FUTURE IOT NETWORK ARCHITECTURE AND APPLICATIONS IN MOBILE SENSING

by

SUGANG LI

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Yanyong Zhang and Dipankar Raychaudhuri

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

OCTOBER, 2018

ABSTRACT OF THE DISSERTATION

Future IoT Network Architecture and Applications in Mobile Sensing

by Sugang Li

Dissertation Director: Yanyong Zhang and Dipankar Raychaudhuri

As the number of networked devices rapidly increases in the past few years, the era of the Internet of Things (IoT) has arrived. IoT integrates a variety of existing technologies such as wireless sensor network, mobile sensing, and wearables, while new challenges arise as a result of this integration. In this thesis, we aim at addressing the following challenges. First, these technologies are isolated within insular management and communication systems, where inter-system communication is either absent or cumbersome. Current network protocols such as IP fail to support the scalability requirement of IoT. Meanwhile, the growth of connected devices imposes a tremendous amount of small packets with repeated or similar content, which leads to inefficient network resource utilization. Finally, due to the deployment cost of IoT infrastructure, IoT sensing service is missing in many suburban areas.

In the first part of this dissertation, we design and implement MF-IoT, a new IoT architecture based upon future internet architecture MobilityFirst, to address the global reachability and scalability challenge. We extend MobilityFirst to resource-constraint devices by adopting shorter device/service identifiers, which we refer as the Local Unique Identifier. At the same time, we maintain the transparency at the application layer, i.e., communication between applications is still based on the full-length Global

Unique Identifier that is used in MobilityFirst. Besides, MF-IoT provides cross-domain rich communication patterns (unicast, multicast, etc.) as well as mobility. Through detailed evaluation, we show that MF-IoT outperforms the existing solution, and also provides the global reachability via id-based communication.

In the second part of this dissertation, we propose AggMEC, an IoT traffic aggregation system that reduces total network traffic for any data collection traffic flow. By introducing a novel cost function, we are able to adopt two clustering-based algorithms to minimize the overall network traffic in any unspecific network topology. In addition, we design our routing plane over MobilityFirst, which avoid obtrusive destination address translation in the IP network. Through detailed evaluation, we show that our first algorithm outperforms two other baseline schemes in both total network traffic as well as end-to-end latency when the resource is specified by the application provider, while the second can achieves better aggregation efficiency if the resource is unspecified.

In the third part of this dissertation, we propose Auto++, a mobile roadside context sensing system to support pedestrian safety and traffic monitoring applications in low population areas. Auto++ analyzes audio stream captured by microphones on smartphones to extract the features (maximum frequency on a particular energy & Time Difference of Arrival) to detect the presence of cars and their arriving direction. Also, Auto++ can also count the pass-by cars on the road in real-life. Through detailed experiments, we show that Auto++ can detect a car's presence 7 seconds before its arrival with a very low false positive rate. We also demonstrate that Auto++ is tolerant to various noisy environments in real-life.

Acknowledgements

Through my amazing five years at WINLAB, I have received tremendous help and guidance from many people. I would like to express my gratitude to all these people who have made my graduate journey magnificent and this dissertation complete.

First, I want express my special thanks to my advisor Prof. Yanyong Zhang and Prof. Dipankar Raychaudhuri. During my five years stay at WINLAB, they have provided me with great support, encouragement and guidance. Prof. Zhang has work with me through my entire Ph.D. process, and worked with me on every steps towards my dissertation. She has taught me how to refine research problem, conduct research and think in a critical way. In addition to the technical aspect, I have received valued suggestions from her on my career path, team work collaboration, management skills, communication skills, etc. She has further impacted me on my values and my life goal, I could not thanks her enough. Prof. Ray has always been very supportive and encourage me to conquer the challenges on my path to success. His broad knowledge and insightful suggestions always leverage my research problem to the next level. Besides research, I also learned a lot from his exceptional management skills, which I believe will lead to the success of both life and career.

I have worked with several industrial collaborators throughout my Ph.D., especially Dr. Ravishankar Ravindran and Mr. Guoqiang Wang. Before I came back to WINLAB to continue my graduate study, I spent a wonderful half year to work with them at Huawei Research Center and continued our collaboration after I came back. We had a quite amount of discussion on system design and evaluation. They insightful suggestions ignited me on doing networking related research.

I would also like to give my special thanks to the excellent colleagues that I had worked with. I started my research with Dr. Chenren Xu when I was a master student.

His passion on research encourage me to continue my research in Ph.D. program. During my Ph.D., his also helped me to overcome the obstacles in research and career path. Dr. Jiachen Chen has impacted me with his strict research methodology and in-depth understanding of computer system. He helps me to set a higher bar for my research, especially for my MF-IoT project, the key aspect of my dissertation. Prof. Janne Lindqvist gives me enormous help on experiment design and paper writing and I learn lots of knowledge to make the work more repeatable. Prof. Marco Gruteser gives me many advices in my first project which leads to my first top-tier conference paper. Dr. Richard E. Howard always inspires me with his fundamental understanding of any research problem that I have encountered.

I also wish to thank many collaborators, colleagues and friends, who have worked with me and influenced my way of thinking during my Ph.D, including Ivan Seskar, Zhenhua Jia, Wuyang Zhang, Xiaoran Fan, Shreyasee Mukherjee, Parishad Karami, Francesco Bronzino, Michael Sherman, etc. I would also like to thank all the undergraduate students and master students that I had worked with.

Finally, I would like to thank my family for their selfless love, support, and patience.

Dedication

To my wife Bin Lin, my parents Xiangzhen Li and Liping Zhou, for their love, continuous support and encouragement

Table of Contents

Chapter 1

Introduction

1.1 Overview

1.1.1 Challenges in The Future Internet of Things

In the era of the Internet of Things (IoT), many of the devices have been networked in one form or another. According to [1], the number of connected devices will grow to 50 billion by 2020. Technologies such as wireless sensor network, wearable, mobile sensors are being integrated as a part of IoT. This integration has introduced new opportunities as well as posed new challenges. In the last two decades, research communities and industry make remarkable efforts in developing new low power technologies(e.g., Zigbee, Bluetooth Low Energy (BLE), 6LoPAN, etc.) to connect resource-constrained devices to the Internet. Nevertheless, each of the IoT architectures is a silo-system due to the network protocol compatibility or the naming/addressing scheme fails short in supporting global reachability. Meanwhile, a growing number of connected IoT devices introduce a new data traffic pattern – the high data traffic volume of small packets with repeated content, which introduce tremendous overhead to the network and the server. In any packet-switch network, the overhead is determined by the number of packets as every incoming packet requires a routing table lookup in routers and an interrupt in the server which results in the same performance cost regardless of the packet size. Moreover, a great amount of sensor reading data are repeated or similar due to the short sensing interval and closed locations. This leads to the low utilization of the network. In the perspective of sensing applications, although dedicated embedded devices (e.g., thermostat, motion sensor, smoke detector)have been widely deployed in sensing infrastructure, their functionality is too isolated and static to provide seamless sensing

service for dynamic users. The use of mobile devices compensates this drawback, but the sensing services are limited by the sensor types on such devices as they are designed to serve specific tasks/applications.

As a result, we believe that there is still much room to improve in this area despite the progress in the past decade. In particular, we look into the challenges in the following aspects:

- **Global Connectivity:** Traditional computer network is designed with a host-to-host communication paradigm. In the context of the Internet of Things, people pay more attention to the services (eg. sensing service or actuating service) provided by the devices instead of the devices themselves. Most of the state of art IoT systems adopt centralized architecture, in which servers play roles in managing the devices and providing the services. As the number of services is increasing exponentially, it raises a scalability challenge to the storage and the network capacity that the current architecture can support. One way to tackle down this problem is via realizing service-to-service or device-to-device self-management. However, such decentralized manner requires global reachability of the resource, as the services may be deployed in the different domains. The state-of-art IP-based solutions [2,3] hinder this by the Network Address Translation (NAT) service because the flat IP address space is not sufficient to name all of services.
- **Intelligent Computing:** IoT traffic demonstrates different patterns compared to user content dissemination due to the massive data traffic volume of smart packets coming from the unprecedented amount of connected devices. Although data aggregation has been widely studied in the area of Wireless Sensor Network (WSN) to maximize the overall battery life, there are few works of literature discussing its infrastructure support in the edge network or the core network. One of the reasons is that current IP router is not aware of the content in the transiting packets due to the cost of checking each packet. At the same time, in order to build an efficient aggregation tree, additional routing control logic needs to be added to the network.
- **Sensing in Real-time:** Pervasive deployed IoT sensing devices benefit the artificial intelligence model with large data sets that are more readily than ever before.

However, in term of timeliness, current cloud-based solutions fail in short due to the inevitable latency between the computing services and the actuating services. Moreover, their stationary setting cannot satisfy the requirements of dynamic mobile users. While mobile devices are seamlessly supporting user-centric sensing with a suite of sensors with real-time processing capability, it requires additional effort to develop new algorithms to support sensing tasks other than those which the sensors are designed for.

Our objective in this dissertation is to proposed the corresponding solutions to address the challenges we mentioned above.

1.1.2 Proposed Solutions

Scalable and Global-Reachable IoT Network

At the core of the proposed IoT architecture is MobilityFirst [4], a next-generation network that focuses on handling device and service mobility on the Internet. In MobilityFirst, each device or service has a 20-byte flat Global Unique Identifier that decouples from its location or Network Address (NA). The translation from GUID to NA is provided by a logically centralized Global Name Resolution Service (GNRS). It enables routers instead of end-hosts perform NA lookup on a delivery failure when an attached end device moves. This process is transparent to the application layer, which means application only needs to focus on the GUIDs rather than the NAs. Although MobilityFirst is well-suited for addressing the scalability and global-reachability challenge for the IoT, MF-IoT extends MobilityFirst to adopt the following requirements in the IoT network: 1) shorter IDs to accommodate the short Maximum Transmission Unit provided by the popular low-rate layer-2 protocols like 802.15.4 [5]; 2) more efficient routing that costs less transmission power and storage than the one adopted by MobilityFirst [6]; 3) feasible GNRS lookup protocol due to the limited connection between IoT device and GNRS.

Aggregating IoT traffic at the Internet Edge

The tremendous amount of IoT data impose a huge challenge to the capacities of the network infrastructure as well as the cloud infrastructure. Furthermore, these data contains considerable redundancy of data representation which poorly utilizes the network resource and the storage resource. To address this problem, we propose a generic data aggregation system over mobile edge clouds – AggMEC, for any application that needs aggregation on unspecific network topology. By introducing the cost function of the total network traffic, we unveil two clustering-based schemes to minimize this cost function. Since current IP network cannot scale well for this problem as the network layer is not aware of the content of the packets, we design and implement our system over a clean-slate Future Internet Architecture (FIA) – MobilityFirst (MF) [4]. In MF, every object/application/service/device is named with a persistent global unique ID, which not only enables the network to aggregate the packets without decoding the packets' content, but also reduces the overhead of obtrusive address resolution in the IP network.

Sensing the Pedestrian Ambient Context in Real-time

IoT sensing service coverage is restricted in density because of the high IoT infrastructure deployment cost, when the information(including pedestrian safety and traffic) in the uncovered is still critical. However, mobile sensors such as microphones on smartphones are widespread and require zero-additional infrastructure support to provide sensing service. As a result, we propose Auto++ – an unsupervised roadside context sensing system using microphones on smartphones, which not only alarms the pedestrian for approaching vehicles and their coming direction, but also senses traffic flow on the nearby road.

Unlike other object sensing technique such as DSRC [7] and LIDAR [8] that require additional hardware and infrastructure support, Auto++ utilizes smartphone built-in microphone to capture the sound from the vehicle and perform local computing to detect the presence and the direction of a vehicle and count the pass-by vehicle in real

time. Our experimental results illustrate that our system can detect a car seven seconds in advance, and determine its direction in most instances.

1.2 Organization

The organization of this dissertation is organized as follows. In Chapter 2, we present our IoT architecture MF-IoT which provide global-reachability of the IoT object/service with persistent flat ID. Next,Chapter 3 proposed the data traffic aggregation system and share our observation with realistic data trace. Thirdly, we introduce our unsupervised roadside sensing system on mobile devices and demonstrate how well our system could work in real life scenarios. Finally, Chapter 5 concludes the dissertation.

Chapter 2

Service-centric Internet of Things Architecture over Information-Centric Networking

2.1 Introduction

With the advent of new generation embedded devices, including wearables, robots and drones, Internet of Things (IoT) has received a great deal of attention in the past few years. The applications of IoT range from individual-level applications such as smart health care and smart homes, to large-scale ones such as smart cities. According to [1], the total number of connected "devices[1]" can reach 50 billion by the year of 2020. Compared to traditional sensors, these new devices are able to cope with more complex logic — sensors can carry out more local computation and actuators can be easily controlled via the network. Additionally, many devices now have higher mobility than before. These changes in the IoT devices have introduced new opportunities as well as posed new challenges to the underlying network design.

In the last few years, several solutions have been proposed to design new IoT systems, which can be generally classified into two categories based on the underlying network design. Solutions in the first category (e.g., those discussed in [5, 9]) try to support the IoT system through traditional Internet Protocol (IP). However, IP has its intrinsic problems in dealing with device mobility since it couples a node's identity with its location. Also, the wide deployment of Network Address Translation (NAT) hinders global device reach-ability in scenarios such as sending invasion alarms to the user's mobile devices. To better cope with mobility and realize global reach-ability,

[1]In this chapter, we refer to these devices as embedded devices, sensor devices, or IoT devices interchangeably.

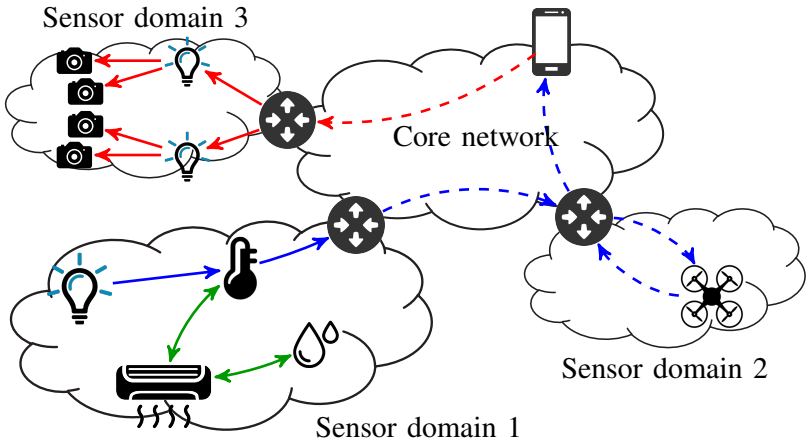


Figure 2.1: IoT Overview (green: air conditioner request data from temperature and humidity sensors; blue: light sensor triggers drone in another domain to take a picture and send it to the cell phone; red: cell phone controls cameras to take pictures at the same time)

therefore, solutions in the second category (e.g., those discussed in [3, 10]) try to alleviate the above problem in the application layer. In these solutions, a variety of Wireless Sensor Network (WSN) protocols are used inside the constrained part of the networks (that mainly consist of resource constrained embedded devices), while the interactions between infrastructure nodes and sensor devices are achieved by a server (or a proxy) that runs application-layer logic. Hence, to allow embedded nodes to initiate communication with infrastructure nodes, these solutions either have to maintain long-lived connections between the client and the proxy, which will likely result in scalability issues, or rely on polling, which will likely cause traffic overhead and long latency. To make matters worse, sensor nodes deployed by different organizations are usually not compatible with each other, and therefore users will end up installing multiple servers (or server applications) to support all the sensor nodes.

In this chapter, we argue that we should design a generic and efficient network architecture to support sensor nodes and infrastructure nodes across domains and organizations. We present the IoT system we envision in Fig. 2.1. Much more than a simple combination of WSN and core network, the proposed network architecture should have the following critical features:

- *global reach-ability* for all the embedded and infrastructure nodes, in that they can

be identified and located via persistent, globally accessible, identities,

- *mobility support* for seamless connection in the presence of node mobility,
- *richer communication patterns* including direct device-to-(multiple/any) device, device-to-(multiple/any) infrastructure, infrastructure-to-(multiple/any) device communication without the necessity of going through the application layer (where the devices may belong to different local IoT domains as shown in Fig. 2.1), and
- *resource efficiency* which supports a large number of embedded devices that are severely constrained in energy, computation, storage, and/or network capacity.

To build an IoT system with these features, we believe the main challenge lies in the network layer design, especially in the data plane. In this chapter, we focus on this particular aspect, while adopting existing algorithms in neighbor/service discovery and routing in the proposed system.

At the center of the proposed IoT network architecture is MobilityFirst [4], a next-generation network that focuses on handling device mobility in the Internet. We choose to build our architecture on MobilityFirst because it is designed to address the inefficiencies of IP when accomodating rapidly increasing mobile and sensor devices in the Internet, which shares a common set of challenges as building a new IoT system out of these devices. In MobilityFirst, each device (or application, or even a piece of content) has a 20-byte flat *Globally Unique IDentifier (GUID)*. MobilityFirst decouples the node identity (GUID) from its location (Network Address, *NA* in short). The translation from GUID to NA is performed by a logically centralized *Global Name Resolution Service (GNRS)*. Different from DNS, GNRS is a network component transparent to the applications, which means the routers rather than the senders can perform late-binding (GNRS re-lookup) on a delivery failure when an end host moves. This design allows quick and local fail recovery to increase the delivery rate, and more importantly, the applications focus only on the GUIDs rather than the changing NAs. MobilityFirst is well-suited to support the aforementioned features of the IoT architecture.

However, we need to address several challenges before we can extend MobilityFirst into the IoT networks:

1. *Long GUIDs:* the usage of 20-byte GUIDs in IoT is too heavy-weight, if at all

possible, for layer 2 protocols like 802.15.4 [11] (which has a mere 127-byte MTU);

2. *Costly GNRS lookup:* MobilityFirst routers perform a GNRS lookup when the destination NA is unknown or the client is no longer at the NA specified in the packet (due to mobility). This operation is not feasible for storage-constrained embedded nodes;

3. *Link-state routing:* MobilityFirst adopts link-state routing [6], similar to OSPF, which poses high computation and storage burdens on embedded nodes.

To address these challenges, in this chapter, we propose *MF-IoT*, a generic network architecture that extends MobilityFirst into the IoT world. We create a resource efficient "dialect" of MobilityFirst to allow sensor nodes to communicate within a local area, referred to as a local IoT/sensor domain (see Fig. 2.1). Gateways are used to translate between MobilityFirst and the dialect but this process is transparent to the applications. Unlike the application layer solutions, the dialect only exists in the network layer. Unlike NAT either, each device in MF-IoT has a GUID, like in MobilityFirst, and this GUID can be used to realize global reach-ability. MF-IoT also takes communication pattern diversity, GUID size, and computation awareness into consideration to provide rich yet light-weight support to IoT applications.

The contributions of this chapter are as follows:

- We identify a list of requirements for a generic IoT architecture;
- We extend the GUID-based communication into the IoT domains to allow global reach-ability and seamless mobility handling, while using Local Unique IDentifiers (LUID) in local IoT domains for efficiency;
- We support a rich set of communication patterns, including unicast, multicast and anycast between sensor nodes and infrastructure nodes *and* among sensor nodes (that may or may not belong to the same domain);
- We adopt *service-based* GUID assignment which facilitates communication with a specific service provided by a node instead of with the node itself, and easier support for functions like caching, service mobility (anycast), etc;
- Through large scale simulations, we show that MF-IoT can signifcantly reduce the

network traffic load and effectively avoid congestion in the network, leading to packet delivery latencies that are orders of magnitude shorter. Indeed, MF-IoT can deliver packets within tens of milliseconds, while IP-based solutions encounter severe congestion and the resulting latencies are more than a few seconds.

The remainder of the chapter is organized as follows: §2.2 summarizes the envisioned requirements of a generic IoT network architecture and describes our main design rationales. The design detail is presented in §2.3 and the architecture is evaluated in §2.4. §2.6 discusses existing IoT network architectures, and §2.7 concludes the chapter.

2.2 Design Rationale

In this section, we first discuss the requirements we envision an IoT architecture should satisfy to efficiently connect billions of devices online and enable diverse interactions among them. We then give a brief description of MobilityFirst and explain why we base our design on MobilityFirst. Finally, we present our design rationales one by one.

2.2.1 Requirements of a Generic IoT Architecture

As the number of embedded devices rapidly increases, they pose a set of new challenges/requirements on the underling network design.

Global reach-ability

One of the salient features offered by IP is its global reach-ability – applications can reach each other by simply specifying the destination IP address in the packet header, without worrying about details such as hardware type, or the application on the destination.

This feature is particularly important for scenarios like emergency notification. In such scenarios, it is desirable that the sensors are able to notify the predefined clients without the need of going through a server or proxy, requiring each client to be associated with a unique and persistent identifier. Global reach-ability is also important for remote actuation applications, where users may need to use their smartphones to

directly control devices such as air conditioners, rather than going through 3rd party protocols [2].

Most of the existing IoT architectures [5, 10]) focus either on the communication within a local sensor network or on the adaptation over the application layer, but we take the viewpoint that an IoT architecture should also enable transparent interactions between sensors and infrastructure nodes at little overhead.

Mobility support

With the rapid deployment of mobile sensors such as robots and drones, an IoT architecture should provide seamless mobility support such that applications can communicate with each other without worrying about the consequence of a node's location change or any network change (e.g., new identity, new route to the target). At the same time, client devices tend to move frequently — the user's smartphone may move to a different network while in the middle of controlling a rice cooker, or listening to the security alarm at home. An IoT architecture should be able to handle mobility of different parts of the system.

Communication diversity

A significant departure from traditional WSNs whose main function is data collection, the new IoT paradigm aims to facilitate a larger variety of use cases and a much richer set of communication patterns.

An important communication pattern in IoT is device-to-device communication; a sensor should be able to directly communicate with an actuator rather than being connected by a server. This communication pattern can cut down response time, traffic volume and potential failures caused by the server, all of which are critical to real-time IoT systems. Additionally, direct communication between a device and multiple devices should also be supported.

Another communication pattern needed by IoT is direct communication between a device and infrastructure nodes. In fact, the *observe* mode described in [12] follows this communication pattern — a client registers a certain event, and when that event

is detected by a device, it would send notifications to the client directly.

Finally, anycast should be supported, delivering messages to any node from a group.

Resource constraints and heterogeneity

Even though today's IoT devices are becoming increasingly more powerful, their resource constraints remain a big issue. Many of the embedded devices still have very limited computation, memory, and communication capability. Moreover, embedded devices vary greatly in their capability. As a result, an IoT architecture should take into consideration these factors.

2.2.2 Background on MobilityFirst

MobilityFirst [4] is proposed as a future Internet architecture with mobility and global accessibility as core design concerns. To achieve these features, MobilityFirst introduced several components into the network:

Globally Unique IDentifier (GUID)

MobilityFirst utilizes persistent GUID to name every network object. The separation between the identifier (GUID) and the locator (network address, *NA*) provides support for mobility and global accessibility. Meanwhile, GUID can be a public key derived from the properties of the object or a human-readable name, hence it allows the objects to be self-certifiable.

Global Name Resolution Service (GNRS)

GNRS is a logically centralized service that maintains the mapping from the GUID of an object to its current NA(s). MobilityFirst routers can perform late binding — querying the GNRS whenever a destination NA could not be resolved in the local scope. This is a network-layer solution which is different from DNS, and it provides better support for mobility since the network has the potential to recover a delivery failure locally. Works in [13, 14] proposed distributed solutions for GNRS implementation which can

have scalability and acceptable lookup performance in the core network.

Routing

MobilityFirst routes packets based on the NA(s). Work in [6] proposed a basic routing solution in MobilityFirst similar to Open Shortest Path First (OSPF). In this solution, each router maintains the global topology and calculates the shortest path to the destination in a distributed manner.

Service ID

To support multiple network services such as unicast, multicast, and in-network computing, service ID is included in the packet header so that each router is capable of making decision based on its policy.

Based on the aforementioned components, MobilityFirst has the potential to be a network architecture for IoT. However, some challenges remain for the deployment in IoT systems in which many resource-constraint devices might exist. First of all, MobilityFirst uses a 20-byte flat GUID. If the network operator tries to run the low-rate network (e.g., IEEE 802.15.4), it will be inefficient in data transmission. Secondly, GNRS operations remains unrealistic for the low-end devices since they may not have direct link to the GNRS server, nor does it have enough storage to support store and forward in late binding. Routing scheme also needs to be optimized to preserve energy consumption if we want to use MobilityFirst in IoT. Therefore, in this work, we propose MF-IoT, a generic network architecture that extends MobilityFirst into the IoT world, providing rich (yet light-weight) support for different applications and communication patterns.

2.2.3 MF-IoT Design Rationales

Based on the requirements, we propose MF-IoT, an architecture that extends MobilityFirst to allow seamless communication among IoT nodes and between IoT and infrastructure nodes. We build our architecture over MobilityFirst because of its inherent support for reach-ability (via 20-byte persistent GUID) and mobility (via late-binding

in the network layer [4]). Accordingly, we create a much lighter-weight protocol embedded devices can use within a local IoT domain to meet their resource constraints. In order to achieve global reach-ability, we use network-layer translators (gateways) to provide transparent translation between the light-weight protocol and MobilityFirst.

GUID vs. LUID

The 20-byte GUID is a key feature in MobilityFirst to provide mobility support. Each device would have a persistent and unique GUID no matter when and where it moves. It is also important for MF-IoT to keep this feature in achieving global reach-ability and mobility handling.

