
Mai A. Alduailij

Western Michigan University, mai.alduailij@gmail.com

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the Computer Sciences Commons

Recommended Citation

https://scholarworks.wmich.edu/dissertations/1162

This Dissertation-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Dissertations by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.
DESIGN OF OPPNET VIRTUAL MACHINE FOR OPPORTUNISTIC RESOURCE UTILIZATION NETWORKS: A UNIVERSAL STANDARD FOR APPLICATION-LEVEL RESOURCE SHARING

by

Mai A. Alduailij

A dissertation submitted to the Graduate College in partial fulfillment of the requirements for the Degree of Doctor of Philosophy
Computer Science
Western Michigan University
December 2015

Doctoral Committee:
Leszek Lilien, Ph.D., Chair
Ala Al-Fuqaha, Ph.D.
Zijiang (James) Yang, Ph.D.
Ikhlas Abdel-Qader, Ph.D.
Opportunistic resource utilization networks or oppnets are a paradigm for specialized ad hoc networks. The Oppnet Virtual Machine (OVM) is a collection of primitives designed to create a middleware for application-level resource acquisition by oppnets and oppnet-enabled systems and devices (where the latter are defined as the computational entities able to communicate with other oppnets or oppnet-enabled entities). Acquisition of communication resources is the foundation for acquisition of other resources.

The OVM primitives can be downloaded by any computational entity—making entity oppnet-enabled. OVM ensures that oppnet-enabled entities can communicate and acquire resources in an opportunistic and ad hoc manner. This OVM-based interaction between entities can support the following required oppnet characteristics, which together distinguish oppnets from other collaborative distributed systems: (i) support for the helper paradigm—as the basis for resource acquisition; (ii) universality—regardless of the system or device make or function, and through any communication technologies; (iii) lack of third-party mediators—since interactions among oppnet-enabled entities take place without (trusted) third parties; and (iv) ad hoc operation. We develop and validate the improved second version of the OVM primitives, capable of supporting a very broad and diverse set of applications. As a proof of feasibility, we use our set of OVM primitives to develop non-monolithic OVM-based oppnet middleware that implements a healthcare scenario. The features that distinguish our oppnet middleware from the previous oppnet work are: (i) implementation of the object-oriented OVM primitives in a non-monolithic fashion; (ii) utilization of a wider number of resources; and (iv) employment of more kinds of communication technologies. Our pseudocode is an object-oriented (OO) pseudocode based on the Java code syntax, which allows using some
desirable OO capabilities in the pseudocode. To the best of our knowledge we are the first to use OO pseudocode.

We compare the performance of the monolithic oppnet middleware with the non-monolithic OVM-based oppnet middleware by simulating both middleware for the above scenario. We apply analytical methods to evaluate and compare the performance for the two middleware. The evaluation criteria include, but are not limited to time overhead, resource usage and success rate for the scenario.
Copyright by
Mai A. Alduailij
2015
ACKNOWLEDGMENTS

At the beginning, I praise Allah Almighty for bestowing an abundant amount of gifts on me. This work wouldn’t have been completed without Allah’s grace.

My deepest acknowledgment goes to Professor Lilien for all the guidance, support, and help he gave me through this process. The great deal of knowledge I learned from him and the professional way he delivered it will be a model for me to follow.

To the esteemed committee members: Professor Ala Al-Fuqaha, Professor Zijiang (James) Yang, and Professor Ikhlas Abdel-Qader: thank you very much for your advice and for taking the time to review this work and provide valuable review and feedback.

My warmest gratitude goes to my first mentor and role model, Professor Stephanie August, Loyola Marymount University. I owe the good solid basis I have in Computer Science to you. I cannot forget how a two-hour class with you went so fast and yet was extremely informative and fun. I will always remember how you helped me and my family in every possible way. May God always bliss you and protect you.

My late father, Abdullah, you have instilled all the great values that made me a better person today, you always told us that there is no limit for what we can accomplish in life, and yet we should always stand for what we believe in. I wish you were here to celebrate with us.

To my mother, Aisha, no words, or deeds could ever show how I am grateful to you. Your kind words, your prayers, and your advice are the light that illuminated my path. Regardless of the thousands of miles that separated us, your unconditional love, and your guidance helped me throughout this journey. From the bottom of my heart, thank you.

To my dearest mother in law, Noura: I think about you every single day; I miss your presence, and love to us and to our children. You loved me as if I was your own daughter. You loved my son more than anyone could ever love their grandchild. You left a big hole
in our life that is hard to be replaced. I haven’t had the chance to say Goodbye, but I pray to Allah to bliss you and for us to meet you in the highest paradise.

To my father in law, Abdullah: thank you for loving me and caring about me like your own. I am deeply grateful to you for supporting our decision to study in the US although this means you will not see your son for a long time. Thank you for your generosity in so many ways, and most importantly, thank you for raising such a wonderful man, my husband.

To my loving husband, Faisal: we finally did it! Thank you for standing by my side throughout this journey. Thank you for the endless sacrifice, love and support. You believed in me and constantly encouraged me even when the task seemed impossible. You have been my pillar through ups and downs and I wouldn’t have done it without you.

