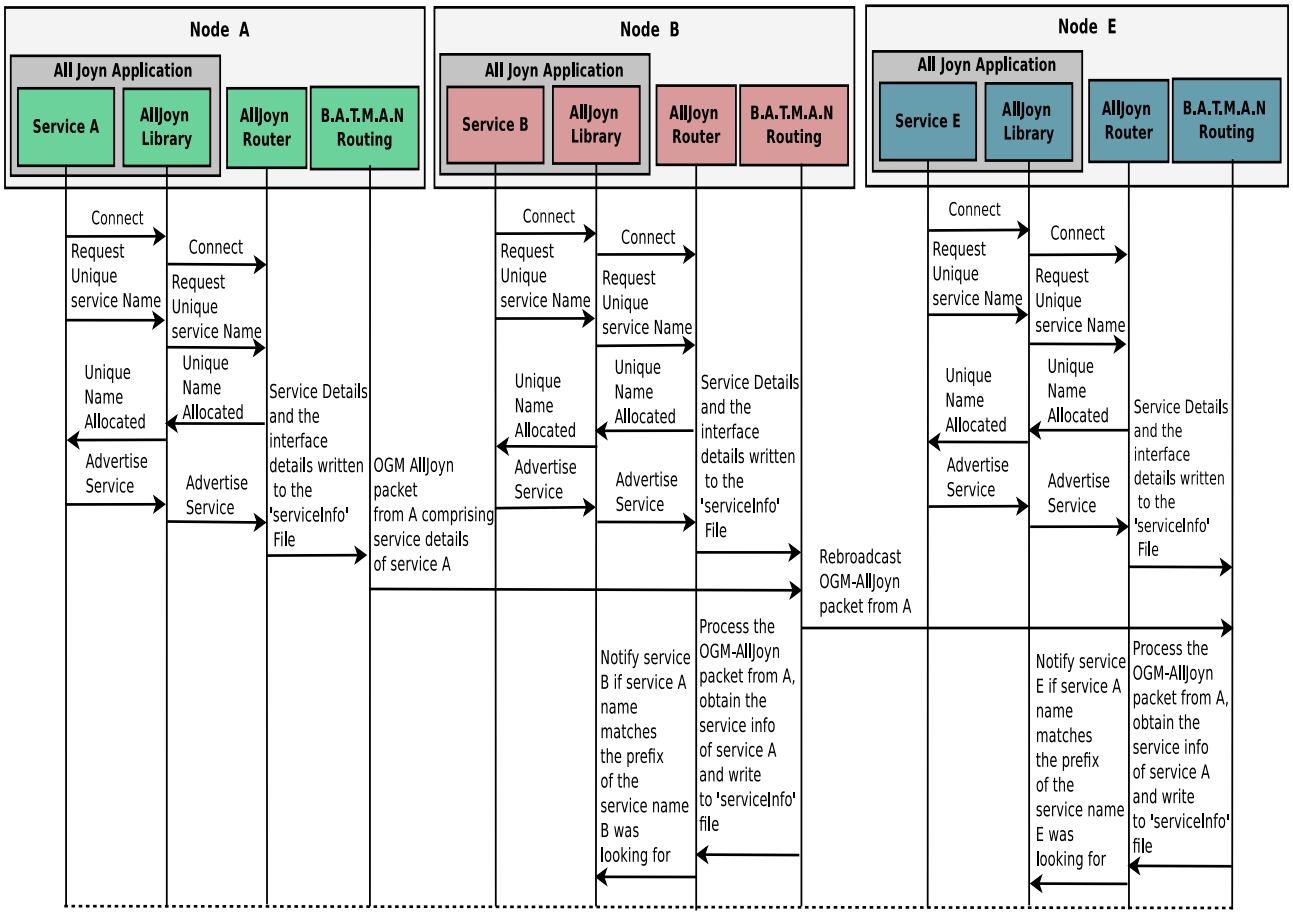


Figure 6.4: Multi hop Propagation of Service Advertisement.

- In out implementation as mentioned before we constitute the basic details of about the service in the AllJoyn router along with the details of the interface it implements.
- The service information is written by the AllJoyn router to the 'serviceInfo' file in the debugfs file system.
- B.A.T.M.A.N routing module reads those details and adds the information corresponding to the sourceIP address and the sourceMAC address corresponding to the interface through which it would be sending the service advertisement.
- For the service A, B and E, we assume that the interface the service implements has a single method. Hence the service information totals upto 112 bytes.
- The service information is appended at the end of 28 bytes of originator message and TVLV information of 28 bytes hence constituting the OGM-AllJoyn packet of size 182 bytes including the Ethernet header.
- The OGM-AllJoyn packet from A is received by B which processes the service information in the packet and writes it into the 'serviceInfo' file as well updates the details in the originator table corresponding to the entry for originator on A. The originator message contents in the OGM-AllJoyn packet from A are used as usual for the operation of B.A.T.M.A.N routing.

- The AllJoyn router on B would read the advertised service information from A from the 'serviceInfo' file and accordingly notify the service B.
- Originator on B would rebroadcast the OGM-AllJoyn packet from A which would be received by A and E both. A would discard the service information as the packet was advertised by itself which is concluded from originator address in the OGM-AllJoyn packet which is A's MAC address.
- E would do the similar processing as originator on B and would rebroadcast the OGM-AllJoyn packet from A. This would further be received by B, however B would discard the packet according to B.A.T.M.A.N forwarding rules as it was the previous sender of the packet.

Conflicts in the read and write operations to the 'serviceInfo' file are handled by the APIs for performing read and write operations to the debugfs file system. In short, Figures 6.3 and 6.4 summarize the entire Naive approach for B.A.T.M.A.N and AllJoyn integration approach.

6.5 Communication between the services (Single-hop or Multi-hop)

Now the naive approach has facilitated the multi-hop advertisement of the services. This in context of AllJoyn can be understood as propagation of IS-AT messages over multi-hops. Similarly the naive approach could be modified to include WHO-HAS messages. However we do not incorporate the WHO-HAS messages considering the fact that through IS-AT messages the nodes come to know of the services on the other nodes. WHO-HAS messages can be considered as a means for explicit generation of IS-AT messages.

Now the nodes have discovered the services existent on its single-hop as well on its multi-hop neighbours through the Naive approach. The next step is the communication between the services which may be single-hop away or multi-hop away to each other. The service can play the role of a consumer as well as that of the provider. If a service (say X) connects to some other service using the port and the protocol it received in the advertisement from the other service (say Y) through Naive approach then service X is a consumer service and that service Y is a provider service. Conversely if Y connects to the service on X on its specified port and using the specified protocol in the advertisement, then Y acts as a consumer service and X as a provider service. The regular mechanism in the communication between two services is through a session establishment procedure where the consumer service connects to the provider service at its advertised port. Once a session is established a session is allocated and its used in all further communications between the producer and the consumer service. The provider service has Bus Objects that provides definitions to the methods in the interface the service implements. Consumer service creates a Proxy Bus Object to communicate with the producer service. Now as producer service in its advertisement includes the interface implements and the methods in those interfaces, consumer service invokes the method using the proxy bus object. The proxy bus object converts the method call to a network message and is communicated to the provider service using TCP / UDP as specified by the provider service in its advertisement. In other words the consumer services invokes methods on the provider service through the Proxy bus Objects which convert the call to a network message that flows over TCP or UDP. We do not delve into the internals of how RPC (Remote Procedure Calls) are established and that how messages are marshalled and unmarshalled. The entire communication between the

services can be considered as UDP / TCP flows abstracting the RPC mechanisms employed within. This is analogous to client server mechanism where a client connects to a server at the port on which it is running using either TCP / UDP. E.g. Consider a service on node A that implements an interface and the method in the interface is implemented through the Bus Object. The method returns the contents of some file stored on node A when invoked. The service advertisement reaches an another node B. B would create a Proxy Bus Object, establish a session with service on A and invoke the method in the interface service A implements. Since method definition does not exist on the same device, hence the Proxy Bus Object is used which converts the call to a network message. This is communicated to service running on node A using TCP / UDP as specified by the service in its advertisement at the port it is running. The service in turn would return the file contents that would be transmitted as TCP / UDP flow and received by the proxy bus object on B and eventually by service on B. At the service level the communication is in the form of method calls but at the transport layer communication is in the form of TCP / UDP flows.

Multi-hop Communication between services : We employ the B.A.T.M.A.N routing module to facilitate communication between services existent on nodes which are single hop or multiple hops away. Corresponding to the way we create a 'serviceInfo' file for service advertisement, we create a file called 'appData' for the exchange of data between the two services. Modifications in batman-adv module are done to create a file called 'appData' in the debugfs file system. This file is created at same path : "/sys/kernel/debug/batman_adv/bat0" where 'serviceInfo' file is created. It is created when B.A.T.M.A.N routing is enabled on a certain interface and for all further discussions assume that it is enabled on "wlan0" interface on a node. Now in the Naive approach, the AllJoyn router reads the service details and the method details of the interface the service implements from the debugfs file system about all the service advertisements received by the B.A.T.M.A.N routing module. The service details among the other details comprises of the sourceIP and the sourceMAC address at which the other service exists. Thus a service knows the IP address and the MAC address at which the other service is existent. Let us explain the mechanism of multi-hop communication between services through the nodes in Figure 6.2. Consider that through the Naive approach each node has received the information about the services existent on other nodes and now that the service on node A wants to communicate with the service on node E. Thus service on node A is a consumer service and that on node E is provider service. Following steps describe about how the communication between services on A and E is facilitated and how B.A.T.M.AN routing module and the AllJoyn router plays the role in it. Similar analogy could be applied for communication between any two services on nodes, single-hop or multi-hops away from each other.

- The service on A creates a Proxy Bus Object to communicate with the provider service E.
- The call to the method on service E is encapsulated as a message by the AllJoyn router. All the messages that are exchanged between any two services during their communication with each other is termed as AllJoyn application data.
- From the AllJoyn router we direct the message to the debugfs file system i.e we write the message into the 'appData' file. Conventionally the message would be send by the router over underlying transport destined towards the provider service as a TCP / UDP flow. The AllJoyn router on the provider side would receive the incoming TCP / UDP flow and redirect

it to the provider service running on a certain port. Now here we want to facilitate the communication between services which reside on nodes which are multi-hops to each other. We leverage the routing tables built by the B.A.T.M.A.N routing to realize the communication between services. Since we need to use the B.A.T.M.A.N routing we write the data into the file 'appData' in the debugfs file system so that routing module can pick the data from there.

- In the implementation, while the AllJoyn router writes the application data to the 'appData' file, it writes the IP address, MAC address at which the provider service is existent as well. Also it writes the port on which it itself is connecting to the provider service, port at which the provider service is running, and the protocol on which the provider service is operating i.e it accepts TCP flows or UDP flows. In our case the AllJoyn router on A would write the IP address and the MAC address of the node E as the service on A wants to communicate with the service on node F. Along with that the port on which the service on A is connecting to the service on F, the port on which service A is running and the protocol (TCP / UDP) the service accept connections on. As discussed previously TCP is assigned as value 0 and UDP as 1.

- B.A.T.M.A.N routing module would detect the write operation by the AllJoyn router on the file 'appData' through the write callback registered by the module on the file 'appData'. The module would read the data from it and constitute a packet from it. The fields in the packet and the size of those fields are explained below :

  1. **Packet Type (Size - 1 byte) :** This corresponds to the packet type. We give a value of 5 in the implementation to the packets carrying AllJoyn application data.
  2. **Version (Size - 1 byte) :** This corresponds to the B.A.T.M.A.N protocol version. This has the value 4 in the implementation. All the OGM-AllJoyn packets as well the default OGM packets have the same value for this field.
  3. **destinationMAC (Size - 6 bytes) :** This corresponds to the MAC address of the wlan0 interface on node F. This information is written by the AllJoyn router on A when the service on A specifies to the AllJoyn router that it has to send the data to node F.
  4. **sourceIP (Size - 4 bytes) :** This corresponds to the IP address of the interface through which the packet would be send. In our case it is the IP address of the wlan0 interface on node A.
  5. **destinationIP (Size - 4 bytes) :** This corresponds to the IP address of the interface to which the AllJoyn application data has to be delivered. This corresponds to the IP address of the wlan0 interface on node F in our example. This information is written by the AllJoyn router on A in the 'appData' file. This destinationIP and destinationMAC basically tells that service on A wants to connect to the service on which IP address and MAC address as service on A can connect to service on B, C , D or E. In our case service on A wants to connect with the service on F and hence the router writes F's IP address and the MAC address into the file.
  6. **sourcePort (Size - 2 bytes) :** The port at which the service on A is connecting to the service on F. This port is different from the port at which the service on A itself is running. This is the port at which it is connecting to some other service.

7. **destinationPort (Size - 2 bytes) :** The port at which the service on F is running.
8. **protocol (Size - 1 byte) :** This corresponds to the protocol the service on F is accepting connnections on. This is also written by the AllJoyn router into the 'appData' file.
9. **data** Corresponds to the actual application data that has to be exchanged between two services. The maximum length possible for this field is 1500 - X where X is the sum of length all the previous fields which is equal to 21 bytes. The maximum length is 1500 - X as the maximum data that can be transmitted in one packet at the Ethernet layer is 1500 bytes.

We term the entire packet carrying the AllJoyn application data and all the above mentioned fields in the order as stated above as the BATMAN-IP packet. The length of the BATMAN-IP packet can be $\leq$ 1500 bytes depending upon the data in the packet. The packet is constituted by the B.A.T.M.A.N routing module when AllJoyn router writes data into the 'appData' file. The maximum length of the BATMAN-IP packet accompanying the Ethernet header is 1514 bytes. When the BATMAN-IP packet is sent by the module the protocol field in the Ethernet header is set to be ETH_P_BATMANIP, a defined value in the implementation. This protocol value is set for other packets (Default OGM packet, OGM-AllJoyn packet) as well both in the Default and the Naive approach. This ensures that B.A.T.M.A.N routing module in the nodes process B.A.T.M.A.N related packets only and hence a value ETH_P_BATMANIP is defined and the module would process only the packets with this value in the Ethernet header.

- Let the value of the fields in BATMAN-IP packet generated by routing module on A be as follows :

  1. packetType : 5
  2. Version : 4
  3. destinationMAC : fMAC. The MAC address of the interface wlan0 on node F.
  4. sourceIP : aIP. The IP address of the interface wlan0 on node A.
  5. destinationIP: fIP. The IP address of the interface wlan0 on node F.
  6. sourcePort : x1. The port on which the service on A is connecting to the service on F.
  7. destinationPort: x2. The port in which the service on F is running.
  8. protocol : 1. The service on F is accepting UDP flows/ connections.
  9. data : xyz. This could take any value based on data to be transmitted from the service.

  Now BATMAN-IP packet of certain size is generated of a certain size. This is appended at the Ethernet header of the 14 bytes, the values of fields in which are determined as follows :

  1. **Ethernet destination address :** This is set using the routing tables generated by the B.A.T.M.A.N routing module. Now that service on A wants to communicate with the service on F. The service A communicates with the AllJoyn router on A specifying that it wants to communicate with the service on F. The router allocates a port at which the service A is connecting to the service on F, the value of which in our example is x1 as stated in the details of the BATMAN-IP packet. The service would then communicate

the data to the AllJoyn router. AllJoyn router would write the data into the 'appData' file along with the IP address, MAC address, port and the protocol corresponding to the service F. When the B.A.T.M.A.N module reads the application data it reads the other fields as well which tells it where the data has to be sent. The IP address, port and the protocol values are filled in the BATMAN-IP packet accordingly. The MAC address obtained is then used to determine the route. In our case it is the F's MAC address and the routing module on A would look up for the best next hop towards destination node F in the routing table. The routing tables are built up using the originator message content as usual from the OGM-AllJoyn packet. Now consider that next best hop in the routing table at A for node F is B. The Ethernet destination address is then set to MAC address of the node B.