However, always carrying the 20-byte GUIDs (40 bytes for a source-destination pair) in the packet header may not be always feasible over a low-rate layer-2 protocol such as 802.15.4. To solve this issue, we first introduce a lighter-weight packet header (total length of 10 bytes, see §2.3.2)) and a 2-byte *locally unique ID* (LUID in short). In this way, we map a device's 20-byte GUID to its 2-byte LUID when we reach the local area IoT domain. To cope with collisions that may occur in this mapping process, we let each domain have its own GUID to LUID mapping which is managed by a gateway deployed at the edge of the domain.

Different from NAT and other existing domain-based solutions, MF-IoT does not change the identity the application uses. The applications, either on constrained IoT devices or on the infrastructure nodes, still use the 20-byte GUID to identify each other, while the network performs translation which is transparent to these applications (see §2.3.3 for detailed explanation). An IoT node carries its GUID no matter where it moves, even when it is relocated to another local IoT domain and is assigned with a new LUID. This ensures the global reach-ability and mobility handling yet still considers resource constraints of embedded devices.

Service-based GUID

In MobilityFirst, a GUID can be used to identify a network node, an application, or even a piece of content. In MF-IoT, we associate a GUID with a specific service,

hence *service-based* GUIDs. Here, service has a finer granularity than a network node since in IoT, a node often carries multiple services — e.g. a robot might carry a light sensor, a temperature sensor and several actuators, each of which provides a service. Its granularity is similar to that of a "port" in the TCP/UDP definition – each application can have its GUID(s) that are exposed to the network and other applications. In MF-IoT, we name each individual service instead of the node GUID + port approach like in TCP/UDP.

With service-based GUIDs, applications on an IoT node can simply listen to one or more GUIDs for different services, e.g. sensor data reporting, actuation, caching, etc. With this approach, we can easily support transparent and energy efficient service migration, without affecting the functionality of the services (see §2.3.4 for detail).

MF-IoT also treats message forwarding and gateway as services, allowing simpler topology management and logic separation in IoT especially when multiple services co-locate on a single IoT node (see §2.3.5).

GUID-centric communication diversity

MF-IoT is well suited to support direct device-to-device communication, no matter if these two devices are in the same domain or not. The applications on the devices can identify each other with corresponding service GUIDs while the underlying network takes care of the translation between GUID and LUID. IoT applications can also reach infrastructure nodes easily, through their GUIDs.

MF-IoT does not distinguish unicast and multicast services. Whenever there are multiple services listening for the same GUID, the network would forward a message to *all* of these services. However, MF-IoT distinguishes *to-all* services from *to-any* services (anycast). This is achieved by a "Service ID" (SID) field in the packet header similar to MobilityFirst.

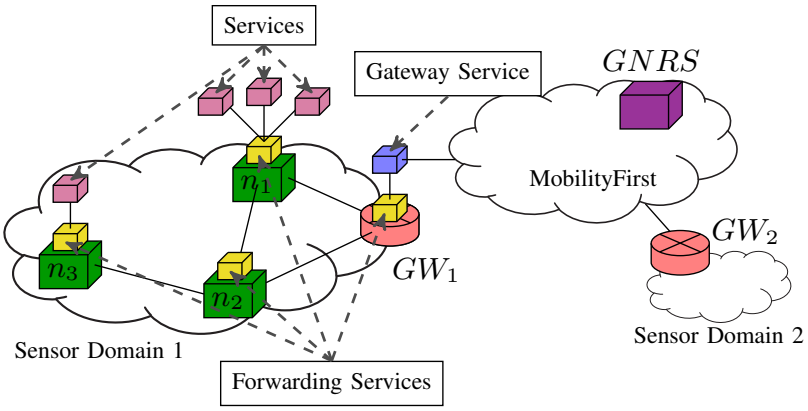


Figure 2.2: MF-IoT architecture overview

2.3 MF-IoT Architecture Design

In this section, we describe the detailed design of MF-IoT. We first present the components in MF-IoT, the data packet format, and then explain how the components work together to provide the features discussed in §2.2.1.

2.3.1 Components in MF-IoT

MF-IoT consists of the following components (Fig. 2.2):

IoT/Sensor domain

We refer to a local area IoT/sensor network as an IoT/sensor domain. A large portion of the nodes in the IoT domain are resource-constrained, where energy-efficient link and physical protocols such as 802.15.4 or Bluetooth Low Energy (BLE) are primarily used.

MobilityFirst domain

MobilityFirst domain refers to the infrastructure network consisting of MobilityFirst routers.

Gateway

A gateway (e.g., GW_1 and GW_2 in Fig. 2.2) serves as the bridge between local IoT domains and the MobilityFirst domain, translating between MF-IoT packets and MobilityFirst packets for each local domain. In order to be compatible with both ends, multiple physical interfaces should be adopted by a gateway node.

Network nodes

In our MF-IoT architecture, network nodes can be categorized in three classes: constrained nodes within an IoT domain, resource-rich nodes within an IoT domain, and infrastructure nodes. Note that a resource-rich node within an IoT domain usually has rich computing and storage resources (e.g., camera), but the network interface is compatible with that of a resource-constrained node.

Service

We treat any resource that might be of interest to users (such as a sensor or an actuator) as a service which is the unit of GUID assignment.

To provide basic network functions while separating them from the application logic, we treat forwarding and gateway also as services (namely, *forwarding service* and *gateway service*). They provide functions like neighbor discovery, routing, packet forwarding and translation between MF-IoT and MobilityFirst packets.

Application services refer to IoT resources such as sensing and actuating. While multiple such services can be provided by a single IoT node (or even within a single application for embedded devices), a single service can also be supported among multiple sensor nodes (details in §2.3.4).

2.3.2 Packet Format

In MF-IoT, we use fixed-length headers instead of Type-Length-Value (TLV), so that less space and computing is needed. As shown in Table 2.1, we define the following fields:

Table 2.1: MF-IoT packet format (4 octets per row)

0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
VER	PKT_TYP	SVC_TYP	PROT	TTL	PKT_LENGTH
SRC_LUID				DST_LUID	
NONCE					
PAYLOAD					

- **Version (VER)**: the version number of MF-IoT packets;
- **Packet Type (PKT_TYP)**: whether a packet is a control packet or data packet;
- **Service Type (SVC_TYP)**: the network service type, such as multicast and anycast;
- **Protocol (PROT)**: the upper layer (e.g., transport layer or application layer) protocols, which helps map the upper layer to the corresponding logic;
- **Time-To-Live (TTL)**: used to prevent routing loops;
- **Packet Length (PKT_LENGTH)**: length of a packet, can allow packet size up to 4kB (2^{12} bytes);
- **SRC_LUID and DST_LUID**: the source and destination LUID;
- **Nonce**: a random number generated by the sender. In MF-IoT multicast, the link layer of the branching node broadcasts the packet instead of unicasting to every next hop. The previous hop will receive the same packet, and drop it if packet with same nonce is seen before.

2.3.3 Transparent GUID/LUID Translation

To enable global reach-ability, MF-IoT uses gateways as the bridge between the MF-IoT and MobilityFirst domains. It maintains the mapping between GUID and LUID via a translation table. The translation table contains 3 columns (as shown in Table 2.2) — the GUID, its LUID in the local domain and the mapping type. The mapping type can be *Local*, meaning the GUID is in this local domain, *Remote*, meaning the GUID is outside of the local domain, or *Local+Remote*, which is usually a multicast GUID and means that there receivers are both inside and outside the domain (see §2.3.6). The LUID in the local domain can be recycled based on Least Recent Used (LRU) or Time To Live (TTL) policies. This ensures the uniqueness of LUID in a local domain

```
1:  procedure SEND(G, d)
      parameters:
      G: destination GUID
      d: data to be sent
2:    tmp ← t_i[G]                          ▷ Lookup local translation cache
3:    if tmp = ∅ then                       ▷ Initiating communication
4:      L ← REQUEST(G, ∅)
5:    else if tmp.State = Stale then        ▷ After move
6:      L ← REQUEST(GL, tmp.LUID)
7:    else                                  ▷ Continue communication
8:      L ← tmp.LUID
9:    end if
                                            ▷ Forward MF-IoT packet based on routing
10:   FORWARD(L_i, L, d)
11: end procedure
```

during a period of time even with a 2-byte length (which allows 65,536 concurrent GUID mappings).

When an embedded device joins a domain, it registers its GUIDs (each service has a GUID) at the gateway. The gateway would give each GUID a LUID and mark them as *Local*.

When an application tries to send a message to a certain GUID (G), it would call the *send* function provided by the host node's forwarding service (see Algorithm 0). Note that in this process, the LUID is transparent to the application. The forwarding service requests G's LUID from the gateway (lines 3–4), and the gateway looks up G in its translation table. If there is already a mapping, the gateway simply replies with the LUID; otherwise it creates an entry {GUID=G, LUID =L, Type=Remote} in the translation table, where L is randomly generated (different algorithms could be adopted here, which is orthogonal to this study). Note that in this stage, the gateway does not have to perform a GNRS lookup and it can respond to the request immediately. After getting G's LUID, L, the forwarding service checks its own routing and neighbor table to forward the packet using L as the destination LUID.

The forwarding service can also have a local translation cache (t_i in Algorithm 0) for frequently communicated parties. Before requesting G's LUID from the gateway, the forwarding service can first check its own cache (lines 2 and 8). The cache could have

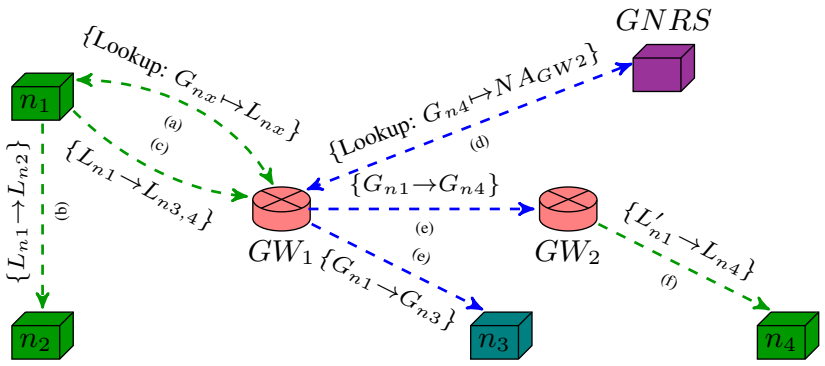


Figure 2.3: Example for global reach-ability (blue lines: MobilityFirst traffic, green lines: MF-IoT traffic)

stale information when a node moves to a new domain, but we try to keep the original LUID information to reduce changes in the routing tables. Therefore, when requesting the LUID of a stale entry in the cache, the forwarding service would carry the original LUID as its preference (lines 5–6). If there is no collision, the gateway would register this original LUID in its translation table.

Upon arrival of a MF-IoT packet, a gateway looks up its translation table and obtain the GUIDs for both source and destination forward the packet using MobilityFirst logic. At this point, it might need to look up GNRS for the destination node's NA if it is unknown. On the other hand, when the gateway receives a MobilityFirst packet whose destination GUID (G_d) is in its domain (the translation table has a matching entry whose type is *Local*), the gateway would create an LUID (L_s) for the source GUID (G_s) and mark the type as *Remote* if the source is not in the translation table, and then send a MF-IoT packet consisting of L_s and L_d. This entry is created such that the destination device can send a message to the sender.

In MF-IoT, the gateway does not differentiate if a MobilityFirst packet is coming from an infrastructure node or an embedded node in another domain. This feature enables global reach-ability and seamless mobility handling. Below we explain how these two objectives are achieved.

Table 2.2: Translation table on GW_1 in Fig. 2.3

GUID	LUID	Type
—=—=—=— G_{n1}	L_{n1}	Local
G_{n2}	L_{n2}	Local
G_{n3}	L_{n3}	Remote
G_{n4}	L_{n4}	Remote
...	...	...

Global reach-ability

Fig. 2.3 depicts three scenarios where an embedded node n_1 wants to send a message to a node in the same domain (n_2), an infrastructure node (n_3), and an embedded node in another domain (n_4), respectively. To simplify the description, we assume that each node has only one forwarding service and one application service, represented by a box. In Fig. 2.3, MF-IoT traffic is represented by green lines and MobilityFirst traffic is represented by blue lines. Note that we use dotted lines here to denote that the traffic is not direct traffic between the two nodes, but there might be relay nodes between them. We next describe the protocol exchange according to the figure.

1. To initiate the communication with n_2, n_3, and n_4, n_1's forwarding service needs to get their LUID from the gateway. For n_2, GW_1 can respond directly since it has a *Local* entry in the translation table. For the other two, GW_1 creates new entries and mark them as *Remote*. Here, GW_1's translation table is shown in Table 2.2.

2. The routing algorithm in the local IoT domain forwards the packet based on the destination LUID. Since n_2 is in the same domain, the local routing algorithm would forward the packet to n_2 eventually.

3. If the destination LUID (L_{n3} or L_{n4}) is not in the same domain, the local routing algorithm forwards the packet to GW_1, which translates the packet to MobilityFirst packets $\{G_{n1} \rightarrow G_{n3}\}$ or $\{G_{n1} \rightarrow G_{n4}\}$.

4. Now GW_1 sends the packets with traditional MobilityFirst logic. In MobilityFirst, the first step is a GNRS lookup for the NA of the destination. If the destination

is an embedded node in another domain (n_4), GNRS would reply with the NA of the corresponding gateway (GW_2). For an infrastructure node (n_3), GNRS would respond directly with its NA (not shown in the figure).

5. After getting the NA, the packet will be forwarded in the MobilityFirst network and eventually reach n_3 or GW_2. Note that thanks to late-binding technique in MobilityFirst, the packet would reach the destination even if n_3 or GW_2 has moved and has a new NA. This provides seamless mobility support when an infrastructure node or an entire local IoT domain moves.

6. When GW_2 receives the packet destined to G_{n4}, it checks the translation table and finds that n_4 belongs to the local domain. It then creates a LUID mapping for G_{n1} (L'_{n1}) and forwards an MF-IoT packet $\{L'_{n1} \rightarrow L_{n4}\}$. Note that the LUID of G_{n1} in this domain does not have to be the same as L_{n1} given by GW_1. However, this new LUID does not affect the communication between n_1 and n_4 since they are communicating with the GUID while LUID is kept transparent from them.

Handling node mobility

Next, we show how MF-IoT handles the situation when embedded nodes move from one domain to another. There are two cases we need to consider, the first involving one of the communication parties moving to a different domain (e.g., n_2 moves to GW_2), and the second involving one of the communication parties moving into the same domain (n_4 moves to GW_1). In both cases, the node that does not move (n_1) will not observe any change in the communication.

In the first case, let us consider the following situation: when n_1 sends a message to n_2, n_2 has moved from GW_1 to GW_2. We assume that n_1 already initiated the communication before n_2 moved, and therefore it already has $G_{n2}toL_{n2}$ mapping in its local cache. When the packet ($\{L_{n1} \rightarrow L_{n2}\}$) reaches GW_1, either via proactive routing (which detects the node departure and updates the routing) or reactive routing (which cannot find n_2 during message forwarding and then redirects the packet to GW_1), GW_1 contains a *Remote* entry for n_2 and it will forward the packet similar to the steps (4,

5) in the previous example. Note that during this process, n_2's LUID has changed, but the application uses only its GUID and is unaware of this change. If GW_1's translation table has not been updated when n_1's packet arrives, GW_1 can store the message and forward it later when n_2 reconnects from the new domain.

In the second case, let us consider the following situation: when n_1 sends a message to n_4, n_4 has moved from a different domain to GW_1 and has registered with GW_1. In this case, GW_1 has assigned a LUID to n_4, L_{n4}. When $n1$ sends a packet $\{L_{n1} \rightarrow L_{n4}\}$, it would reach n_4 without going to GW_1, without n_1's active involvement.

Having considered infrastructure node mobility, IoT domain (as a whole) mobility (described in the previous example), and embedded node mobility, we believe that MF-IoT can provide seamless mobility support for an IoT system.

2.3.4 Service-based GUID

With the wide deployment of IoT devices where each device can have more than one sensor services (e.g., a robot may have a temperature sensor, a humidity sensor, and several actuators), there is a need to communicate with a specific sensor service rather than lumping all the sensor services together. Thus, MF-IoT gives each of these services a GUID, which enables seamless service migration and service caching. Such features would be particularly useful in extreme cases like disaster management. Below, we will discuss 2 typical use cases to illustrate the benefits of service-based GUIDs compared to the traditional ID (IP) + port solution.

Service migration

In this case, we have an embedded node n_1 which has a temperature sensor (T) and a smoke sensor (S), and a backup node n_2 with a temperature sensor that is not in use in the beginning of the example (see Fig. 2.4). In this example, an infrastructure node n_3 queries T from time to time to get the current temperature. When the temperature sensor on n_1 fails, n_2 will serve the same functionality. In the traditional IP + port solution, the new T would have n_2's IP address with a specific port and accordingly, GW and/or n_3 would need to know the change. Note here that it is often not feasible

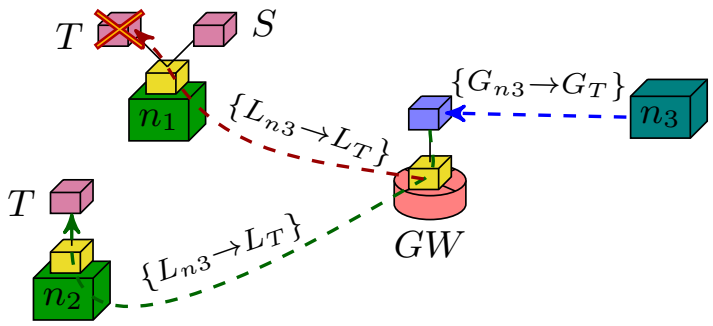


Figure 2.4: Service migration — seamlessly migrate to a backup temperature sensor on a sensor failure

to migrate the whole node and let the new node (n_2) use the original node's (n_1's) IP address because n_2 only provides a subset of services that n_1 supports. Thus, the traditional solution is inconvenient for users and application designers.

In MF-IoT, we can have the temperature sensor on n_2 take over the service T by inheriting T's GUID G_T and LUID L_T. In this way, the routing algorithm would find T's new location without any extra overhead from n_3 and GW.

Service caching

There are also cases that the more powerful devices can help lower-end devices cache the sensor readings, or low-end devices collaborate and cache for each other to save energy, which we refer to as *service caching* in MF-IoT. The caching node will listen for the specific service GUID and the source sensor can update the caching node with the same GUID.

Fig. 2.5 shows a local IoT domain containing 3 services (a humidity sensor service H from node n_1, a temperature sensor service T from node n_2, and a light sensor service L from node n_3). During normal operations (Fig. 2.5(a)), each of three nodes has its own power source and can serve data requests from other parts of the system (e.g., an infrastructure node n_4). When power outage happens (Fig. 2.5(b)), to extend the lifetime of the entire domain, they can elect a representative (n_1 in the example) and cache the latest readings from all three sensor services on n_1. n_2 and n_3 can then go to the sleep mode and wake up periodically to get the sensor reading and update the cache. In Fig. 2.5(b), n_2 is updating the cache and n_3 is in the sleep mode. Since

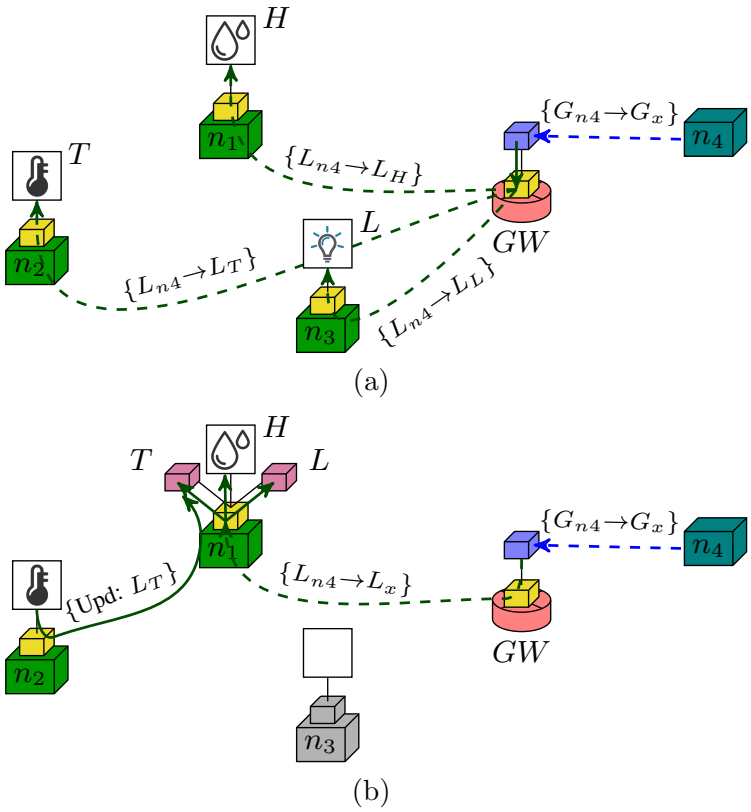


Figure 2.5: Service caching during power outage:(a)Before power outage (each device is serving its own sensor service) (b)During power outage (n_2 and n_3 can be turned off and update the caching service on n_1 periodically)

n_1's cache is listening for the LUID for T and L, the corresponding requests would be forwarded to n_1 and the caching service can respond with the cached value. When the battery on n_1 drops lower than a threshold, n_1 may also offload the caching service to other nodes (e.g., n_3) and go to the sleep mode. Of course, the caching service can also be placed on the gateway (GW).

2.3.5 Forwarding Service and Gateway Service

MF-IoT treats the basic network functionalities within a local IoT domain, such as forwarding and gateway, as services. This leads to easier topology management and better separation between application functions and network functions.

Fig. 2.6 illustrates a local IoT domain with 4 nodes ($n_1 - n_4$), each of which has some or no services, and a gateway (GW). Unlike the traditional solutions in which the services have to take care of neighbor discovery and routing, in MF-IoT, each node's forwarding service collectively performs these tasks. To send a packet, an application simply sends the data to its forwarding service and the network functions reside only in forwarding services. This clear separation would help developers focus on a specific part of the system.

When the embedded nodes move, this design can also help simplify topology management. For example, when n_4 moves away from n_1 and they cannot reach each other, this solution only has one link change (between forwarding service of n_1 and n_4) while the tradition solution would have 3 link changes.

On the gateway, we also separate the forwarding service (that relays packets for IoT nodes) from the gateway service (that translates between MF-IoT packets and MobilityFirst packets). Therefore, only the packet that will be forwarded out will go through the gateway service. This reduces the response time and energy consumption on the gateway and leads to simpler modification of routing algorithms on a gateway node.

Similar to popular WSN designs, MF-IoT separates neighbor discovery from routing. The forwarding service on the embedded nodes maintains two basic data structures — a neighbor table (see Table 2.3) and a FIB (see Table 2.4). We denote the forwarding

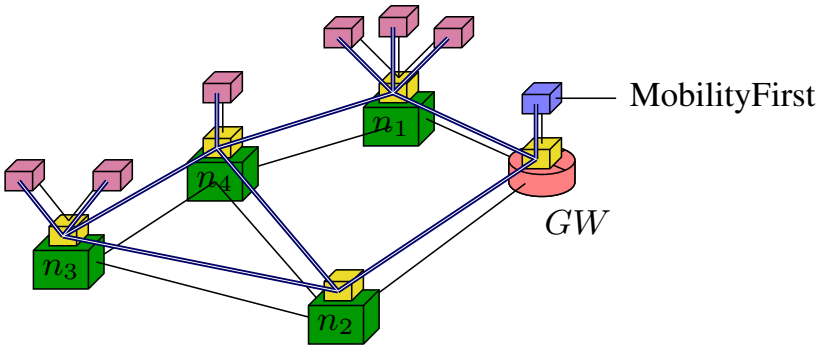


Figure 2.6: Virtual topology with forwarding as a service (virtual topology marked in blue)

Table 2.3: Neighbor table on n_{4F}

Neighbor	Identity
—=—=— L_{n1F}	MAC(n_1)
L_{n2F}	MAC(n_2)
L_{n3F}	MAC(n_3)
L_{n4S1}	PID($n4S1$)

Table 2.4: FIB on n_{4F}

Destination	Next Hop
—=—=—=— L_{n3S2}	L_{n3F}
L_{n4S1}	L_{n4S1}
L_{MF1}	L_{n1F}
...	...

service on node n_i as n_{iF} and the application service on node n_i as n_{iSj} in the tables. The neighbor table maintains the direct neighbors in the virtual topology. In the example, the forwarding service on n_4 has 3 neighbor forwarding services with LUID $L_{n1F} - L_{n3F}$ (the first 3 rows in the table). Since the application service on n_4 (L_{n4S1}) is also a neighbor to the forwarding service, we require the neighbor table on n_4 to maintain an extra entry that maps L_{n4S1} to the Process/Thread ID (PID) of the service (the last row). The FIB maintains the next hop(s) for each active LUID.

To forward a message, the forwarding service would first get the next hop(s) for the destination LUID according to the FIB, and then forward the packet either through wireless media or through Inter-Process Communication (IPC) according to the neighbor table. According to Tables 2.3 and 2.4, if the destination of the packet is a service on n_3 (L_{n3S2}) the forwarding service on n_4 would forward it to the forwarding service on n_3 (1st row in FIB) through layer 2 with MAC($n3$) (3rd row in neighbor table). If the destination is the service on its own device, the forwarding service would know $n4S1$ is a direct neighbor (2nd row in FIB) and forward it though IPC (4th row in neighbor table). If the destination is a node not in the domain (e.g., $MF1$), the FIB

would have an entry pointing towards the gateway (to $n1F$, 3rd row in FIB) and it will be forwarded to the MAC address of n_1 (first row in neighbor table).

Here, the forwarding service takes care of neighbor discovery and forwarding, and the update of the FIB is performed by the routing module (see §2.3.7). An extra TTL field can be added to the table to allow soft-state neighbor and routing management,

With the forwarding and application services separated, one can better focus on either component without interfering with the other.