To my kids: Abdullah, Noura, and Abdulaziz; thank you for making my life colorful and thank you for hanging in there for me, I promise I will make it up for you for all the nights I have been working instead of being with you.

To my brothers, Abdulaziz, and Abdulrahman, you not only helped me accomplish this goal by your constant advice and support, you also provided me with fatherly love that I so much long for. Thank you for your love, help and support.

My sisters, Munira, Reem, Maha and Mona, even though we are miles apart, you have always managed to put a smile on my face. Your kind words, encouragement and prayers helped me on the tough days. Each one of you has a special place in my heart that cannot be replaced.

To my sisters and brothers in-laws, my extended family members and my friends, thank you very much for all your prayers and love.

Mai Abdullah Alduailij
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ............................................................................................... ii  
LIST OF TABLES ....................................................................................................... viii  
LIST OF FIGURES ...................................................................................................... ix  
1. INTRODUCTION .....................................................................................................1  
   1.1 Definitions and Terms ...................................................................................1  
   1.2 Background Information on Oppnets .........................................................3  
      1.2.1 Basic Oppnet Operations ....................................................................3  
      1.2.2 Classification of Helpers ....................................................................4  
      1.2.3 Oppnet Virtual Machine (OVM) .......................................................6  
      1.2.4 Monolithic MicroOppnet v.2.2 ....................................................6  
   1.3 Motivation for Oppnet Virtual Machine .....................................................9  
   1.4 Problem Statement and Research Hypothesis ...........................................11  
   1.5 Scope and Limitations of the Proposed Research .......................................12  
   1.6 Dissertation Organization ..........................................................................13  
2. LITERATURE REVIEW .......................................................................................14  
   2.1 A Review of Technologies Related to Oppnet ...........................................14  
      2.1.1 Resource-Sharing Technologies ....................................................14  
      2.1.2 Connectivity-Based Technologies ...............................................20  
      2.1.3 Specialized and Other Networks ....................................................26  
   2.2 Previous Oppnet Work ..............................................................................29  
3. OPPNET VIRTUAL MACHINE (OVM) ...............................................................31  
   3.1 Definition of OVM ....................................................................................31  
   3.2 Overview of OVM Primitives ....................................................................32  
   3.3 First-Generation OVM .............................................................................32  
      3.3.1 Motivation and Overview ..................................................................32  
      3.3.2 OVM Primitives ............................................................................33  
   3.4 Second-Generation OVM ..........................................................................34  
      3.4.1 Motivation and Overview ..................................................................34
Table of Contents - continued

3.4.2 OVM Primitives ................................................................. 34
3.5 Third-Generation OVM .......................................................... 37
  3.5.1 Motivation and Overview ................................................. 37
  3.5.2 Requirements ................................................................. 37
  3.5.3 OVM Primitives ............................................................... 38
  3.5.4 Applied Object-Oriented Techniques ............................... 40
3.6 Template for Class Hierarchy of the OVM Primitives ............... 42
3.7 Chapter Conclusions .............................................................. 44

4. PROOF OF FEASIBILITY OF OVM ................................................. 45
  4.1 First-Generation OVM .......................................................... 45
    4.1.1 Implementation of OVM Primitives ................................. 45
    4.1.2 MicroOppnet v.2.2 ..................................................... 45
  4.2 Second-Generation OVM ...................................................... 46
    4.2.1 Implementation of OVM Primitives ................................. 46
    4.2.2 MicroOppnet v.4 ....................................................... 47
    4.2.3 Challenges and Lessons Learned .................................. 50
  4.3 Third-Generation OVM ......................................................... 51
    4.3.1 Implementation of the Third-Generation OVM ................. 51
    4.3.2 Implementation of OVM Primitives ................................. 51
    4.3.3 MicroOppnet v.5 ....................................................... 60
    4.3.4 Challenges, Lessons Learned, and Limitations ............... 77
  4.4 Chapter Conclusions .............................................................. 78

5. PERFORMANCE EVALUATION OF THE OVM PRIMITIVES AND COMPARISON OF THE MONOLITHIC VS. MODULAR OVM .......... 79
  5.1 Simulation Scenario ............................................................ 79
  5.2 Simulation Design .............................................................. 79
    5.2.1 Simulation General Assumptions ................................... 80
    5.2.2 Assumptions for Simulated Nodes ................................. 80
Table of Contents - continued

5.2.3 Simulation Helper Assumptions ..............................................81
5.2.4 Simulation Message Assumptions ...........................................81
5.2.5 Simulation Technology/Standard-Related Assumptions ..........88
5.2.6 Simulation Task Assumptions/Setup .......................................95
5.2.7 Simulation Input Parameters ....................................................98
5.2.8 Random Variables for the Simulation .....................................101
5.2.9 Simulation Measures ..............................................................103
5.2.10 Simulation Code .................................................................107

5.3 Simulation Results and Their Discussions .................................107
5.3.1 Average Helper Integration Time for All Inviting Nodes
for All Iterations ......................................................................108
5.3.2 Average Task Runtime for All Iterations ..............................110
5.3.3 Average Time for Oppnet to Succeed for All Iterations .......111
5.3.4 Average Number of Helpers Requested for Help by All
Inviting Nodes for All Iterations .............................................113
5.3.5 Average Number of Admitted Helpers by All Inviting
Nodes for All Iterations .........................................................114
5.3.6 Average Number of Helpers Refusing to Help by All
Inviting Nodes for All Iterations .............................................116
5.3.7 Average Number of Helpers that Performed Urgent
Tasks for All Iterations ..........................................................117
5.3.8 Average Number of Helpers that Performed Unhurried
Tasks for All Iterations ..........................................................118
5.3.9 Success Rates .....................................................................120