2. **Ethernet source address :** This corresponds to the MAC address of the interface wlan0 on node A as the packet would be sent through that interface.
3. **Protocol :** The value of this field is set to ETH_P_BATMAN.

Now since the Ethernet destination address in the packet is of node B, the BATMAN-IP packet generated by originator on node A would be sent to originator on node B.

- Originator on B would receive the packet and it would check the destination address. If it is not the MAC address of interface wlan0 on B, originator would discard the packet. If the packet is not discarded value of packetType field in the received packet is then inspected. Since we allocate a separate type (value = 5 in the implementation) for the application data packets, routing module in B would come to know that it is BATMAN-IP packet and the module would process the packet accordingly. The module would inspect the destinationIP field in the packet which in our case is 'fIP' which is the IP address of interface wlan0 on node F. Routing module in B would conclude that the received BATMAN-IP packet is not destined to the originator on B and that it has to be routed towards its destination. Now B would check the destinationMAC field in the BATMAN-IP packet. The destinationMAC field contains the value 'fMAC' and hence B would route the BATMAN-IP packet to the best next hop towards F. The idea behind including the destinationIP and the destinationMAC both in the BATMAN-IP packet is that the destinationIP lets a node determine that whether the packet is destined to it. And if the value in the destinationIP field does not matches with the receiving originator's IP address the destinationMAC field enables the node to route the packet towards its destination. Now the routing module at node B has determined from the B.A.T.M.A.N routing table that the next best hop towards originator on F is originator on E. The BATMAN-IP packet from A would be routed to E as it is with the change in the values in the Ethernet header only. Ethernet destination address for the BATMAN-IP packet would now be MAC address of the wlan0 interface on E and the Ethernet source address be the MAC address of the wlan0 interface on B.

- Routing module on E would receive the BATMAN-IP packet and do the similar processing as B. It would identify that the destinationIP value in the received packet is not the IP address of the interface wlan0 on E. It would determine the next best hop towards which the packet has to be routed. looking at the destinationMAC field it would identify that destinationMAC address

is F's MAC address. From its routing table it would determine the next best hop towards F and that would be F only. Thus the BATMAN-IP packet would be sent to originator on F with the Ethernet destination address in the packet be the MAC address of the wlan0 interface on F and the Ethernet source address be the MAC address of the wlan0 interface on E.

- Routing module on F would receive the BATMAN-IP packet and it would determine that the packet is destined to the originator wlan0 on F by checking the destinationIP address field in the BATMAN-IP packet. The routing module would write the application data from the BATMAN-IP packet into the 'appData' file. It would also write the values of the field sourcePort, destinationPort, sourceIP, destinationIP, protocol which in our case are x1, x2, aIP, fIP, 1 respectively. AllJoyn router on F would then read the data from the 'appData' file. It would check that whether there exists any service running on port x2 and accepting UDP flows (as 1 corresponds to UDP in the implementation). If it determines that there exists any service running on port x2 it would redirect the received data to the service running on port x2. The redirect mechanism is the same as that in the AllJoyn implementation where in the AllJoyn router receives the data and it is redirected to the service running on a certain port and accepting either TCP or UDP flows. The difference that comes is earlier the data was communicated from AllJoyn router to another AllJoyn router, while here the data is routed through the B.A.T.M.A.N routing module using the debugfs file system , generation of BATMAN-IP packets and the use of B.A.T.M.A.N routing tables. The values of the field sourcePort and the sourceIP are preserved by the AllJoyn router so that the return data from the service on F could be communicated. Now if the service on F wants to sent some data back to the service on A it would request the AllJoyn router specifying that it needs to send data back to A on the port (x1) at which it received data from A. AllJoyn router on F would write the data from the service F and the details such as IP address, MAC address, port and the protocol. This details correspond to the service at which the data is to be sent. If service on F wants to send data to the service on A at the port it which received data from the value of these fields would be aIP, aMAC, x1 and 0. Thus accordingly the BATMAN-IP packet would be generated by the B.A.T.M.A.N routing module at F with the field value as follows :

  1. packetType : 5
  2. Version : 4
  3. destinationMAC : fMAC. The MAC address of the interface wlan0 on node A.
  4. sourceIP : fIP. The IP address of the interface wlan0 on node F.
  5. destinationIP : aIP. The IP address of the interface wlan0 on node A.
  6. sourcePort : x2. The port on which the service on F is running.
  7. destinationPort : x1. The port on A at using which the service on A connected to the service on F.
  8. protocol : 1. Service F is accepting UDP flows and hence the reverse flow to service on A is also the UDP flow.
  9. data : xyz. This could take any value based on data to be transmitted.

- The BATMAN-IP packet from F destined to A would be routed through the intermediate nodes as the packet from A to F was routed explained by the entire mechanism above. If the service on F wants to connect the service on A with service on F acting as the consumer service and A as the provider service, service F would request the AllJoyn router specifying that it wants to connect to the service on A. The router would then allocate a port through which the service on F could connect to the service on A. Let the advertised port on which the service A is running be y1 and that router on F allocated port y2 to the service on F to connect to service on A. The sourcePort value would be then y2 and the destinationPort be y1 and the BATMAN-IP packet would have these values. This constitute a separate flow altogether with the service on F acting as the consumer service and that on A as the provider service. The mechanism illustrated above is for the service on A as the consumer and the service on F as the provider service. Similar mechanism would be followed for the communication between the service A and F where service on F is acting as the consumer service and that on A as the provider service.

6.6 Conclusion about the Naive Integration Mechanism and its drawbacks

In the Naive integration mechanism we facilitate multi-hop service advertisement. A new packet called OGM-AllJoyn packet of 182 bytes is generated every originator interval and it comprises of the regular originator message information, TVLV information and the AllJoyn service related information. The OGM-AllJoyn packet is broadcast and the rebroadcast according to the B.A.T.M.A.N specific forwarding rules. The originator message content and the TVLV information helps in the operation of the regular B.A.T.M.A.N routing, while the AllJoyn service information enables the discovery of the services existent on the single-hop or multi-hop neighbours. This service information is passed to the AllJoyn router through the debugfs virtual file system and the advertisement from the router is received as well through the debugfs file. This is facilitated by read and write operations by AllJoyn router and the B.A.T.M.A.N routing module into the 'serviceInfo' file that exists at the following path "/sys/kernel/debug/batman_adv/bat0" in the implementation. Besides the multi-hop service advertisement communication mechanism between services which may be existent on nodes which are single-hop or multi-hop away from each other is designed and developed. Modifications are performed in the AllJoyn implementation and the B.A.T.M.A.N routing module to route the application data from the AllJoyn service using the B.A.T.M.A.N routing module.

Drawbacks : In the Naive approach the service information is appended at the end of originator message content and the TVLV information. As we consider that the there exists a single service and that the service implements an interface which contains a single method. The size of this information is 112 bytes in the implementation as discussed in the previous section. Clearly the broadcast of the service information with every originator message content and the rebroadcast of the same at every node would result in large amount of data being circulated in the network. Every node would be rebroadcasting the OGM-AllJoyn packet from every other data. The rebroadcast of the originator message and TVLV information is required for the B.A.T.M.A.N operation and hence that could not be avoided. However the service information in the OGM-AllJoyn packet is the advertisement data about the service and it itself would consume a lot of bandwidth with its broadcast at every originator

interval and rebroadcast by the other nodes. With the increase in the number of nodes in the network the size of data would increase flowing in the network rapidly. Also with the increase in the number of services and the existence of more methods in the interface service implements, size of AllJoyn service related information flowing across the network would grow enormously. As discussed before if there are X services on a device and that the interface the service implements has Y properties, the total size required for conveying the information about the services would grow up to X * (56 + Y * 56) as 56 bytes are taken in the implementation to convey basic service information and 56 bytes for conveying the details of every property in the interface service implements. This would result in a lot of bandwidth being consumed for service advertisement itself. Clearly, mechanisms are needed to optimize the service advertisement mechanism so as to conserve the bandwidth being consumed in the relay of service advertisement over multiple hops. The following two approaches : Improved approach and Ask / Reply approach are enhancements over the Naive approach with the goal to convey the service information to multiple hops as well conserving the bandwidth consumed in the advertisement. Certain implementation decisions have been made from the source code of the Default approach.

6.7 Improved Approach

We use the source code of the B.A.T.M.A.N implementation available at the following source [39]. The data communication mechanism between the AllJoyn services over multiple hops used in the Naive approach is used as it is the Improved approach as well. Thus the mechanism for communication between services existent on nodes multiple hops away, generation and processing of BATMAN-IP packet, read / write operation to the 'appData' file remain the same in the Improved approach. The difference lies in the service advertisement mechanism in the Improved approach. In the naive approach service information is transmitted every originator interval. One way to optimize the advertisement mechanism is to send the service information every 'x' originator intervals where x can take value ≥ 2. Now the question that arises is what value to choose for x. Larger value of x may result into the delay in the service advertisement and that nodes would come to know of the service existent on others in a larger time duration. However it would limit the amount of service advertisement related information flowing in the network. Smaller value would reduce the time in which the nodes comes to know of the services existent on other nodes at the cost of larger information flow related to service advertisement in the network. We take the value of x = 2 in the Improved approach. This is taken based on the inferences drawn from the running the B.A.T.M.A.N routing over a real network, compiling the source code obtained from [39] and enabling the B.A.T.M.A.N routing on the nodes. It is explained through a network of nodes shown in Figure 6.5 with B.A.T.M.A.N routing operational on them.

Consider Fig 6.5 with B.A.T.M.A.N routing operational on wlan0 interface on the nodes A-D assuming that wlan0 interface is existent on all of them. Now when the routing module is operational, originator on each node would send originator messages content every originator interval. The originator message content referred here is the one that exists in the Default OGM packet, of size 66 bytes in the Default approach. Now originator on A would receive originator message content from originator on node B. Originator on node A would not conclude the originator on B as the next best hop towards B just on the receival of one Default OGM packet from the originator

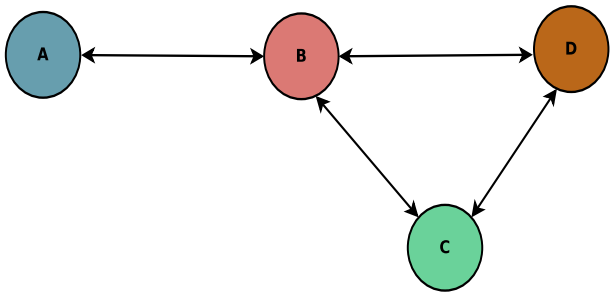


Figure 6.5: A network of nodes with B.A.T.M.A.N routing operational on them.

on node B. In the implementation of the default approach, on receival of the Default OGM packet from B, calculations are performed using the number of Default OGM packets received from B of B, transmission quality value in the received packet from B of B and the number of Default OGM packets that were send by A are rebroadcast by B. Transmission quality is a field in the Default OGM packet, the value of which is changed at every hop by the node receiving it before rebroadcasting the packet. Based on the calculations on the mentioned parameters a value is obtained and if that value falls above a certain threshold only then B is concluded as next hop neighbour towards B. From the real-time operation by running the B.A.T.M.A.N routing on a group of nodes, it was observed that the computed value would fall above the threshold value if atleast 2 Default OGM packets be received by B. For the transmission of the two Default OGM packets B would take 2 originator intervals. Originator on A thus would take 2 or more than 2 originator intervals (in case the OGMs from originator on B are lost) to conclude the originator on B as the next best hop neighbour towards originator on B although the two are single-hop neighbours. It takes more than 2 originator intervals for an originator to conclude a single-hop neighbour to be the best next hop towards itself. Now originator on A takes more than 2 originator interval to conclude B as the next best hop towards B, now even if it receives the service advertisement in one originator interval from B about the service existent on B it would be of no use. After one originator interval service on A received the information of the service on B and it wants to communicate with it. However there would be no route available at A to communicate the data to service on B as it would take more than 2 originator intervals for the determination of next best next hop towards B to be determined.

Hence in the Improved approach we append the service information to the originator message content every alternate originator intervals. Thus the service information is not appended when the node sends first originator message content. It is appended with second originator message content that is being sent by the node and from there on in an alternating fashion. The above observation can be concluded for multi-hop neighbours as well. Consider the Figure 6.5, where A and C are multiple hops away from each other. Originator on A would receive Default OGM packets from originator on C via originator on B. Originator on A would not conclude B as the next best hop neighbour towards C just on the receipt of one Default OGM packet of C from B. Calculations are performed considering the number of Default OGM packets from C received via B, the transmission quality value in the Default OGM packet received from B of C, and the number of rebroadcasts from B of the Default OGM packets send by A. The value so obtained is compared with the threshold value and if the value falls above the threshold only then originator on B is concluded as the next best hop towards C. For the value to fall above the threshold it takes atleast two Default OGM packets

Ethernet Header	14 bytes
Originator message	28 bytes
TVLV information	28 bytes

Table 6.5: Size of different segments in the OGM packet.

be received from B of C. For the transmission of 2 Default OGM packets C would take atleast 2 originator intervals. Originator on A would then take 2 or more than 2 originator intervals (if Default OGM packets from originator C are lost on path C-B or B-A) to conclude the originator on B as the next best hop towards originator on C. Now according to the Naive approach C transmitted the service information appended in the originator message content after the first originator interval, forming an OGM-AllJoyn packet. The OGM-AllJoyn packet being rebroadcast by B would reach A. Although A receives service information about the service on C via B, the service on A cannot communicate with the service on C. As originator on A would conclude B as its next best hop neighbour towards C only when two or more originator messages from C are received via B. The transmission of 2 Default OGM packets from C would take atleast 2 originator intervals and after 2 or more than 2 originator intervals only A would have a next hop towards C. It is only then A could route the data towards C through the chosen best next hop (in this case B).

Based on the above inferences in the Improved approach (Version III), every originator appends the service information every twice the originator interval. As mentioned previously the originator message content in the Default approach is of 24 bytes, combined with TVLV information and the Ethernet header, forms a Default OGM packet of size 66 bytes. Now here the service information is appended every twice the originator interval, thus OGM-AllJoyn packet is generated every twice the originator interval. Now as discussed in the Naive approach, the actual originator message content in the OGM-AllJoyn packet is of 28 bytes due to the addition of 4 bytes of serviceLen field into the originator message content in the Default OGM packet. Now the originator interval at which the service information is not appended the serviceLen field has the value 0. In essence, the serviceLen field of 4 bytes is needed, which on having value 0 concludes that there is no service information appended in the end. This results into a new packet called as OGM packet of size 70 bytes. The details of the segments in the OGM packets and their sizes are shown in table 6.5. OGM packet has no AllJoyn service information appended and the serviceLen in the originator message content has value 0.

Thus in the Improved approach (Version III) an originator would send OGM packet and OGM-AllJoyn packet in alternating intervals. OGM packet is send in the first originator interval after the B.A.T.M.A.N routing is enabled, and OGM-AllJoyn packet in the second originator interval and so on. The receiving nodes would rebroadcast the OGM packets and the OGM-AllJoyn packets as according to the B.A.T.M.A.N rebroadcast rules only.

6.8 Ask / Reply Approach

Ask / Reply approach is an enhancement over Naive and Improved approach. One thing that can be observed in the Naive Approach and the Improved approach is that the service information is rebroadcast by intermediate nodes along with the rebroadcast of the originator message content and the TVLV information. The rebroadcast of the service information may not be necessary all the time as most of the service information that is received about the service on a particular node is

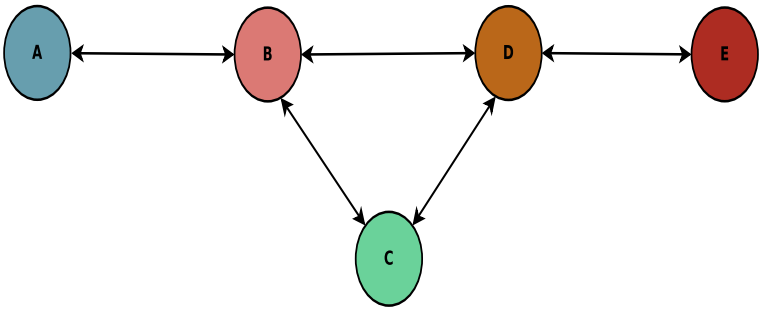


Figure 6.6: A network of 5 nodes with B.A.T.M.A.N routing operational on them.

constant and is there existent with the node. Hence it need not to be rebroadcast by the receiving node. Taking this as the fundamental idea, we design a new approach for the service advertisement using the B.A.T.M.A.N routing termed as Ask / Reply approach (Version IV).