2.3.6 Additional Communication Patterns

As described in §2.3.3, MF-IoT can support direct device to device communication (both intra- and inter-domain) and the communication between devices and infrastructure nodes. In this subsection, we describe additional communication patterns that are supported in MF-IoT.

Multicast

Since we use service-based GUIDs which are independent of any specific node, everyone in the same local IoT domain can listen to the same service GUID. Therefore, multicast can be supported naturally in MF-IoT and we further lump unicast and multicast together and refer to them as a *to-all* service. The forwarding service on the branching point would have more than one entry in the FIB for a GUID if there are more than one receiver. It then replicates the packet and sends a copy to each next hop (either it is on another node or an application in the same node). MF-IoT also takes advantage of the broadcast media all the wireless nodes are using. When the number of next hop nodes is larger than a threshold, a node can broadcast the packet instead of replicating and sending the packet multiple times. The next hop nodes will look up their FIB and discard the packet if no matching entry is found.

Anycast

In addition to unicast and multicast, MF-IoT also supports anycast. The listeners in anycast work in the same way as in multicast — they would listen to the same GUID

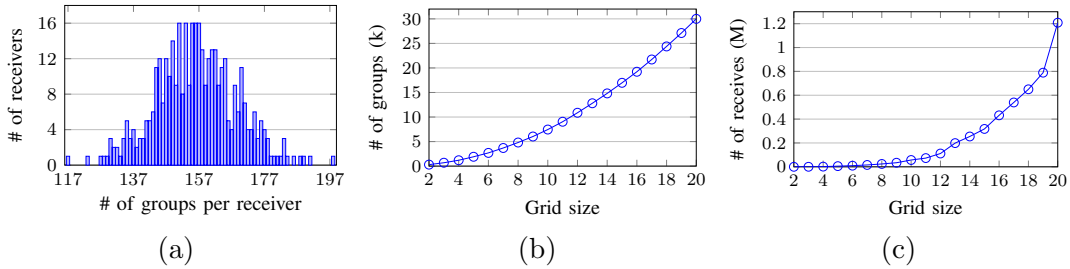


Figure 2.7: Intra-domain simulation setup. (a)Groups receiver distribution (b)Groups vs. grid size (c)Receives vs. grid size

and a tree would be formed by the routing protocol either proactively (e.g., OSPF-like) or reactively (e.g., AODV-like). When sending an anycast packet, the sender would place a different `SVC_TYP` value in the packet header and the intermediate nodes would only forward it to one of the next hop nodes based on its policy (e.g., shortest path, to a node with highest energy level, etc.).

"Observe" mode

According to [12], the observe mode is important for WSN and IoT applications. In this mode, the observer registers a specific event at a sensor and when the event is detected, the sensor notifies the observer. Usually one registration in the observe mode can get multiple notifications from a single sensor.

The observe mode can also be supported in MF-IoT, and furthermore, we can provide additional mobility handling and multicast support. The observers (either in the same local IoT domain, in the core network or even in different local IoT domains) can listen to the same specific GUID. When an event is triggered, the subject can send the notification to all the receivers through multicast. With the mobility support and an inherent push model, we allow the notifications to be sent in a timely and efficient manner.

2.3.7 Routing

MF-IoT does not restrict the routing in the network. The application designers can feel free to use any existing routing mechanism or design their own according to the

communication pattern they envision. Here, we suggest several mechanisms that we have in mind:

RPL [15] is widely used in the existing IoT systems for home and building automation [16]. The solution builds a tree among the nodes and usually the gateway is seen as the root. It can provide easier management with lower memory and energy consumption thanks to the tree topology. For applications which mostly depend on sensor to gateway and sensor to infrastructure communication, the solution has its benefits since all the traffic has to go through the gateway. MF-IoT can also adopt such kind of routing — RPL algorithm can run as a separate module and modify the FIB of on the forwarding engines. The data plane does not need to be modified.

AODV [17] is used by Zigbee [18] as the default routing. It provides on-demand distance vector routing to accelerate the direct sensor-to-sensor communication (they do not need to go to the root of the tree as RPL). However, to find a path on demand, a request has to be flooded in the whole network which made the solution less efficient when the network is large. AODV also can be used in MF-IoT in small domains for direct communication.

With the advent of Software Defined Networking (SDN), the concept of a central controller eases the traffic engineering and policy enforcement in the network. At the same time, it allows the forwarding engines to be simpler and more efficient. This concept can also be used in IoT world since the gateway usually has more resources, no power constraints and possibly larger radio coverage. The sensors can report the link changes to the gateway and after calculation, the gateway will send the forwarding rules back to the sensor nodes either proactively or on demand. This solution can reduce the amount of flooding in AODV, and support efficient sensor-to-sensor communication compared to RPL. It also has the flexibility to support communication based on policies and resource constraints on the sensors. At the same time, the sensors do not have to calculate the routing and it can save energy in resource constraint nodes. We will use this kind of routing as the default routing for MF-IoT in the evaluation.

2.4 Evaluation

To evaluate the performance of MF-IoT, we modified our event-driven simulator that was used in [19, 20], to represent typical IoT usecases. In the evaluation, we compare MF-IoT with IP and another state-of-the-art IoT architecture that is based upon a clean-slate Named Data Network (NDN) future Internet architecture. We will show that compared to IP and NDN, MobilityFirst provides a better support for IoT.

Specifically, We consider three scenarios in the evaluation: intra-IoT domain communication, IoT device mobility, and communication between embedded devices and infrastructure nodes.

2.4.1 Intra-IoT Domain Device-to-Device Communication

Simulation setup

We first report the performance of MF-IoT within a local IoT domain. We simulate a domain that has 400 sensor nodes, forming a 20×20 grid, and a gateway (or proxy in the IP case). Each sensor node can communicate with 8 direct neighbors through 802.15.4, with bandwidth of $250kbps$. To perform stress tests, we first generate 30,000 communication groups each containing a sender and 1–50 receivers (therefore the traffic containing both unicast and multicast traffic). The number of receivers per group follows a Zipf distribution with α=0.35. As a result, each node in the network belongs to 117–198 such kind of groups (see Fig. 2.7(a)). We also generate a set of messages for each group. The number of messages per group also follows a Zipf distribution but with α=0.81 [21]. The trace then has 138,662 messages and 1,208,203 receive events. The size of the messages varies between 1 and 100 bytes, and follows a normal distribution with $\mathbb{E}$=50. The arrival of the messages follows a poison distribution with a mean inter-arrival time of 1/160 seconds.

We compare the following three networking solutions:

IP: We use UDP over 6Lowpan [5] to represent the state of the art IP-based IoT solutions. According to [22], we use RPL [15] as the routing mechanism. To send a message to multiple receivers, the sender would send multiple unicast messages. If a

message cannot fit into a 802.15.4 MTU (127 bytes), it needs to be segmented and re-assembled later.

NDN: NDN [23] uses a query/response model and therefore the receivers of a group have to poll the sender for updates in the group. We choose a polling period of 2 seconds to get a 1-second average latency, which might be acceptable for many event notification cases but still large for emergent real-time cases. To reduce the average latency, NDN has to poll more frequently and the network traffic would increase rapidly. This is an inevitable trade off in NDN. Unlike IP, it can form a temporal multicast tree if the requests have temporal locality. We place a 10kB Content Store on each sensor node. NDN packet uses TLV format as described in [24] and we do not place signature and signed info in the Data packet to make them feasible to be transmitted in 802.15.4. Similar to IP, if an NDN packet cannot fit into a MAC packet, segmentation and reassembling would happen on the end hosts.

MF-IoT: We consider two variations of MF-IoT in the evaluation — MF-IoT w/o multicast (MF-IoT-U) and full fledged MF-IoT. In MF-IoT-U case, we only use unicast feature in the network and to send a message to multiple receivers, the sender has to send multiple unicasts. We use centralized routing described in §2.3.7 in both variations.

We use the end-to-end latency as well as the aggregate network traffic transmitted by all the nodes as the performance metrics of our evaluation.

Simulation Results

To show each solution's performance trend, we use different grid sizes ranging from 2×2 to 20×20, and plot the number of groups and receive events for each grid size in Fig. 2.7(b) and Fig. 2.7(c). The performance results are reported in Fig. 2.8.

From Fig. 2.8(a), we observe that with larger grids, the average latency for each solution becomes larger since the sender-receiver pairs are farther apart. Among the four solutions, MF-IoT outperforms the other three. Specifically, MF-IoT-U caused minor congestion in the 20×20 grid and the average latency grows by around 15ms. However, the average latency in the IP solution grows even faster since the traffic has to go through the proxy. When the grid size reaches 13×13, the traffic load reaches the

capacity limit on the gateway, causing congestion. The average latency goes up to 18.32 seconds eventually. Here, we assume the proxy has an infinite queue so that IP would not drop packets. NDN does not cause serious congestion in the network thanks to its intrinsic flow balance design. However, due to the polling frequency, the average latency remains around 1 second, and when the network grows larger, the average latency goes up by around 100ms partially caused by some minor congestion.

Fig. 2.8(b) shows the aggregate network traffic generated by each solution. We observe that MF-IoT and MF-IoT-U generate much less traffic compared to IP and and the difference becomes more pronounced when the network size becomes larger. Finally, NDN causes a lot of wasteful traffic due to the polling mechanism.

To summarize, MF-IoT has achieved lower traffic overhead and average latency compared to other solutions. As a result, we believe that IoT systems that adopt MF-IoT will accomadate higher traffic load and larger network size than the other start-of-the-art solutions.

2.4.2 Communication between IoT and Infrastructure Domains

Next, we demonstrate that MF-IoT supports efficient communication between an embedded device and infrastructure, even when the infrastructure node is mobile. We consider the use case that involves a sensor node n trying to send data to an infrastructure node I once every 100ms. We report the latencies observed at I at different times in Fig. 2.9.

In the start phase of the simulation run, marked as "Initialization", n first requests I's LUID from the gateway, and sends the packet to the gateway. The gateway then conducts a remote GNRS lookup to find the network address for I. The overall latency for this initialization phase is around 550ms.

To deal with node mobility, each GNRS entry cached on the MobilityFirst router would expire after some time. Here, we set the expiration time as 5 seconds. We thus observe the reciever-side latency has spikes each time the cached GUID-to-NA mapping expires and the gateway has to perform remote GNRS lookups (marked as "GNRS lookups").

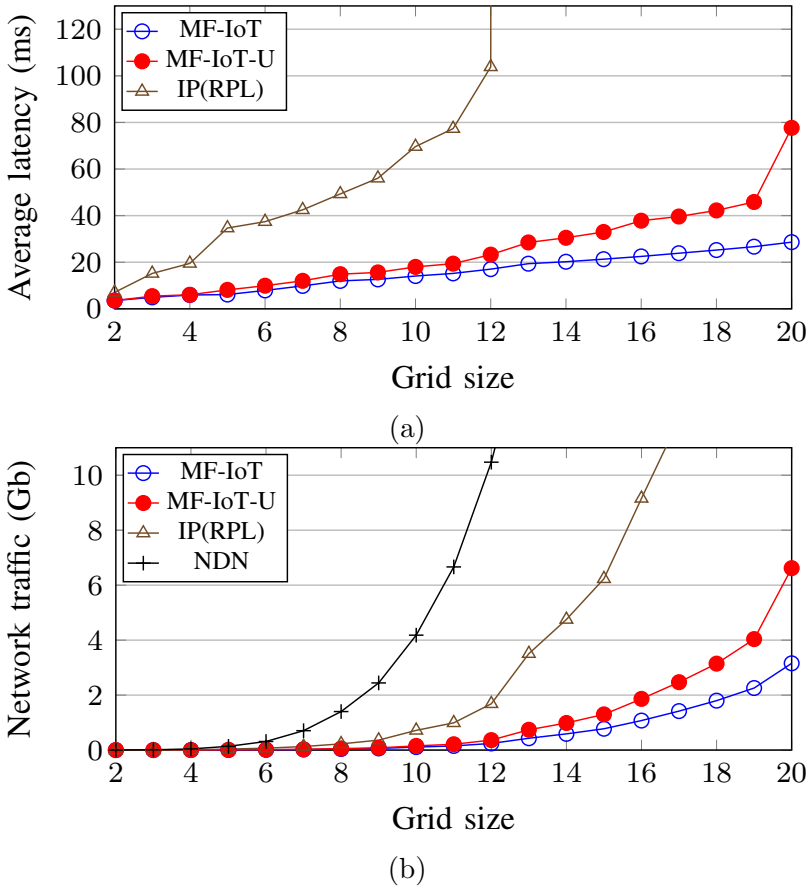


Figure 2.8: Intra-domain simulation with varying # of sensor nodes. (a)Average latency (b)Aggregate network traffic

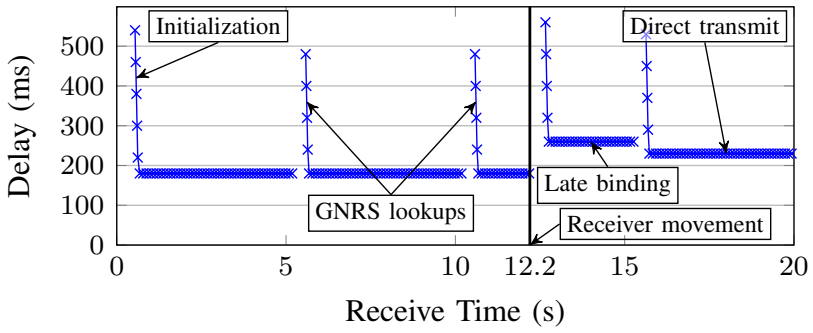


Figure 2.9: Sensor to infrastructure communication, infrastructure node (i) moved at 12.2s

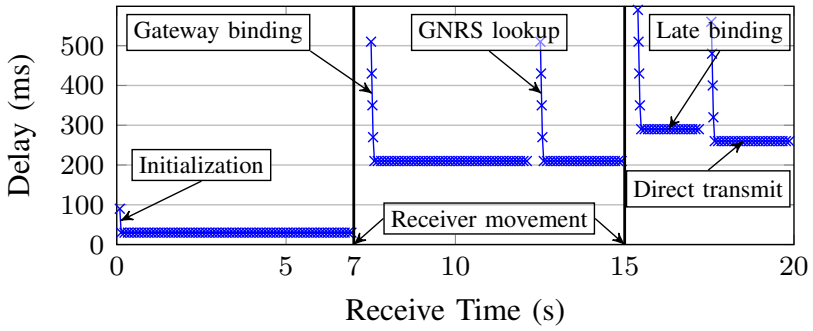


Figure 2.10: Sensor to sensor communication, receiver (n_2) moved at 7s, 15s

In the simulation setup, I moves to a different network at 12.2 second and the gateway is not aware of this. Therefore, the gateway would send the packet to the first-hop router towards I's original NA (denoted as R) where R performs late-binding. We assume that the movement happens instantly and R can obtain I's new NA through another GNRS lookup. With this additional remote GNRS lookup, the average latency at these times (marked as "Late binding") also increases.

Finally, we note that this entire process is transparent to the sensor node n.

2.4.3 Inter-IoT Domain Device-to-Device Communication

Finally, we demonstrate that MF-IoT can efficiently support communication between two embedded devices, even when one of them moves to a different domain. Specifically, we consider two embedded nodes n_1 and n_2, and n_1 sends a packet to n_2 every 100 ms. The receiver node n_2 is within the same local IoT domain as n_1, and it moves to another IoT domain at 7s, and to a third IoT domain at 15s. We plot the latency

observed at n_2 at different times in Fig. 2.10.

In the beginning of the simulation (marked as "Initialization"), n_1 requests n_2's LUID from the gateway, and its latency is rather low since they are in the same domain.

n_2 moves to another domain at time 7. Here, n_1 still sends the MF-IoT packet with n_2's original LUID, but the packet will be forwarded to the gateway as n_2 has left the domain. The gateway then performs a remote GNRS lookup for the NA of n_2's new gateway (marked as "Gateway binding"). From this point on, the gateway has to perform a GNRS lookup every 5 seconds as cached GNRS entries expire every 5 seconds. As a result, the latency increases around here.

When n_2 moves to a third domain at time 15, the gateway in the second domain would perform a late binding (like in the case of the infrastructure node movement), and the first gateway obtains the NA for n_2's latest gateway at time 17 (with a spike in the latency numbers).

This process is entirely transparent to both nodes. Finally, we note that in the two mobility usecases we have considered, it is the receiver that has moved during the communication. MobilityFirst-IoT also works seamlessly if the sender moves to a different domain, wherein the sender simply needs to register the reciever with its new gateway (see §2.3.3).

2.5 System Implementation

We have implemented MF-IoT over a realistic system to show its feasibility of real world deployment. In order to demonstrate a end-to-end system that runs actual application, we have also included MobilityFirst core network components. Our system implementation contains different modules including IoT network, IoT applications, gateway, MobilityFirst network, camera applications and client applications. In Fig. 2.11, we show both the logical relationship among GUIDs (application layer) and how the packets are routed in the physical network (network layer). Next, we will decribes our system implementation by following the application message flow.

We have implemented the MF-IoT protocol on Atmel SAM R21 XPro with RIOT

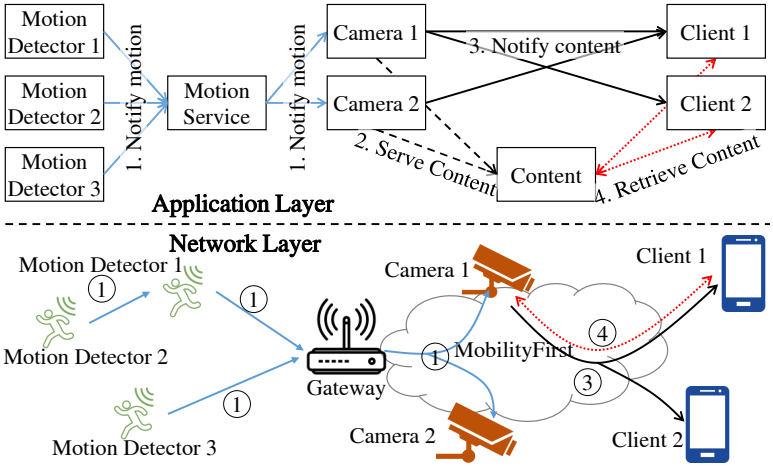
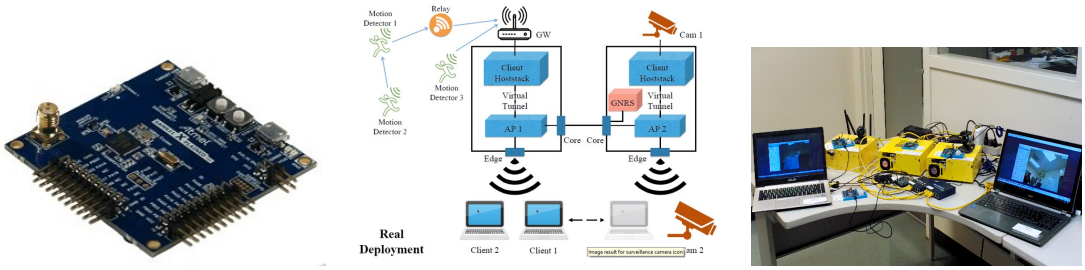


Figure 2.11: Communication diagram in different views

OS (shown in Figure 2.12(a)). Each board is used as a micro-controller and network adapter for a PIR sensor, or as a relay. The MF-IoT is implemented as a network layer module in RIOT similar to 6LowPAN. The module listens to the normal MobilityFirst `send` packets from the application layer, performs GUID to LUID mapping and send MF-IoT packets over 802.15.4. On receiving MF-IoT packets, the module would lookup the local routing table and forward it to neighbor(s). The module would also map the LUID to GUID and forward a MobilityFirst packet to the applications listening to the GUID.

The gateway is implemented as an IoT board connected to a PC via USB. The gateway application on the board reads packets from MF-IoT module and prints the binaries to the debug console. The PC reads the debug output via the USB and reconstructs MobilityFirst packets accordingly. The MobilityFirst packets would be sent to the core network and reach the destination(s) listening to the dst GUID. The gateway also manages the GUID-LUID mapping in the domain.

The application running on motion detector node listens to the motion sensor via UART. During the period that motion is detected, the applications send MobilityFirst messages periodically to a *GUID* representing Motion Service (step 1 "Notify Motion"). The MF-IoT module performs the GUID to LUID translation and forwards the packets based on the routing. The packets will be relayed on the intermediate nodes and reach the gateway eventually. They will be translated back to MobilityFirst packets on



(a) (b) (c)

Figure 2.12: (a) SAM R21 Xpro (b) Demo overview (c) Realistic setup

the gateway and be forwarded to the application(s) listening on this service (cameras). With the motion service GUID, the cameras do not need to change their reconfigurations when a new motion detector is added to the system.

On receiving the notification from the motion detectors, the cameras would start recording the video until no notification is received for a timeout period. The cameras would combine a certain number of frames into a chunk and get a content GUID from Name Certification Service (NCS) for each chunk. After saving the chunk, the camera would notify GNRS that it is serving the chunk GUID (step 2 "Serve content" in Fig. 2.11). The camera would then send a notification to the GUID of the camera service. The content GUID is placed in the payload of the notification. Whoever is interested in (or has the right to receive) the camera data would listen to the camera service GUID and get the corresponding notifications (step 3 "Notify content" in Fig. 2.11). Note that we allow different cameras to create different services to enforce the policy and/or satisfy user interests. In the figure, Client 1 is interested in both cameras while Client 2 is only allowed to see Camera 2. Therefore, Client 1 listens to GUIDs of both cameras and Client 2 only listens to GUID of Camera 2.

When a client receives the notifications from the camera service(s), he would look into the payload and get the GUID of the content. He can then query the MobilityFirst network with the content GUID whenever he wants to watch the captured video (step 4 "Retrieve content" in Fig. 2.11). Similar to other Information-Centric Networks (ICNs), MobilityFirst would route the request to the nearest content provider or even get the content from the cache in the network.

2.5.1 Demo Setup

We would use the normal surveillance camera function to show the feasibility and efficiency of the design. During the demo, we would also dynamically adjust the network to demonstrate the flexibility of using service-oriented communication. We have deployed our demo in a network that constructed by 4 SAM-R21 Xpro sensor nodes and 5 linux machines(shown in Figure 2.12(a) & (b)).

Scenario 1 *Normal Surveillance Camera Function:* We use 3 motion detectors and several relays to form IoT domain (Fig. 2.11). Due to the space limit in the demo site, we pre-configured a virtual topology so that the IoT nodes would only accept the packets from neighbors. The core MobilityFirst network comprises 2 MobilityFirst routers and several end-hosts (cameras and clients). We showed that when any of the motion sensors detect the motion, messages would be sent (via relays) to the gateway using MF-IoT over 802.15.4. A sniffer node would be placed to show how the packets and sent among the IoT devices. On the gateway, we would capture both the MobilityFirst and MF-IoT packets to explain how the translation is performed. The cameras gets the messages via MobilityFirst multicast and generate contents with the images they capture. Via the log on the cameras, we showed the messages the cameras receive, the new content GUIDs they create and the requests form for the contents. On the client side, we allowed the clients see the video immediately when the video is generated (real-time mode). The clients can also choose to play the video recorded earlier (playback mode).

Scenario 2 *Dynamic Configuration Adjustment:* In the demo, we use motion service and camera service GUIDs. When we add a new motion detector or use another camera to replace the existing one, no configuration would be needed. We demonstrated how the network enables the automatic adjustment.

2.6 Related Work

In this section, we discuss related work in state-of-the-art IoT system and architecture design.

2.6.1 State of the Art IoT Architectures

Existing work on IoT systems can be broadly classified into two categories: network adaptation for constrained devices and application-layer approaches for resource accessibility and manageability. Network adaptation solutions like 6LoWPAN [5] and ZigBee IP [9] compress the packet header to allow IPv6 (with MTU $\geq$ 1280 bytes) to operate on resource-limited networks like IEEE 802.15.4 (with only 127-byte MTU). However, the tight coupling between identifier and locator in IP makes it difficult for these solutions to provide efficient mobility support.

To deal with resource mobility, studies in the second category seek solution in the application layer. Constrained Application Protocol (CoAP) [10] proposes a specialized web transfer protocol to cope with constrained nodes and networks. It provides a query/response interaction model between endpoints, where resources could be accessed via URIs. State of the art IoT platforms such as IoTivity [3] usually involve generic interfaces to accommodate different lower layer protocols and a centralized server to facilitate efficient resource retrieval. The downside of these overlay approaches is that they usually rely on a server, which is an additional deployment overhead. Also, they are not well suited to support event notification or pushing type of communication pattern.

As a result, we believe that in order to support next-generation IoT systems, we need to consider a network architecture that can naturally support IoT's inherent demands, one that is fundamentally different from IP.

2.6.2 Information-Centric Networking and Its Use in IoT

Information-Centric Networking (ICN) is a clean-slate Internet architecture, which is proposed to evolve the network from host-centric to content-centric model where data are identified and accessed by names. Named Data Networking (NDN) [23] (or Content-Centric Network (CCN) [25]) is one of the popular ICN proposals. It uses human-readable, hierarchically-structured Content Names as the identity in the network. NDN provides a query/response communication pattern by using two types of packets in the

network: Interest (query) and Data (response). It also introduced a new forwarding engine model having three data structures — Forwarding Information Base (FIB) which maintains the outgoing (inter)faces for each name prefix; Pending Interest Table (PIT) which keeps the unsatisfied Interests and there incoming (inter)faces; and Content Store working as an in-network cache. Data consumers issue Interests with Content Names. The intermediate forwarding engines forward the Interest towards Data provider by according to FIB. Bread crumbs are left on the path via PIT. On receiving an Interest, a node that provides the requested Data (either an intermediate caching router or a Data provider) can reply the Interest. The Data packet travels through the reverse path according to PIT and it consumes the entries in the PITs.