5.4 Chapter Conclusions ...............................................................121

6. CONCLUSIONS, SIGNIFICANCE, EXPECTED IMPACTS, AND
FUTURE WORK .............................................................................124
6.1 Conclusions ..............................................................................124
Table of Contents - continued

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2 Significance</td>
<td>126</td>
</tr>
<tr>
<td>6.3 Expected Impacts</td>
<td>126</td>
</tr>
<tr>
<td>6.4 Future Work</td>
<td>127</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>129</td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A. FIRST-GENERATION OVM PRIMITIVES</td>
<td>137</td>
</tr>
<tr>
<td>B. SOURCE CODE INFORMATION</td>
<td>140</td>
</tr>
<tr>
<td>C. DETAILS OF SIMULATION RESULTS</td>
<td>142</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>3.1.</th>
<th>List of second-generation CC primitives.</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.</td>
<td>List of second-generation seed primitives.</td>
<td>35</td>
</tr>
<tr>
<td>3.3.</td>
<td>List of second-generation helper primitives.</td>
<td>36</td>
</tr>
<tr>
<td>3.4.</td>
<td>List of third-generation OVM primitives.</td>
<td>39</td>
</tr>
<tr>
<td>4.1.</td>
<td>Nodes in MicroOppnet v.5.</td>
<td>62</td>
</tr>
<tr>
<td>4.2.</td>
<td>Implemented/emulated devices in MicroOppnet v.5.</td>
<td>74</td>
</tr>
<tr>
<td>4.3.</td>
<td>Implemented/emulated links in MicroOppnet v.5.</td>
<td>74</td>
</tr>
<tr>
<td>5.1.</td>
<td>Assumed helper categories and communication protocols for simulated nodes.</td>
<td>81</td>
</tr>
<tr>
<td>5.2.</td>
<td>Java String Size.</td>
<td>84</td>
</tr>
<tr>
<td>5.3.</td>
<td>Size of message payloads.</td>
<td>86</td>
</tr>
<tr>
<td>5.4.</td>
<td>Message payload reading times for used technologies.</td>
<td>89</td>
</tr>
<tr>
<td>5.5.</td>
<td>Delivery parameters of used communication technologies.</td>
<td>91</td>
</tr>
<tr>
<td>5.6.</td>
<td>Tasks and their running times.</td>
<td>96</td>
</tr>
<tr>
<td>5.7.</td>
<td>Input parameters.</td>
<td>98</td>
</tr>
<tr>
<td>5.8.</td>
<td>Other simulation parameters.</td>
<td>100</td>
</tr>
<tr>
<td>5.9.</td>
<td>Random variables for the simulation.</td>
<td>101</td>
</tr>
<tr>
<td>5.10.</td>
<td>Simulation measures.</td>
<td>103</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1. Seed oppnet and expanded oppnet [LGKY10]......................................................... 4
1.2. MicroOppnet v.2.2 [LGKY10] ................................................................................. 7
3.1. The template for the class hierarchy of the third-generation OVM primitives .................. 43
4.1. MicroOppnet v.2.2 [LGKY10] ............................................................................... 46
4.2. Packages containing the full OVM primitive hierarchy. ............................................. 55
4.3. The class hierarchy for OVM primitives in the Java package. ................................... 56
4.4. The class hierarchy for the OVM primitives in the Android package ......................... 57
4.5. The class hierarchy for the OVM primitives in the NesC package. ............................ 57
4.6. The healthcare scenario............................................................................................. 65
4.7. MicroOppnet v.5 components................................................................................... 68
4.8. Seed pseudocode...................................................................................................... 69
4.9. Pseudocode for NODE_searchHelpers()................................................................ 70
4.10. Pseudocode for NODE_releaseAllChildren()........................................................ 71
4.11. Helper pseudocode.................................................................................................. 72
4.12. Pseudocode for NODE_selectPerformTasks(oppnetTasks). ............................... 73
5.1. Average helper integration time for all inviting nodes for all iterations 
   (the standard deviation indicated as well).................................................................... 109
5.2. Average task runtime for all nodes for all iterations................................................. 110
5.3. Average time for oppnet to succeed for all iterations. ............................................ 112
5.4. Average number of helpers requested for help by all inviting nodes for all iterations.......................................................... 113
5.5. Average number of admitted helpers by all inviting nodes for all iterations.......................... 115
5.6. Average number of helpers refusing to help by all inviting nodes for all iterations.......................................................... 116
5.7. Average number of helpers that performed urgent tasks for all iterations.......................... 118
List of Figures - continued

5.8. Average number of helpers that performed *unhurried* tasks for all iterations .............................................................................................................. 119

5.9. Success rate........................................................................................................... 120
1. INTRODUCTION

This chapter starts by giving definitions of terms used in this Dissertation. The chapter then provides background information on Opportunistic Resource Utilization Networks (oppnets). We then present the motivation for Oppnet Virtual Machine (OVM) implementing the oppnet middleware, the problem statement and the research hypothesis, and the scope and limitations of the proposed research. Finally, the chapter explains the organization of this Dissertation.