We use the source code of the B.A.T.M.AN implementation available at the following source [39]. Communication mechanism between services that are existent on node that are single hop or multiple hops away is the same as that in the Naive approach. Enhancements are performed for service advertisement mechanism. Below we present the service advertisement mechanism in the Ask / Reply approach. Again, the assumption is that B.A.T.M.A.N routing is enabled on a single interface, wlan0 on a node. The approach would work with the routing enabled on multiple interfaces but for the ease of explanation, we consider that it is enabled on a single interface. Thus every node has one originator.

- Every originator on a node would generate the OGM packet in the first originator interval after the B.A.T.M.A.N routing is enabled and OGM-AllJoyn packet in the second originator interval. This is the same as in the Improved approach.

- Nodes receiving the OGM-AllJoyn packet from an originator on some other node will come to know about the service existent on that node through the service information in the OGM-AllJoyn packet. When it first time receives the service information about the service on a certain originator, the service information is added in the originator table corresponding to the entry for the originator. The OGM -AllJoyn packet is rebroadcast by the receiving originator to other.

- When a node further receives the OGM-AllJoyn packet from a certain originator, the node obtains the service information from the packet. The service information is then compared with the existent information in the originator table corresponding to the entry of that originator. If a change in the service information is found, only then the OGM-AllJoyn packet is rebroadcast as it is to other originators. If there is no change in the service information obtained in the OGM-AllJoyn packet compared to the existing service information in the originator table, then the service information from the OGM-AllJoyn packet is removed. Thus the 112 bytes of the service information is removed from the end of the packet and accordingly the serviceLen field is made to 0. This results in the conversion of OGM-AllJoyn packet to OGM packet by the intermediate node. The OGM packet so formed is rebroadcast to others as the originator message content and the TVLV information in it is needed for the operation of B.A.T.M.A.N routing.

The above mechanism is explained through the topology shown in Figure 6.6. The nodes are annotated as A-E and the B.A.T.M.A.N routing is operational on the interface wlan0 on them.

- Each node from A - E would send the OGM packet and the OGM-AllJoyn packet in alternating originator intervals starting from OGM packet. Consider two nodes A and D.

- Node D would come to know of the service existent on node A when it first receives the OGM-AllJoyn packet from A, either via B or C. Through the service information in the OGM-AllJoyn packet D would come to know of the service existent on A. Since it did not had any service information about A, the received information is new information for D. The service information is added in the entry corresponding to the originator on A in the originator table at D. The OGM-AllJoyn packet is rebroadcast by the node D and E would receive the OGM-AllJoyn packet.

- Now later when D received the OGM-AllJoyn packet from A via B or C, D would check the existence of the service information in the received OGM-AllJoyn packet. If there is some new service information obtained only then the OGM-Alljoyn packet is rebroadcast as it is. If the received service information about the service on A is already existent in the originator table at D, D would truncate the service information in the OGM-AllJoyn packet from A and convert it into OGM packet and rebroadcast it.

- Similar mechanism would be followed by other nodes for the OGM-AllJoyn packet they receive. In the above example B and C would also rebroadcast the OGM-AllJoyn packet from A when they observe a change in the service information they have for the service on A. Accordingly D would receive OGM packet or OGM-AllJoyn packet.

Thus in the Ask / Reply approach every twice the originator interval OGM-AllJoyn packet is sent by a node to its neighbours. If the neighbours detect a change in the received service information only then OGM-AllJoyn packet is rebroadcast further as it is. If the neighbours do not detect a change in the service information the service information is truncated from the OGM-AllJoyn packet and it is rebroadcast as the OGM packet.

This would work fine until the topology is the same or there is no packet loss. Following two conditions arise when the nodes may not come to of the service on other nodes due to truncation of the service information in OGM-Alljoyn packet.

- **Case I :** Consider the Figure 6.6. When node B first received the OGM-AllJoyn packet from A it got a change in the service information as the service information about the service on A was received for the first time. Node B would rebroadcast as it is to nodes D and C but consider that the rebroadcast OGM-AllJoyn packet is lost due to interference or packet drop in the wireless medium. Nodes D and C would thus never come to know of the service on A as because when node B would receive the next OGM-AllJoyn packet from A, it would not detect the change in the received information assuming that the same service information is send by A. B would truncate the service information in the received OGM-AllJoyn packet from A and rebroadcast it as OGM packet. If the first OGM-AllJoyn packet sent from node A to node B is lost in the link then node B also would not come to know of the service on node A. However since node B is a single-hop neighbour, B would come to know of the service on node A when

next time node A transmits the OGM-AllJoyn packet. Thus in the Ask / Reply approach a node sends out OGM-AllJoyn packet every twice the originator interval to its neighbours. If the neighbours detect a change in the service information in the OGM-AllJoyn packet it is rebroadcast further, else not. This would work fine and every node would come to know of the service on other node unless there is no packet loss. However if there is packet loss of OGM-AllJoyn packet then not all the nodes may know of the service on others because here a node would rebroadcast the received OGM-AllJoyn packet as it is only when the information it received in the OGM-AllJoyn packet is new to the node or a change in the existing information at the node. Figure 6.7 illustrates the same problem through the OGM-AllJoyn packet and OGM packet between the nodes A, B, C and D. Note that for the sake of explanation, it is shown that A emits OGM-AllJoyn packet and OGM-packet at alternating intervals, however every node would be doing so every alternating originator intervals. The loss of OGM-AllJoyn packet results into non reachability of service information to nodes C and D.

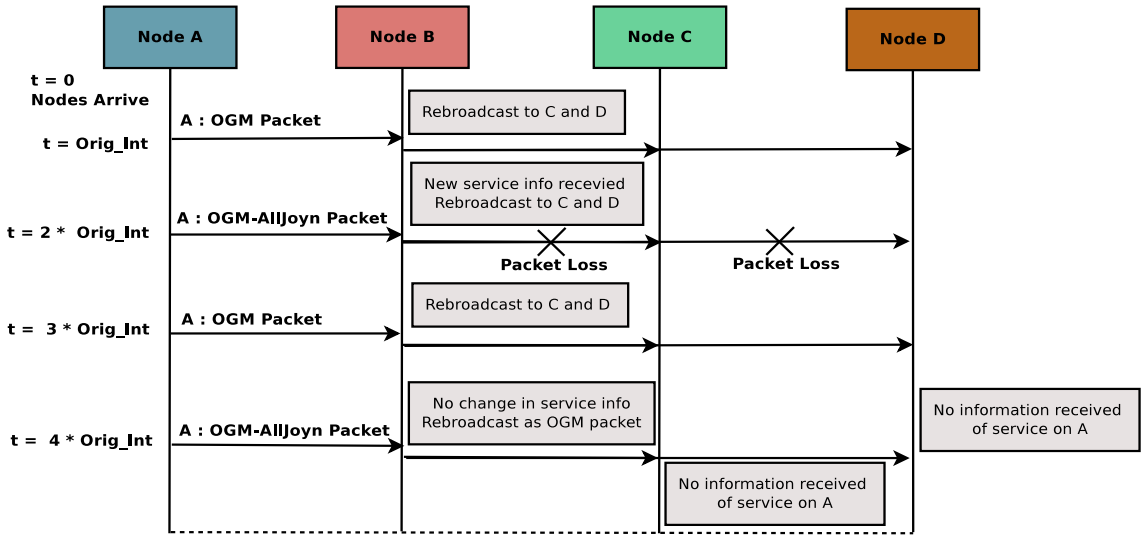


Figure 6.7: OGM-AllJoyn packet and OGM packet exchange between nodes A, B, C and D.

- **Case II :** Another case in which the nodes may not come to know of the service on other nodes is due to the arrival of new nodes. Consider the topology shown in Figure 6.6 and due to the arrival of the two nodes F and G the topology changes to the topology shown in Figure 6.8.

  Now nodes F and G would send the OGM-AllJoyn packets every twice the originator interval. When B first received the OGM-AllJoyn packet from A it would be receiving the service information from F for the first time. The OGM-AllJoyn packet of F would propagate from B to other nodes and every one would rebroadcast it as they would be receiving the service information about the service on F for the first time. All nodes would come to know of the service on F provided there is no loss of OGM-AllJoyn packet from F. If there is a loss of OGM-AllJoyn packet the nodes may not come to know of the service on F. Likewise the OGM-AllJoyn packet from G would propagate through F into the entire network. F would come to know of the service on B as B would be sending OGM-AllJoyn packet every twice the originator interval. F would be receiving the service information about the service on B for the first time, F would rebroadcast and G would also come to know of the service on B.

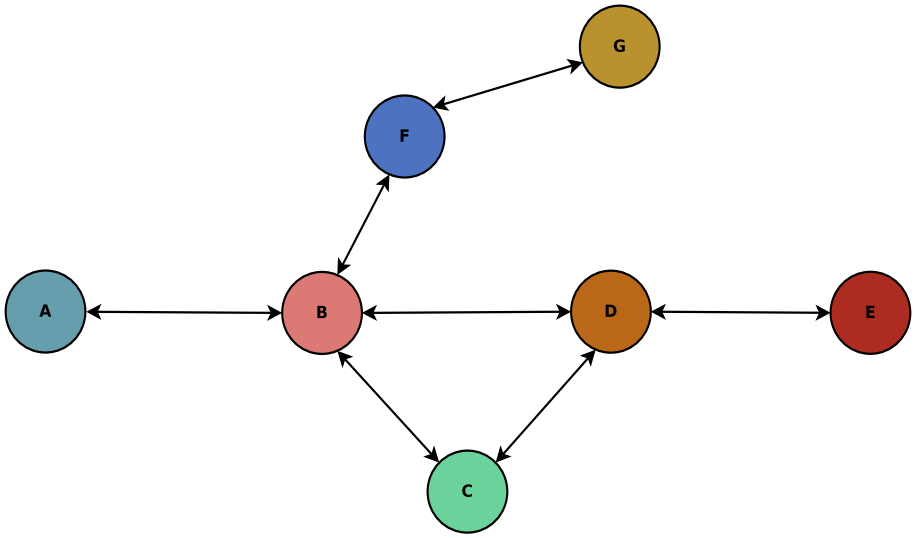


Figure 6.8: A network of 7 nodes with B.A.T.M.A.N routing operational on them. Arrival of nodes F and G in the topology shown in the Figure 6.6 results in the formation of this network.

However F and G would not come to know of the service on A, C, D and E. This is because by the time F and G arrived in the network other nodes were known of the service on each other and hence only OGM-AllJoyn packets were converted to OGM packets. Take the example of OGM-AllJoyn packet from D. It would be received by node B. Since by the time F and G arrived in the network B already had the service information about the service on D. The OGM-AllJoyn packet from D were not changing the existent service information about the service on D at the originator table in B. Thus B would not rebroadcast the OGM-AllJoyn packet as it is but it will convert it into the OGM packet and rebroadcast it. Thus the nodes F and G would not never come to know of the service on D. Extending the same analogy for the OGM-AllJoyn packets from other nodes which would be converted to OGM packets by its one hop neighbours, nodes F and G would not come to know of the service on other nodes. Figure 6.9 represent this problem considering node A, B and F into account. Due to late arrival of the node F in the network F would not receive service information from A as B would not be rebroadcasting as it may not observe a change in the service information it receives from A. However F would know of service on B as B would be sending its OGM-AllJoyn packet every twice originator interval and hence F would come to know of it. The timeline for B is not presented in the picture as we want to convey that A would come to know of the service on F but F would not come to know of the service on A. The other combinations that might arise can be concluded from the mentioned example and those would be the similar cases only as the case shown in Figure 6.9 corresponding to the nodes that arrive later into the network.

In order to remedy the problem generated in the above two cases where in the nodes may not come to know of the service on other nodes, we introduce two new packets called as Ask / Reply Packets. The fields in the Ask packet and the sizes of those are as mentioned below :

1. **packetType (Size - 1 byte) :** Packet type to determine that the packet is an Ask packet. This has been given the value 6 in the implementation.

2. **version (Size - 1 byte) :** Version corresponding to the B.A.T.M.A.N protocol version.

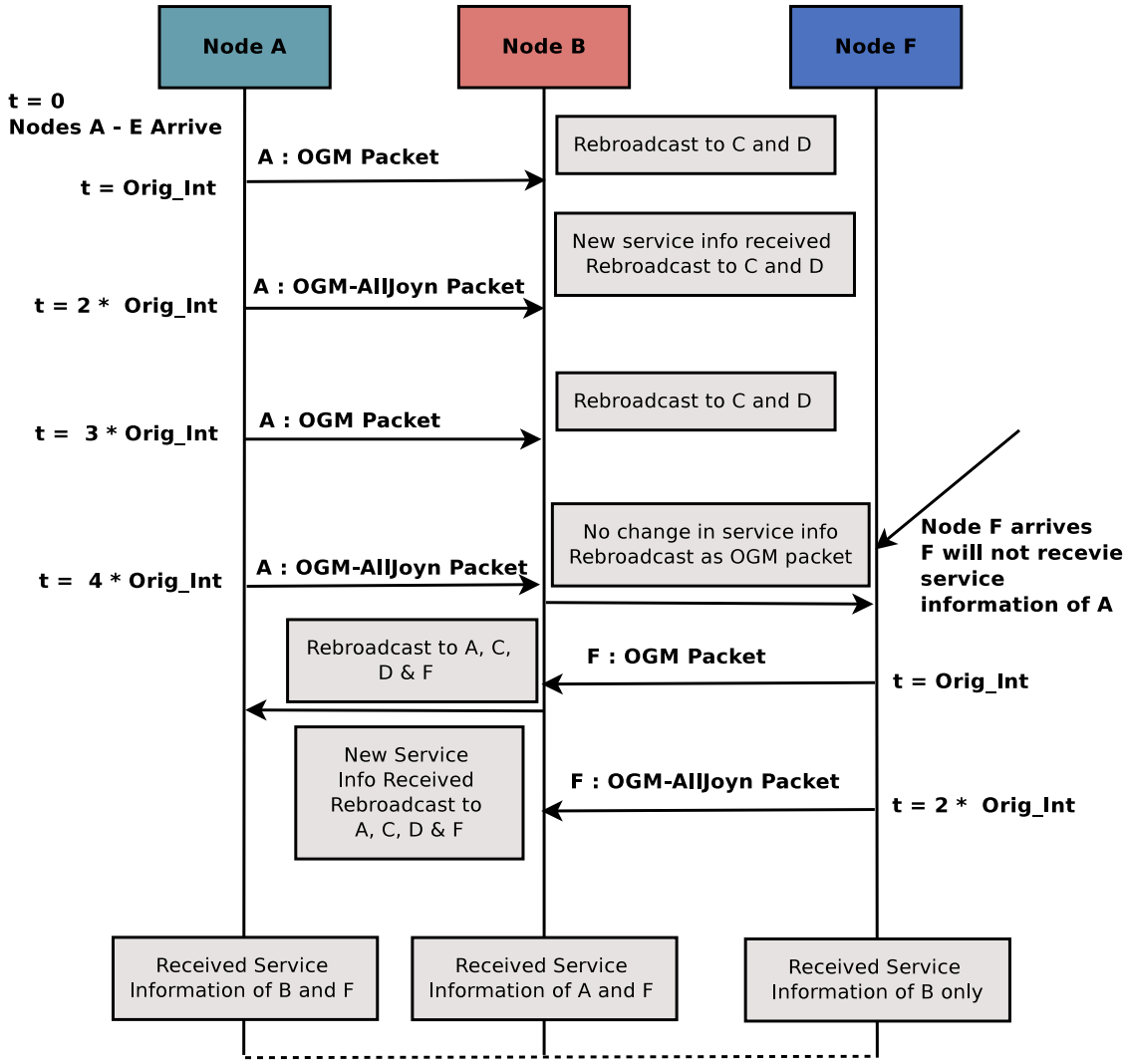


Figure 6.9: OGM-AllJoyn packet and OGM packet exchange between nodes A, B and F.