Work by Zhang *et al.* [26] defines several architectural requirement for the IoT including global accessible name and mobility, which indicates ICN has the potential to be used as the underlying network for IoT since it integrates named-based routing, compute, and caching/storage as part of the network. To adapt to the resource-constraint devices, CCNLite [24] is proposed as a lightweight implementation of NDN which simplifies the original code base and data structure. In order to support multi-source data retrieval, work in [27] proposes a framework that multiple data producer can answer to the same Interest packet. However, the similar to NDN, these solutions only focus on data retrieval, and it is difficult for them to achieve functions like event notification and multicast in IoT. Moreover, NDN requires the Data packet to be transmitted on the reverse path as the Interest, it causes difficulties in IoT where links might be asymmetric. The need for PIT and Content Store also puts burden on storage-constraint devices.

Work in [28] proposes a generic IoT middleware architecture based on NDN and MobilityFirst to support basic IoT function such as service discovery and naming service. These functions can also be used in MF-IoT in the application layer and is orthogonal to the design in this paper.

2.7 Conclusion

In this chapter, we propose MF-IoT, a generic network architecture that satisfies the requirements placed by the emerging IoT systems, namely, global reach-ability, mobility, communication diversity, and resource efficiency. We achieve this goal by creating a network-layer dialect (Local Unique IDentifier, LUID) in a local IoT domain and adopt a gateway to efficently translate between GUID's that are used in the core network and the corresponding LUID's. The translation is transparent to the applications so that MF-IoT can have efficient global reach-ability and mobility support. With service-based GUID assignment, we further enable seameless service migration, service caching and the separation between application logic and network functions. Our simulation results show that MF-IoT can greatly improve the performance of IP-based and NDN-based solutions.

Chapter 3

Aggregating IoT Traffic in The Edge Network

3.1 Introduction

The rapid growth of the Internet of Things (IoT) will inevitably lead to a large volume of IoT data, which will in turn dramatically increase the number of packets that are injected to the Internet. For example, a city scale IoT system – such as an environmental monitoring system or intelligent transportation system – may involve millions of embedded/mobile devices. These devices frequently collect data and transmit them to the distant cloud for processing and storage. According to [29, 30], 75 % of IoT sessions transfer less than 1 KB data, while the number of connected devices will reach 70 billion by 2025. The large volume of small traffic generated by these IoT systems thus poses a great burden to the network as well as the cloud.

The IoT traffic exhibits unique features that are quite different from traditional network traffic. Firstly, IoT packets are usually small in size. Work by Sivanathan et. al [29] indicates most of the IoT packets in smarthome and enterprise applications are less than 500 bytes. Secondly, since the data is generated by the sensors that are often geographically co-located, there is tremendous redundancy within the data. Improperly handled, the IoT packets can potentially lead to very poor utilization of the network bandwidth as well as the storage resources. In this chapter, we propose to address these new challenges – i.e., large volume, poor utilization, etc., – by aggregating the IoT packets along their transportation paths through in-network aggregation.

In-network aggregation targets at combining multiple data packets from various sources (that usually share the same destination) and generating summary data for these packets [31–33]. It can enhance the network utilization and efficiency by reducing the total number of packets. Usually, specific aggregation functions are determined by

the application providers, based on the unique properties of the data that are collected by the application. As a result, we do not focus on the actual design of the aggregation function in this chapter. Instead, we set out to investigate where in the network we should deploy the aggregation service, with the objective of minimizing the overall IoT traffic. Given a certain capacity of the total aggregation services, a good placement strategy must be able to balance the traffic reduction due to aggregation and the traffic inflation due to the extra routing needed to reach the aggregation service. Furthermore, this problem becomes much more challenging when we consider the fact that an IoT system main consist of a large number of (i.e., thousands or more) sources.

The problem of optimizing network traffic has been largely overlooked by earlier studies on in-network aggregation. Existing studies mainly focus on optimizing energy efficiency in the Wireless Sensor Network (WSN) domain and achieving state synchronization in complex distributed systems. For example, aggregation solutions in [34, 35] create clusters to aggregate data via a time division based Media Access Control (MAC) protocol to achieve optimal energy consumption, without considering the network traffic efficiency. As another example, studies in [36, 37] propose state synchronization solutions by aggregating the information that is shared by all nodes in the system.

Meanwhile, Mobile Edge Clouds (MECs) loom on the surface and aim to provide computing services at the proximity of mobile users to facilitate low-latency context-aware IoT applications. Through MECs, computation and network traffic are offloaded from the application provider's central clouds [38] to the Internet edge. Leveraging this nature, we envision that MECs can help aggregate the IoT packets. In addition, we can offer the abstraction of IoT data with different granularity levels, to satisfy both local subscribers and remote subscribers. For example, the finer-grained data generated by roadside traffic sensors can be used to feed the local traffic management systems, while much coarser-grained data may be needed by a remote traffic analytic application.

In this work, we propose a generic data aggregation system over mobile edge clouds – AggMEC for any application that needs aggregation on unspecific network topology. By introducing the cost function of a aggregation network, we unveil a clustering-based strategy of aggregation nodes placement and data sources assignment, which lead to

minimizing the total network traffic. In order to support such aggregation scheme, we design and implement our routing layer over a clean-slate Future Internet Architecture (FIA) – MobilityFirst (MF) [39], which can reduce the overhead of obtrusive address resolution in the current IP network.

The contributions of AggMEC are as follows:

- We proposed an architecture for leveraging MECs to support efficient data aggregation across the distributed network environment.
- We conducted detailed analytic studies that unveil the aggregated IoT traffic model.
- We introduced two heuristic machine learning based assignment and placement algorithm using the cost function as distance metric, that can approximate the minimum cost of the total network traffic.
- We proposed an application-specific routing protocol that directs data flows to their desired destinations with low control overhead.

The chapter is structured as follows: Section 3.2 details the system components overview and the application-specific routing protocol over MobilityFirst. Section 3.3 studies the analytic model of the aggregated IoT traffic. We introduces the placement schemes in Section 3.4. Section 3.6 provides related works on data aggregation and mobile edge clouds. Finally, we evaluate the proposed algorithms efficiency and effectiveness for various network setting in Section 3.5.

3.2 System Overview

In this section, we first introduce AggMEC's system design overview. Next, we define a generic network traffic cost function for our aggregation network, and unveil an placement and assignment algorithm to approximate the minimum network traffic with given computing resource. Finally, we propose a clean-slate network architecture MobilityFirst which can support our system efficiently.

3.2.1 System Overview

Figure 3.1(a) demonstrates an overview of the proposed system. We envision AggMEC can reside on any MEC architecture, where virtual machines or containers are one/two hops away from the data sources. Here, a data source could be a NB-IoT sensor, a smartphone, or a WSN gateway, which has direct connection to edge networks. AggMEC involves two key components: computing node and application specific routing (ASR) controller. The former aims at providing in-network computing service on the fly, while the later computes and control the data flows base upon application criteria.

Computing Node: Any available MEC can be a computing node. It is capable of performing both routing operation and in-network computing operation. When it routes a packet, this packet will be forwarded to the computing layer if a service ID flag is turned on. Each application/service/data flow could have it own data handler indexed by a persistent ID or URL. Such data handler can be initiated by application/service controller. Specifically, for an aggregation service, we envision a data handler need to have four modules: 1. a decoder to interpret network packets into specific data format; 2. a data queue to store aggregated data for a certain period of time; 3. a computing function to generate data summary over the data queue; 4. an encoder to convert the data summary back into network packets. Once a computing node is selected as a aggregation node, we call it **AggNode**.

Application Specific Routing Controller: The main objective of ASR controller is to control the data flow routing based on application requirements. Here, we explain the procedure of the aggregation service in the ASR controller. As we will show latter in Section 3.3, the cost between data sources and their first AggNodes depends on the data rate and their hop counts, hence the very first step is to sample the data rate at the data sources. Secondly, ASR perform our algorithm with the knowledge of available AggNodes, network topology, and the data rates to find the optimal computing routers to run our aggregation service. Next, it generates the routes between the data sources and the server. Finally, the control message is disseminated to the network to set up the paths.

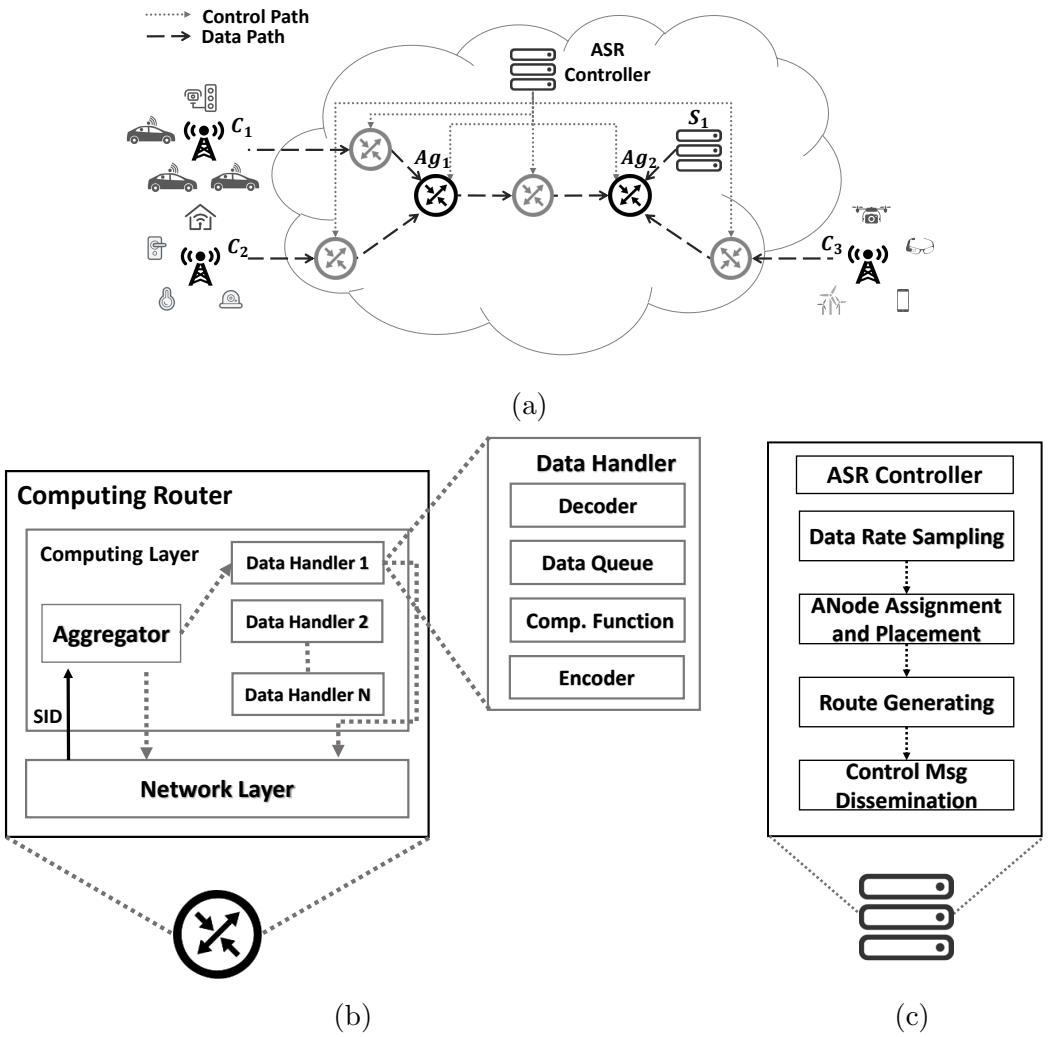


Figure 3.1: (a) Computing router overview. (b) An example procedure within ASR controller.

3.2.2 System Implementation

To implement a such aggregation network in the current IP network, data sources may use persistent service URL to push packets towards ANodes through HTTP protocol instead of using IP address. By introducing service URL, we can decouple with any physical device locator, which grands the flexibility for the data sources as the IP addresses of the ANodes may change from time to time. It also enables data flows being aggregated for multiple times without alternating the destination address. However, it requires http session establishment as well as high Domain Name Service (DNS) lookup

overhead, which is not scalable in a high-dense network. As a result, we decide to implement our aggregation network over a clean-slate network architecture-MobilityFirst [39], which naturally integrates with the similar idea of persistent service URL.

MobilityFirst is proposed as future internet architecture with mobility and global accessibility. To achieve these features, MobilityFirst introduces several components into the network.

Globale Unique Identifier (GUID): MobilityFirst utilizes persistent Global Unique Identifier (GUID) to name services/devices/applications in the network. The separation between the identifier (GUID) and locator (NA) provide support for mobility and global accessibility. Furthermore, GUID can be a publice key derived from the properties or the name of a object, which allows the objects to be self-certified.

Global Name Resolution Service: MobilityFirst uses logical centralized Global Name Resolution Service (GNRS) to provide resolution from names (GUID) to routable Network Address (NA). MobilityFirst can perform late binding, *i.e.*, querying the GNRS whenever the destination NA is not available in the local scope. GNRS is proposed as a network layer service and is transparent to application layer. Work in [13,14] introduce distributed solution with acceptable scalability and lookup performance in the core network.

Routing: MobilityFirst routes packets based on NA(s). Work in [6] proposes a storage-aware intra-domain routing scheme which is similar to Open Shortest Path First (OSPF) routing. Work in [40] proposes an edge-aware inter-domain routing. Both of them have shown that MobilityFirst can provide scalable and efficient routing for the next generation mobile network.

Service ID (SID): MobilityFirst network utilizes service ID to support multiple network services such as multicast, unicast, anycast, and in-network computing. It is worthy to mention that SID is different from service GUID. It is in a flag field in MobilityFirst packet header to indicates the router to perform the corresponding operation, while service GUID is in the destination field .

3.2.3 Aggregation Routing Protocol

With the help of the features we list above, we design a application-specific routing protocol for the proposed aggregation network over MobilityFirst. We use a virtual aggregation service GUID (G_{AS}) to represent this service. We also assign GUIDs to application clients that need this service, ANodes, and destination servers. ASR controller adopts the algorithm we propose in Section 3.3 to compute and assign the first ANode for each group of the data source. Although we have shown that the major gain of aggregation comes from the first ANode, we aim at designing a routing protocol that allow a data flow to encounter more than one ANode. Since a single service GUID ((G_{AS})) may associate with multiple network entities (ANodes or servers), we use the hash of self-GUID and service-GUID as the key to ensure only one dedicated next destination for each query. For each segment of the route, ASR controller creates a key-value pair $< Hash(Self-GUID, Service-GUID), Next-GUID >$ and populate it into GNRS. During the packet routing phase, each ANode lookups the value of the key: $Hash(Self-GUID, Serice-GUID)$ in GNRS if there is no cache locally. We next depicts an example procedure according to Figure 3.2.

1. Client G_{C_1} sends a data packet to aggregation service (G_{AS}) and set the source GUID as G_{C_1}.

2. C_1's first hop router use the hash of (G_{C_1}, G_{AS}) to lookup the next ANode/destination in GNRS. GNRS returns G_{Ag_1} as a result.

3. Its first hop router send to G_{Ag_1} by using Open Shortest Path First (OSPF) routing.

4. When ANode Ag_1 receives the packet, it appends the packet content into the data structure indexed by G_{AS}. When it reaches the preset time window, it performs the computing function over this data structure and generates a corresponding meta data.

5. Ag_1 sends this packet to G_{AS} and repeat step 2, 3, and 4 until it reaches the server G_{S_1}

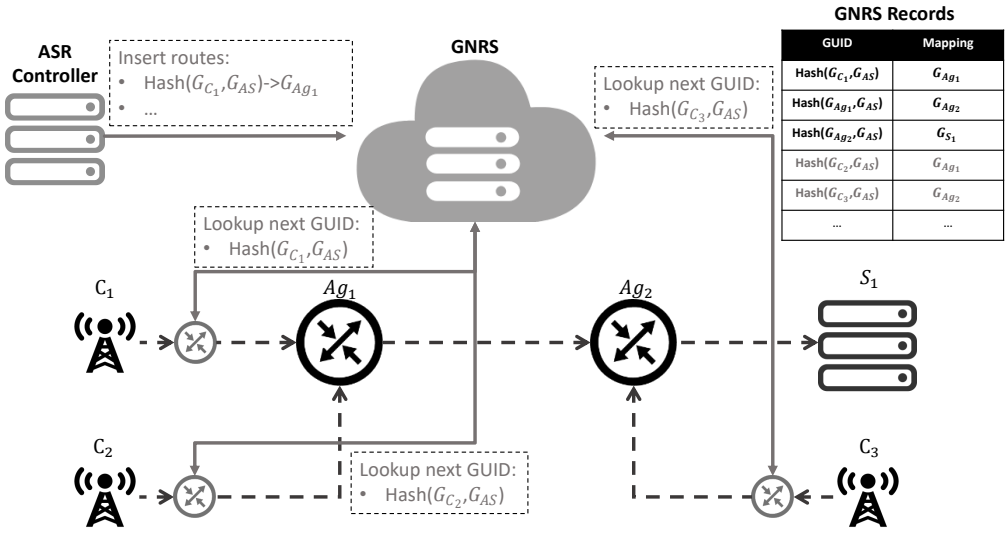


Figure 3.2: Routing example.

Similarly, C_2 and C_3 can follow the same rule to send their data flow through the ANodes computed by ASR controller.

3.3 Model Analysis

We envision a many-to-one data collection scenario with single data destination node **DN**, multiple intermediate computing nodes **CN** and multiple data sender nodes (**SN**). We can set each computing node either as an **AggNode** or a router node(**RN**). Each data flow may pass more than one **AggNode**. Each **AggNode** can aggregate multiple flows at a time. In order to host multiple data collection service concurrently, we identify each data collection service with a persistent ID, hence multiple data collection services can co-exist at the same **CN**(s).

First we introduce some auxiliary notations used to describe the network. Let $\mathcal{S}$ be the set of all nodes of network $\mathcal{N}$, $\mathcal{S}_S$ be the set of all **SN**s, $\mathcal{S}_C$ be the set of all **CN**s, $\mathcal{S}_A$ be the set of **AggNode**s and $\mathcal{S}_R$ the set of all **RN**s. Thus, $\mathcal{S} = \mathcal{S}_S \cup \mathcal{S}_C$ and $\mathcal{S}_C = \mathcal{S}_A \cup \mathcal{S}_R$. Let $K_C = |\mathcal{S}_C|$, $K_R = |\mathcal{S}_R|$ and $K_A = |\mathcal{S}_A|$, where $|\Xi|$ is number of elements of set Ξ. Thus, $K_R + K_A = K_C$.

The data traffic is routed by the **CN**s such way that the data sent by any **SN** can

Table 3.1: Abbreviations and notations

symbol and abbreviations	denotion
AggNode, **CN**, **RN** and **SN**	*aggregation node, computing node, router node and sender node*
DN or s_0	*destination node*
$\mathcal{N}$	*network*
$\mathcal{S}$	*set of all nodes of network* $\mathcal{N}$
$\mathcal{S}_A$, $\mathcal{S}_C$, $\mathcal{S}_S$ and $\mathcal{S}_R$	*set of all* **AggNode***s,* **CN***s,* **SN***s and* **RN***s correspondingly*
T	*time window*
$\lvert\Xi\rvert$	*number of elements of set* Ξ
K_A, K_C and K_R	$\lvert\mathcal{S}_A\rvert$, $\lvert\mathcal{S}_C\rvert$ *and* $\lvert\mathcal{S}_R\rvert$ *correspondingly*
$\mathcal{S}_C = \{s_{K_C}, \ldots, s_1\}$	*set of* **CN***s for network with bus and star topology*
$\mathcal{S}_A = \{s_{a_{K_A}}, \ldots, s_{a_1}\}$	*set of* **AggNode***s for network with bus and star topology*
$d(s, \tilde{s})$	*number of hops employed in the network to get node* $\tilde{s}$ *by data forwarded from node* s
$x(s)$	*rate of sending data by* **SN** s
$X(s)$	*rate of total output traffic traffic re-forwarded by* **RN** s
$\mathrm{CN}(s)$	*the first of* **CN***s such that data sent by* **SN** s *to* **DN** *passes through*
$\mathcal{S}_C(s) = \{s_{K_C(s)}, \ldots, s_1\}$	*subset of* $\mathcal{S}_C$ *consisting only of such* **CN***s, that data traffic, forwarded by* **SN** s *to* **DN***, passes through*
$\mathcal{S}_A(s) = \left\{s_{a_{K_A(s)}}, \ldots, s_{a_1}\right\}$	*subset of all* **AggNode***s of set* $\mathcal{S}_C(s)$
$K_A(s)$ and $K_C(s)$	$\lvert\mathcal{S}_A(s)\rvert$ *and* $\lvert\mathcal{S}_S(s)\rvert$ *correspondingly*
$\mathrm{AN}(s)$	*such* **AggNode** *from* $\mathcal{S}_A(s)$ *where data sent by* **SN** s *is aggregated the first time*
$\overline{\mathcal{S}}_A$	*subset of* **AggNode***s such that each of them does not get in input data traffic any data aggregated by other* **AggNode***s*
$\mathcal{N}(S)$	*minimal sub-network of network* $\mathcal{N}$ *containing set of nodes* S *and* **DN**
$D(S)$	*the total number of hops in subnetwork* S
$\sigma_R(s)$, $\sigma_A(s)$ and $\sigma_S(s)$	*sets of all* **RN***s,* **AggNode***s and* **SN***s forwarding data flow to node* s *which does not pass through other* **RN***s or* **AggNode***s*
$C(\mathcal{S}_A)$	*total traffic in network if set of aggregation nodes is* $\mathcal{S}_A$
$C(s, \mathcal{S}_A)$	*individual end-to-end traffic generated by* **SN** s *toward* **DN** *if set of aggregation nodes is* $\mathcal{S}_A$

reach to the **DN** without any loop passing a sequence of **CN**s. Also, **SN** cannot sent data directly to **DN** without assistance at least one **CN**. We tell that node s and node $\tilde{s}$ are connected if data forwarded by node s to **DN** passes through node $\tilde{s}$. Denote by $d(s, \tilde{s})$ the number of hops employed in the network to get node $\tilde{s}$ by data forwarded from node s. If these nodes are not connected then let $d(s, \tilde{s}) = \infty$. We tell that a **CN** $s \in \mathcal{S}$ has *index* $i = \mathrm{idx}(s)$ if a data flow, to get to the **DN**, has to pass through $i - 1$ other **CN**s. Thus, **CN** with index one forwards data directly to **DN** without passing other **CN**s. For **SN** s, denote by $\mathrm{CN}(s)$ the first of **CN**s data sent by **SN** s to **DN** passes through. We assume that **SN** s sends data with rate $x(s)$. Such rate generates individual traffic $x(s)d\left(s, \mathrm{CN}(s)\right)$ of **SN** s between node s and $\mathrm{CN}(s)$

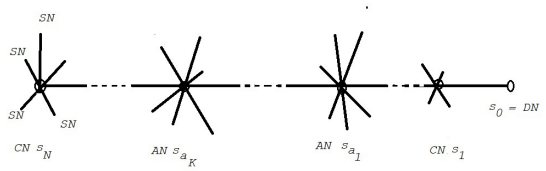


Figure 3.3: Bus line and star network topology

3.3.1 Bus line and star topology

To define cost function associated with installation of aggregation nodes in the network. let us first to consider an example of network $\mathcal{N}$ with a particular topology, namely, *bus line and star topology* (Fig. 3.3). A linear bus component of this topology consists of a main cable where **CN**s are sequentially attached to forward data in direction to the **DN**, while **DN** is attached as the root node of this line. Meanwhile, each **CN** serves as a root note for the nearby **SNs** to form a star topology. Let $\mathcal{S}_C = \{s_{K_C}, \ldots, s_1\}$ with $\text{idx}(s_n) = n$ for $n = 1, \ldots, K_C$. Let $\mathcal{S}_A = \{s_{a_{K_A}}, s_{a_{K_A-1}}, \ldots, s_{a_1}\}$ with $K_C \geq a_{K_A} > a_{K_A-1} > \ldots > a_1 \geq 1$.

Total network traffic

Each **AggNode** aggregates all the received data for every time window T, and sends 1 data out to the next **CN** in the direction to the **DN**. This results into an output traffic generated by the **AggNode** with rate $\frac{1}{T}$. Therefore, the total network data traffic consists of two parts: traffic of aggregated data and traffic of non-aggregated data. Let us consider each of these parts separately.

Traffic of aggregated data starts at **AggNode** $s_{a_{K_A}}$, passes sequentially through all the other **AggNode**s $s_{a_{K_A-1}}, \ldots, s_{a_1}$ and ends at **DN**. Thus, the total traffic of aggregated data is given as follows:

$$\mathcal{T}_{\text{agg}} = \frac{1}{T} d(s_{a_{K_A}}, s_0), \tag{3.1}$$

where

$$s_0 = \textbf{DN}.$$

Traffic of non-aggregated data combines all sub-traffics starting at a **SN** and ending either at such **AggNode**s where forwarded data are aggregated for the first time, or at **DN**. Thus, the total traffic of all non-aggregated data is given as follows:

$$\begin{aligned}\mathcal{T}_{\text{non agg}} = &\sum_{s\in\mathcal{S}_S:\text{idx}(s)>a_{K_A}} x(s)d(s,s_{K_A}) \\ &+ \sum_{s\in\mathcal{S}_S:a_i>\text{idx}(s)\geq a_{i-1},1\leq i\leq K_A} x(s)d(s,s_{a_i-1}),\end{aligned} \tag{3.2}$$

where $a_0 = 0$.