1.1 Definitions and Terms

The following terms used in this Dissertation require definitions:

- *Opportunistic resource utilization networks or oppnets* is a new paradigm for specialized ad hoc networks. The salient feature of oppnets is their use of “helpers” to expand in an opportunistic and ad hoc manner. The helpers can be either predefined or ad hoc, and collaborate in acquisition of various resources.

  - An oppnet starts as a *seed oppnet* (a.k.a a *seed* or *seed nodes*), which is a pre-designed (not ad hoc) collection of nodes that initiate oppnet activity. A *seed* can be as small as a single node.
    - *Controller* or a *control center (CC)* consists of an arbitrary subset of seed nodes that controls and manages oppnet (but its tasks can be delegated to other nodes, if desirable). Among others, CC can decide to start oppnet expansion (growth), as described next.

  - A seed oppnet can grow into an *expanded oppnet* by inviting, admitting, and integrating so-called helpers.
    - *Helper* is an entity that can assist an oppnet in its activities, and accepts an invitation to help oppnet.
Before it joins an oppnet, a future helper is a *candidate helper*; a candidate helper can possess diverse capabilities and “skills” potentially useful for the oppnet (such as communication, computing, storing, sensing or actuating capabilities).

A helper selected from among (potentially a broad set of) candidate helpers and integrated into an oppnet becomes an *actual helpers*.

There are two types of helpers w.r.t. their capabilities:

- **Regular helpers**, which are powerful, able to perform all oppnet activities, including inviting and integrating other helpers.

- **Lightweight helpers** a.k.a. *lites* have limited oppnet capabilities, including restricted oppnet communication capabilities (usually due to their weak native).

There are two types of helpers w.r.t. their refusal rights:

- **Reservist helpers** a.k.a. *reservists* that can be ordered to assist an oppnet requesting for help, and are obliged to comply (can be subject to appropriate penalties if they refuse to help).

- **Voluntary helpers** a.k.a. *ad hoc* that must be asked for help, and are free to refuse (possibly except the cases when an oppnet requesting help faces a life-or-death situation; this will be elaborated upon later).

*Oppnet Virtual Machine (OVM)* is a collection of primitive functions (building blocks) whose subset can be used by any oppnet-based or oppnet-enabled application as its middleware supporting all oppnet activities, including universal platform for application-level communication and resource acquisition.

*Oppnet-based entities* (systems and devices) are entities that include OVM primitives by design, and thus are capable of oppnet activities, including communicating with other oppnet-based or so-called oppnet-enabled entities (defined next).
• *Oppnet-enabled entities* are entities that—after downloading and installing a required subset of OVM primitives in an ad hoc manner—are capable of oppnet activities, including communicating with other oppnet-based or oppnet-enabled entities.

• *Opportunistic networks* (defined by other researchers) are a proper subset of oppnets, where opportunism is limited to a small subset of oppnet capabilities, such as only opportunistic communication capabilities, or opportunistic message forwarding.

### 1.2 Background Information on Oppnets

*Oppnets* were proposed [LKBG06] as a novel paradigm for specialized ad hoc systems. The salient feature of oppnets is their expansion in an opportunistic and ad hoc manner, based on the idea of finding “helpers.” The helpers need not be predefined, and collaborate in acquisition of various resources.

#### 1.2.1 Basic Oppnet Operations

An oppnet starts as a collection of *seed* nodes, which constitute the initial pre-designed *seed oppnet* (cf. Fig.1.1a); it can be as small as a single node. A *controller* or *control center (CC)* can consist of an arbitrary subset of seed nodes. When needed, CC can decide to expand its oppnet by asking other systems and devices to assist oppnet as its *helpers*.

By inviting foreign nodes or systems to help in reaching oppnet goals, the initial seed oppnet grows into an *expanded oppnet* (cf. Fig.1.1b). An entity that accepts an invitation to assist an oppnet makes an obligation to keep on helping as long as the oppnet needs it, and becomes a *helper*. When a helper completes its tasks, it is released from its obligation by the node that invited it to join the oppnet, and is free to return to its original duties.
1.2.2 Classification of Helpers

Helpers can be classified in many dimensions. In the first dimension, they are either actual helpers or candidate helpers. The actual helpers for an oppnet are selected by it from the very broad set of candidate helpers. Candidate helpers can be wired or wireless, free-standing or embedded, transient or pervasive, etc. They possess diverse capabilities and “skills,” such as communications, computing, storing, sensing and actuating.

In the second dimension, oppnet helpers can be classified as either regular helpers or lightweight helpers (or lites). The latter are distinguished by having limited oppnet capabilities, including restricted oppnet communication capabilities (usually due to their weak native hardware/software) [LGKY10, KGLY07].

In the third dimension [LGKY10], oppnet helpers can be classified as voluntary helpers (a.k.a. volunteers) or reservist helpers (a.k.a. reservists).¹ The former are asked for help and are free to refuse (except when the inviting oppnet faces a life-or-death

¹ This is an analogy to Army, Air Force, and other military reserves.
situations), while the latter can be ordered to help and are obliged to comply (and are subject to appropriate penalties if they refuse to help)\(^2\).