This has the value 4 in the implementable same as that in Default, Naive and the Improved approach.

3. **origMACAddress (Size - 6 bytes) :** MAC address corresponding to the originator of which the service information has to be obtained. This field is used by a node to ask its neighbours about the service information corresponding to the MAC address specified in this field. Since B.A.T.M.A.N implementation that we use operates at the MAC layer and in the originator message that are generated by the node the identity of the actual generator of the packet is through MAC address. Similarly the originator tables maintained at the nodes have key value as the MAC address for every entry that is maintained in the originator table.

Ask Packet is generated by a node (say X) when it wants to know of the service information on a other node (say Y). It comes to know of the existence of the Y through the OGM packets corresponding to Y i.e originator address field in the packet corresponding to MAC address of originator on Y. However X does not have the service information corresponding to service on Y due to truncation and conversion of the OGM-AllJoyn packet emitted by Y into OGM packet by

its single-hop neighbours or other intermediate nodes in the path(s) from Y to X. Ask packet is broadcast by a node to its neighbours. The neighbours would receive the Ask packet and check if they have the service information corresponding to the MAC address in the Ask packet by looking at its originator table. Accordingly the Reply packet would be generated appending the service information corresponding to the node of which the service information is asked. The fields in the Reply packet and the sizes of those are as mentioned below :

1. **packetType (Size - 1 byte) :** Packet type to determine that the packet is a Reply packet. This has been given the value 7 in the implementation.

2. **version (Size - 1 byte)** : Version corresponding to the B.A.T.M.A.N protocol version.

3. **origMACAddress (Size - 6 bytes) :** MAC address corresponding to which the service details are provided if it exists with the node generating the Reply packet.

4. **serviceLen (Size - 4 bytes) :** The field represent the number of bytes of service information (comprising of basic service details and the property details of the interface, service implements) that is appended in the Reply packet.

5. **TTL / Time To Live (Size - 1 byte) :** The number of hops the packet can reach with the value decremented by 1 at every hop. The packet is dropped at the hop where the value becomes 0.

   The size of Ask packet combining the Ethernet header in the implementation comes up to 22 bytes. The size of Reply packet can vary depending upon whether it contains the service information or not. If it does not contain the service information the size of all the fields described for the Reply packet comes upto 16 bytes in the implementation due to structure padding. This combined with the Ethernet header totals upto 30 bytes. Thus the size of Reply packet which does not have service information is 30 bytes with serviceLen field in it set to 0. If the node has the service information corresponding to the origMACAddress in the Ask packet, it would include the service information in the Reply packet. This service information comprising the basic service details and the property details of the interface, the service implements totals up to 112 bytes. The fields in basic service details and the property details are the same as explained previously. Also the same assumption is taken that there exists a single service on nodes and the interface the service implements have a single property. Thus if the Reply packet contains the service information its size comes up to be 128 bytes which combined with Ethernet header of 14 bytes forms a packet of size 142 bytes. Eventually, Reply packets can be of size 30 bytes or 142 bytes depending upon whether they contain the service information or not.

Now considering the two cases mentioned above where in service information is not reached to the nodes, below we present how Ask / Reply packet could resolve the problem for the cases. Other conditions that might arise could be resolved through the Ask / Reply packet mechanism. This two cases serve to be the generic cases in which service information is not received.

Ask / Reply Packet Mechanism to remedy the problem in cases I and II

- **Case I :** Consider Figure 6.6. Originator on A is sending the OGM-AllJoyn packet at every twice the originator interval. Now B on receival of first OGM-AllJoyn packet from A would obtain the service information from A for the first time. B would update the service information in the originator table as well rebroadcast the OGM-AllJoyn packet. The rebroadcast from B is lost and in that case C and D would not receive the OGM-AllJoyn packet. C and D thus would not know the service information corresponding to service on A. This is because B would not rebroadcast further OGM-AllJoyn packet from A as it would not observe the change in service information from A provided that the same information is received from A as the one that exists in the originator table at B. However B would be sending OGM packets to C and D. C and D this would notice the presence of originator on A and they will have the entry corresponding to the originator on A in their originator table. C would generate the Ask packet under following conditions:

  1. It has transmitted its one OGM packet and one OGM-AllJoyn packet. This represents that C has its 2 originator intervals passed.
  2. It has received 2 or more OGM packets of A from either of its neighbours. This concludes that A has its 2 originator intervals passed but C has not received service information from A. Ideally if there was no packet loss of the OGM-AllJoyn packet of B on link B-C , C would have received OGM-AllJoyn packet from A and hence the service information of the service on A in 2 originator intervals. This conditions just checks that A has its 2 originator intervals passed but still C has not received the service information from A. Same condition are used by node D and other nodes before generation of Ask Packets.

  C would then generate Ask packet with the origMACAddress field in the Ask packet corresponding to the MAC address of node A. The Ask packet would be broadcast to its neighbours by C. B would receive the Ask packet and check the origMACAddress field in the packet which is A's MAC address. It would check its originator table whether it has the service information in the entry corresponding to A's MAC address. Since B has the service information about service on A, B would generate a reply packet of size 142 bytes that containing the service information about service on A and would unicast it to node C. C would thus have the service information of service on A. Since it is obtained by C for the first time, it would rebroadcast the Reply packet so that its neighbours can know of the service on A. This would continue until either TTL expires or the node receiving the information already has the service information about A. Thus through the Ask / Reply packet the service information can be propagated to the nodes who did not have the service information about an other node but was receiving the OGM packet generated by them. Ask / Reply packet is a one time overhead compared to the rebroadcast of the service information with every OGM-AllJoyn packet in Naive approach and the Improved approach. Note that the Ask packet generated by C asking about the service information on A would also be received by D as it is broadcast. However D does not have the service information about service on A, D would generate a Reply Packet of size 30 bytes with no service information and the serviceLen field set to 0. The generation of Ask packet by D asking about service information on A would follow the same mechanism as that by C. Based on this the generation of Ask and Reply Packet by other nodes can be interpreted in case they do not know of the service on a certain node from which they are receiving OGM packet.

- Consider Figure 6.6 and Figure 6.8 where Figure 6.6 corresponds to a topology that existed before, which changes to topology in Figure 6.8 with the arrival of nodes F and G. Now service information from F would reach other nodes via B when it would receive the OGM-AllJoyn packet from F. When B would receive the service information from F for the first time about the services on F it would rebroadcast the OGM-AllJoyn packet from F. The service information about the services on F is propagated to other nodes provided that the OGM-AllJoyn packet from F is not lost. In case it is lost the node which does not receive the service information of services on F but receives the OGM packet of F would generate Ask packet according to the mechanism in the Case I. Now nodes F and G would receive the service information of the services on B as B would be sending OGM-AllJoyn packet every twice the originator interval. Thus F would receive the service information of service on B which would also be propagated to G. However F and G would not receive the service information about services on A, C, D and E. Since B would not propagate the OGM-AllJoyn packets from A, C and D as B would not see a change in the received service information from A, C and D and it would truncate the respective OGM-AllJoyn packets to OGM packets. The non receival of service information of A, C and D at F arises due to the late arrival of node F in the network. And the time at which arrives B has the service information of services on A, C and D and it would not rebroadcast the OGM-AllJoyn packets from them unless it observes a changes in the received service information in those. Similarly D would truncate the OGM-AllJoyn packet from E to OGM packet. Thus F would generate Ask packets asking about service information of services on A, C, D and E. These Ask packets would be broadcast to its neighbours B and G. G any ways does not have service information and it would send Reply packets of size 30 bytes with no service information appended in it and the serviceLen field value be 0. B would receive different Ask packets with origMACAddress value A, C, D and E. B would look into its originator table and obtain the service information corresponding to MAC address of originators on A, C, D and E. It would generate separate Reply packets of size 142 bytes for each Ask packet it received including the service information of services at A, C, D and E. B would send those Reply packets as unicast packet to node F. F would thus come to know of the services at A, C, D and E. As F would come to know of the services at A, C, D and E for the first time, the service information received is a change in the service information that existed at F and hence F would rebroadcast the Reply packets to G and B. B already has those service information and hence B would drop these packets. G however would receive the service information for first time and it would update its originator table and rebroadcast them again. F would not rebroadcast it again as it already has those information. Thus through Ask / Reply packets a node could Ask its neighbours about the services existent on nodes of which it received the OGM packets but not the OGM-AllJoyn packets. Reply packets corresponding to the origMACAddress in the Ask packet would provide the details of service existent on that node. Reply packet would be rebroadcast if a receiving node sees a change in the service information so that it neighbours knows of the change in the service information. Note that in above case G would also be generating Ask packets which would be received by F. F would provide service information in Reply packets if it has the service information corresponding to the origMACAddress in the Ask packets. If it does not have service information corresponding to origMACAddress in the Ask packets it would send Reply

packets with no service information and serviceLen field value set to 0. The mechanism for generation of Ask / Reply packets and rebroadcast of Reply packets for other cases that might arise can be interpreted from the explained mechanism.

The mechanism for advertisement of the service information from the AllJoyn router by the B.A.T.M.A.N module and that of conveying the AllJoyn router of the received advertisement by the B.A.T.M.A.N routing module is the same in Improved and Ask / Reply approach as in the case of Naive approach through read / write operations in the 'serviceInfo' file in debugfs file system. Similarly the data communication between the service existent on nodes multi-hop or single-hop are same in Improved and Ask / Reply approach as in Naive approach.

6.9 Purging of service information

A node would purge the service information of services on other nodes with the purging of the entry corresponding to originator in the originator table. This in Naive approach would happen when a node does receive any OGM-AllJoyn packet of a know originator within PURGE_TIMEOUT interval. In case of Improved approach and Ask / Reply approach this would occur when a node does not receive OGM packet or OGM-AllJoyn packet of a known originator within PURGE_TIMEOUT interval. The PURGE_TIMEOUT value is kept as 10 * WINDOW_SIZE * Orig_Int where WINDOW_SIZE corresponds to sliding window and Orig_Int represents the Originator Interval.

6.10 Generic Terminologies

Below we present certain generic terminologies drawn from the Naive, Improved and Ask / Reply integration approaches for B.A.T.M.A.N - AllJoyn integration. This terminologies would be used further in the thesis in testing and the result chapters.

1. **Originator Interval (Orig_Int) :** The periodic interval in which the originator message content are send by a node is referred to as Originator Interval. The originator message content in the Default OGM packet is of 24 bytes, that in OGM packet and OGM-AllJoyn packet is of 28 bytes.

2. **AllJoyn Service Interval (AS_Int) :**The periodic interval in which the AllJoyn service information is send by a node is referred to as the AllJoyn Service Interval. In the Naive approach, the AllJoyn service information is transmitted every Orig_Int and hence AS_Int = Orig_Int for naive approach. In the Improved and Ask / Reply approach the AllJoyn service information is transmitted every alternating Orig_Int and hence AS_Int = 2 * Orig_Int for Improved and Ask / Reply approach.

3. **Control data :** All the information exchange between the nodes for B.A.T.M.A.N routing and AllJoyn service advertisement is termed as Control data. This information is required for B.A.T.M.A.N operation for computation of next best hop neighbour as well the service advertisement between AllJoyn routers. In this context the originator message content, TVLV information, AllJoyn service information, information in Ask / Reply packets is termed as Control data.

4. **Control packet :** All the packets carrying the Control data are termed as Control packets. Thus Default OGM packet, OGM packet, OGM-AllJoyn packet, Ask packet and Reply Packet are all termed as Control packets. The traffic generated from the transmission of control packets is termed as Control traffic / Control overhead.

5. **Application Data :** All the information exchange between AllJoyn services running on nodes single-hop or multi-hops away from each other is termed as Application data.

6. **Data Packet :** The packets carrying application data are termed as Data packets. Thus the BATMAN-IP packets which carry the data corresponding to the flow between services in all the approaches are Data packets. The traffic generated from the transmission of BATMAN-IP packets is termed as Data traffic / Data overhead. Tables 6.6 and 6.7 presents the control and data packets in each of the approaches. The Default approach is the normal B.A.T.M.A.N routing module generated from the compilation of B.A.T.M.A.N source code [39]. It does not have AllJoyn integration and thus no application data flows exist in it and hence no BATMAN-IP packets. Naive, Improved and Ask / Reply approach each have their separate source code and the corresponding kernel modules are generated from the compilation of those source codes. During the testing process, modules corresponding to different approaches are loaded into the kernel at different times, with one same type of module existing in the kernel across all devices at a time.

Approach	**Control Packet**	**Size**
Default approach (Version I)	Default OGM packet	66 bytes
Naive approach (Version II)	OGM-AllJoyn packet	182 bytes
Improved approach (Version III)	OGM packet OGM-AllJoyn packet	70 bytes 182 bytes
Ask / Reply approach (Version IV)	OGM packet OGM-AllJoyn packet Ask packet Reply packet	70 bytes 182 bytes 22 bytes 30 / 142 bytes

Table 6.6: Control packets in each approach

Approach	**Data Packet**	**Size**
Default approach (Version I)	-	-
Naive approach (Version II)	BATMAN-IP packet	$\leq$ 1514 bytes
Improved approach (Version III)	BATMAN-IP packet	$\leq$ 1514 bytes
Ask / Reply approach (Version IV)	BATMAN-IP packet	$\leq$ 1514 bytes

Table 6.7: Data packets in each approach

6.11 Conclusion

The chapter discusses about the realization of the multi-hop service advertisement in the AllJoyn framework using the B.A.T.M.A.N routing protocol. Details of how the B.A.T.M.A.N kernel module is loaded and enabled on the nodes to use the B.A.T.M.A.N routing, are provided. The traditional serviced advertisement mechanism in the AllJoyn framework is described further. The source code of

the B.A.T.M.A.N kernel module and the AllJoyn framework are obtained from the following sources [39] and [25] respectively.

Chapter 7

Testing of Integration approaches

7.1 Introduction

This chapter discusses about the real-time testing of the developed integration mechanism for the AllJoyn framework and B.A.T.M.A.N routing protocol. Various topologies on which the testing has been performed have been illustrated. Device configurations of the devices in the real-time experimental set up have been highlighted along with the details of the wireless network scenario in which the experiments have been performed. Logging mechanism for logging the packets and the events occurring in the network have been described at the end of the chapter. It involves incorporation of modules in **batctl** (a tool for B.A.T.M.A.N configuration and debugging) for capturing and logging packets over all the B.A.T.M.A.N enabled interfaces on the devices in the test-bed.

7.2 Topology Description

Integration mechanism with various approaches is tested over real-time test-bed with following network topologies and configurations:

- **Topology I :** 3 Node Topology
- **Topology II :** 5 Node Topology
- **Topology III :** 7 Node Topology

Figures 7.1, 7.2, 7.3 present the Topology I, II and III respectively. Nodes have been annotated with symbols A - G and the MAC address in the figures represent the MAC address of the external Wi-Fi adapter attached to the devices in the real-time test-bed.