Summing up (3.1) and (3.2) implies that the total data traffic is given as follows:

$$\begin{aligned}C(\mathcal{S}_A) = &\frac{1}{T}d(s_{a_{K_A}},s_0) \\ &+ \sum_{s\in\mathcal{S}_S:\text{idx}(s)>a_{K_A}} x(s)d(s,s_{K_A}) \\ &+ \sum_{s\in\mathcal{S}_S:a_i>\text{idx}(s)\geq a_{i-1},1\leq i\leq K_A} x(s)d(s,s_{a_i-1}).\end{aligned} \tag{3.3}$$

This total network data traffic $C(\mathcal{S}_A)$ can be considered as the cost function of such network topology with set of aggregation nodes $\mathcal{S}_A$. In particular, for two boundary cases this cost function (3.3) can be simplified. Namely,

(a) if there is no aggregation nodes installed, i.e., $\mathcal{S}_A = \emptyset$, then

$$C(\emptyset) = \sum_{s\in\mathcal{S}_S} x(s)d(s,\mathbf{DN}), \tag{3.4}$$

(b) if each computing node is aggregation node, i.e., $\mathcal{S}_A = \mathcal{S}_C$, then

$$C(\mathcal{S}_C) = \frac{1}{T}d(s_{K_C},s_0) + \sum_{s\in\mathcal{S}_S:\text{idx}(s)=i,1\leq i\leq K_C} x(s)d(s,s_i). \tag{3.5}$$

Individual traffic generated by a sender node

We can also specify individual traffic generated by a sender node, namely, individual contribution of a sender node into the total data traffic. Note that before first data aggregation such contribution of a **SN** is defined by rate of data forwarded by the **SN**, while after aggregation contribution of the **SN** is given by its portion in output data

traffic generated by the **AggNode** with rate $1/T$. That is why, to derive individual traffic of **SN** s we have to consider separately two cases: (1) $\text{idx}(s) \geq a_{K_A}$ and (2) $\text{idx}(s) < a_{K_A}$:

Case (1): Let **SN** s be such that $\text{idx}(s) \geq a_{K_A}$. Then, traffic generated by **SN** s between s and **AggNode** a_{K_A} is

$$x(s)d\left(s, s_{a_{K_A}}\right). \tag{3.6}$$

The rate of total input data traffic at **AggNode** $s_{a_{K_A}}$ is $\sum_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})$. The **AggNode** aggregates all the input data and generates an output data traffic with rate $\frac{1}{T}$ to forward it to the next **CN**. Thus, the contribution of the **SN** s into such output data traffic of **AggNode** $s_{a_{K_A}}$ is

$$\frac{x(s)}{T\sum\limits_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})}, \tag{3.7}$$

and traffic of **SN** s from **AggNode** a_{K_A} to **AggNode** a_{K_A-1} is

$$\frac{x(s)}{T\sum\limits_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})} d\left(s_{a_{K_A}}, s_{a_{K_A-1}}\right). \tag{3.8}$$

The total input data rate in **AggNode** a_{K_A-1} is $\frac{1}{T} + \sum_{a_{K_A}>\text{idx}(\tilde{s})\geq a_{K_A-1}} x(\tilde{s})$ while output data traffic after aggregation has rate $\frac{1}{T}$. Therefore, by (3.7), the contribution of the **SN** s into this output traffic is

$$\frac{1}{T}\frac{\dfrac{x(s)}{T\sum\limits_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})}}{\dfrac{1}{T} + \sum\limits_{a_{K_A}>\text{idx}(\tilde{s})\geq a_{K_A-1}} x(\tilde{s})} \tag{3.9}$$

and the traffic of **SN** s from **AggNode** a_{K_A-1} to **AggNode** a_{K_A-2} is

$$\frac{1}{T}\frac{\dfrac{x(s)}{T\sum\limits_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})}}{\dfrac{1}{T} + \sum\limits_{a_{K_A}>\text{idx}(\tilde{s})\geq a_{K_A-1}} x(\tilde{s})} d\left(s_{a_{K_A-1}}, s_{a_{K_A-2}}\right) \tag{3.10}$$

and so on. Thus, individual traffic generated end-to-end to **DN** by **SN** s such that $\text{idx}(s) \geq a_{K_C}$ is

$$\begin{aligned} C(s,\mathcal{S}_A) &= x(s)d\left(s, s_{a_{K_A}}\right) \\ &+ \frac{x(s)}{T \sum\limits_{\text{idx}(\tilde{s})\geq a_{K_A}} x(\tilde{s})} \\ &\times \sum_{j=0}^{K_A-1} \frac{d\left(s_{a_{K_A-j}}, s_{a_{K_A-j-1}}\right)}{T^j \prod\limits_{r=1}^{j} (\frac{1}{T} + \sum_{a_{K_A-r+1}>\text{idx}(\tilde{s})\geq a_{K_A-r}} x(\tilde{s}))}. \end{aligned} \tag{3.11}$$

Case (2): Let **SN** s be such that $\text{idx}(s) < a_{K_a}$. Then, there is k such that $a_{k+1} > \text{idx}(s) \geq a_k$. In this case, the input flow at each **AggNode** s_{a_j} for each $j \leq k$ is $\frac{1}{T} + \sum_{a_{j+1}>\text{idx}(\tilde{s})\geq a_j} x(\tilde{s})$. Then, following case (1) we can prove that individual end-to-end traffic generated by **SN** s toward **DN**, where $a_{k+1} > \text{idx}(s) \geq a_k$, is given as follows:

$$\begin{aligned} C(s,\mathcal{S}_A) &= x(s)d\left(s, s_{a_{a_k}}\right) \\ &+ x(s) \sum_{j=1}^{k} \frac{d\left(s_{a_{k-j+1}}, s_{a_{k-j}}\right)}{T^j \prod\limits_{r=1}^{j} (\frac{1}{T} + \sum_{a_{k+2-r}>\text{idx}(\tilde{s})\geq a_{k+1-r}} x(\tilde{s}))}. \end{aligned} \tag{3.12}$$

3.3.2 General network

In this section, we extend our discussion of cost functions for general networks. First let us introduce some axillary notations.

$\mathcal{S}_C(s)$ denotes the subset of $\mathcal{S}_C$ consisting of all the **CN**s on the paths from all the **SN**s to the **DN**. $\mathcal{S}_A(s)$ denotes the subset of all **AggNode**s in the set $\mathcal{S}_C(s)$, while $\text{AN}(s)$ denotes an **AggNode** from $\mathcal{S}_A(s)$ where data sent by any **SN** s is aggregated the first time. Let $K_A(s) = |\mathcal{S}_A(s)|$ and $K_C(s) = |\mathcal{S}_C(s)|$. $\overline{\mathcal{S}}_A$ denotes the sub set of AggNodes whose incoming traffic is not coming from other AggNodes. For a fixed subset S of $\mathcal{S}_C$ $\mathcal{N}(S)$ denotes the minimal sub-network of network $\mathcal{N}$ containing nodes

S and **DN**. $D(S)$ denotes the total number of hops of network S. Again, we first discuss the total traffic of the entire network, then study the contribution of individual SNs.

Total network traffic

The total traffic generated by all the nodes is given by

$$C(\mathcal{S}_A) = \frac{1}{T} D(\mathcal{N}(\overline{\mathcal{S}}_A) \tag{3.13}$$

$$+ \sum_{s \in \mathcal{S}_S : \mathcal{S}_A(s) \neq \emptyset} x(s) d(s, \mathrm{AN}(s)) \tag{3.14}$$

$$+ \sum_{s \in \mathcal{S}_S : \mathcal{S}_A(s) = \emptyset} x(s) d(s, \mathbf{DN}). \tag{3.15}$$

In particular, we can simplify the cost function for two boundary cases, namely:

(i) if there is no aggregation nodes, i.e., the set $\mathcal{S}_A$ is empty, then $C(\emptyset)$ is given by (3.4).

(ii) if each computing node is aggregation node, i.e., $\mathcal{S}_A = \mathcal{S}_C$, then

$$C(\mathcal{S}_C) = \frac{1}{T} D(\mathcal{N}(\overline{\mathcal{S}}_C)) + \sum_{s \in \mathcal{S}_S} x(s) d(s, \mathrm{CN}(s)). \tag{3.16}$$

Individual traffic generated by a sender node

To derive the individual traffic of a **SN**, we introduce the following auxiliary notations. For a $s \in \mathcal{S}_C$, denote by $\sigma_R(s)$, $\sigma_A(s)$ and $\sigma_S(s)$ the sets of all **RN**s, **AggNode**s and **SN**s sending data to **CN** s without passing through other **RN**s or **AggNode**s Note that some of these sets can be empty. Denote by $X(s)$ the output data traffic of a **RN** s. Due to **RN** just re-forwards all input data traffic to the next **CN** in direction to **DN**, output traffic from **RN** s is given as the sum of input traffic, i.e.,

$$X(s) = \sum_{\tilde{s} \in \sigma_S(s)} x(\tilde{s}) + \sum_{\tilde{s} \in \sigma_R(s)} X(\tilde{s}) + \sum_{\tilde{s} \in \sigma_A(s)} \frac{1}{T}. \tag{3.17}$$

Then, if data sent by **SN** s to **DN** is not aggregated, i.e., $\mathcal{S}_A(s)$ is empty, then individual data traffic generated by such **SN** s is given as follows:

$$C(s, \mathcal{S}_A) = x(s) d\left(s, \mathbf{DN}\right). \tag{3.18}$$

While, if the data sent by **SN** s is aggregated at least once, i.e., $\mathcal{S}_A(s)$ is not empty, then the traffic generated by such **SN** s is given as follows:

$$C(s, \mathcal{S}_A) = x(s)d\left(s, s_{a_{K_A(s)}}\right) \tag{3.19}$$

$$+ \sum_{j=1}^{K_A(s)} \frac{1}{T^j} \frac{x(s)d\left(s_{a_{K_A(s)-j+1}}, s_{a_{K_A(s)-j}}\right)}{\prod_{r=K_A(s)-j+1}^{K_A(s)} X(s_{a_r})}, \tag{3.20}$$

where $\mathcal{S}_C(s) = \{s_{K_C(s)}, \ldots, s_1\}$ and $\mathcal{S}_A(s) = \left\{s_{a_{K_A(s)}}, \ldots, s_{a_1}\right\}$.

3.3.3 Optimization Objective

Our objective is to minimize the Eq. 3.14, i.e., the total cost from the sources to their first AggNodes, as the cost of Eq. 3.13 and Eq. 3.15 are negligible for when the aggregation time window T is large and the number of SN that has direct connection to the DN is small. Assume that we have a limited number of such AggNodes to place, we can also represent it as follow:

$$\sum_{s \in \mathcal{S}_S : |S_A(s)| = K} x(s)d(s, \mathrm{AN}(s)) \tag{3.21}$$

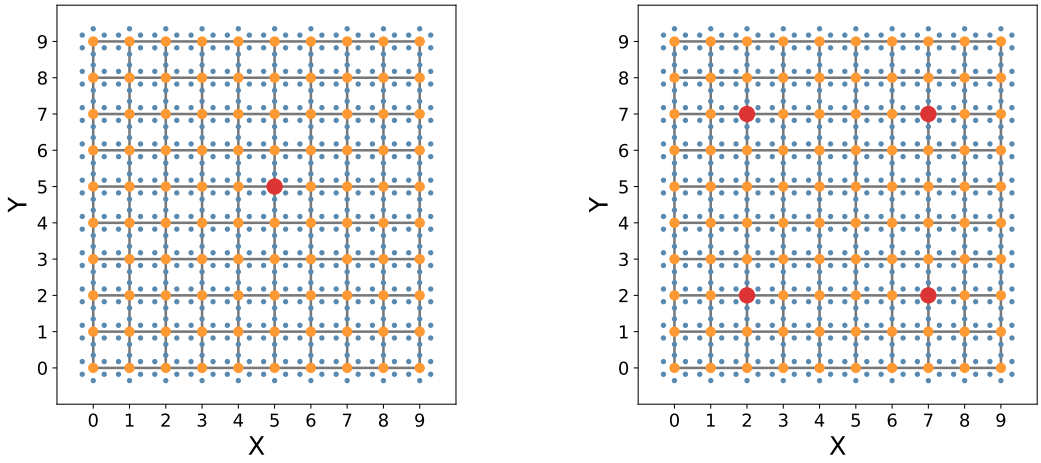
Where K denotes the total number of AggNode we can deploy in the edge network.

Theorem 1. *The problem of finding K AggNodes in a network to minimize sum of square cost is* NP-Hard.

Proof. In the minimum sum-of-squares clustering (MSSC) problem, we try to find K means/medians to form the clusters so that the intra-cluster sum of square Euclidean distance is minimized. Formally speaking, we try to minimize the following cost function:

$$\sum_{k=1}^{K} \sum_{i \in C_k} [d(a_i, b_k)]^2 \tag{3.22}$$

The MSSC problem in general dimension for $k \geqslant 2$ is referred as an NP-Hard problem [41]. We map this problem from euclidean space onto a graph $G = (V, E)$, where



(a) (b)

Figure 3.4: (a) K-AggNodes's results when $K = 1$ (b) K-AggNodes's result when $K = 4$

V is a set of vertexes and E is a set of edges. In this graph, any pair of $v \in V$ have an edge $e \in E$.

Next, we will show that our problem is a special case of MSSC. Given a network, we have $V_n \subseteq V$ and $E_n \subseteq E$, where V_n is the set of nodes and E_n is the set of links. For each data source node $a_i \in V_n$, there is one and only one corresponding data rate x_i. As a result, finding K AggNodes in V_n in order to minimize Eq. 3.21 can be reduced from MSSC in polynomian time. We can conclude that our problem is NP-Hard.

3.4 Heuristic Aggregation Service Placement Algorithms

The AggNodes placement problems we introduced in the previous section are NP-Hard. As a result, there is a clear need to formulate heuristics that can efficiently find a close to optimal solution for the placement problems. Here, we propose two heuristic algorithms to partition the senders into a number of groups, and select the "centered" nodes as the AggNodes to minimize the total cost. Starting from the practical setting in realistic networks, the aggregation service provider has to face two constrains of resource, *i.e.*, the limitation on available number of AggNodes and the capacity of each AggNode. Our heuristics aim at fitting these constrains one at a time.

3.4.1 Provider-defined Placement

Our first algorithm tackles the placement problem when the number of AggNode is fixed. Here, we assume the number of AggNode is determined by the application provider, and try to minimize the total network traffic under a given number. In order to solve this problem, we borrow the idea from K-Means/K-Medoids clustering algorithm. In K-Means [42] or K-Medoids algorithm [43], the objective is to find the centroid points to form clusters so that their intra-cluster sum of square distance to each of the data point in their cluster are minimum. Similarly, we search the K AggNodes to decrease the cost function Eq. 3.21 until the cost remains the same value after a certain number of consecutive iterations. Formally speaking, we aim at updating the AggNodes by using the following formula:

$$an_{new} = \underset{an}{\operatorname{argmin}} \sum_{sn \in SN_k} d(sn, an) x_i, an \in \{an | CN - CN_k\}, \tag{3.23}$$

where an_{new} denotes the updated AggNode of the group, SN_k denotes the assigned source nodes set of the original AggNode indexed by k. Specifically, we use traffic $((\#ofpackets) \times (\#ofhops))$ between nodes as the edges in such graphs. Given that we have K aggregation nodes to be placed in a network with N nodes, the algorithm can be represented as Alg. 1.

This algorithm aims at minimizing the intra-cluster sum of cost function for each group, resulting in a minimized total cost. The algorithm starts with initial estimates for the k AggNodes, which can be randomly selected from the candidate nodes. It then iterates between two steps: *Source Assignment* and *AggNodes Update*. In *Source Assignment* step, each data source is assigned to its AggNode with the lowest cost to form k clusters. In *AggNodes Update* step, the AggNodes are updated by recalculating the cost of its attached sources to its neighbor nodes. The current AggNodes are swapped with candidate nodes if the costs are lower.

Figure 3.4 (a) and (b) demonstrate the example results AggNodes when $K = 1$ and $K = 4$. In a grid network with uniform traffic distribution(*i.e.*, each computing node attaches the same number of source nodes and each source node generates the traffic at the same rate), the selected AggNode (s) should reside at the center of the network

Algorithm 1 K-AggNodes Placement

```
procedure INITIALIZE(K, CN, SN)
    RandomSelectFrom(K, CN)
    AssignSenderNodes(CN_k, SN)
    for sn in SN do
        Assign sn to the neareast cn in CN_k
    end for
end procedure
procedure FINDOPTIMALS
    while C(CN_k) is not stable do
        AssignSenderNodes(CN_k, SN)
        for cn in CN_k do
            for cn_t in CN − CN_k do
                if C(cn_t)) < C(cn) then
                    Update(cn_t, cn)
                end if
            end for
        end for
    end while
end procedure
```

or the sub-network. Our results indicate that K-AggNodes algorithm is capable of identifying the optimal AggNodes in the ideal settings.

Key Parameter: Homogeneity score is a common metric to evaluate the choice of K in the classical K-Means or K-Medoids clustering, however, it is not meaningful in the AggNode placement problem. This is due to our objective is solely minimizing the total sum of square cost and increasing K will always decrease this total sum.

3.4.2 Network-defined Placement

The second algorithm is design to allocate AggNodes to the network traffic density center in order to achieve the optimal aggregation efficiency. Here, the number of AggNode is not determined by the application provider, instead, it merely depends on the network traffic distribution. By locating the network traffic density centers, we could determine both the location and the number of AggNodes, and assign data sources to the corresponding AggNodes. While K-AggNodes starts from initializing randomly selected K AggNodes, this algorithm follows a bottom-up approach by initializing all computing router nodes as candidate AggNodes. Figure. 3.5(a) depicts the process

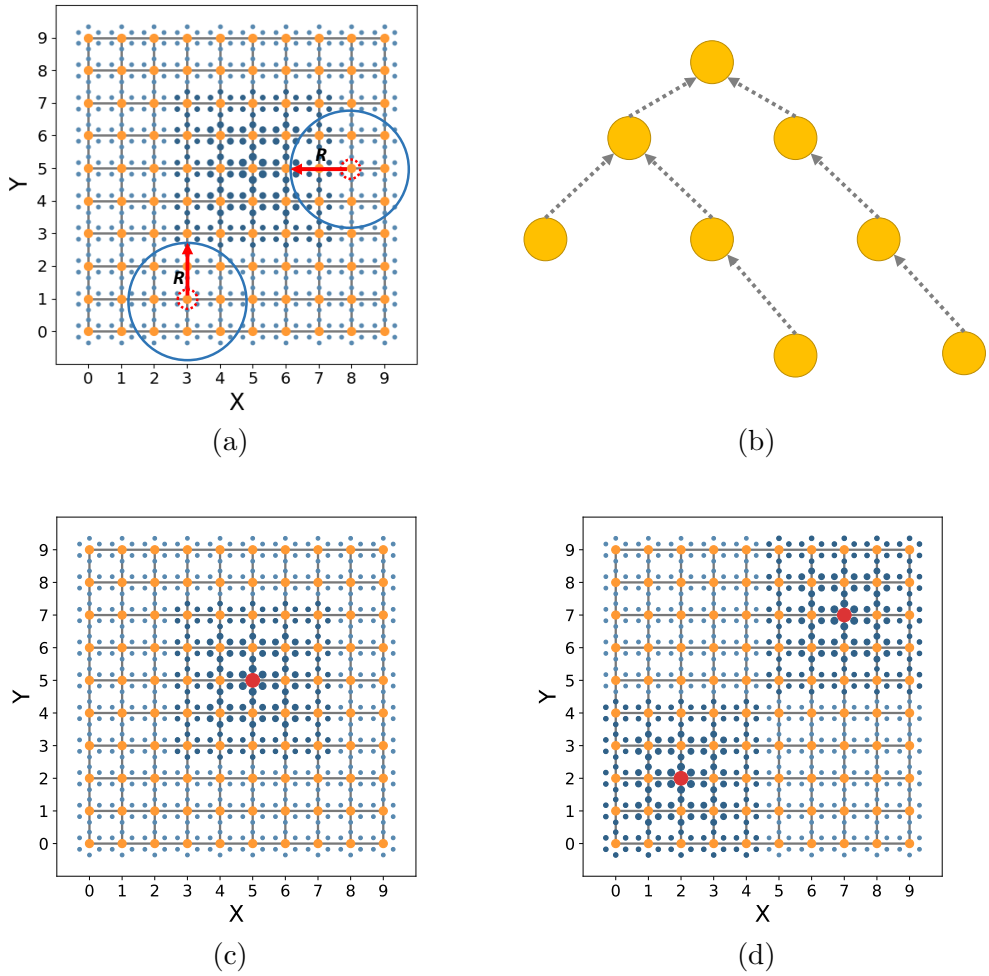


Figure 3.5: (a) An example process of AggNodeShift (b) A merging Graph (c) AggNodeShift's result when there is one network traffic density center (d) AggNodeShift's result when there are two network traffic density centers

of the algorithm. Each AggNode covers a set of data source nodes within a range. By recalculating the centroid nodes in the circles, we shift AggNodes towards these centroid nodes and merge two clusters if their new AggNodes point to the same node. As a result, we call this placement algorithm AggNode-shift. The idea of such mode seeking approach has been widely adopted in numerous clustering algorithm such as MeanShift [44] and CAMShift [45]. We depict the detailed procedure in Algorithm 2.

In the initialization step, we set all candidate nodes as AggNodes and attach the data source node within a radius R to these candidates, where R is a tun-able parameter. The distance metric we used here is $d(sn, cn)x$ since we aim at minimizing the total

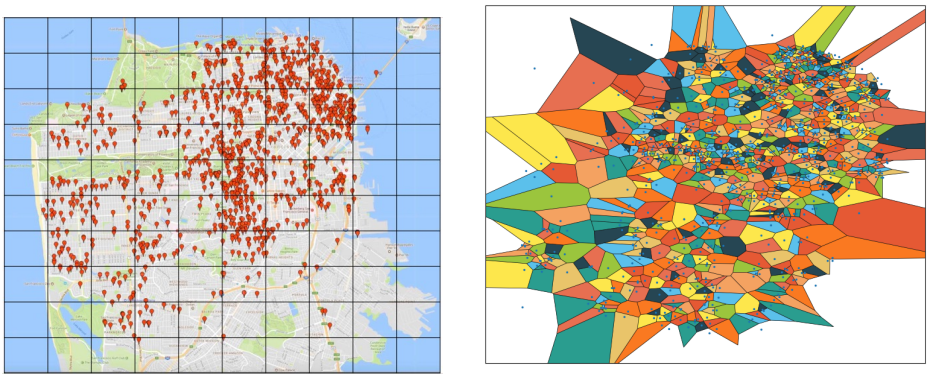
Algorithm 2 AggNodeShift Placement

```
procedure INITIALIZE(R, CN)
    for cn in CN do
        Set cn as AggNode
        AssignSenders(R, cn)
    end for
end procedure
procedure FINDCENTERS
    while AN are updated do
        for cn in AN do
            FindLeastCostNeighbor
            AddtoMergedGraph(cn, cn_target)
        end for
        TopologicalSort
        MergeGraph
    end while
end procedure
```

sum of cost. It is worthy to mention that a data source node may be attached to more than one AggNode during the iteration. By calculating the cost from source nodes to a AggNode's neighbor, we shift the AggNode to the neighbor with the lowest cost and merge their attached data sources set. We repeat this process until the result AggNodes set AN remains unchanged for a fixed number of interations.

*Merging Order*Before actually merging two AggNodes and their corresponding attached data sources, we store their merging relationship, *i.e.*, $cn \rightarrow cn_{target}$ in to a directed acyclic graph (DAG) (shown as Figure 3.5(b)) and merge all the AggNodes by following their topological order in the graph. This is due to if we merge every pair independently, one pair merging may conflict with another pair merging. By applying topological sorting to this DAG, we are capable of starting merging from the leave nodes in order to avoid such conflicts.

Key Parameter: Radius R determines the range of data sources that each AggNode should start with. An oversized radius could lead to network traffic density centers to be merged, while an undersized radius could generate additional local network traffic density centers. A reasonable value of R needs to be determined empirically. The distance metric of this radius could vary depending on the application requirement. In our implementation, we choose the number of hops as it is robust to different network



(a) (b)

Figure 3.6: (a) Access Point locations at San Francisco.(b) Voronoi diagram for the Access Points

traffic distribution.

Load Re-distributed: Seeking the network traffic density centers help us determine the number of required AggNodes, but it may lead to another non-trivial problem – for a center with extreme high incoming traffic load, an AggNode's capacity is not sufficient to handle all of them. In order to tackle this challenge, we propose two re-distribution approaches in addition to AggNodeShift when a AggNode's assgined load exceeds its capacity. In the first algorithm, we re-distributed the work load of such AggNode with its nearest available nodes in a Round-Robin manor. To find such available nodes, we apply Breath-First Search that starts from the AggNode. This approach is motivated by the fact that if an AggNode is the local optimal, its nearest available nodes should be the sub-optimal. In the second approach, we apply K-AggNodes on the data source set which is assigned to this AggNode as it seeks to minimize the cost. In both approaches, the number of AggNode for one such AggNode after re-distritbution is calculated as $N_a = L/C_T$, where L denotes the load before re-distribution, and C_T denotes the capacity threshold.