Before an oppnet can grow, it finds a set of potential reservists or voluntary helpers available to it. Finding reservists requires looking them up in a reservist directory. On the other hand, finding voluntary helpers is a true discovery, involving scanning communication spectra for signals or beacons, and collecting enough information to contact their senders [LBAD14].

It is obvious that any candidate can be asked to join in any situation. It should also be obvious that any candidate can be ordered to join in life-or-death situations\(^3\).

Using reservists requires a further explanation. First of all, we envision different application-related categories of reservists, based on their declared application areas. Reservists sign up for one or more application areas (or just for individual application). For example, an “emergency reservist” is obliged to help (can be ordered to help by) any oppnet deployed for an emergency, while an “entertainment reservist” is obliged to help (can be ordered to help by) any oppnet running an entertainment application.

In addition, there is a ranking of reservist categories. The reservists from the lower-ranking categories are obliged to comply with orders from oppnets working in higher-ranking application areas (or applications). In the above example, the emergency-reservist category has a higher rank than the entertainment-reservist category. This means that an entertainment reservist must comply with orders from oppnet involved in an emergency operation. In contrast, an emergency reservist can refuse a help request coming from an oppnet serving somebody’s entertainment needs.

\(^2\) Reservists include many kinds of computing and communication systems. For instance, reservists signed up for emergency-response applications are systems serving police, firefighters, the National Guard, etc. Also, the federal/local governments can make some of their systems available upon an order from an emergency oppnet. Similarly, business and private citizens can volunteer their systems as emergency reservist. As is the case for human reservists, owners of reservists sign them up for an oppnet reserve (in a reservist directory) for some incentives, be they financial, moral, etc.

\(^3\) It is an analogy to citizens being required by law to assist with their property (e.g., vehicles) and labor in saving lives or critical resources.
Once the reservists sign up, they are “trained” for an “active duty.” This means providing the reservists with “oppnet-enabling” facilities for assisting oppnets in their discovery and for using the reservists by oppnets. For example, a standard Oppnet Virtual Machine (OVM) middleware (cf. Section 1.2.3) is installed on the reservists. The “training” makes reservists highly prepared for their oppnet duties.

By employing helpers working for free, oppnets can be extremely competitive economically in their operation. Full realization of this crucial property requires determining the most appropriate incentives for both voluntary and reservist helpers (but this issue is beyond the scope of this Dissertation).

1.2.3 Oppnet Virtual Machine (OVM)

The Oppnet Virtual Machine (OVM) is a collection of primitive functions (building blocks) that aims first of all to create a universal middleware for application-level communication within and among oppnets (this communication is then the basis for opportunistic utilization of all resources) [LKBG06, LGKY10, BMJS14]. OVM primitives can be downloaded to any system or device to make it oppnet-enabled, that is, able to communicate and share resources with other oppnets or oppnet-enabled. OVM is envisioned as a standard for assisting oppnet application development, facilitating a widespread use of oppnets.

1.2.4 Monolithic MicroOppnet v.2.2

MicroOppnet v.2.2 is a small-scale system [LiGY07, KGLY07, LGKY10] constructed as a proof of concept and a testbed for resource utilization opportunistic network. In this section we present an overview of its design and structure, and we explain its test case scenario.
1.2.4.1 Overview of MicroOppnet v.2.2

MicroOppnet v.2.2 built by Lilien et al. [LGKY10] is both a small-scale proof of concept and a testbed for resource utilization opportunistic network since it not only allows opportunistic communications but also supports sensing by opportunistically accessed sensornet nodes (a sensor network). Since sensing is the only resource utilized, MicroOppnet v.2.2 is rudimentary in its resource utilization opportunism.

![Diagram of MicroOppnet v.2.2](image)

Figure 1.2. MicroOppnet v.2.2 [LGKY10].

The seed oppnet for MicroOppnet v.2.2 (cf. Fig. 1.2) consists of Workstation A with a Bluetooth (BT) adapter and a serial port connection to Sensornet Base Station BS1, and BS1 itself. In MicroOppnet v.2.2, the sensornet consists of 10 Mica2 Motes and 6 Stargate sensornet gateways.

The seed searches for BT devices and initiates a connection with them. Alternatively, a BT-enabled device—a Victim cell phone in our example—can find the seed and initiate a connection. Once a connection has been established, the Victim cell phone can send a message to the seed, for example, the help message. This message is then forwarded via Base Station BS1, and then through the sensor network.

Some of the gateways are also connected to Mica2 Motes. Base Station BS2 at the other end of the sensornet is connected to Laptop B. Once the help message is propagated via BS2 to Laptop B, a Java TCP/IP client socket connection is initiated with Remote
Java (Database) Server. The help message and the location of a device that sent it are logged on this server.

Remote Java (Database) Server can be queried by remote users employing either traditional computing devices or Java-enabled devices. In our example, the Responder cellphone employs T-MobileTM Virtual Private Network (VPN). The seed broadcast the help message to the sensornet.