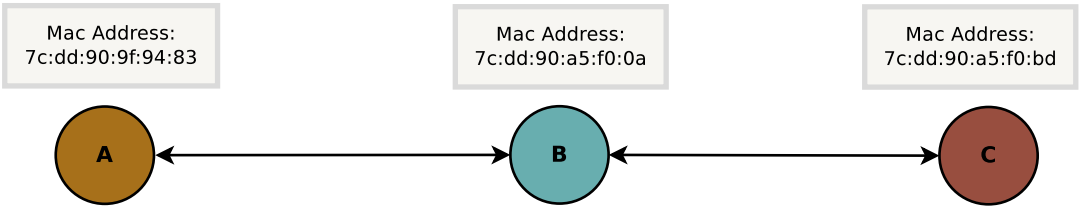


Figure 7.1: Wireless Ad hoc network set up between 3 nodes.

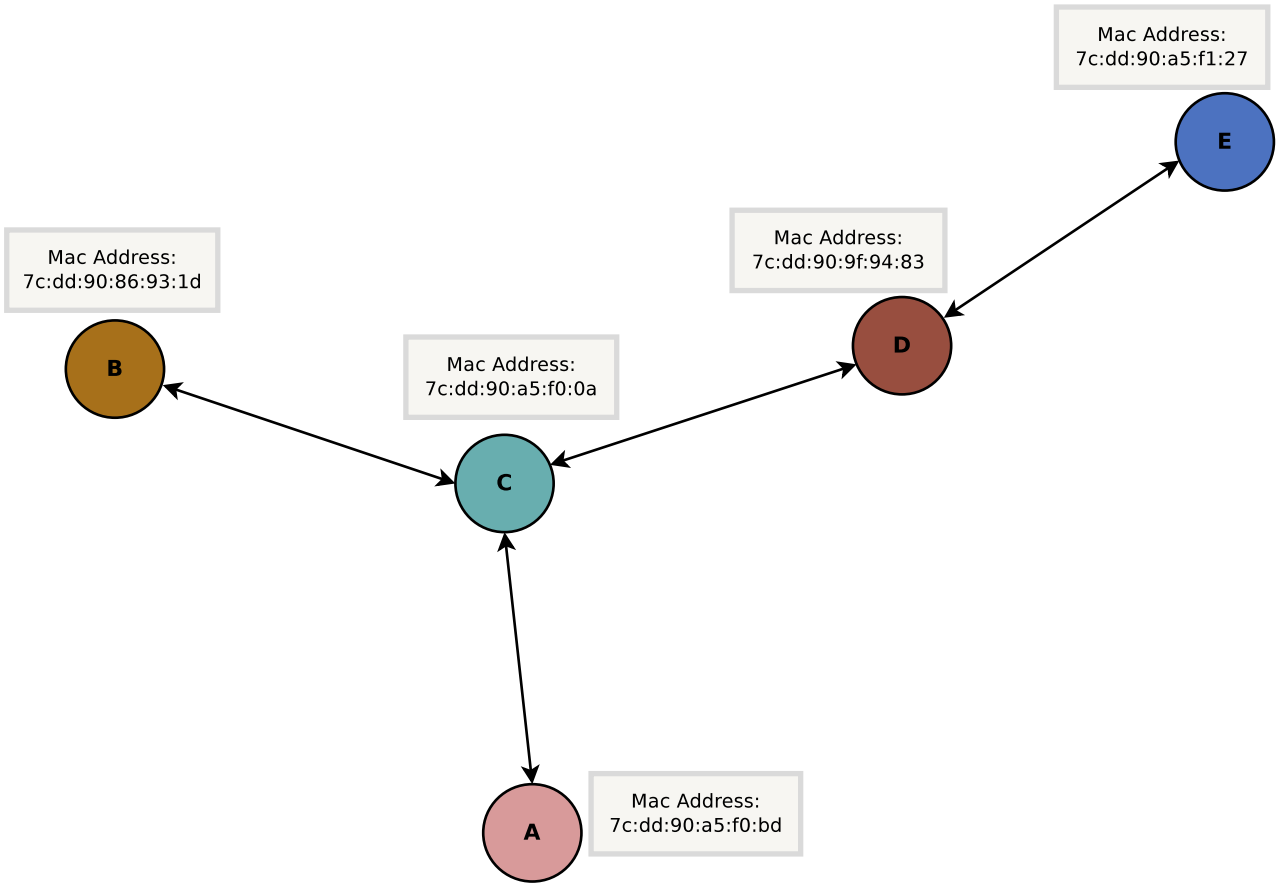


Figure 7.2: Wireless Ad hoc network set up between 5 nodes.

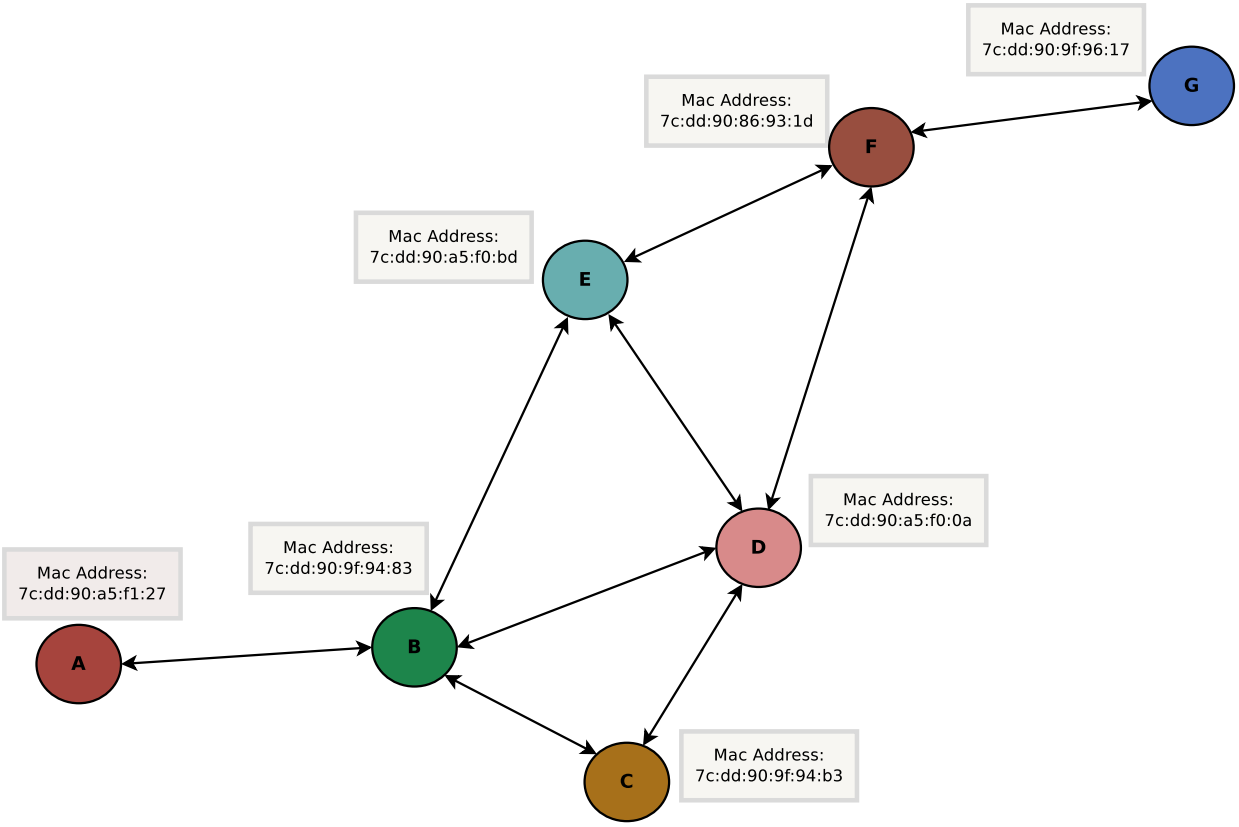


Figure 7.3: Wireless Ad hoc network set up between 7 nodes.

	Raspberry PI	Linux Machine I	Linux Machine II
OS	Raspbian OS	Ubuntu 14.04	Ubuntu 15.10
CPU Architecture	ARM Cortex A7	Intel i5	Intel i5
RAM	1 GB	4 GB	2 GB
No. of Cores	Quad Core	Quad Core	Dual Core
CPU Freq.	900 MHz	2.70 GHz	2.50 GHz

Table 7.1: Hardware and Software specifications of the devices used in the real-time test-bed.

Wi-Fi Standard	IEEE 802.11n
Wi-Fi Mode	Ad Hoc
Wi-Fi Channel	6
IP Subnet	10.10.20.0 / 24
Tx Power	20 dBm
Frequency	2.437 GHz

Table 7.2: Wireless network configuration .

7.3 Device Configurations

Nodes shown in the Topology I, II and III annotated as A - G in the real-time test-bed are one of the following:

- Linux Machine (Desktop / Laptop)
- Raspberry PI

Table 7.1 lists the details of the hardware and the software specifications of these devices.

7.4 Wireless Network Configurations

Table 7.2 describes the wireless network configuration of the real-time test-bed of Topology I, II and III. We attach external Wi-Fi adapter to all the devices in the experimental set up and allocate static IPs to the external Wi-Fi interface from the subnet shown in table 7.2. B.A.T.M.A.N routing is enabled on the attached external Wi-Fi interface. An ad hoc Wi-Fi network is set up in each of the topologies I, II and III. Channel 6 was chosen in order to have the effect of interference from the regular Wi-Fi on the ad hoc network been set up in the topologies. It was observed through the Wi-Fi analyser tool that there were other Wi-Fi access points operating on channel 6 at certain locations in the physical area over which the topology was spread. Consideration of the interference accounts for more realistic scenario. In the practical application scenarios of AllJoyn, Wi-Fi interface available on a node can be configured to allocate IP address to itself through the link local addressing mechanism [42]. Along with these those Wi-Fi interfaces needs to be operating on ad hoc mode in order for the multi-hop communication between AllJoyn applications. For the embedded devices like Raspberry PI and the laptops this could be realized easily through the operation of external or inbuilt Wi-Fi module in ad hoc mode. However for mobile phones using Android and iOS, this might not be possible as they do not support operating of the Wi-Fi interface in ad hoc mode. With the support for ad hoc mode on mobile phones in future, AllJoyn applications on phones could communicate over multi-hop provided that B.A.T.M.A.N routing module is present in the kernel.

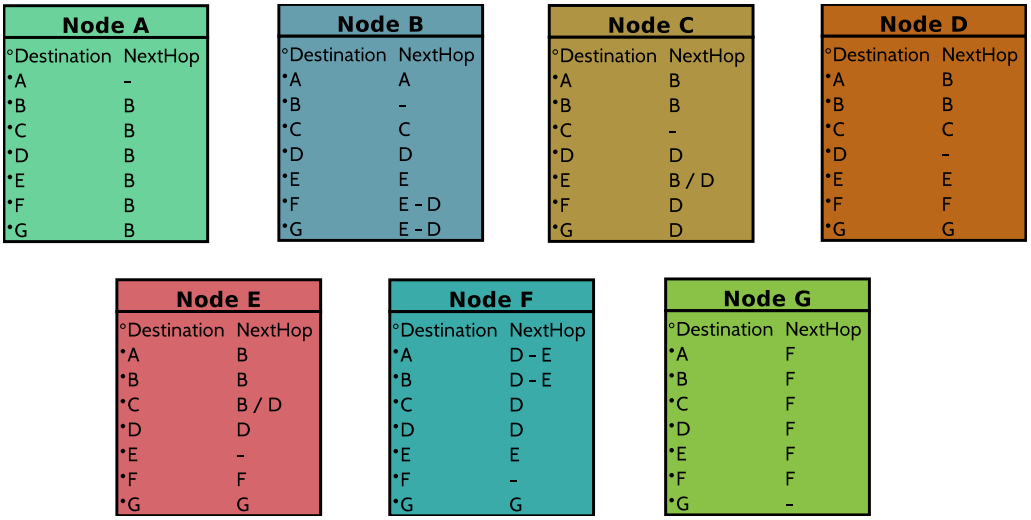


Figure 7.4: Routing Table at the nodes for Topology III - 7 nodes topology.

7.5 Routing Tables

Routing tables at the nodes for Topology I (7.1) and II (7.2) can be deduced from the topology itself as each node has one single-hop neighbour towards multi-hop neighbours in these topologies. Figure 7.4 presents the routing table for the Topology III - 7 nodes topology generated at the nodes in the real-time test-bed. These routing tables in Figure 7.4 are representative of the multiple routing tables gathered over periodic intervals on multiple iteration of B.A.T.M.A.N operation on Topology III. Multiple iteration (10 iterations) of the B.A.T.M.A.N operation on the real-time test-bed of 7 node topology were performed with similar physical locations of the nodes in the topology. Each iteration of the B.A.T.M.A.N operation involved letting the routing mechanism to run for a duration of 10 mins and the routing tables were logged at periodic intervals of 30 secs in each iteration. Based on those multiple routing tables, 3 types of entries are generated: single valued entries, entries of the form X / Y and X - Y as can be seen in Figure 7.4. Entries of the form X / Y are put where in 50% of routing tables, X and in 50% of the routing tables Y is observed as the next hop to a certain destination node. Entries of the form X - Y are concluded where in 75% of the routing tables, X is present and in 25% of the routing tables Y is present as the next hop. Presence of one value certain number of times and other value at other times might be due to change in the link quality due to interference from the regular Wi-Fi on those links and hence a update in the next best hop neighbour. Single valued entries in the table is concluded from the observation of that entry in 99% of the logged routing tables.

E.g. From the Figure 7.4 in the routing table of node C, next hop entry towards the destination node E is B / D. This means that in 50% of the routing tables node B and 50% of the routing tables node D is chosen as the next best single-hop towards node E. Likewise in the routing table of the node B next hop entry towards destination F is E - D. This represents that 75% of the times node E is chosen as the next best hop towards node F and 25% of the times node D is chosen as the next best hop towards node F.

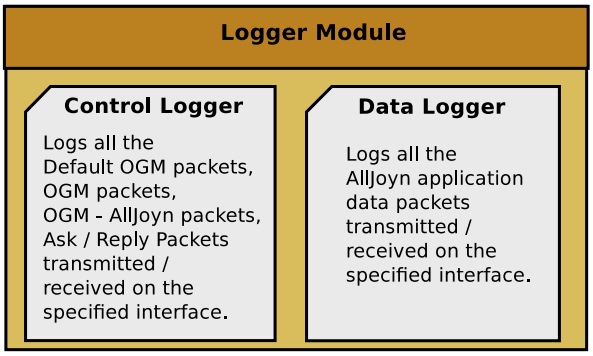


Figure 7.5: Logger Module incorporated into the batctl.

7.6 Logging Mechanism

Batctl is a tool for debugging and configuration of the operation of B.A.T.M.A.N routing module [43]. While using batctl for configuration one can add or remove interfaces using B.A.T.M.A.N routing or change parameters such as originator interval in the batman kernel module. Batctl can be used for debugging as well where it has utilities such as *tcpdump* to capture originator messages on a specified interface, retrieve logs information from the batman kernel module, etc. In order to realize the integration mechanism of B.A.T.M.A.N and AllJoyn new packets are generated / received at the nodes in the integration mechanisms. These packets have been described in the previous chapter covering the multi-hop realization. In order to capture these packets over an interface we incorporate a module called Logger module shown in Figure 7.5 into the batctl code obtained from the following source [39].

Control Logger shown in the Figure 7.5 has 2 parts, one for logging all the transmitted control packets (**controlSend logger**) and one for logging all the received control packets (**controlReceive logger**). Data logger similarly has 2 parts, one to log sent AllJoyn application data packets (**dataSend logger**) and the other to log received AllJoyn application data packets (**dataReceive logger**). Collectively control and data loggers are referred to as **loggers**. These loggers capture the packets on the specified interface provided as an input argument to them.
Since these loggers are integrated into batctl tool they can be used through a command line argument to the batctl tool. Accordingly, the format of the command to use controlSend logger in order to capture all the sent control packets on a particular interface is as follows: **"batctl controlSend <intfName>".** Similarly **batctl controlReceive <intfName>** would invoke contolReceive logger to capture all the received control packets on the specified interface. Likewise **"batctl dataReceive <intfName>"** and **"batctl dataSend <intfName>"** would log all the received and sent AllJoyn application data packets.