3.5 Evaluation

In this section, we study the performance of our proposed heuristic in a city-scale taxi data collection scenario. In this scenario, thousands of taxis report their sensor

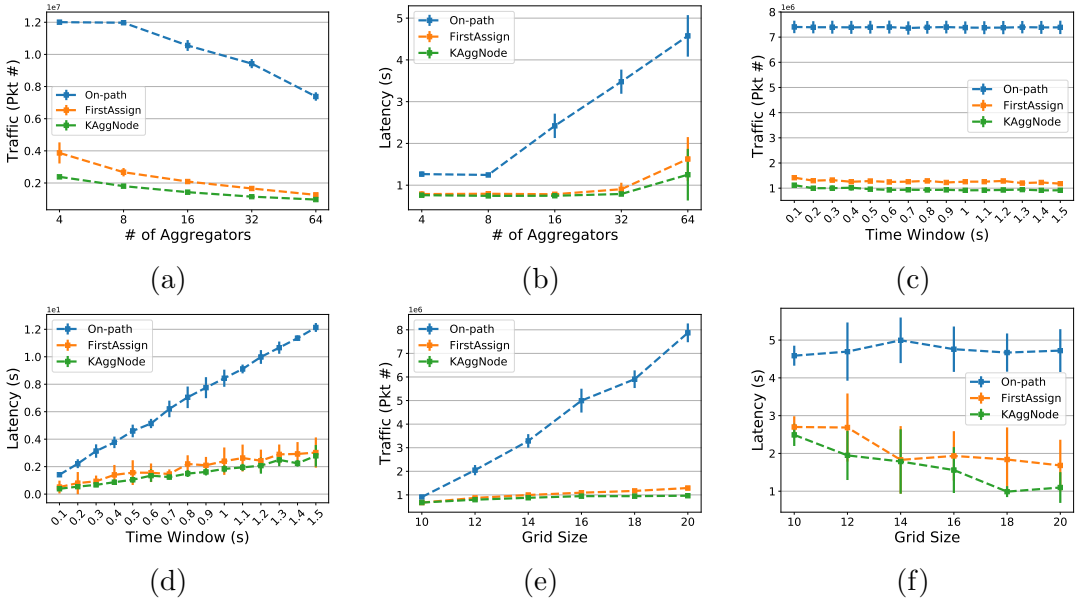


Figure 3.7: (a)K-AggNodes achieves 20% traffic of On-path when $K = 4$(b)K-AggNodes achieves the lowest end-to-end latency among three schemes (c) Total network traffic is insensitive to the aggregation time window (d)The latencies of K-AggNodes and FirstAssign change slower than that of Onpath when the time window varies (e) K-AggNodes and FirstAssign delivers stable total network traffic in a dense AggNode deployment when $K = 64$ (f)K-AggNodes outperforms the other two schemes when the grid size is varied.

reading to a single application server via mobile edge network, where the application provider can install AggNodes in the VMs at any edge routers. The objectives of such experiments aim to understand: Firstly, given the number of AggNodes to be installed is defined by the application provider, where are the optimal locations so that we can achieve the minimum the total network traffic? Secondly, given a specific network traffic distribution, how can we select the proper number of AggNodes in order to achieve the maximum aggregation efficiency? Finally, how can we re-balance the work load of AggNodes when their capability of processing data in a time unit is constraint.

3.5.1 Experiment Setup

In order to simulate a realistic IoT data collection scenario, we extracted the geographical information of the taxis from San Fransico Taxi mobile traces [46]. We took a snapshot of the trace that contains all taxis at 3:00 PM every day for 10 days. Due

to the limited number of taxi in the trace (less than 400), we performed spatial interpolation on the snapshots to increase the number of taxis in order to simulate a more realistic city-scale taxi data collection scenario(shown as Figure 3.6(a)). Next, we generate a voronoi diagram [47] for the San Francisco Access Point data set [48] by using access point locations as centroids. Within each voronoi cell, we attached the taxis to the access point. We assumed all taxis send data with the same interval X, which follow Poisson distribution with $\lambda = 1$. As a result, the data sending interval at each access point is determined by its attached taxi number Y, which follows Poisson distribution with $\lambda = Y$.

For the network topology, we used a $N \times N$ grid topology to deploy computing routers. We divided the San Francisco map into $N \times N$ equal-size grid and attached the access points to the corresponding grid node. Finally, we linkd the server node to one of the corner node of the grid. We extended a Java-based even-driven netork simulator[1] that was used in [49] to adopt our requirement.

3.5.2 Provider-defined Placement

We first investigate the scenario where the number of AggNodes is defined by the application provider. By varying the parameter of the network, we aims at understand the performance of K-AggNodes algorithm in terms of reducing the total network traffic with minimum end-to-end latency.

As comparisons, we also implement two other aggregation schemes as follow:

- **On-path Aggregation:** We place AggNodes on the merging nodes on the shortest paths between the data sources and the server. However, the number of merging nodes and their location are determined by the location of the data sources, the location of the server, and the network topology, it could be inefficient if we place K AggNodes closed to only part of the data sources. Instead, we place K AggNodes around the server node so that most of the data flow can be aggregated at least once before received by the server.

[1]Download link: https://github.com/sugangli/Simulator.git

- **First-Assignment Aggregation:**We randomly select the locations for K AggNodes, and assign data sources to their nearest AggNode based on their cost function ($HopCount \times DataRate$).Once a data packet arrives at its assigned AggNode, it follows the shortest path to the server and will be aggregated if there is AggNode(s) on the path.

Impact of AggNode Number: We first investigate the impact of available AggNode number. We use end-to-end latency and network traffic collected by all nodes as the performance metrics of our evaluation. We set the aggregation time window to 0.5 seconds and collect the received packets at each node for one minute. To show the trend of each aggregation scheme, we vary the number of AggNodes from 4 to 64, and report the result in Figure 3.7.

Figure 3.7(a) shows that total traffic can be reduced by placing more AggNodes. Among three schemes, KAggNodes outperforms the other two. Specifially, KAggNodes has only 62% and 20% total traffic of FirstAssign and On-path in the case of 4 AggNodes. This is due to KAggNodes is capable of approximating the best placement locations of AggNode hence achieve large traffic reduction gain in a sparse scenario (4 AggNodes vs. 400 Computing Routers). On-path gives the worst performance since aggregation only happens at the last few hops to the server.

Figure 3.7(b) to (d) shows the end-to-end latency increases as the AggNode number increases. As shown in Figure 3.7 (b), On-path aggregation gives the highest average latency among three schemes because the every data flow are aggregated iteratively at the last few hops, and the latency is mainly contributed by the aggregation. Figure‘3.7(c) and (d) depicts the CDF when the number of AggNode is 4 and 64, respectively. We observe that, in the case of 4 AggNodes, KAggNodes scheme achieves similar average latency with smaller variation compared to FirstAssign scheme. While in the case of 64 AggNodes, KAggNodes scheme results in lower average latency. This is due to KAggNodes scheme is capable of searching the center nodes which have lowest cost sum to all other nodes within the clusters.

Impact of Aggregation Time Window: Next, we study the performance impact of aggregation time window. We vary the time window from 0.6 seconds to 1.5 seconds at

all 64 AggNodes. Figure 3.7(c) depicts the total traffic of three schemes. KAggNodes outperforms the other two schemes with less than 10^6 average total packets.

The latency, however, is sensitive to aggregation time window as shown in Figure 3.7(d). KAggNodes and FirstAssign increase slower than On-path as they have lower chance to be aggregated iteratively. It indicates that we can choose smaller aggregation time window for latency-sensitive applications without sacrificing noticeable network traffic.

Impact of Network Size: We investigate the impact of network size. We place 64 AggNodes at networks with grid size of 10×10,12×12,14×14,16×16,18×18, and 20×20, respectively. We also keep the same number of data sources and the same data rate at each data source. Figure 3.7(e) shows the total traffic of three schemes. KAggNodes outperforms the others since it can approximate the locations with lowest cost regardless the network size. The slopes of both KAggNodes and FirstAssign are much smaller than On-path, because On-path wastes resources on iterative aggregation. In On-path scheme where all AggNodes surround the server, the number of hops without aggregation is increasing proportional to the increment of the grid size. In a dense-deployed (64 out of 100 Computing Routers are AggNodes) network, the difference between On-path and the other two schemes is marginal due to the improving space is limited.

Figure 3.7(f) describes the latency of the same experiment. We observe that KAggNodes can achieve the lowest latency in all the cases (grid size from 12 to 20), since it allocates the AggNodes more sparsely than the other two algorithm which results in reducing the times of a flow being aggregated.

3.5.3 Network-defined Placement

In this section, we study the performance of AggNodeShift, the algorithm to cluster sender nodes based on the network traffic density estimation. Unlike the application-provider-defined scheme we show above, the number of installed AggNodes is calculated based on the number of network traffic density center. By identifying the network traffic density centers, we can allocate the AggNodes with better efficiency. Here, we introduce

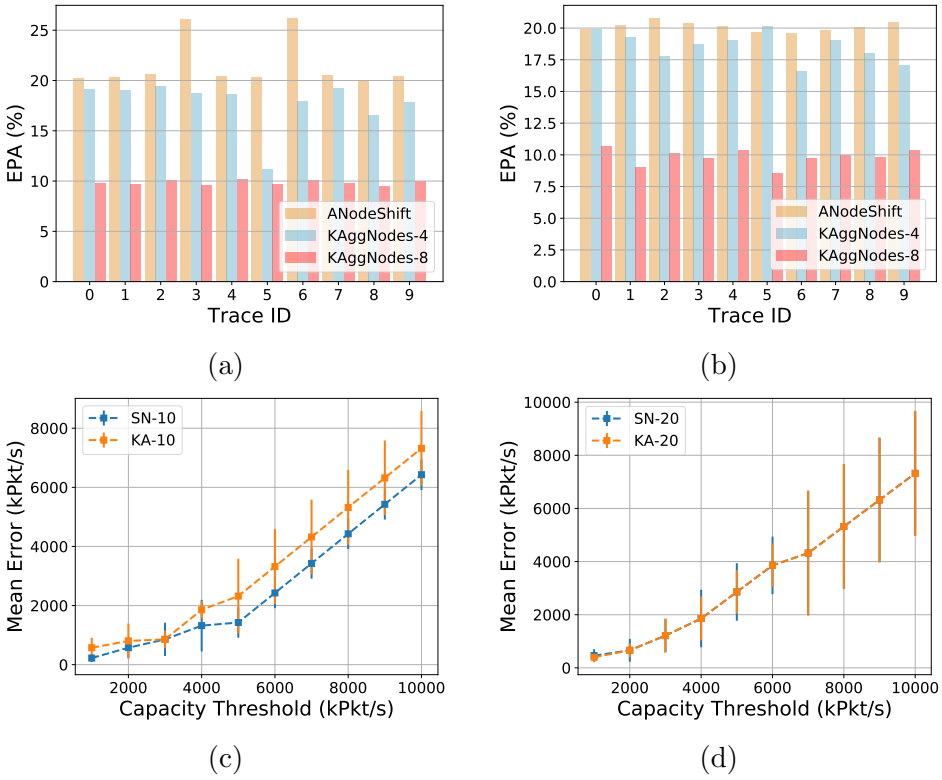


Figure 3.8: (a)EPA when the grid size is 10×10(b) EPA when the grid size is 20×20 (c) Mean Error of load when the grid size is 10×10 (d)Mean Error of load when the grid size is 20×20

a new evaluation metric *Efficiency per AggNode (EPA)*, which is defined as follows:

$$EPA = \frac{T_{NA} - T_A}{T_{NA} \times K} \times 100\%, \tag{3.24}$$

where T_{NA} denotes the total network traffic without any aggregation, T_N denotes the aggregated network traffic, and K denotes the number of AggNode.

As a comparison, we evaluate KAggNode algorithm with $K = 4$ and $K = 8$ on the same data traces and the network topologies. We collected the received packets at each node for one minute to calculate the total network traffic.Figure 3.8(a) depicts the EPA over ten different traces when the grid size is 10×10. AggNodeShift outperforms the other two setting over all traces. KAggNodes-4 achieves only slightly worse EPA due to $K = 4$ is closed to the average number of AggNode estimated by AggNodeShift. When the grid size is enlarge to 20×20 (shown in Figure 3.8 (b)), the result demonstrates the similar pattern which indicates both algorithm is insensitive to the network size

and robust at varied scale. *Load Re-distribution:*Here, we further evaluate the performance of load re-distribution among AggNodes with SharewithNeighbors(SN) scheme and KAggNodes (KA) scheme. Please note that both algorithms are triggered after AggNodeShift is executed in order to re-balance the load for those AggNodes that reside in dense network nodes and reach the capacity threshold. In order to evaluate the effectiveness of load re-balancing, we introduce *Mean Error (E)*, which is defined as follows:

$$E = \frac{1}{n} \sum_{i=1}^{n} \frac{L_R}{C_T}, \tag{3.25}$$

where L_R denote the load after re-balancing, C_T denotes the capacity threshold and n denotes the number of AggNodes.

Figure 3.8 (c) and (d)show the $MeanError$ at different capacity threshold when the grid size is 10×10 and 20×20, respectively. SN achieves smaller mean errors than KA due to the nature of SN's Round-Robin assginment,which equally assign the load to each of the candidate node. In the larger network (20), KA output the similar set of AggNodes and their assigned set compared to SN, which results in similar mean errors.

3.6 Related Works

WSN Data Aggregation: In the area of wireless sensor network, many works have been proposed to aggregate sensor data in order to reduce the total number of transmissions which further save energy consumption. First type of the studies i mainly focusing on protocol design of aggregation networks. TAG [50] approach introduce a data-centric aggregation protocol. It constructs an aggregation tree and is specially designed for monitoring applications.TAG adopts the idea of the selection and aggregation functions of the database query language (SQL). As for most of the tree based solutions, TAG falls to in short when dynamic topologies or link failures are considered. Directed Diffusion [51, 52] is a pull-based network protocol. The routing of the data is specifically for the scenarios where there are one or few sinks query some information by flooding the network with the queries. Due to the query broadcast is mainly implemented in MAC layer, the network is easy to be congested when the node's density is

increased. As a result, such solutions can only be applicable when the local traffic or node's density is kept at an acceptable low level in order to avoid collision or congestion. PEGASIS [53] introduce a idea to organize the sensor nodes in a chain. Nodes take turns to act as chain leader which is the only node allowed to transmit data directly to the sink. In PEGASIS, each node receives data from its neighbor and aggregates with it own sensor reading into one packet. Next, such aggregated packet is transmitted to the next node until the packet arrives at the chain leader. However, the performance of the system is sensitive to the link failures. Our approach overcome these disadvantages by an logical centralized ASR controller. It can collect nodes' states and configure the routing without flooding the entire network, hence is robust against link failures and topology dynamics.

Another type of studies pays attention to designing aggregation algorithm for minimizing certain network metrics. Work by Lindsey et.al [34, 54] studies a self-organizing and adaptive clustering algorithm to aggregate data at the head of each cluster. It outperforms traditional clustering algorithms by considering adaptive clusters and rotating cluster heads depending on the signal strength. However, it fails to account the various of data rates at data sources, hence the overall network traffic is not minimized. The author of [35] introduce a clustering algorithm that similar to the previous solution. In addition, it allows more than one hop between each node and its cluster head by adopting the Ad Hoc On Demand Distance Vector (AODV) routing. Like most of the clustering algorithm, it is also affected by the network dynamics and the unbalanced network traffic. Our approach address these problems by periodically computing the network traffic cost function and re-configure the network topology with low control overhead.

Data Synchronization: In order to synchronize the data or the states efficiently, aggregation is also studied in large-scale decentralized systems or networks which are not as resource-constrained as WSN. DIAS [55] is an agent-based middleware which combine the local availability of collective and the summary about the state of the whole system to perform decision making. It addressed the communication and the storage overhead by introducing aggregation memberships in bloom filters. Work by Lochert

et.al [56] proposes a algorithm to identify the appropriate positions of static roadside units in traffic information systems in vehicle network, in order to overcome the problem of limited bandwidth and minimal initial deployment and eventually shorten the travel time. Works in [37, 57] propose gossip-based protocols for computing aggregate values over network components in a fully decentralized fashion. In order to reduce aggregation time and archive high ranking accuracy with low memory cost, Zhou et.al [58] introduce a Bloom-filter based reputation aggregation architecture in peer-to-peer systems. These solutions fall short in efficiently collecting data for single data consumer as they require data flooding across all computing nodes. Our application specific routing overcomes this challenge by computing and populating data routes from a centralized controller hence avoid flooding the entire network.

3.7 Conclusion

In this chapter, we have described AggMEC, a generic aggregation architecture for efficient data collection in large-scale MECs. To overcome the dilemma between limited computing resource and high-volume network traffic, we define a cost function and propose a novel clustering-based algorithm to approximate the minimum cost. In addition, we design a application-specific routing protocol over a clean slate network architecture – MobilityFirst, which grands low control overhead and high scalability. Through detailed evaluation, we show that AggMEC can reduce 80% of total network traffic from the baseline scheme with low end-to-end latency.

Chapter 4

Roadside Context Sensing Using Microphones on Smartphones

4.1 Introduction

Using smartphones to sense users' surroundings and learn about their contextual information has received much attention in the past few years [59–63]. A large number of such studies have been conducted for users when they are driving [64–66], while much less effort has been devoted to users when they are walking. In this study, we focus on sensing and learning an important context information for pedestrian users using their smartphones – whether there is an approaching car nearby.

The ability to detect oncoming cars using smartphones can enable/enhance several important ubiquitous applications. Firstly, with such information available, notifications could be sent out to alert distracted users (through sounds or vibrations) about approaching cars when they are about to enter an intersection or wander into roads (illustrated in Figure 4.1). Secondly, we could aggregate sensed cars in different areas to perform fine-grained traffic monitoring. Although navigation apps (*e.g.*, Google Map [67] and Apple Map [68]) have been widely adopted to collect vehicle traces and provide real-time traffic information, some geographic areas (such as residential areas, college campuses, etc) may not have sufficient data because drivers in those areas do not turn on their navigation apps frequently enough. In such cases, pedestrian users are able to sense cars in their vicinity and provide additional traffic data that was not available earlier. With these additional data, we could better assess the traffic condition in our neighborhood, such as outside of the post office or a grocery store. Thirdly, the ability to detect incoming cars could also enhance emerging augmented reality (AR) applications/games by enriching their virtual maps – *e.g.*, marking a place as 'walkable'

or 'accessible' on a Pokemon Go map.

In fact, there has been investigation on detecting cars in other communities. For example, solutions based on Dedicated Short-Range Communications (DSRC) have been proposed and studied [7, 69], which utilize a special radio channel to establish bidirectional communications between a car and people who are walking nearby. Such solutions, however, require additional hardware modules to be included in both the vehicle and smartphones, which may not offer an immediately available solution. Computer vision based techniques have also been studied, such as those in [70, 71], which detects vehicles via images captured by smartphone cameras. The disadvantage of such solutions, however, stems from several factors, including the requirement of high CPU overhead that is needed for real-time vehicle recognition, as well as the requirement of inflexible camera facing direction, etc.

In this chapter, we propose a system, referred to as *Auto++*, that accurately detects approaching vehicles by processing real-time audio stream directly captured by off-the-shelf smartphone microphones without any prior training or additional infrastructure support. *Auto++*does not have the shortcomings of a radio based or vision based system; in addition it offers several clear advantages: (1) it is self-contained and does not need any infrastructure support; (2) it applies completely unsupervised classification techniques and no prior training is needed for the system to operate; (3) it is accurate and has low false positive rate; and (4) it is robust as it works with different cars, roads, and different environmental noise. The main challenge we face is due to the characteristics of car sounds. Today's engines are becoming increasingly quiet, and the perceivable sounds are generated mainly by tire-road friction. This tire noise lacks both distinguishable temporal and frequency structures. As a result, many popular acoustic techniques such as Doppler Shift [72] or features such as MFCC [73] cannot be applied in our system. In this study, we establish a new feature that can discriminate between car sounds and other environmental sounds. The feature is the maximal frequency component that crosses a power thread; when a car drives closer to the user, this frequency continuously goes up. Through edge detection, we can robustly extract this feature from the spectrogram of the audio signal.

Figure 4.1: *Auto++*detects approaching cars using audio data recorded by smartphones. This capability can enable applications such as alerting distracted users (possibly through sounds or vibrations). Please note that the application logic such as alerting users is not part of *Auto++*.

The car sound's property does not only present challenges for car detection, but it also provides an opportunity for our detection algorithm to work cross different cars and roads – tire noise from different cars is likely similar to each other. Once the system detects a car, we can estimate its driving direction by applying cross-correlation function over audio streams from two microphones (many of today's smartphones have dual microphones). We can also cluster detections that are within a short time period to count the cars around the user.

We implement *Auto++*over Android platform and collect 330 minutes of audio data from smartphone over the course of 14 months. Our evaluation involves seven vehicles whose types cover a broad range of cars that are seen on US highways, including SUVs, medium sedans, sports cars, compact cars, and electric cars. The audio is recorded by *Auto++*in a range of different outdoor scenarios, from quiet parking lots, to noisy outdoor shopping center during business hours. Our results show that *Auto++*is robust across different cars and various types of environmental noise. Specifically, we demonstrate that we can detect 91% of the cars even when they are more than four seconds away from the user. On average, our system can detect cars when they are 6.8 seconds away (while a naive scheme could only detect cars that are 2.4 seconds away). We note that *Auto++*can also detect electric cars with 100% when they are 7.8 seconds away. Finally, we are able to estimate a car's driving direction with an average success rate of 84%.

4.2 Motivation and Challenges

In this section, we first present the motivation and system assumptions. Then we discuss several challenges that arise due to car sound signal properties and limitations of the underlying hardware platform.

4.2.1 Motivation and System Assumptions

*Auto++*performs the following three tasks in real-time: (1) car detection – whether there is a vehicle approaching the user, (2) car tracking – what is its direction, and (3) car counting – how many cars are passing or have passed the user within a given time window. It performs these tasks by analyzing the acoustic signal captured by built-in microphones. The ability to answer these questions can enable several new applications. For examples, we could alert the distracted pedestrian users (through sound, vibration, etc); we could collect fined-grained traffic data from pedestrian smartphones; we could further enhance augmented reality tools or games.

In our system, we assume that the user carries a mobile device that is equipped with one or two microphones. Today, all smartphones and tablets have at least one microphone, and some have two microphones, such as Google Nexus 6 and Nexus 6P. Further, we assume that the microphones have a similar degree of sensitivity in all directions. Also, we assume that the smartphone is held in hand by the user, or in bag/pocket with connected headset. Finally, we assume that the smartphones have the orientation service available so that *Auto++*can calculate the smartphone's relative direction with respect to the road.

4.2.2 Background on Acoustic Signal Processing

Sound Pressure Level (SPL) is the direct form of a sound signal measured by the microphone and describes the local pressure deviation caused by the sound wave.

Time Difference of Arrival (TDoA) [74] provides a measurement of the location of sound source. By calculating the Cross-correlation Function (CCF), Inter-channel Phase Difference (IPD), or Maximum Likelihood Function [75–77] of the signal observed

at difference microphones, the possible location of the sound source can be estimated. With three microphones, the sound source's exact position can be estimated via triangulation. Due to the limitation of today's smartphone, we could only use two microphones to infer the sound direction.

Short-Time Discrete Fourier Transform (STDFT) is used to divide a long signal into shorter segments of equal length and then compute the discrete Fourier Transform on each of the segments. STDFT is widely used in audio analysis [75] as it can reveal both temporal and spectral information of the signal at the same time. It is particularly useful in detecting and tracking a moving car, because we can observe how the frequency changes when a car is approaching. We provide details on how we could potentially adopt these features in our system, or, why we can't adopt them at all. Finally, we also provide a brief discussion on commonly used acoustic signal processing tools.

Mel-frequency cepstrum (MFCC):

Doppler Shift: Doppler shift is the frequency change of a wave at the receiver when the source moves, which is often used in sound source localization [78, 79]. The use of Doppler shift requires that the signal has one or more frequency ranges on which energy concentrates. Car sound signal, however, does not exhibit such skewed energy distribution, and thus we can not use Doppler shift to detect or track a moving car.

4.2.3 Challenges in Designing *Auto++*

In designing *Auto++*, we face several significant challenges. When a car runs at a steady speed (without sudden acceleration), its sound consists mostly of tire noise instead of engine noise, which has very different traits from many other sound signals that have been studied. Furthermore, this problem is made even worse by the fact that we are capturing and processing the audio data using off-the-shelf smartphones that have hardware limitations.

Lack of Energy Concentration on Frequencies: Many sound signals have distinctive energy concentration on certain frequencies, such as those studied in [75,80]. In such cases (illustrated in Figure 4.2(a)), we can focus on detecting energy on those

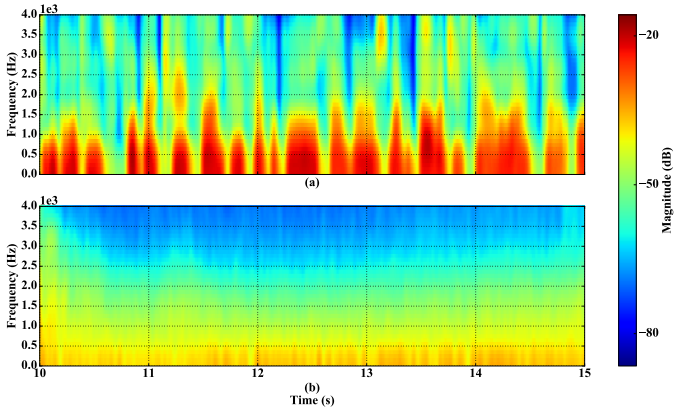


Figure 4.2: (a) A typical speech signal has one or more frequency ranges on which acoustic energy is concentrated. (b) Car sound signal does not have clear energy concentration across different frequency ranges

frequencies to detect the presence of the sound source. To localize such sound sources, the angle of arrival can be easily computed via the inter-channel phase difference [75], or Doppler shift [72] can be applied to estimate the distance between the sound source and the device. However, the sound of a vehicle, as shown in Figure 4.2, does not contain obvious energy concentration on any specific frequencies and therefore calls for different techniques.

Lack of Temporal Structure: Several studies rely on the sound signal's temporal features to localize the sound source, such as the sound pressure level (SPL) peaks [76, 81], sound emitting times [76], etc. However, we do not observe obvious temporal features in our car sound signal due to its noise-like nature.