MicroOppnet v.2.2 integrates three communication technologies and frequency ranges: BT (2.4 GHz), a sensor network (916/896/433 MHz), and the wireless Internet using 802.11b/g (at 2.4 GHz, i.e., using the same frequency as BT). More information on implementation details of MicroOppnet v.2.2 is available in Ref. [Kama08]

1.2.4.2 Test Case for MicroOppnet v.2.2

MicroOppnet v.2.2 was tested using the following emergency scenario. Suppose that a fire starts in an office building and some workers are unable to evacuate. Most of them tried to use their cell phones to call for help. Many succeeded but some failed to get a connection since the cell phone infrastructure was overloaded at that time with calls being made by thousands of workers and passers-by outside of the building.

Firefighters can put a MicroOppnet to use. They deploy around the office building the MicroOppnet seed, consisting of laptops and networks connecting them. MicroOppnet v.2.2 uses BT connectivity to discover all kinds of BT-enabled helpers. An owner of any such helper can now communicate with the firefighters via the expanded MicroOppnet (consisting of the seed plus all helpers that joined it).

1.2.4.3 Limitations of MicroOppnet v.2.2

Although MicroOppnet v.2.2 served as a proof of concept to the resource utilization opportunistic networks paradigm, its development suffers from two major limitations:

1) MicroOppnet v.2.2 was developed in the monolithic fashion, without using OVM primitives.
2) Since sensing is the only resource utilized, MicroOppnet v.2.2 is rudimentary in its resource utilization opportunism.

In the following subsections, we discuss benefits of using a modular (non-monolithic) OVM-based oppnet middleware, thus improving upon MicroOppnet v.2.2.

1.3 Motivation for Oppnet Virtual Machine

With the wide spread of pervasive devices and ubiquitous computing comes the need to break the software and hardware gap to exploit the wealth of resources available in heterogeneous devices that are within each other’s reach.

We envision a universal middleware for oppnet that is capable of breaking the software and hardware barriers by integrating heterogeneous oppnet-enabled systems and devices—serving diverse applications. Such devices can reply to each other’s “cries for help,” and then actually help each other by sharing their hardware and software resources (incl. communication resources). To this end, we need to develop a standard application-level middleware for supporting implementation of oppnets and oppnet-enabled systems and devices that will provide a convenient basis for these diverse applications.

Previous work on oppnets does not provide an implementation of the OVM in a non-monolithic fashion; moreover, the proof-of-concept system (MicroOppnet v2.2) is very limited in its resource utilization opportunism. Many of the available systems and technologies (as shown in Chapter 2—Literature Review) provide powerful capabilities for application-level communication; however, we are not aware of any single platform or technology that incorporates all the functionalities proposed by OVM. Some systems require a mediation of a directory, of a control center,\(^4\) or of a trusted third party to arrange any interaction among devices. Many platforms need to be tweaked before use to adapt them for new applications, as they are either geared towards a particular application

\(^4\) CC is a single point of network control. It could be as small as a single software application, or it could include one or more workstations with various monitoring tools for oppnet control.
or support only a particular communication protocol (e.g., just Wi-Fi, or just a manufacturer-specific protocol). In contrast to other solutions, our OVM-based oppnet middleware employs the “helper paradigm” which means the ability of heterogeneous systems or devices to interact with each other in an ad hoc manner, discover other systems or devices, admit them into the oppnet network, listen to cries for help from other systems or devices, and help each other by sharing their resources and capabilities without the mediation of a directory, a control center or a trusted third party. We aim to create a standard middleware which facilitates interoperability of oppnet-enabled systems or devices regardless of their make, and through any communication medium. Unlike the current systems or devices, our oppnets should be able to easily extend their capabilities to different applications.

We need to design and develop the oppnet middleware in a non-monolithic fashion, and structure it into convenient modules called OVM primitives; each primitive is an atomic element [BhLi81] of the oppnet middleware that implements a well-defined activity of an oppnet device.

The optimal set of OVM primitives is implemented for four separate node categories to avoid the situation where a node is forced to play a role of a node from another category. This separation ensures security and saves resources. We want to find the optimal set of OVM primitives. In other words, we want to divide the oppnet middleware into the best minimal set of non-overlapping or minimally-overlapping modules.

If a system or a device is not oppnet-enabled, we want an easy way of making it oppnet-enabled. Downloading and installation of needed (a subset of all) OVM primitives into the system or device is all what is needed for the system or the devices to become oppnet-enabled.

Different systems or devices will need and will pick different subsets of the OVM primitives. For example, lightweight or “thin” devices need a small subset of the OVM primitives, while full-strength or “fat” systems or devices need a large subset (even the full set) of the OVM primitives. Thus modularity of OVM (its separation into OVM primitives) provides customization of the oppnet middleware to the systems or devices using it.
Our goal is assuring that any two oppnet systems or devices can easily communicate with each other and help each other. We contrast truly oppnet-based systems or devices—which include OVM primitives by design, with systems or devices that become oppnet-enabled in an ad hoc fashion—by downloading and installing needed OVM primitives. We will investigate in the future if we should require that not only the former but even the latter must be able to easily communicate with and help each other. If we decide that they should, then—for example—an ad hoc oppnet-enabled printer will be able to help another ad-hoc oppnet-enabled device; also, the same printer will be able to help an oppnet-enabled device and vice versa.