Data Collected in the log files : Loggers store all the packets in a file and each of the different loggers logs the packets in different files. If controlSend is specified as an argument to batctl tool, a file is created for storing the information of all the control packets sent on the specified interface. For each packet that is sent /received the timestamps at which it is sent through / received at the interface is logged along with the size of packets, size of different segments in the packet, parameters such as sequence numbers, relevant details based on the packet and the Ethernet source and destination of the packet. The timestamp in the log files is the time at which the packet is sent /

received at the MAC layer in the TCP / IP stack.

Capturing of packets and identification of control and data packets : Each of the loggers, viz. controlSend, controlReceive, dataSend, dataReceive are provided as a network interface name as an argument and it captures the packet on that interface. This interface name is the interface on which the B.A.T.M.A.N routing is operational. Capturing of the packets on that interface is done through obtaining the MAC address of the interface and then using the ioctl, bind and socket APIs in Linux. This is analogous to the operation of tcpdump. Now since controlSend logger has to monitor all the sent control packets on particular interface, it checks that the Ethernet source address in the packet been captured is the same as that of the MAC address of the specified interface. If it matches then the packet is logged into the file corresponding to that logger else otherwise discarded. Similarly controlReceive logger has to log only the received control packets, it checks whether the destination Ethernet address of the packet is the MAC address of the specified interface and it logs if the two matches. controlReceive logger would also log the packet if destination address is the broadcast address and source Ethernet address is not the MAC address of the specified interface. This corresponds to the packets being broadcast by other nodes and has to be logged by the controlReceive logger. If the destination address is the broadcast address and the source address is the MAC address of the specified interface, then the corresponding packet need not to be logged by controlReceive logger as it is the packet being broadcast by the specified interface. dataSend and dataReceive logger operate in a similar manner as the controlSend and controlReceive logger however they capture the sent and the received AllJoyn application data packets. Identification of the control and data packets is done through a packet type field in the packets. Different approaches for integration have different control packets and each different control packet have a unique packet type value assigned to it. A particular control packet which is there in one implementation and also exist in some other implementation have the same packet type as illustrated in previous chapter. Data packets have been assigned a separate type and are identified through that type.

Different Control Logger implementations: Corresponding to the different B.A.T.M.A.N AllJoyn integration approaches there are different control logger implementations incorporated into the batctl tool. This is because there would be different control packets generated in each of the approaches, the data packets generated though would be the same. Eventually there are 4 different batctl implementations with different control logger implementations in each of them, however data logger implementation is same in each of them. These batctl implementations are termed as batctl-I, batctl-II, batctl-III and batctl-IV. controlSend and controlReceive loggers in batctl-I would capture the sent and received default OGM packets respectively. Similarly controlSend and controlReceive loggers in batctl-II would capture OGM-AllJoyn packets, those in batctl-III would capture OGM packets, OGM-AllJoyn packets and that in batctl-IV would capture OGM packets, OGM-AllJoyn packets, Ask packets and Reply packets. batctl-I is used for logging when the batman module corresponding to the default B.A.T.M.A.N implementation is operating on the nodes. batctl-II, batctl-III and batctl-IV are used when the batman module corresponding to the Naive approach, Improved approach and Ask / Reply approach, respectively are operating on the nodes.

Naming convention of the log files generated through the loggers: In order to ease the

process of data collection from the different nodes and also through running multiple approaches of B.A.T.M.A.N - AllJoyn integration, we keep a syntax for the names of the log files getting generated. The syntax of the same is as follows: **<A-B-C-MacAddr >**. Here A corresponds to a number corresponding to the integration mechanism that is being run. The default batman routing is assigned as number 1. Naive approach for B.A.T.M.A.N AllJoyn integration is assigned as number 2, Improved approach as number 3 and Ask / Reply approach as number 4. B can take the following values : "controlSend", "controlReceive", "dataSend" and "dataReceive". C corresponds to the originator interval that B.A.T.M.A.N module is operating on and the MacAddr corresponds to the MAC address of the specified interface in the argument to the logger. Since we have 4 different batman modules, one corresponding to the default B.A.T.M.A.N routing and the other 3 corresponding to the B.A.T.M.A.N AllJoyn integration mechanisms. Accordingly there are 4 batctl implementations batctl I,II, III and IV. These 4 versions will generate the log files with prefix 1, 2, 3 and 4 so that A can take the values 1, 2, 3 and 4. **E.g.** Assume that module corresponding to the Improved approach for B.A.T.M.A.N AllJoyn integration with originator interval = 5 secs is compiled and loaded into the kernel module. Accordingly we compile the batctl III implementation that would generate log files with prefix '3' as improved approach has been assigned number 3. Now if we want to log the sent control packets on interface say wlan0 the appropriate command is issued for the same. It would generate the log file with the following name: 3-controlSend-5-aa:bb:cc:dd:ee:ff where aa:bb:cc:dd:ee:ff is the MAC address of the interface wlan0. Originator interval is obtained in the logger implementation through the kernel module when the logger is run and appended to generate the log file with the name according to the specified syntax. The naming syntax of the log files ensures uniqueness among control/data send/receive logs, across different originator intervals, different integration approaches and different nodes. This would ease the collection of generated log files from the nodes in the topology. These loggers are used in the experiments over the real-time test-bed and the statistics from the corresponding log files are extracted to compare the performance of different integration approaches on certain identified parameters. The next chapter details about the performance evaluation and comparison analysis of the integration approaches on the topologies and wireless network configuration specified in this chapter.

Chapter 8

Performance Evaluation and Results

8.1 Introduction

This chapter discusses about the performance evaluation of the integration mechanisms on the various topologies mentioned in the previous chapter. Parameters on which the performance comparison of the approaches is done are described initially. A comparative study through the various plots and graphs is presented thereafter in the order, starting from 3 node topology first, then for the 5 node topology and lastly for the 7 node topology.

8.2 Performance Metrics

8.2.1 Service Discovery Time

Consider any two nodes 'i' and 'j' in a particular network. Service discovery time for node i to know about a particular service, say 'x1' on node j is denoted as SD[i][j_{x1}]. SD[i][j_{x1}] is computed using the following equation:

$$\mathbf{SD[i][j_{x1}] = T_{iFj}[x1] \text{ - } maxArr(arrT_i, arrT_j)}, where \tag{8.1}$$

- T_{iFj}[x1] = Time at which node 'i' first comes to know about a service 'x1' on node 'j'.
- $arrT_i$ = Time at which nodes 'i' arrives in the network and $arrT_j$ = Time at which nodes 'j' arrives in the network.
- maxArr(t_a,t_b) = A function that returns maximum of the time value between t_a and t_b.

Based on the equation (8.1), **Service Discovery Time** for node 'i' to discover the service 'x1' on node 'j' is defined as the time at which node 'i' first comes to know of the service 'x1' on 'j' from the arrival time of the node which entered later into the network among node 'i' and 'j'. Similarly, **Service Discovery Time** for node 'j' to discover the service 'y1' on node 'i' is defined as the time at which node 'j' first comes to know of the service 'y1' on 'i' from the arrival time of the node which entered later into the network among node 'i' and 'j'. In our implementation a service on a node is available with the arrival of that particular node in the network.

Now, extending the equation (8.1) for multiple services, following equation can be generated,

$$\mathbf{SD[i][j] = (SD[i][j_{x1}] + SD[i][j_{x2}] \text{ ..... } + SD[i][j_{xn}]) \text{ / } n}\ , where \tag{8.2}$$

- **SD[i][j]** denotes the service discovery time for node 'i' to know about all the services on node 'j' and
- **x1, x2 ..... xn** are the n services available on node 'j'.

Similarly, SD[j][i] denotes the service discovery time for node 'j' to know about all the services on node 'i' and can be represented as:

$$\mathbf{SD[j][i] = (SD[j][i_{y1}] + SD[j][i_{y2}] \ldots\ldots + SD[j][i_{yn}]) \ / \ n} \ , where \tag{8.3}$$

- **y1, y2 ..... yn** are the n services available on node 'i'.

8.2.2 Network Service Discovery Time / Average Service Discovery Time

Based on the notations and the equations in the previous section we compute the network service discovery time or average service discovery time. Average service discovery time is denoted as $\mathbf{SD_N}$ and its value is determined through following steps :

- Let V denote the set representing the nodes in the network and W = V * V. Generate W from V.
- Compute SD[i][j] for every pair of nodes (i,j) ϵ W, where node i,j ϵ V. Clearly SD[i][i] = 0 i.e. service discovery time of node 'i' to know about its own services.
- Then, $\mathbf{SD_N} = \frac{\Sigma_{\forall(i,j)\ \epsilon\ \mathrm{W}}\ \boldsymbol{SD[i][j]}}{\boldsymbol{|W| - |V|}}$, where $|W|$ and $|V|$ denotes the size of set W and V respectively.

Example : Consider the 5 node topology shown in the Figure 7.2. Set V for the 5 node topology can be represented as V = {A,B,C,D,E}. Accordingly the set W can be calculated and would be the following W = { (A,A), (A,B), (A,C), (A,D), (A,E), (B,A), (B,B), (B,C), (B,D), (B,E), (C,A), (C,B), (C,C), (C,D), (C,E), (D,A), (D,B), (D,C), (D,D), (D,E), (E,A), (E,B), (E,C), (E,D), (E,E)}. Compute SD[i][j] for every pair (i,j) ϵ W through the equation 8.2 and sum of these values be **Z**. Since in the implementation we consider the existence of a single service on each node the value of n in the 8.2 and 8.3 reduces to 1 and the value of SD[i][j] and SD[j][i] can be straightly computed through $SD[i][j_{x1}]$ and $SD[j][i_{y1}]$ respectively. Consequently $\mathbf{SD_N = \frac{Z}{25 - 5}}$ as $|W| = 25$ and $|V|$ for the 5 node topology.

SD_N represents the average service discovery time of the network at a particular time with certain number of nodes. It may change with the addition of new nodes and removal of certain existing nodes from the network over time. On addition of new nodes set V would change and accordingly set W and hence SD_N would have a new value. Similarly on removal of nodes set V and W would change and SD_N would be computed using SD[i][j] values for existing (i,j) pairs in W only.

8.2.3 Control overhead

Control overhead is the amount of traffic generated from the control packets. Terms : Control packet, Control data and Control overhead have been explained previously in implementation details.

Different approaches would have different control overheads based on their functioning and the size of control packets send in each approach. Control overhead correspond to the data that is transmitted for the operation of B.A.T.M.A.N routing and AllJoyn service advertisement. In the Default approach control overhead would be just from the operation of B.A.T.M.A.N routing. In the Naive, Improved and Ask / Reply approach, control overhead would be from the operation of B.A.T.M.AN routing as well from the advertisement of AllJoyn service information. However in Ask / Reply approach control overhead would be generated form the Ask and Reply packet as well. Control information corresponding to B.A.T.M.A.N enables computation of next best hop and the routing tables at the nodes while control information corresponding to the AllJoyn service information lets nodes discover service existent on other nodes. Lesser the control overhead more beneficial it is,

8.2.4 Network Throughout

Network Throughput is the rate of successful message delivery over a communication channel. Here, Network Throughput is determined by the amount of control traffic generated during the transmission of certain application data. Lot of control traffic would have impact on the the data traffic. This is because the network has a certain capacity and if a lot of control information is generated it would result in drop of control packets. This at the same time would result in drop of application data packets. So the network throughput here is concerned with the amount of data traffic being transmitted. If there is low control traffic in the network, application data flows would not be affected and hence a higher network throughput can be obtained. On the other hand, if there exists a lot of control traffic, it would impact the flow of application data packets, resulting in packet drops and a less network throughput. Approach in which less control traffic is generated would yield higher network throughput. We define the network throughput as the amount of application data transmitted in the network per unit time (Kbps / Mbps).

8.3 Node Arrival Time

In the topologies in the experimental set up we provision the entering of nodes at different times i.e nodes arrive in the network at different instant of times. Arriving of a node in the network means that B.A.T.M.A.N routing module is there present in the kernel and that B.A.T.M.A.N routing is enabled on a particular network interface (in our case it is the Wi-Fi interface) on that node. It means that other existing nodes already had B.A.T.M.A.N routing enabled on them except in the case of the arrival of first node in the network. Arrival is also possible when a node had B.A.T.M.A.N routing enabled on it, but was not in the wireless range of other nodes which were operating using the B.A.T.M.A.N. The mobility of node would bring it into the wireless range of other B.A.T.M.A.N operating nodes and it would mark the arrival of the node in the network.

In the experiments, for the 3 node topology shown in Figure 7.1 the arrival time of the nodes is shown through the below timeline:

t = 0 • Node **B** and **C** arrives in the network

t = 2 * Orig_Int • Node **A** arrives in the network

For the 5 node topology shown in Figure 7.2, during the experiments nodes arrive in the order as shown through the below timeline.

t = 0	Node **A** and **C** arrives in the network
t = 2 * Orig_Int	Node **B** arrives in the network
t = 4 * Orig_Int	Node **D** and **E** arrives in the network

Similarly the arrival time of the nodes in the 7 node topology shown in the Figure 7.3 is represented through the following timeline:

t = 0	Node **E** and **D** arrives in the network
t = 2 * Orig_Int	Node **B** arrives in the network
t = 4 * Orig_Int	Node **C** arrives in the network
t = 6 * Orig_Int	Node **F** and **G** arrives in the network
t = 8 * Orig_Int	Node **A** arrives in the network

In our case the arrival of nodes is through the enabling of B.A.T.M.A.N routing on the Wi-Fi interface. The arrival time of the nodes in the network have been set differently in order for the generation of the Ask and the Reply packets in the Ask / Reply approach. This helps in determining the overhead due to the generation of these packets and at the same time would impact the service discovery time $\mathbf{SD_N}$. Since these order was kept for the generation of Ask and Reply packets in Ask / Reply approach, similar order was kept while testing the Naive approach and the Improved approach for a proper comparison. Control overheads and average service discovery time are calculated based on the above arrival order.

8.4 Performance Results

8.4.1 Average Service Discovery Time

Here we present the average service discovery time in different approaches at originator intervals of 1, 3 and 5 secs for Topology II and III. The arrival time of the nodes in the topologies is set as explained in the previous section. Based on the B.A.T.M.A.N approach that is being tested (Version I - IV explained previously) corresponding batctl implementations are run (batctl-I to batctl-IV). controlReceive logger at different devices is run to capture the control packets. Based on the receival time logged in the controlReceive logger at different nodes, value of SD[i][j] is determined for every (i,j) pair and finally the SD_N is calculated through mechanism explained in 8.2.2. 5 experiments are conducted with every approach at different originator intervals on all the topologies. SD_N values in different experiments for a certain approach and a particular originator interval is averaged and a final value is determined. Multiple experiments are conducted rather than just conducting single experiment so as to have a better representative value for discovery time for different approaches at different originator intervals. Clocks across all the devices were synced before conducting the experiments so that SD[i][j] can be accurately determined through the timestamps in the logs.