System Limitations: We run *Auto++*on mobile devices, and the system design is thus limited by the available features on the hardware platform, e.g., how many microphones a device has, how far apart are these microphones, how sensitive are these microphones, what is the maximum sampling rate, etc.

In this study, we have carefully addressed these challenges, and our evaluation results in Section 2.4 show that *Auto++*is accurate, timely, and robust.

4.3 Car Event Notification and Updating

For a sensing system such as *Auto++*, it is quite natural to raise the concern of (1) having inaccurate event detection, and (2) having excessive notifications or updates even when the detection is accurate. While the main body of this paper is devoted to addressing the former concern by improving the detection accuracy (results shown in Section 2.4), here we would like to discuss how we can address the latter to improve the system's usability. Towards this purpose, we argue that *Auto++*should decouple sensing and event notification/updating, and that the notification/updating module should be made tunable based on the exact context the user is in.

The event notification/update frequency may need to vary, when we use different applications in different situations –*e.g.*, a user might prefer less frequent alerts about incoming cars when walking in a crowded downtown area during peak hours, compared to walking on a suburban street during off-peak time; games such as Pokemon Go might need more frequent updating than health applications such as Fibit.

With this in mind, we design *Auto++*such that it dynamically adjusts the notification/updating frequency as desired. First, by giving users the ability to configure their preferred setting, we could trigger *Auto++*only in specified situations - e.g., when the user is about to enter an intersection or walking on the sidewalk in a less crowded area. Leveraging existing technologies such as geo-fencing [82], the system could detect whether the user is in such a situation and then activate/deactivate car detection accordingly.

In addition to user-specified settings, *Auto++*can also filter out unnecessary event notification/updating through detailed context sensing. For example, the report will only be sent out if an approaching car is detected while the user is walking towards/along the same road. Such sensing can be achieved by using built-in inertial sensors, GPS, etc.

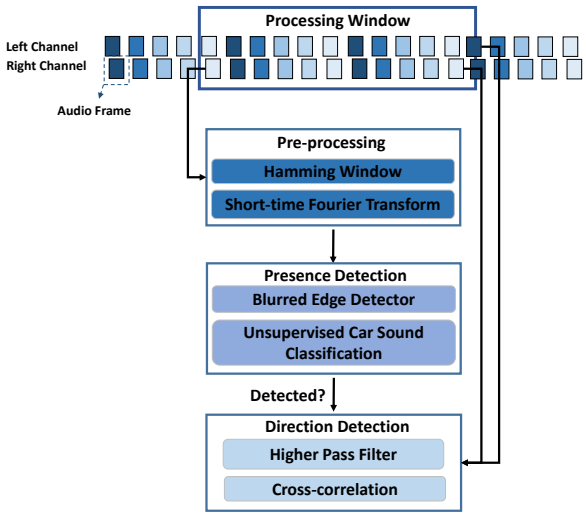


Figure 4.3: The *Auto++*system involves the following four components: (1) signal pre-processing, (2) feature extraction, (3) presence detection, and (4) driving direction estimation. *Auto++*does not require any prior training for any of these steps.

4.4 *Auto++* Design

In this section, we present the detailed design of *Auto++*, which consists of four main steps: pre-processing, feature extraction, car detection, and car direction estimation.

4.4.1 Overview of *Auto++*

Fig. 4.3 depicts the overview of *Auto++*, which consists of the following components:

1. **Pre-processing:** Before we can use the captured sound signal to detect, track, or count the cars, we first need to perform several pre-processing tasks, including segmenting the continuous stream of sound signal into smaller chunks, multiplying each chunk with a suitable window function, and conducting required processing on each chunk, such as discrete Fourier transform (DFT). These steps prepare the data for subsequent signal processing.

2. **Feature Extraction:** Next, we extract features that can be used for car detection. Unfortunately, we find that those features that are commonly used to detect and localize sound sources are not suitable for our system (explained in Section 4.2) as car sound is dominated by tire noise and lacks distinctive spectral

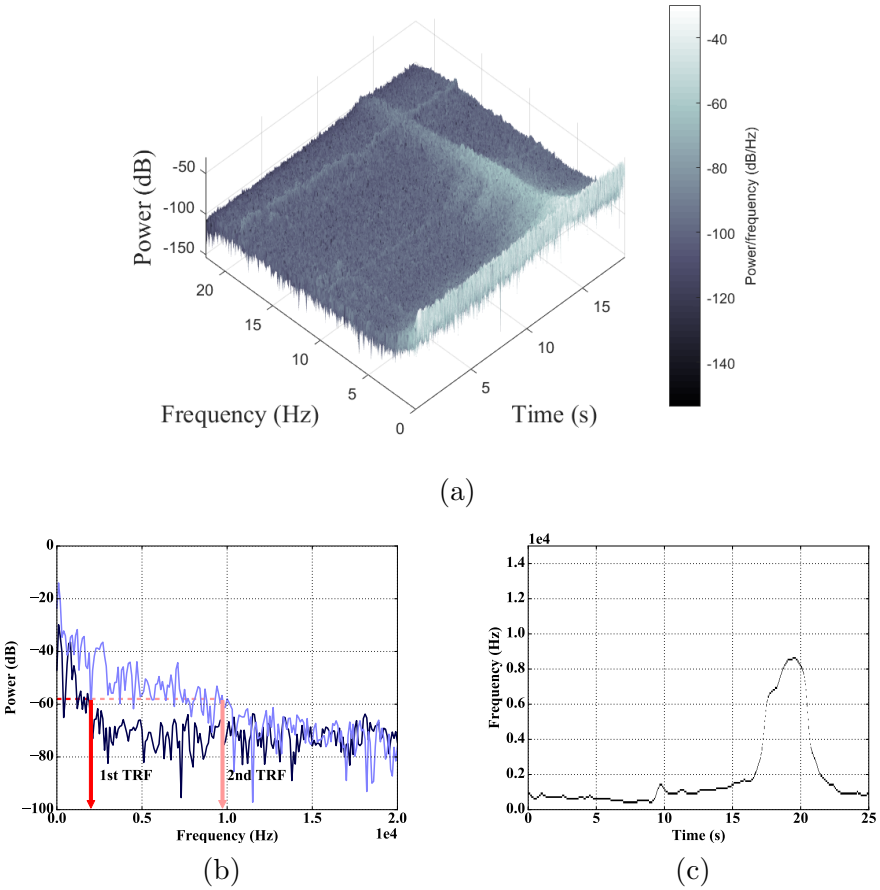


Figure 4.4: (a) The STDFT result of the sound signal recorded when a 2007 Toyota Camry drove from 100 meters away from the user until a few meters past the user. (b) Two STDFT snapshots taken at different timestamps, in which we observe higher TRF when the car gets closer. (c) The TRF trace extracted by the blurred edge detection algorithm.

and temporal features. In this work, we have proposed new features that are unique to car sound signals and that facilitate accurate, timely, and unsupervised car detection.

3. **Presence Detection:** Our presence detection algorithm is unsupervised. It can successfully discriminate between scenarios that involve an approaching car and those that do not involve a car (but with a high ambient noise), as early as possible.

4. **Direction Estimation:** Our direction detection algorithm is based on Time Difference of Arrival (TDoA) of the sound signal. By computing the cross-correlation

function over signals from two channels, we can provide a coarse direction estimation of the sound source.

4.4.2 Sound Signal Pre-Processing

Firstly, we segment the continuous stream of audio data into chunks of equal duration, each chunk referred to as a processing window. In our system, a typical window length is .5 second, which means we run our car detection algorithm every .5 second. Secondly, our detection algorithm involves the calculation of the short-time Discrete Fourier Transform (STDFT) of the audio samples in a processing window. For this purpose, we further divide a processing window into slots of equal length, and multiply the hamming window function with each slot. We then perform the Fourier transform over each slot to obtain the STDFT of the processing window. The slot duration determines the temporal granularity of our STDFT calculation.

4.4.3 *Auto++*Feature Extraction

Next, we propose our car detection feature that addresses the unique challenges of car sound signals, and our feature extraction method that facilitates accurate, timely, and unsupervised car presence detection.

Top-Right Frequency (TRF): Maximum Frequency Whose Power Reaches a Certain Threshold

We discover our feature after carefully examining the car sound signal's STDFT results. Fig. 4.4(a) plots the STDFT result of the sound signal from a 2007 Toyota Camry during its drive of 100 meters. We started collecting audio samples when the car was 100 meters away, and ended recording when it drove past the user. The recording was done using a Nexus 6 smartphone. From the figure, we observe that when the car gets closer to the receiver, the energy across all frequency components rapidly increases, forming a mountain-like shape. The energy reaches the maximum when the car passes by the user's location.

Let us look at two STDFT snapshots in Fig. 4.4(b). For each STDFT snapshot, we observe that the signal power drops as we go from lower frequency components to higher frequency components. This is because higher frequency components experience higher path loss. Let us first choose a particular power threshold value, *e.g.*, $-58dB$, and then find the corresponding highest frequency component whose power level is at or above this threshold. We refer to this frequency value as the *Top-Right* frequency, or *TRF* in short, because if we draw a rectangle whose height is given by the power threshold value (rectangle illustrated in red in Fig 4.4 (b)), the frequency of the top right corner of this rectangle is the frequency we are looking for. When the car gets closer to the user, the signal's TRF value becomes higher. In general, TRF can be calculated as:

$$TRF(n) = \max \left\{ \underset{f}{\text{argmax}}(S(n, f)) \right\},$$

where $S(n, f)$ is the power of the signal at frequency f and time window n. The term $S(n, f)$ is less than the power threshold T, and TRF(n) is the maximum f among all the frequencies whose $S(n, f)$ reaches the power level T. We calculate the TRF value for each slot to form a TRF trace, and *use the TRF trace as the feature for subsequent car detection*.

Blurred Edge Detection Based Feature Extraction

A straightforward way of generating a TRF trace is to adopt a fixed power threshold across all slots. Implementing this method in practice, however, is very challenging because it is hard to choose a suitable power threshold that works across cars and road conditions. Furthermore, adopting a single power threshold is not robust in real environments; in an outdoor environment, activities in the environment (*e.g.* bird singing, footsteps, etc.) can easily lead to high frequencies whose power exceeds the power threshold. As a result, the resulting TRF trace will not be smooth, sometimes even discontinuous.

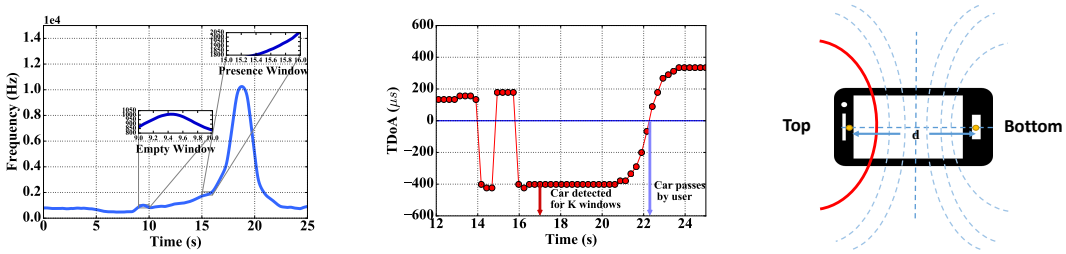
In order to address the challenges associated with the simple threshold-based TRF trace extraction, we next develop an edge detection based method to extract TRF

values from the STDFT result robustly and efficiently. As shown in Figure 4.4 (a), when driving closer to the user, the car generates sound signal that exhibits a continuous, blurred slope. The edge of this slope is the TRF trace of the car. We then adopt a Blurred Edge Detector (BED) to extract this edge, which is more robust than the power threshold based approach.

Our BED-based TRF extraction approach relies on both spectral and temporal information to identify the intensity gradients of the spectrogram. These gradients contain the desired TRF trace. Specifically, we adopt a Canny edge detector [83], a widely used technique, in *Auto++*. It includes the following steps: (1) We suppress those frequency components whose power is below a threshold by setting them to -100 dB. (2) We apply a Gaussian filter to smooth out noisy dots and lines. (3) We calculate the intensity gradient of the spectrogram to obtain the outline of the area that has a consistent power increase. (4) We apply non-maximum suppression to thin the area into edges. (5) We track the main edges by removing all the edges that are weak and not connected to strong edges. After finding the edge (shown in Fig. 4.4(c)), we fill the missing points on the edge with their previous neighbours' values to form a consecutive series. We call this series a *TRF trace*. We conduct such edge detection at the granularity of a *processing window*. Each processing window consists of multiple slots; we detect the edge within each window and uses the moving average over 200 slots to smooth the results.

4.4.4 Unsupervised Car Presence Detection and Counting

Next, we explain our car detection scheme, which is light-weight and unsupervised. Our car presence detection scheme includes two steps– processing window classification and event detection. In the window classification phase, we classify the current processing window (that contains all the audio samples recorded in the last .5 second) as either 'empty' (meaning no car is detected within this window) or 'car present' (meaning at least one car is detected within this window). Our window classification is based upon whether the TRF values within the window consistently increase. If a large fraction of TRF values are greater than or equal to their immediately preceding TRF value, then we



(a) (b) (c)

Figure 4.5: (a) TRF values in a car-present window often exhibit different patterns from those of an empty window that has short-term ambient noise. In the former case, the TRF values often continuously increase while in the latter case, the TRF values first increase and then decrease. (b) The TDoA values when the car gets closer to the user. The red arrow marks the time when the car has been detected in K consecutive windows, while the blue arrow marks the time when the car passes the smartphone. (c) Example hyperbolas we have for direction estimation. The red solid hyperbola on the left plane indicates that a car is driving from the left side.

label the window as a car-present window. This approach ensures only the continuously increasing portion of a TRF trace is counted as a car presence event, eliminating the peaks caused by ambient noise and other activities. For example, Fig. 4.5 (a) shows the TRF values in a typical empty window with a high noise level, as well as the TRF values in a window that contains an approaching car.

In some cases, it is important to find out how many cars are around a user. In order to get the car count, we merge consecutive car-present windows within a short time interval to avoid counting the same car multiple times.

Finally, we note that our detection scheme can work with other features. In our evaluation (Section 2.4), we also used sound pressure level (SPL) in our detection algorithm.

4.4.5 Unsupervised Car Direction Estimation

After detecting an approaching car, we estimate the car's driving direction. We attempt to do so by measuring the time difference of arrivals (TDoA) between sound signals captured by the microphones (if multiple are available on the mobile device). Traditionally, TDoA has been used to provide precise location information for the sound source [72]. To calculate signal TDoA, we apply the cross-correlation function (CCF) on these two

signals $f(\tau)$ and $g(\tau)$ to calculate the lag between them:

$$(f \star g)[\tau] = \sum_{t=-\infty}^{\infty} f^*[t]g[t+\tau], \tag{4.2}$$

where f^* denotes the complex conjugate of f and τ is the delay. The corresponding t value of the highest peak is the estimated delay between the two channels.

However, when applying the above method on smartphones in our problem, we face the following constraints:

1) *Unstable SPL measurements*: TDoA calculation is based on the SPL values from both channels. However, as SPL itself can be easily affected by ambient noise especially when the sound source is far away from the receiver (as in our case), the cross-correlation results of the two channel is usually not accurate enough.

2) *Limited Number of Microphones*: Precise angle calculation requires at least three microphones – we can triangulate using the three sets of TDoA values. However, most of the off-the-shelf smartphones are equipped with one or two microphones. With two embedded microphones, we can only calculate one set of TDoA values.

3) *Low Sampling Rate*: The TDoA resolution is highly related to the sampling rate of the recording device. The smartphone's sampling rate is not higher than $44Khz$, which is not sufficient for high TDoA resolution.

4) *Tiny Distance between Two Microphones:* The distance d (as shown in Fig. 4.5 (c)) between two microphones determines the granularity of the distinguishable TDoA as well as the effective detection distance. On smartphones, the distance between microphones is usually too small to provide sufficient resolution.

In order to address the challenge of unstable SPL measurements, we propose to start TDoA calculation only when we have detected the same car for K consecutive windows. At this point, the SPL measurements are more stable, leading to more accurate TDoA results and direction estimation.

Fig. 4.5 (b) shows how the TDoA value varies as the car approaches while the user remains stationary on the sidewalk. As soon as a car has been detected in 4 consecutive windows at the 17th second (marked by the red arrow), the TDoA value becomes much more stable than before. It remains stable until the 21st second. In this example,

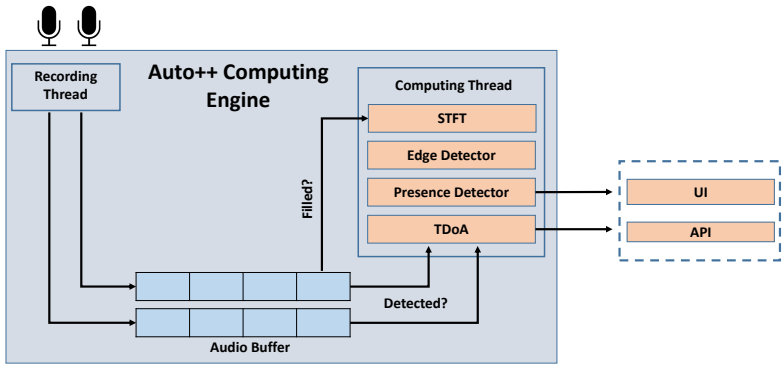


Figure 4.6: Implementation of *Auto++*Computing Engine on Android platform

the car passes the user at the 22nd second (marked by the blue arrow). Even when the TDoA value stabilizes, its absolute value may not be directly used to localize the car, as it is limited by many geometric parameters such as the angle and the distance between the smartphone and the road. Instead, we only consider the sign of the first valid TDoA value and estimate from which direction the car is driving based on the sign. Fig. 4.5 (c) illustrates several possible hyperbolas determined by the measured TDoA values. When we have a negative TDoA value, it indicates the sound source is located at a hyperbola on the left side (marked by red). That is, a car is coming from the smartphone's top side.

Finally, we note that, when applications require high TDoA resolution and direction estimation, we could address the hardware-related challenges 2), 3) and 4) by connecting external microphones with higher specs to smartphones.

4.5 Android Implementation of *Auto++*

We have implemented *Auto++*using the Java and OpenCV native library on Android platform. The raw audio is recorded at a 44 KHz frequency, with a 16 bit pulse-code modulation (PCM). To achieve real-time processing and event updating, we have optimized the code base of *Auto++*and adopted concurrent threads to avoid stalling either sensing/recording or signal processing. As shown in Fig 4.6, the Recording thread handles data collection from microphones and populates the processing buffer, while

the Computing thread processes the buffered data and sends the results (car detection + direction estimation) to a user interface or applications (through API).

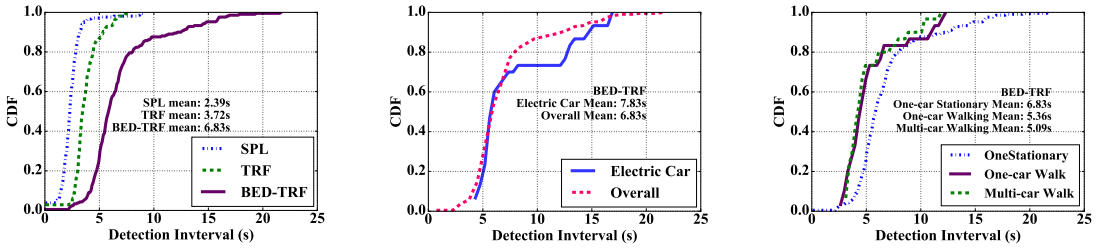
Memory and CPU Usage Profiling: We have tested *Auto++* on both Nexus 6 and Nexus 6P. The core processing engine of *Auto++*runs on-demand, *i.e.*, it is only triggered when specific contexts such as walking on streets are sensed through GPS and inertial sensors (as discussed in Section 4.3). As a result, we consider the processing engine is in one of the three stages: idle, presence detection, and direction estimation.

The processing engines stays idle most of the time, and enters the car detection stage when the user-specified context is detected. In the presence detection stage, the engine performs sound recording, STFT calculation, edge detection, presence detection. When *Auto++*detects a car-present window, it enters the direction estimation stage. In the direction estimation stage, the system needs to first detect the car's presence in $K-1$ consecutive windows, and then perform TDoA calculation via cross-correlation function.

Table 4.1 summarizes the memory and CPU usage profiling of *Auto++*on the two phones. Memory allocation is dynamically done by Android OS when the app boots up. During the execution of *Auto++*, its total memory consumption remains within the pre-assigned range, which is 36.4 MB on Nexus 6 and 32.8 MB on Nexus 6P. Direction estimation consumes the most CPU cycles than the other two stages, and its CPU utilization is only 3.2% for Nexus 6 and 1.8% for Nexus 6P. As a result, we believe our *Auto++*computing engine is lightweight enough to become one of the default background services on mobile platforms.

Table 4.1: *Auto++*'s memory and CPU usage profiling. Among the three stages, direction estimation consumes the most CPU cycles, but still very low, 3.2% on Nexus 6 and 1.8% on Nexus 6P. The memory consumption remains within the pre-assigned memory range during the execution.

	Nexus 6/6P (Idling)	**Nexus 6/6P (Presence Detection)**	**Nexus 6/6P (Direction Estimation)**
Memory (MB)	36.4/32.8	36.4/32.8	36.4/32.8
CPU (%)	0/0	2/1.1	3.2 /1.8



(a) (b) (c)

Figure 4.7: (a) CDF of the three algorithms' detection intervals shows that BED-TRF provides the earliest detection; (b) CDF of the electric car (Volt)'s detection intervals show that *Auto++*can detect electric car as well as other cars (BED-TRF used here); (c) Average detection interval becomes shorter when the user is walking, but an average detection interval of 5.36 second is still very timely. When there are multiple cars, the average detection interval is only degraded by 6%.

4.6 Evaluation

Our experiments are aimed at evaluating the two key functionality for *Auto++*: (1) detecting the approaching car on a road when the user is walking towards the road or on the sidewalk, and (2) estimating the direction from which the car is approaching. We have conducted extensive data collection to evaluate *Auto++*'s performance in these cases. Our results show that *Auto++*can reliably detect a vehicle's presence 6.8 seconds in advance on a parking lot. On more crowded roads, *Auto++*can count the approaching cars with an average error of .6 cars per minute, when the actual number of passing cars is no more than 6 per minute. *Auto++*can estimate an approaching car's direction with 84.3% probability; its accuracy increases to 93.3% when multiple cars are driving in the same direction.

4.6.1 Accurate and Early Car Detection

Setting I: Stationary User & Single Car: We first evaluated how well *Auto++*can detect approaching cars on an outdoor parking lot, where the user stood still on the sidewalk while the car drove closer. We conducted the following experiments to collect audio data when a car was approaching the user. We started an experiment by having the emulated user wave to the driver, who then began to drive the car from 150 meters away towards the user's direction. In this set of experiments, we used a Google Nexus 6,

which was fixed on a tripod, to record the sound via both microphones. To reduce the impact of wind, we mounted windscreens on the microphones. We ended an experiment after the car drove 20 meters past the user. Throughout the entire experiment, we controlled the driving speed to be constant and under the required speed limit. We repeated this experiment to collect data over 7 vehicle models which covered a large fraction of vehicles operated in North America, including electric cars, sedans, sports cars, and sport utility vehicles.

In total, we collected 210 audio tracks that contain an approaching car using the above method, each audio track 20 seconds long. We evaluated the two proposed detection algorithms: TRF, which performs feature extraction based upon a fixed power threshold, and BED-TRF, which performs feature extraction using blurred edge detection. We compared their performance with a baseline sound pressure level (SPL) based detection algorithm – classifying an audio track as 'containing a car' when its SPL is above a threshold. Considering that SPL is highly sensitive to ambient noise, we applied a high-pass filter with the cutoff frequency of 1000 Hz to reduce the impact of noise.

In the evaluation, our objective is to report whether the detection algorithm can accurately detect whether a car is present in an audio track, and how early can it detect the car. The metric we use here is *detection interval*, the interval between the time when a car is first detected and the time when the car is expected to pass the user's location. The larger the detection interval is, the earlier we could detect the car. We present the detection interval breakdowns in Table 4.2. Among the three algorithms, BED-TRF fares the best, while SPL the worst. Even when the car is 4 seconds way, BED-TRF can detect its presence with 91% probability. When the car is 2 seconds away, BED-TRF and TRF can detect car presence with the 100% and 97% probability, respectively. With the data sets we have, we observe the false positive rates are 0 for all algorithms. This shows that BED-TRF has a reliable detection performance.

Fig. 4.7(a) depicts the Cumulative Distribution Function (CDF) of the detection intervals of all three algorithms. The average detection interval of SPL, TRF, and

BED-TRF is 2.4, 3.7 and 6.8 seconds, respectively, with BED-TRF significantly outperforming the other two. BED-TRF has the best performance because its feature extraction is the most robust. In the rest of the evaluation, we thus focus on BED-TRF.

Finally, Fig. 4.7(b) shows even *an electric car can be accurately and timely detected* by *Auto++*. We find that the mean detection interval for the electric car (Chevrolet Volt) is 7.8 seconds, which is even better than the overall mean detection interval. This result is very encouraging as there is a fear that electric cars are too quiet, and thus dangerous to the pedestrian.

Setting II: Mobile User & Multiple Cars: Next, we evaluated how well *Auto++*can detect approaching cars while the user is walking on the sidewalk of a parking lot (smartphone in his hand), with one or more cars coming from the same direction. When we had multiple cars, we made sure that they follow the preceding car after a safe distance (more than 2 seconds away). In this setting, we collected 60 audio tracks in total.