1.4 Problem Statement and Research Hypothesis

With the high demand for pervasive computing comes the need to break the software and hardware barriers to exploit the wealth of resources available in heterogeneous devices that are within each other’s reach. Over the years, technologies enabling pervasive computing have been researched and experimented with to meet demands for ubiquitous communication, computation, and data processing. However, there is not a single platform that encompasses all the required features to unleash the wealth of software and hardware resources.

The research hypothesis for this dissertation is:

OVM-based oppnet middleware can provide efficient ad hoc application-level resource sharing among diverse systems and devices.
1.5 **Scope and Limitations of the Proposed Research**

We design and develop the optimal set of OVM primitives so that the primitives can act as building blocks to construct oppnet-based and oppnet-enabled systems and devices that serve diverse applications and yet—for efficiency and effectiveness—are specialized to serve four different node categories.

The oppnet middleware satisfies the following requirements:

1) The oppnet middleware is implemented in a modular (non-monolithic) fashion using the improved OVM primitives.

2) The oppnet middleware is divided into a minimal set of OVM primitives in such a way that the primitives are either non-overlapping or minimally-overlapping.

3) The oppnet middleware is easily customizable. Different systems or devices can use different subsets of the OVM primitives depending on their capacity and need.

4) Object-oriented principles and constructs—such as modularity, inheritance and polymorphism—were applied during the design of the OVM primitives, including OVM module design and OVM pseudocode development. This enhances efficiency, reliability and scalability of the OVM primitive implementation.

5) The oppnet primitives can be downloaded to and installed on any system or device in an ad hoc way to make it oppnet-enabled.

6) Any two oppnet-based or oppnet-enabled entities (systems or devices) can interact with each other.

This OVM-based interaction between entities relies on the following salient oppnet features:

a) Support for the helper paradigm as the basis for resource acquisition.

b) Universality of communication and resource acquisition—regardless of the system or device make or function, through any communication medium, protocol, etc., and for any resource.
c) Lack of third-party mediators—since interactions among oppnet-based or oppnet-enabled entities take place without third parties (either untrusted or trusted).

d) Ad hoc operation—except predesigned formation of the oppnet seed.

There are a few limitations in our development of the oppnet middleware that are beyond the scope of this Dissertation, and have to be left for future investigation. The major limitations are as follows:

1) Our implementation of the OVM primitives does not consider security of oppnet nor the privacy of data exchanged among the oppnet-based or oppnet-enabled devices.

2) Our implementation of the OVM primitives is limited by using only some of the possible types of entities (systems and devices) in our proof of feasibility (cf. Chapter 4).

3) Our implementation of the OVM primitives is limited by using only some of the possible communication technologies in our proof of feasibility.

1.6 Dissertation Organization

The rest of the Dissertation is organized as the following. Chapter 2 presents the literature review which includes the technologies related to the oppnet middleware and the differences between such technologies and the oppnet middleware. Chapter 3 introduces Oppnet Virtual Machine (OVM) and its primitives; we will also present a set of proposed OVM primitives and compare that set with the original OVM primitives. Chapter 4 explains the design and implementation of the proof of feasibility of the OVM-based oppnet middleware. Chapter 5 compares performance of the monolithic oppnet middleware with the modular (non-monolithic) OVM-based oppnet middleware by simulating both, using a healthcare scenario. Chapter 6 presents our conclusions, indicates the significance and impact of this work, and outlines future work.
2. LITERATURE REVIEW

This chapter reviews the technologies most relevant to the OVM-based oppnet middleware and explains the similarities and distinctions between these technologies and the oppnet middleware. We then present previous work on the oppnet middleware, and contrast oppnet with the other reviewed solutions.

2.1 A Review of Technologies Related to Oppnet

In this section we categorize the technologies related to the OVM-based oppnet middleware into: resource-sharing technologies, connectivity-based technologies, and specialized and other networks. For each category we review the most important systems that are related to the oppnet middleware.

2.1.1 Resource-Sharing Technologies

This subsection discusses resource-sharing technologies that are most relevant to the OVM-based oppnet middleware. We then point out the differences between these technologies and the oppnet middleware in a way that indicate the novelty of the OVM-based oppnet middleware.

2.1.1.1 Peer-to-Peer (P2P) Systems

Peer-to-Peer (P2P) systems allow sharing content—such as files—between devices, or networks that interact via an “appropriate communication and information channel” [SuGo05]. Such resource-sharing and application-level communication is achieved by direct sharing between “peers” without depending on centralized servers [Kama08].
Developing platforms and systems that apply the P2P technology is an active research area. The following paragraphs describe some of the recent work on P2P networks.

*FlashLinq* [Flas15] is a P2P platform developed to complement traditional cellular-based services. The technology advances a concept known as *proximal communication* that enables devices to discover each other as well as communicate at a high speed without intermediary infrastructure. FlashLinq can create a “neighborhood-area network,” where fixed and mobile peer applications can interact directly.

Hunag *et al.* [HuYT12] propose a distributed telematics P2P networking system. The system provides service discovery for mobile users on the road. Mobile users in vehicles are assumed to go through many regions, with a region server associated with each region. Information used by all service providers in a region is stored in the region server. The proposed system is supposed to provide service discovery over the vehicular network for mobile users.