Figures 8.1 and 8.2 represent the SD_N values in different approaches for 5 node and 7 node topologies respectively at originator interval of 1, 3 and 5 secs. Clearly SD_N is observed to be higher for both Improved Approach and Ask / Reply approach in comparison to Naive approach for both 5 node and 7 node topologies as can be seen from Figures 8.1 and 8.2. A difference in SD_N values is observed between Improved and Ask / Reply approach due to generation of Ask / Reply packets. This is due to different arrival time of nodes and as explained in the Ask / Reply approach nodes

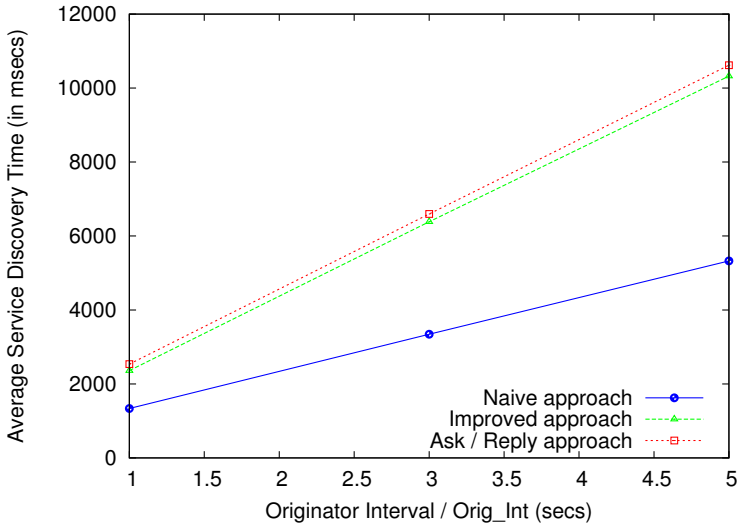


Figure 8.1: Average Service Discovery Time (SD_N) for 5 node topology at Originator interval of 1, 3 and 5 secs

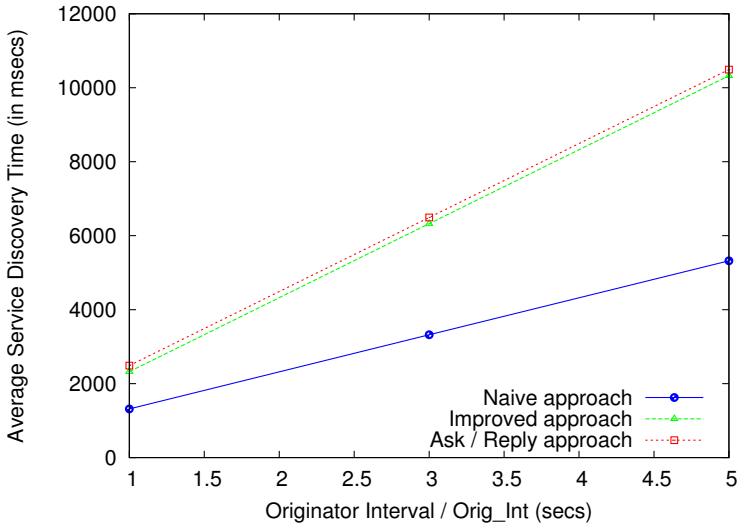


Figure 8.2: Average Service Discovery Time (SD_N) for 7 node topology at Originator interval of 1, 3 and 5 secs

arriving later are not known of the services existent on nodes that arrived before. This results in generation of Ask / Reply packets and hence comparatively higher discovery time is observed in Ask / Reply approach compared to Improved approach. With the increase in originator interval there is proportional increase in the average service discovery time is observed across all approaches. Besides these not much difference in SD_N values is observed between 5 node and 7 node topologies across different approaches. This can be attributed to the compact topology in case of 7 nodes as compared to a linear topology in case of 5 nodes.

8.4.2 Control overhead

To determine control overhead in different approaches, modules corresponding to the approaches : Default, Naive, Improved and Ask / Reply are loaded into the kernel at different times and B.A.T.M.A.N routing is made operational. At a time the kernel module corresponding to a certain approach is loaded across all the nodes. Five experiments are performed with originator interval of 1 sec, five with originator intervals of 3 secs and another five with originator intervals of 5 secs for Default approach on a particular topology and in each experiment B.A.T.M.A.N routing was made operational for a duration of 10 mins. controlSend logger at all the nodes is run to determine the control information being sent. Likewise, the same exercise is repeated for all the approaches, across all the originator intervals of 1, 3 and 5 secs and on every topology (Topology I - III). The arrival time of the nodes is kept as explained in the previous section. This is to have the generation of Ask and Reply packets in the Ask / Reply approach and hence to have its overhead for comparison. Due to late arrival of certain nodes B.A.T.M.A.N routing may not operate for a duration of 10 mins on those nodes. However the duration for which it would run on the late arriving nodes would be same across different approaches and hence comparison in control overhead can be done. Average of Control overhead in the 5 experiments for a certain approach, certain originator interval and on a certain topology is then determined.

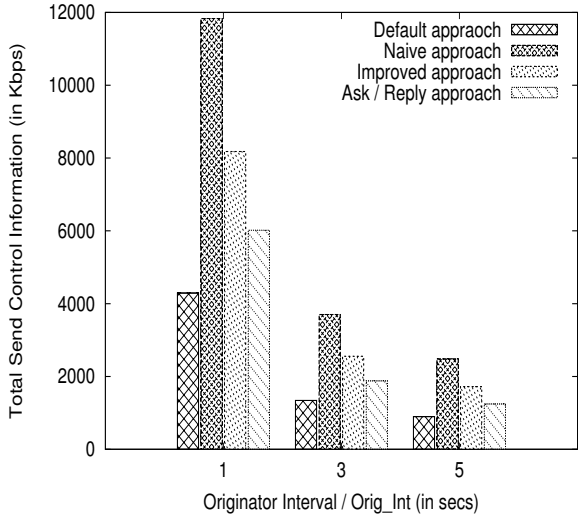
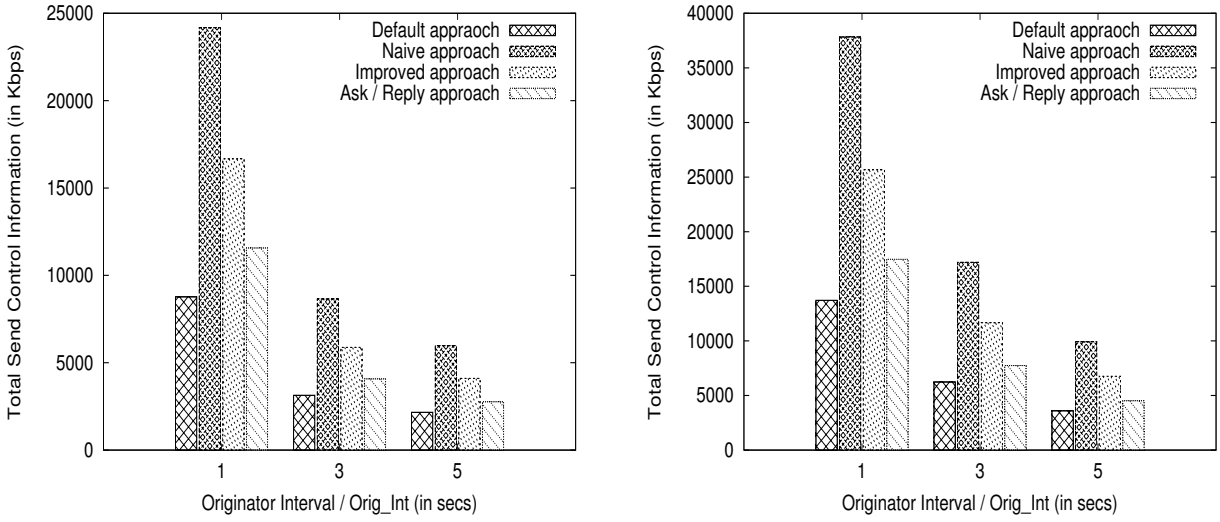


Figure 8.3: Control overhead comparison for 3 node topology.

Figures 8.3, 8.4 (a) and 8.4 (b) represent the control overhead across different approaches at originator intervals of 1, 3 and 5 secs for 3 node, 5 node and 7 node topology respectively. Control overhead in the figures for a certain approach, with certain originator interval and on a particular topology is the average of the control overhead in 5 experiments conducted corresponding to that approach, originator interval and the topology. Clearly there is a significant control overhead in the Naive approach which increases further with increase in number of nodes as can be seen form the Figures 8.4 (a) and 8.4 (b). Significant drop is observed in control overhead in the Improved and Ask / Reply approach. Hence they can be concluded as better integration approaches in comparison to the naive approach in which a lot of control traffic is generated for the AllJoyn service advertisement. However less control overhead in the Improved approach and Ask / Reply approach comes at the



(a) 5 node topology (b) 7 node topology

Figure 8.4: Control overhead comparison.

dispense of increases service discovery time and hence there exists a trade off between the two. Ask / Reply approach has the least control overhead as in it, nodes does not propagate service information through rebroadcast if there is no change observed in the received service information and the existing service information at a certain node.

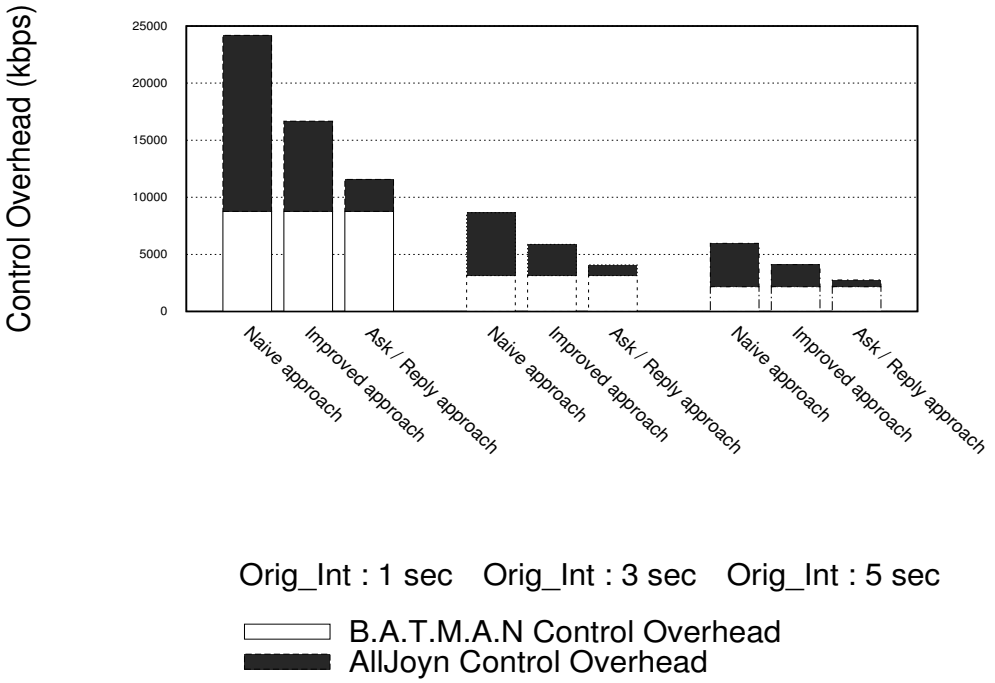


Figure 8.5: Control overhead comparison for 5 node topology at Originator interval of 1, 3 and 5 secs

Figures 8.5 and 8.6 present an alternative view of control overhead comparison. This is presented

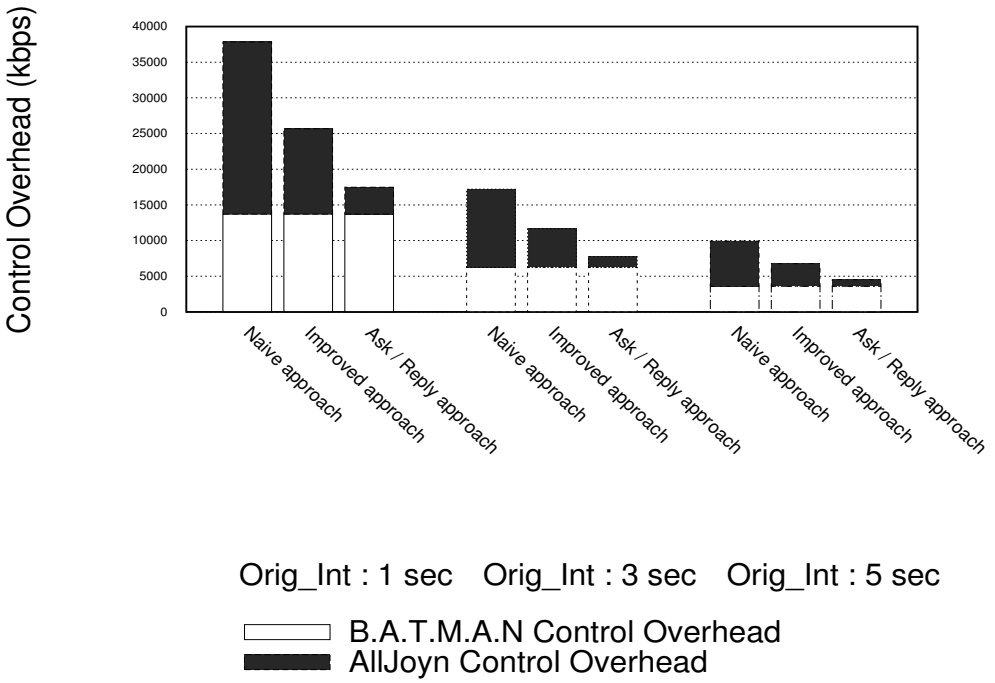


Figure 8.6: Control overhead comparison for 7 node topology at Originator interval of 1, 3 and 5 secs

only for 5 and 7 node topology as they involve larger number of nodes. Control overhead is segregated into the control overhead from the B.A.T.M.A.N routing which is termed as B.A.T.M.A.N control overhead and the control overhead from the advertisement of AllJoyn service information which is termed as AllJoyn control overhead. In Default approach there would be only B.A.T.M.A.N control overhead while other approaches would have both of them. In Naive approach AllJoyn control overhead is from the service information appended in every OGM-AllJoyn packet and 4 bytes of serviceLen field that is incorporated to account for the size of service information appended in the end of OGM-AllJoyn packet. B.A.T.M.A.N control overhead is from the originator message content of 24 bytes (excluding the 4 bytes of serviceLen field) and 28 bytes of TVLV information. In Improved approach, AllJoyn control overhead is from service information in the OGM-AllJoyn packet and serviceLen field of 4 bytes both in OGM packet and OGM-AllJoyn packet. B.A.T.M.A.N control overhead is from 24 bytes of originator message content and 28 bytes of TVLV information in both OGM and OGM-AllJoyn packet. AllJoyn control overhead and B.A.T.M.A.N control overhead in Ask / Reply approach is from the same information as that in Improved approach. Besides this, Ask and Reply packets are also included in AllJoyn control overhead in Ask / Reply approach. Clearly AllJoyn control overhead is least in Ask / Reply approach compared to Naive and Improved approach. Though it has overheads of Ask and Reply packets, but that is a one time overhead. Control traffic generated in Ask / Reply approach is comparatively very less in comparison to that generated in Naive and Improved approach.

8.4.3 Network Throughput

To determine network throughput, application data flows were created between the services on the nodes. For the 5 node topology, following 3 flows are created :

- C-A
- B-E
- D-B

Similarly for 7 node topology, following 3 flows are created :

- 1 E-C
- A-G
- F-A

X - Y denotes application data flow from service at X to service at Y. Flows are UDP flows each of size 10 MB. Likewise 5 experiments are conducted for 5 and 7 node topology, every Orig_Int (1, 3 and 5 secs) and using every approach. Average values are then determined. Figures 8.7 and 8.8 represent the network throughput for 5 and 7 node topologies respectively. Network throughput is the amount of application data transmitted and is expressed in Kbps / Mbps.Figures also show the corresponding control information being transmitted. Network throughput improvement of 4.37 % and 7.17% in Improved and Ask / Reply approach is observed, respectively in comparison to Naive approach for the 5 node topology at Orig_Int = 1 sec (Figure 8.7). Ask / Reply approach 7.17

Similarly, network throughput improvement of 5.15 % and 7.95% in Improved and Ask / Reply approach is observed, respectively in comparison to Naive approach for the 7 node topology 8.8 at Orig_Int = 1 sec. These improvements are due to less control traffic in Improved and Ask / Reply approach as compared to Naive approach. Less control traffic in Improved and Ask / Reply approach results in less drop of application data and hence a higher network throughput value. Improvements are slightly higher in case of 7 node topology as the difference in control overhead across different approaches is slightly higher in 7 node topology than the difference in control overhead across approaches in 5 node topology. Network throughput improvements at Orig_Int = 3 and 5 secs can be similarly deduced from the Figure 8.7 and 8.8. However here the network is not heavily loaded and not notable improvements are observed. Significant higher improvements in network throughput can be obtained when the network is heavily loaded with large number of flows.