Compared to the first setting, we varied two parameters in this setting: user activity (standing still vs. walking), and number of approaching cars (one vs. two). We report the CDF of the detection intervals in Fig. 4.7(c). We observe that, user activity has larger impact on the average detection interval – a walking user's average detection interval is 5.4 seconds while a standing user's detection interval is 6.8 seconds, but we are still able to detect the presence with 90% of probability when a car is 3 seconds away. The number of cars has a much smaller impact. For a walking user, when the number of approaching cars changes from 1 to 2, the average detection interval changes

Table 4.2: Detection interval distribution breakdown. BED-TRF can detect cars when they are 4 seconds aways with 91% of probability

detection interval	SPL	TRF	BED-TRF
>0.5s	0.96	0.97	1
>2s	0.71	0.97	1
>4s	0.03	0.3	0.91
>6s	0.02	0.07	0.46
>8s	0.02	0.25	0.2

from 5.4 seconds to 5.1 seconds, and we can detect a car's presence with 93.3% of probability when a car is 3 seconds away.

Setting III: Noisy Environments: In the third set of experiments, we investigated whether our presence detection algorithm is robust enough to detect cars in rather noisy environments. Towards this goal, we recorded the ambient sound in several typical environments frequented by pedestrians, including shopping centers, campus roads, and residential areas during busy hours on a weekday. Typical sound sources of the ambient noise in these scenarios include steps, talking, door open/close activities, car sound from a far distance, etc. We recorded the sound in each environment for a total length of 10 minutes, consisting of 30 20-second clips. We overlaid these noise clips over those that contain an approaching car (which we collected in Setting I), emulating a 'car-running-in-a-noisy-environment' scenario.

Table 4.3 summarizes BED-TRF's true positive rate and false positive rate in the original quiet environment (parking lot in Setting I) as well as three noisy environments. We observe that, in noisy environments, the detection performance does degrade, but the results are still quite encouraging. For example, on campus roads with students walking/chatting and cars/school buses running, we can detect approaching cars with an average true positive rate of 97.2%, an average false positive rate of 3.3%, and an average detection interval of 4.2 seconds for true positive events. As expected, the results at the shopping center are worse than those on a campus road and in a residential area, because there are many more activities there that generate ambient noise, such as people going in/out of stores, cars running in the shopping center parking lot, etc. In our future work, we will try to improve the detection performance at a shopping center by learning and recognizing noise generated by these different activities.

Setting IV: Counting cars by Mobile User in Crowded Areas: Finally, we investigate whether *Auto++*can detect and count approaching cars in areas with a relatively high traffic volume, while the user walking around in the area (on sidewalks) with the smartphone in hand. In this set of experiments, we collected data on a campus main road, where cars were not driven by our participants, but by regular drivers. In

total, we recorded 180 minutes of audio data, with each audio track 1 minute long. We extracted the ground truth (car passing the user and the corresponding timestamps) by locating the TRF peaks in each trace offline. We then compare the number of cars detected by *Auto++*with the ground truth, and call the absolute difference of the two as *counting error*. If two cars are very close to each other (less than 2 seconds away from each other), we consider them as one car.

We report the counting error histogram with respect to different traffic volumes in Fig. 4.8 (d). The results show that the average counting error is 0.58 car per minute when the user is standing still, and 1.04 car per minute when the user is walking, when the actual traffic volume is 5 or 6 cars per minute. When the actual car number per minute is greater than 8, the counting error is 0.91 (standing) and 2.02 (walking) per minute. This result indicates that *Auto++*can accurately detect and count approaching cars when the traffic volume is moderate. As a result, we believe that *Auto++*offers a viable approach for fine-grained traffic monitoring when existing traffic monitoring tools are insufficient.

4.6.2 Driving Direction Estimation

Next, we evaluated the accuracy of our car direction estimation. In this use case, we used the same audio tracks that we collected in Sec. 4.6.1 in the first three settings. In our experimental setup, the user was walking on the sidewalk along the road, so the car was approaching either from the back or the front. For each audio track, our estimation can be correct or incorrect. We report the number of correct estimations or the correct estimation rate (the ratio between the number of correct estimations over

Table 4.3: *Auto++*'s detection accuracy at campus roads and residential areas is only slightly worse than the performance on quiet parking lots. We see more degradation in detection accuracy when we use *Auto++*at a shopping center because there are a multitude of activities that generate strong ambient noise.

	Quiet Parking Lot	**Campus Road**	**Residential Area**	**Shopping Center**
Detection Interval (s)	6.8	4.2	3.6	3.2
TPR (%)	100	97.2	94.8	83.8
FPR (%)	0	3.3	13.3	13.3

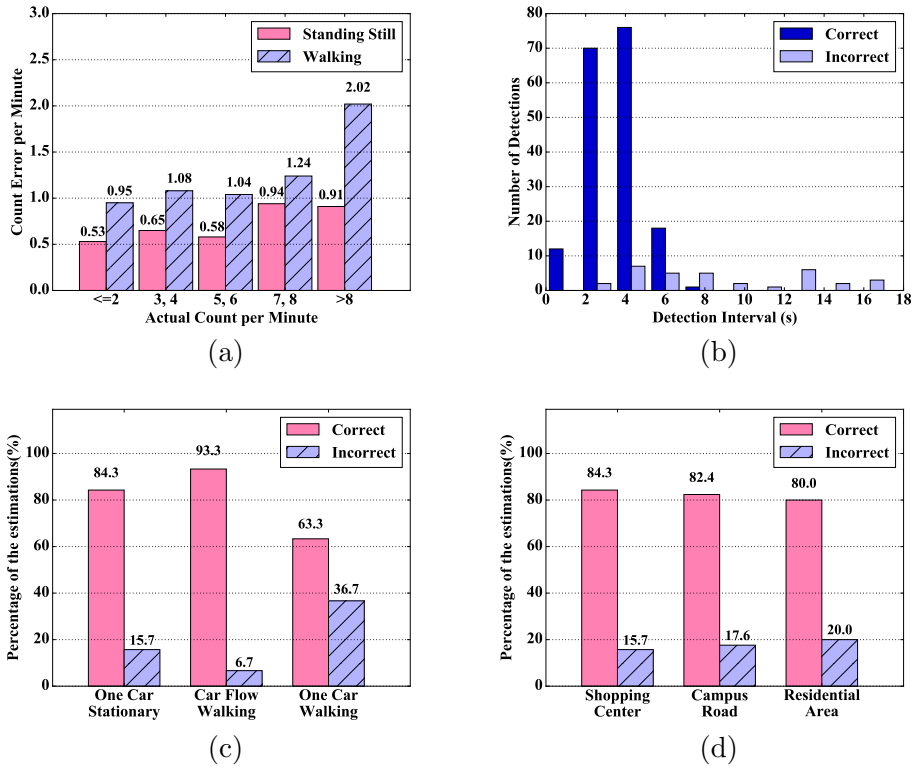


Figure 4.8: (a) The average counting error is 0 for standing users and 0.49 car per minute for walking users, when the actual traffic $\leq$ 2 cars per minute. The average counting error increases to 0.9 for standing users and 1.5 for walking users when the ground truth is 5 or 6 cars per minute. (b) We correctly estimated the car driving direction with probability of 84% (177 out of 210) for all single car test traces. (c) We correctly estimated the driving direction of car flow and single with probability of 93% (28 out of 30) and 67%(20 out of 30), when the user is walking. (d) We correctly estimated the driving direction of cars with probability of 84.3%.

the total number of audio tracks) to evaluate the estimation accuracy.

Setting I: Stationary User & Single Car: We first demonstrate that *Auto++*can estimate a car's driving direction when the user is stationary. Fig. 4.8 (a) presents the histograms of correct and incorrect direction estimations (out of 210 total audio tracks) at different times. The results show that, *Auto++*can correctly estimate the driving direction with 84.3% of the time when the car is 3.5 seconds away.

Setting II: Mobile User & Multiple Cars: Next, we evaluate *Auto++*'s direction estimation accuracy when multiple cars approach a walking user in Fig. 4.8 (b). When a user is walking, the SPL measures become less stable, and the correct estimation rate goes down, i.e., 63.3% in our case. However, when multiple cars follow each other to

approach the walking user, *Auto++*can estimate their direction with a much higher correct estimation rate, 93.3%.

Setting III: Noisy Environments: Finally, we show that *Auto++*can even estimation the driving direction with a reasonable accuracy in various noisy environments. Fig. 4.8(c) shows that its estimation accuracy is similar across three scenarios, as well as on a quiet parking lot, suggesting our direction detection algorithm is not sensitive to environmental noise.

4.7 Related Work

Car-Centered Context Sensing: In the area of pedestrians safety study, many works have been proposed to detect the presence of pedestrian proactively. Different sensing technologies such as piezoelectric sensor, ultrasonic detector, microwave radar can be deployed on the infrastructure or the cars [84]. Piezoelectric sensor requires the contact of the cars or the users, which limits its effective area. Ultrasonic detector and microwave radar are commonly used in the latest model of cars, but it remains a challenge to adopt these sensors in the old ones.

Another category of study focuses on using computer vision to detect the pedestrian. Work by Viola et al. [85] introduced a pedestrian detection platform, which could differentiate a pedestrian from the environment based on the walking motion. Dollar et al. [86] proposed a set of evaluation metrics to investigate pedestrian detection in most of the challenging scenarios. Although pedestrian detection based on computer vision are promising, their performance in the low light environment remains questionable.

Pedetrian-Centered Context Sensing: Besides car-centered methods, user-centered sensing techniques try to extract roadside context from the other angle. Work by Riaz et.al [87] proposed a hybrid system that utilizing wireless sensor network and GPS to reduce the vehicle/pedestrian collisions. This approach requires a scaled deployment, which is difficult to be generalized in realistic. Jain et.al [88] proposed a mobile system that detects the event when a user steps in an intersection. However, it requires the user to mount an additional sensor on the shoes, which adds extra cost and is unlikely

to be ubiquitous.

In contrast, *Auto++*only relies on smartphone without requirement any infrastructure support. Also, it is more flexible and efficient, since microphone is capable of collecting audio data from any angle and audio processing costs less computing resource.

Sound Source Localization: The topic for signal localization is well addressed in the field of signal processing. Earlier work in this domain focused on using the signal difference among multiple receivers and the geometric structure of multiple receivers to locate the incoming signal. Schau et.al [89] propose a solution for source location bu giving time-of-arrival difference measurements when the distance from the source to any arbitrary reference is unknown. This technique was widely applied to military. Damarla et al. [90] developed a special receiver array device to localize the location of a sniper. Typical way to find the time-of-arrival (ToA) difference is to use cross correlation function in time domain [76]. However, due to the hardware limitation and insufficient temporal information in our system, we can not accurately locate the car using these techniques. Another work [75] propose a method to find the ToA difference by using internal phase difference (IPD). This approach requires a high signal to noise rate, but the signal is heavily prone to the surrounding noise in our usage scenarios.

To detect a moving object with acoustic signal, Doppler shift is widely used in many studies. Work by Cheng et.al [72] proposed a system a system to localize an underwater moving object using acoustic sensor network. Chan et.al [91] proposed a system to localize a moving object by sending and receiving RF signals to calculate the Doppler shift. These method, however, require a pre-known distinctive frequency component of the signal, which is difficult to find in our scenario.

4.8 Conclusions

We have presented *Auto++*, an unsupervised approach for detecting approaching cars. *Auto++*depends on non-obvious observation of the sounds that car tires make, and thus is applicable even to electric vehicles, which are otherwise quiet compared to gas

powered cars. The novel feature Top-Right Frequency (TRF) extracted by blurred edge detector (BED), was able to detect a car with 91% accuracy even four seconds away. Further, our direction detection algorithm can detect the driving direction of approaching cars in most of the instances. Finally, *Auto++*can provide car counting service with only 0.9 counting error per minute when the actual count per minute is beyond 8. We believe that the design, implementation and evaluation of our results present important practical contributions towards enabling and enriching more pedestrian-centric applications.

Chapter 5

Conclusion and Proposed Research

5.1 Summary

In conclusion, this dissertation investigates the challenges in future Internet of Things. Specifically, we make the following contributions:

- *MF-IoT:* We propose a generic network architecture that satisfies the requirements placed by the emerging IoT systems, namely, global reach-ability, mobility, communications diversity, and resource efficiency. We archive this objective by creating a network-layer dialect (Local Unique IDentifier, LUID) in a local IoT domain and adopt gateway to efficiently translate between GUID's that are used in the core network and the corresponding LUID's.
- *AggMEC:* Next, to overcome the challenge raised by a tremendous amount of IoT data, we introduce a clustering-based algorithm that utilizes our proposed cost function to minimize the total network traffic. In order to support efficient in-network aggregation, we design and implement our system over MobilityFirst – a clean-slate network architecture.
- Auto++*:* We leverage the pervasiveness of mobile devices to design and implement an approaching car sensing system, which is based on non-obvious observation of the sounds that car tires make. This new user context information enables a number of applications such as safety alert, traffic monitoring, and AR application enhancement.

5.2 End Note

After a decade of development of the IoT, a large variety of communication technologies has gradually emerged, yet there is a large diversity of application domains and network

domains. Such heterogeneity and fragmentation of the IoT systems are one of the main challenges in the next decade. In the era of 5G, with the availability of ubiquitous and scalable connectivity technology, Internet of Things is powered to emerge globally and seamlessly. The network architecture design should not only adopt the evolution of cellular technology to remove the obstacles across different domains, but also integrates with application logic to provide richer services for the users. For example, a user should be able to subscribe and receive environmental sensing data via networks, which is generated by agnostic sensors or services. Moreover, this data could be pre-processed data, data summary, or even result generated by Artificial Intelligent (AI) model. Ultimately, the future Internet of Things will connect objects beyond "things", and deliver smarter representations of the physical environment that we live in.

References

[1] "The Internet of Things — How the Next Evolution of the Internet Is Changing Everything," https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, Apr. 2011.

[2] "Alljoyn framework," https://allseenalliance.org/framework.

[3] "Iotivity," https://www.iotivity.org/.

[4] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: A Robust and Trustworthy Mobility-Centric Architecture for The Future Internet," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 16, no. 3, pp. 2–13, 2012.

[5] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFCs 6282, 6775. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[6] S. C. Nelson, G. Bhanage, and D. Raychaudhuri, "GSTAR: Generalized Storage-Aware Routing for Mobilityfirst in the Future Mobile Internet," in *Proceedings of the sixth international workshop on MobiArch*. ACM, 2011, pp. 19–24.

[7] X. Wu, S. Subramanian, R. Guha, R. G. White, J. Li, K. W. Lu, A. Bucceri, and T. Zhang, "Vehicular communications using dsrc: challenges, enhancements, and evolution," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 9, pp. 399–408, 2013.

[8] J. Han, D. Kim, M. Lee, and M. Sunwoo, "Enhanced road boundary and obstacle detection using a downward-looking lidar sensor," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 3, pp. 971–985, 2012.

[9] Z. Alliance, "ZigBee IP Specification," *ZigBee 095023r10, Work in Progress, July*, 2010.

[10] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt

[11] "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)," *IEEE Std 802.15.4-2003*, pp. 0_1–670, 2003.

[12] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)," RFC 7641 (Proposed Standard), Internet Engineering Task Force, Sep. 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7641.txt

[13] T. Vu, A. Baid, Y. Zhang, T. D. Nguyen, J. Fukuyama, R. P. Martin, and D. Raychaudhuri, "DMap: A Shared Hosting Scheme for Dynamic Identifier to Locator Mappings in the Global Internet," in *IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 698–707.

[14] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A Global Name Service for A Highly Mobile Internetwork," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 247–258.

[15] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550 (Proposed Standard), Internet Engineering Task Force, Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6550.txt

[16] J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks," RFC 5867 (Informational), Internet Engineering Task Force, Jun. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5867.txt

[17] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561 (Experimental), Internet Engineering Task Force, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[18] Z. Alliance, "ZigBee 2007 Specification," *Online: http://www. zigbee. org/Specifications/ZigBee/Overview. aspx*, vol. 45, p. 120, 2007.

[19] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. Ramakrishnan, "COPSS: An Efficient Content Oriented Publish/Subscribe System," in *Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2011, pp. 99–110.

[20] M. Arumaithurai, J. Chen, E. Monticelli, X. Fu, and K. Ramakrishnan, "Exploiting ICN for Flexible Management of Software-Defined Networks," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 107–116.

[21] L. A. Adamic and B. A. Huberman, "Zipfs law and the Internet," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[22] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A Survey on the IETF Protocol Suite for the Internet of Things: Standards, Challenges, and Opportunities," *Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[23] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos *et al.*, "Named Data Networking (NDN) Project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[24] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information Centric Networking in the IoT: Experiments with NDN in the Wild," in *Proceedings of the 1st International Conference on Information-centric Networking*, ser. ICN '14. ACM, 2014, pp. 77–86.

[25] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNext)*. ACM, 2009, pp. 1–12.

[26] Y. Zhang, D. Raychadhuri, R. Ravindran, and G. Wang, "ICN based Architecture for IoT," IETF Internet Draft draft-zhang-iot-icn-architecture-00. IRTF, Tech. Rep., 2013.

[27] M. Amadeo, C. Campolo, and A. Molinaro, "Multi-source Data Retrieval in IoT via Named Data Networking," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 67–76.

[28] S. Li, Y. Zhang, D. Raychaudhuri, R. Ravindran, Q. Zheng, and L. Dong, "IoT Middleware Architecture over Information-Centric Network," in *Globecom Workshop on Information Centric Networking Solutions for Real World Applications (ICNSRA)*. IEEE, 2015, pp. 19–24.

[29] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying iot traffic in smart cities and campuses," in *Computer Communications Workshops (INFOCOM WKSHPS), 2017 IEEE Conference on*. IEEE, 2017, pp. 559–564.

[30] "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025," https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[31] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, no. 2, 2007.

[32] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 3–3.

[33] I. Stojmenovic, "Machine-to-machine communications with in-network data aggregation, processing, and actuation for large-scale cyber-physical systems," *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 122–128, 2014.

[34] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on wireless communications*, vol. 1, no. 4, pp. 660–670, 2002.

[35] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM Sigmod record*, vol. 31, no. 3, pp. 9–18, 2002.

[36] E. Pournaras, M. Warnier, and F. M. Brazier, "A generic and adaptive aggregation service for large-scale decentralized networks," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, p. 19, 2013. [Online]. Available: http://www.casmodeling.com/content/1/1/19

[37] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[38] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[39] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 16, no. 3, pp. 2–13, 2012.

[40] S. Mukherjee, S. Sriram, and D. Raychaudhuri, "Edge-aware inter-domain routing for realizing next-generation mobility services," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[41] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine learning*, vol. 75, no. 2, pp. 245–248, 2009.

[42] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[43] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.

[44] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[45] G. R. Bradski, "Real time face and object tracking as a component of a perceptual user interface," in *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*. IEEE, 1998, pp. 214–219.

[46] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," Downloaded from https://crawdad.org/epfl/mobility/20090224, Feb. 2009.

[47] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[48] "Wigle: Wireless network mapping," Downloaded from https://wigle.net/.

[49] W. Zhang, J. Chen, Y. Zhang, and D. Raychaudhuri, "Towards efficient edge cloud augmentation for virtual reality mmogs," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 8:1–8:14. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134463

[50] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[51] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking.* ACM, 2000, pp. 56–67.

[52] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Transactions on Networking (ToN)*, vol. 11, no. 1, pp. 2–16, 2003.

[53] S. Lindsey and C. S. Raghavendra, "Pegasis: Power-efficient gathering in sensor information systems," in *Aerospace conference proceedings, 2002. IEEE*, vol. 3. IEEE, 2002, pp. 3–3.

[54] F. Xiangning and S. Yulin, "Improvement on leach protocol of wireless sensor network," in *Sensor Technologies and Applications, 2007. SensorComm 2007. International Conference on.* IEEE, 2007, pp. 260–264.

[55] E. Pournaras, M. Warnier, and F. M. Brazier, "A generic and adaptive aggregation service for large-scale decentralized networks," *Complex Adaptive Systems Modeling*, vol. 1, no. 1, p. 19, 2013.

[56] C. Lochert, B. Scheuermann, C. Wewetzer, A. Luebke, and M. Mauve, "Data aggregation and roadside unit placement for a vanet traffic information system," in *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking.* ACM, 2008, pp. 58–65.

[57] S. Kashyap, S. Deb, K. Naidu, R. Rastogi, and A. Srinivasan, "Efficient gossip-based aggregate computation," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* ACM, 2006, pp. 308–317.

[58] R. Zhou, K. Hwang, and M. Cai, "Gossiptrust for fast reputation aggregation in peer-to-peer networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 9, pp. 1282–1295, 2008.

[59] J. Chon and H. Cha, "Lifemap: A smartphone-based context provider for location-based services," *IEEE Pervasive Computing*, vol. 10, no. 2, pp. 58–67, 2011.

[60] S. Nath, "Ace: exploiting correlation for energy-efficient and continuous context sensing," in *Proceedings of the 10th international conference on Mobile systems, applications, and services.* ACM, 2012, pp. 29–42.

[61] C. Xu, S. Li, G. Liu, Y. Zhang, E. Miluzzo, Y.-F. Chen, J. Li, and B. Firner, "Crowd++: Unsupervised speaker count with smartphones," in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing.* ACM, 2013, pp. 43–52.

[62] J. Froehlich, M. Y. Chen, S. Consolvo, B. Harrison, and J. A. Landay, "Myexperience: a system for in situ tracing and capturing of user feedback on mobile phones," in *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007, pp. 57–70.

[63] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *IEEE Communications magazine*, vol. 48, no. 9, 2010.

[64] F. Suard, A. Rakotomamonjy, A. Bensrhair, and A. Broggi, "Pedestrian detection using infrared images and histograms of oriented gradients," in *Intelligent Vehicles Symposium, 2006 IEEE*. IEEE, 2006, pp. 206–212.

[65] C. Wojek, S. Walk, and B. Schiele, "Multi-cue onboard pedestrian detection," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 794–801.

[66] M. Bertozzi, L. Bombini, P. Cerri, P. Medici, P. C. Antonello, and M. Miglietta, "Obstacle detection and classification fusing radar and vision," in *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE, 2008, pp. 608–613.

[67] "Google map," https://maps.google.com, 2017, accessed: 2017-01-31.

[68] "Apple map," http://www.apple.com/ios/maps/, 2017, accessed: 2017-01-31.

[69] Q. Xu, T. Mak, J. Ko, and R. Sengupta, "Vehicle-to-vehicle safety messaging in dsrc," in *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*. ACM, 2004, pp. 19–28.

[70] T. Wang, G. Cardone, A. Corradi, L. Torresani, and A. T. Campbell, "Walksafe: a pedestrian safety app for mobile phone users who walk and talk while crossing roads," in *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. ACM, 2012.

[71] E. Romera, L. M. Bergasa, and R. Arroyo, "A real-time multi-scale vehicle detection and tracking approach for smartphones," in *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE, 2015, pp. 1298–1303.

[72] W. Cheng, A. Thaeler, X. Cheng, F. Liu, X. Lu, and Z. Lu, "Time-synchronization free localization in large scale underwater acoustic sensor networks," in *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on*. IEEE, 2009, pp. 80–87.

[73] F. Zheng, G. Zhang, and Z. Song, "Comparison of different implementations of mfcc," *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582–589, 2001.

[74] F. Gustafsson and F. Gunnarsson, "Positioning using time-difference of arrival measurements," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*. IEEE, 2003.

[75] W. Zhang and B. D. Rao, "A two microphone-based approach for source localization of multiple speech sources," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 18, no. 8, pp. 1913–1928, 2010.

[76] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015.

[77] Y. T. Chan and K. Ho, "A simple and efficient estimator for hyperbolic location," *Signal Processing, IEEE Transactions on*, vol. 42, no. 8, pp. 1905–1915, 1994.

[78] B. Kusy, A. Ledeczi, and X. Koutsoukos, "Tracking mobile nodes using rf doppler shifts," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007.

[79] W. Huang, Y. Xiong, X.-Y. Li, H. Lin, X. Mao, P. Yang, and Y. Liu, "Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014.

[80] J. N. Holmes, W. J. Holmes, and P. N. Garner, "Using formant frequencies in speech recognition." in *Eurospeech*, vol. 97, 1997, pp. 2083–2087.

[81] R. Kaune, "Accuracy studies for tdoa and toa localization," in *Information Fusion (FUSION), 2012 15th International Conference on*. IEEE, 2012, pp. 408–415.

[82] A. Sheth, S. Seshan, and D. Wetherall, "Geo-fencing: Confining wi-fi coverage to physical boundaries," in *International Conference on Pervasive Computing*. Springer, 2009, pp. 274–290.

[83] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[84] F. Bu and C.-Y. Chan, "Pedestrian detection in transit bus application: sensing technologies and safety solutions," in *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*. IEEE, 2005.

[85] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *International Journal of Computer Vision*, vol. 63, no. 2, pp. 153–161, 2005.

[86] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009.

[87] Z. Riaz, D. Edwards, and A. Thorpe, "Sightsafety: A hybrid information and communication technology system for reducing vehicle/pedestrian collisions," *Automation in construction*, vol. 15, no. 6, pp. 719–728, 2006.

[88] S. Jain, C. Borgiattino, Y. Ren, M. Gruteser, Y. Chen, and C. F. Chiasserini, "Lookup: Enabling pedestrian safety services via shoe sensing," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015.

[89] H. Schau and A. Robinson, "Passive source localization employing intersecting spherical surfaces from time-of-arrival differences," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 35, no. 8, pp. 1223–1225, 1987.

[90] T. Damarla, L. M. Kaplan, and G. T. Whipps, "Sniper localization using acoustic asynchronous sensors," *Sensors Journal, IEEE*, vol. 10, no. 9, pp. 1469–1478, 2010.

[91] Y. T. Chan and J. Towers, "Passive localization from doppler-shifted frequency measurements," *Signal Processing, IEEE Transactions on*, vol. 40, no. 10, pp. 2594–2598, 1992.