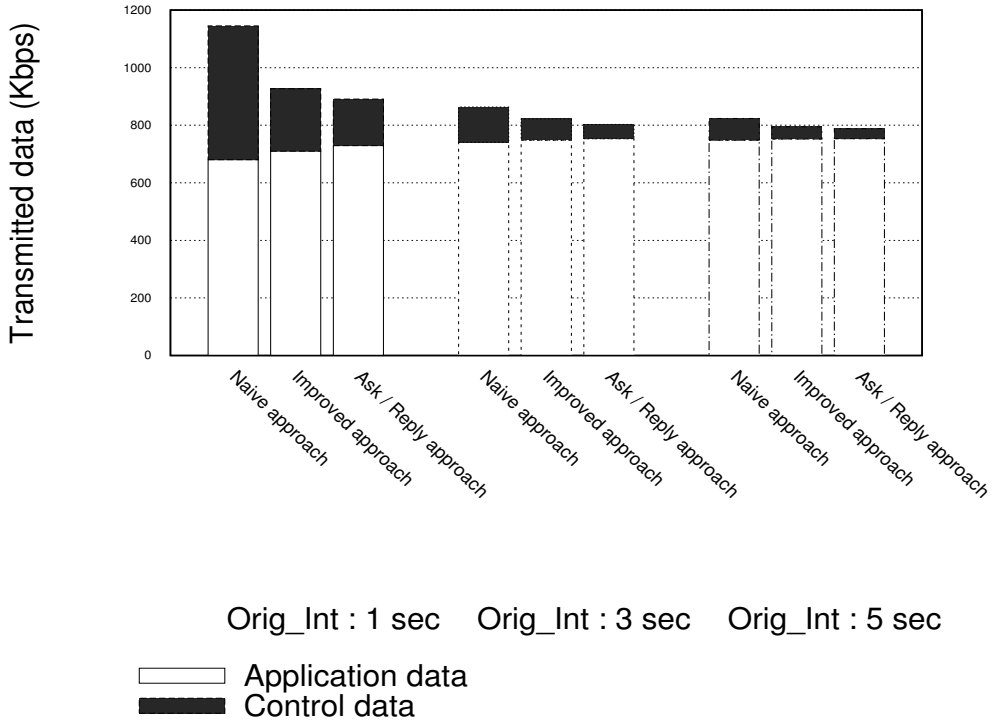


Figure 8.7: Network Throughput : Effect of control overhead in different approaches on transmission of application data for 5 node topology

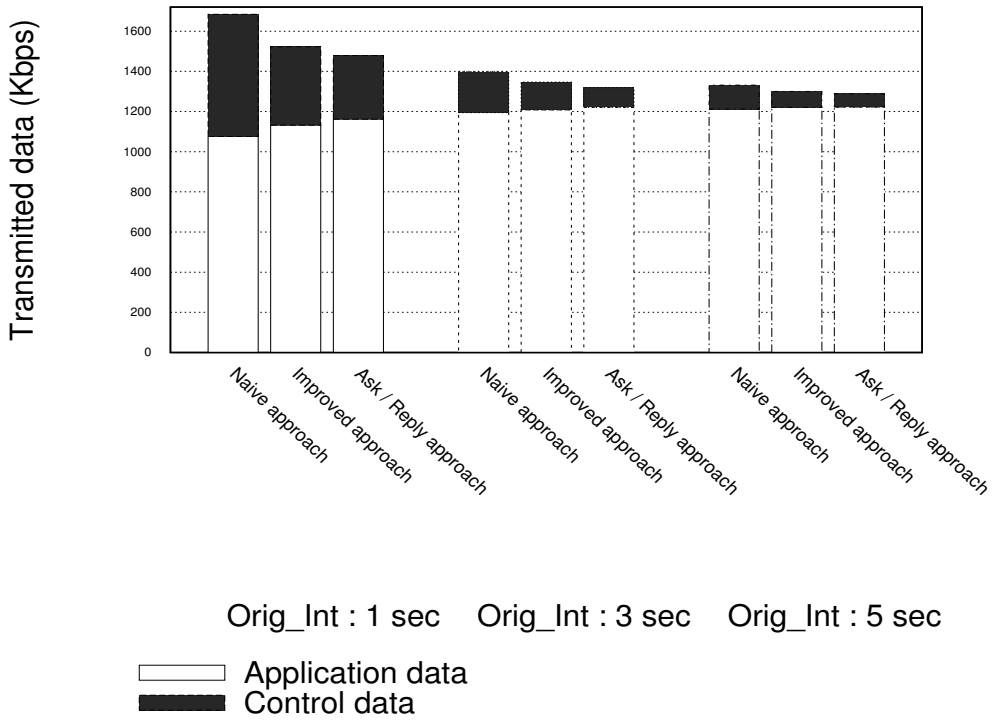


Figure 8.8: Network Throughput : Effect of control overhead in different approaches on transmission of application data for 7 node topology

Chapter 9

Conclusions and Future Work

9.1 Conclusions :

The thesis presents AllJoyn framework as a potential platform for developing proximity based applications. The developed application Min-O-Mee serves as an intended realization of a MSNP through AllJoyn framework. A plethora of applications and use cases in context of IoT and other domains can utilize the framework due to its rich feature set and support for diverse platforms. Comprehensive analysis of the framework is done with similar other technologies in the literature. AllJoyn framework though emerges out to be a better alternative in the comparison, it has scope for further technological enhancements. Lack of support for short range technologies like Bluetooth and Wi-Fi direct and support for multi-hop communication are identified as key missing features in the framework. Multi-hop communication support is identified as a key incorporation that could be made into framework due to its potential application in scenarios such as vehicular networks, gaming, disaster networks, IoT and many such more applications. Accordingly a popular routing protocol in ad hoc networks B.A.T.M.A.N is chosen as underlying routing protocol for multi-hop communication. Various integration mechanisms are presented to integrate the AllJoyn framework operating at the application layer with the B.A.T.M.A.N kernel module operating at the kernel level. This approaches are termed as Naive, Improved and Ask / Reply approach. Naive approach and Improved approach blindly rebroadcast service information appended in the received packets resulting into a lot of control overhead. Ask reply approach operates in an intelligent manner by rebroadcasting service information only if it observes a change in the received information. However it has its own overhead of generating of Ask / Reply packets but that exists as a one time overhead. Besides the multi-hop service advertisement, a mechanism is devised to enable communication across services existent on nodes which are single hops or multiple hops away from each other using the Batman routing. Developed enhancements to the framework are tested on real-time test-bed comprising of different types of devices. Logger modules are incorporated into the batctl implementation to gather logs during the experiments. Comparison between the integration approaches is made on identified parameters such as service discovery time, control overhead and network throughput. A trade off between discovery time, control overhead and network throughput is observed across the approaches.

9.2 Future Work :

Multi-hop realization through other routing algorithms in ad hoc networks : The thesis proposes mechanisms to realize multi-hop communication in AllJoyn framework through the use of B.A.T.M.A.N routing. Integration of the framework with other routing algorithms such as OLSR, AODV, etc. will be a noteworthy future course of direction. A thorough evaluation can then be performed across these implementation on parameters such as service discovery time, control overhead and the application performance (throughput, delay, etc). This would enable a cross comparison in real-time across different algorithms and classification could be made of which integration would be a better alternative in different real application scenarios of AllJoyn framework.

Incorporation of Support for short range technologies such as Bluetooth, Wi-Fi Direct and ZigBee : Technologies such as Bluetooth, Wi-Fi Direct are readily available in various devices these days. AllJoyn support for these transports on different platforms will enable ad hoc, dynamic, infra structureless proximity based networks in true sense. A variety of applications and scenarios in real-time could benefit from the support of these transports. ZigBee is another popular communication protocol used for communication in various IoT due to its lower power consumption and hence useful for embedded devices which are typically resource constrained. Incorporation of support for ZigBee protocol in AllJoyn framework will lead to utilization of the benefits of both the technologies, lower power consumption of ZigBee and lightweight application framework, support for diverse platforms and devices from the AllJoyn framework.

References

[1] Forbes. Forbes IoT forecasts and market estimates 2015. http : / / www . forbes . com / sites / louiscolumbus / 2015 / 12 / 27 / roundup-of-internet-of-things-forecasts-and-market-estimates-2015/.

[2] E. Balandina, S. Balandin, Y. Koucheryavy, and D. Mouromtsev. IoT Use Cases in Healthcare and Tourism. In Business Informatics (CBI), 2015 IEEE 17th Conference on, volume 2. 2015 37–44.

[3] P. P. Gaikwad, J. P. Gabhane, and S. S. Golait. A survey based on Smart Homes system using Internet-of-Things. In Computation of Power, Energy Information and Commuincation (ICCPEIC), 2015 International Conference on. 2015 0330–0335.

[4] A. Azzara, M. Petracca, and P. Pagano. The ICSI M2M Middleware for IoT-Based Intelligent Transportation Systems. In Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on. 2015 155–160.

[5] F. Bing. Research on the agriculture intelligent system based on IOT. In Image Analysis and Signal Processing (IASP), 2012 International Conference on. 2012 1–4.

[6] S. Fang, L. D. Xu, Y. Zhu, J. Ahati, H. Pei, J. Yan, and Z. Liu. An Integrated System for Regional Environmental Monitoring and Management Based on Internet of Things. *IEEE Transactions on Industrial Informatics* 10, (2014) 1596–1605.

[7] Wikipidea. IPv6 2016. https://en.wikipedia.org/wiki/IPv6.

[8] S. Ziegler, C. Crettaz, and I. Thomas. IPv6 as a Global Addressing Scheme and Integrator for the Internet of Things and the Cloud. In Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on. 2014 797–802.

[9] A. Alliance. AllJoyn Framework 2016. https://www.allseenalliance.org.

[10] M. Villari, A. Celesti, M. Fazio, and A. Puliafito. AllJoyn Lambda: An architecture for the management of smart environments in IoT. In Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on. 2014 9–14.

[11] F. K. Santoso and N. C. H. Vun. Securing IoT for smart home system. In Consumer Electronics (ISCE), 2015 IEEE International Symposium on. 2015 1–2.

[12] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks* 54, (2010) 2787–2805.

[13] S. Dominikus and J.-M. Schmidt. Connecting passive RFID tags to the Internet of Things. In Interconnecting Smart Objects with the Internet Workshop, Prague. 2011 .

[14] R. Weinstein. RFID: a technical overview and its application to the enterprise. *IT professional* 7, (2005) 27–33.

[15] Y. Wang, A. V. Vasilakos, Q. Jin, and J. Ma. A Wi-Fi Direct Based P2P Application Prototype for Mobile Social Networking in Proximity (MSNP). In Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on. IEEE, 2014 283–288.

[16] F.-Y. Chen, C.-C. Chao, K. Wattanachote, and K.-C. Li. Using Gesture Annotation to Achieve Real-Time Communication under MSNP. In Ubi-Media Computing (U-Media), 2011 4th International Conference on. IEEE, 2011 101–105.

[17] Y.-C. Yu. A mobile social networking service for urban community disaster response. In Semantic Computing (ICSC), 2015 IEEE International Conference on. IEEE, 2015 503–508.

[18] M. Nekovee and B. B. Bogason. Reliable and effcient information dissemination in intermittently connected vehicular adhoc networks. In Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th. IEEE, 2007 2486–2490.

[19] M. Picone, M. Amoretti, and F. Zanichelli. GeoKad: A P2P distributed localization protocol. In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on. IEEE, 2010 800–803.

[20] N. Eagle and A. Pentland. Social serendipity: Mobilizing social software. *Pervasive Computing, IEEE* 4, (2005) 28–34.

[21] A. I. Wang, T. Bjornsgard, and K. Saxlund. Peer2me-rapid application framework for mobile peer-to-peer applications. In Collaborative Technologies and Systems, 2007. CTS 2007. International Symposium on. IEEE, 2007 379–388.

[22] J. Scott, J. Crowcroft, P. Hui, and C. Diot. Haggle: A networking architecture designed around mobile users. In WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services. 2006 78–86.

[23] D. J. Dubois, Y. Bando, K. Watanabe, and H. Holtzman. ShAir: Extensible middleware for mobile peer-to-peer resource sharing. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013 687–690.

[24] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. MobiClique: middleware for mobile social networking. In Proceedings of the 2nd ACM workshop on Online social networks. ACM, 2009 49–54.

[25] A. S. Alliance. "AllJoyn source code" 2016. `https://allseenalliance.org/framework/download`.

[26] P. Papadimitratos, L. Buttyan, T. S. Holczer, E. Schoch, J. Freudiger, M. Raya, Z. Ma, F. Kargl, A. Kung, and J.-P. Hubaux. Secure vehicular communication systems: design and architecture. *Communications Magazine, IEEE* 46, (2008) 100–109.

[27] S. Akiyama. Vehicular communication system for communicating information among electronic devices installed in vehicle 2003. US Patent 6,629,032.

[28] W. James. Intelligent transport system 2004. US Patent 6,810,817.

[29] T. Fujiwara, H. Makie, and T. Watanabe. A framework for data collection system with sensor networks in disaster circumstances. In Wireless Ad-Hoc Networks, 2004 International Workshop on. IEEE, 2004 94–98.

[30] C. B. Nelson, B. D. Steckler, and J. A. Stamberger. The evolution of hastily formed networks for disaster response: technologies, case studies, and future trends. In Global Humanitarian Technology Conference (GHTC), 2011 IEEE. IEEE, 2011 467–475.

[31] Wikipedia. "B.A.T.M.A.N routing protocol" 2016. `https://en.wikipedia.org/wiki/B.A.T.M.A.N.`

[32] Wikipedia. "Freifunk" 2016. `https://en.wikipedia.org/wiki/Freifunk`.

[33] D. Johnson, N. Ntlatlapa, and C. Aichele. A simple pragmatic approach to mesh routing using BATMAN .

[34] IETF. "Batman RFC" 2008. `https://tools.ietf.org/html/draft-wunderlich-openmesh-manet-routing-00`.

[35] L. Barolli, M. Ikeda, G. D. Marco, A. Durresi, and F. Xhafa. Performance Analysis of OLSR and BATMAN Protocols Considering Link Quality Parameter 307–314.

[36] A. Sharma and N. Rajagopalan. A Comparative Study of BATMAN and OLSR Routing Protocols for MANETs. *International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)* 2, (2013) 13–17.

[37] R. Sanchez-Iborra, M. D. Cano, and J. Garcia-Haro. Performance Evaluation of BATMAN Routing Protocol for VoIP Services: A QoE Perspective. *IEEE Transactions on Wireless Communications* 13, (2014) 4947–4958.

[38] T. Honda, M. Ikeda, E. Spaho, M. Hiyama, and L. Barolli. Effect of Buildings in VANETs Communication: Performance of OLSR Protocol for Video Streaming Application 323–327.

[39] M. H. Linus Lssing. "B.A.T.M.A.N and Batctl source code" 2016. `https://www.open-mesh.org/projects/open-mesh/wiki/Download`.

[40] IETF. "B.A.T.M.A.N advanced documentation" 2016. `https://www.open-mesh.org/projects/batman-adv/wiki/`.

[41] A. S. Alliance. "The AllJoyn Framework - System Description" 2016. `https://allseenalliance.org/framework/documentation/learn/core/system-description/`.

[42] Wikipedia. "Link Local Addressing" 2016. `https://en.wikipedia.org/wiki/Link-local_address`.

[43] M. H. Linus Lssing. "Batctl Usage" 2016. `https://www.open-mesh.org/projects/batman-adv/wiki/Using-batctl`.