Castro *et al.* propose *Scribe* [CDKR02] a large-scale and decentralized application-level multicast infrastructure built upon *Pastry* (a scalable, self-organizing P2P location and routing substrate). Scribe provides efficient application-level multicasts and is capable of scaling to a large number of groups and group members as well as multicast sources. Scribe and Pastry provide a decentralized P2P model where all nodes have equal responsibilities.

Although the OVM-based oppnet middleware shares many characteristics with P2P systems, such as the decentralized sharing of resources, our approach is distinguished by the following key features:

1) It allows collaboration of devices to achieve a common goal rather than meeting individual node requirements.

2) It is designed to allow acquiring all kinds of resources—such as communication, sensing, storage, etc.—rather than just sharing computing cycles and information.

3) It distinguishes four types of nodes (i.e., CC, seed, regular helpers, and lites) in which the nodes have different capabilities and duties.
2.1.1.2 Grid Computing

*Grid Computing* refers to the hardware and software infrastructure that enables the aggregation of distributed computing resources to achieve a common goal [Abba04]. It can use a set of open standards and protocols to gain access to applications and data through application-level communication while sharing processing power, storage capacity and a vast array of other computing resources over the Internet [IBMS02]. The following paragraphs review some of the grid computing systems.

*Globus Toolkit (GT)* [Fost05] is a service-oriented distributed computing encompassing applications and infrastructures. The main components addressed by GT are: security, resource access, resource management, data movement, and resource discovery. These components enable a broader “Globus ecosystem” of tools and components that build on or interoperate with core GT functionality to provide a wide range of useful application-level functions. These tools can be used to develop a wide range of grid infrastructures supporting distributed applications.

Mo et al. [MHWW10] uses *GridGain* which is an open source middleware solution that enables big data analysis on a distributed computing architecture [Techo15] via the core middleware layer. It facilitates building a grid computing platform for sharing computing resources for, e.g., *Back Propagation Neural Networks (BPNs)*. The authors claim that the proposed platform can provide solutions also for large-scale high-performance applications other than BPN.

*Nimbits* [Nimb14] is a *Platform as a Service (PaaS)* that can be used to develop software and hardware solutions for connecting seamlessly to clouds and to other Nimbits servers. Nimbits provides *Representational State Transfer (REST)* Web services for logging and retrieving time and geographically-stamped data (such as readings from a temperature sensor). A Nimbits server records incoming data, and each new value triggers an event such as an alert, a calculation, or a message. Newly calculated values can be automatically propagated to another channel, with more and more cascading triggers and computations.
Although the OVM-based oppnet middleware shares many characteristics with some of grid computing systems (such as the ability to grow by joining, and the use of “foreign” resources), our approach is distinguished from grid computing systems by the following key features:

1) It is designed to deal with a wide range of application areas while grid computing systems are concentrated on computationally-intensive operations.

2) It relies on helpers joining a system and remaining a part of it until released while grids allow sites to join and leave as they choose [Kama08].

3) Its entities collaborate in highly heterogeneous environments while grid entities work in a more homogenous environment.

2.1.1.3 Spontaneous Networks

*Spontaneous Networks* are a subset of ad hoc networks created when a group of people meet for a collaborative activity using heterogeneous wireless mobile computing devices [FeAW01]. Spontaneous networks are limited in space and time (the space and time of the meeting). They are comprised of heterogeneous devices connected using a variety of wireless technologies, such as Wi-Fi, infrared, or Bluetooth.

Rewadkar and Karve [ReKa14] propose a spontaneous network based on a wireless ad-hoc network that enables devices to communicate with each other without the availability of a central server (as in a client-server communication) and without a prior configuration (as in P2P networks). In this spontaneous network, laptops are placed close to each other at a particular place for a short period of time to access available services and resources within the network without Internet connection. *Energy Efficient Self Configure Secure Protocol (EESCSIP)* is proposed to create and manage the spontaneous network, which is self-configured to integrate services and resources into the network.

*Tree-based Overlay over Multiple and Heterogeneous (TOMH)* [QIGB13] is a framework for building spontaneous networks that supports smart network management features on top of heterogeneous multi-network environments. TOMH creates and maintains a dynamic and mobility-aware tree-based overlay network to integrate different
communication technologies while maintaining the reliability of the connection and supporting quality-sensitive applications.

Aloi et al. [AFLP14] introduce *Spontaneous Smartphone-based Networks (SSNs)*, where smartphones are responsible for supporting communication services. SSNs capture the key features of traditional spontaneous networks (like spontaneity in creation of the network, and defining the role of the devices in a way that allows them to adapt to the network and user requirements). The authors identify the challenges to adopting their paradigm.

Many features of spontaneous networks are shared by the OVM-based oppnet middleware, for example, heterogeneity of devices, collaboration among devices without a central server, and the use of diverse communication technologies. However, our approach has three key features that are not found in spontaneous networks: (i) categorization of nodes into four different node categories; (ii) the administrative capabilities and role of the seed/CC nodes; and (iii) opportunistic growth of oppnets not limited by space or time as severely as in spontaneous networks.

### 2.1.1.4 Internet of Things (IoT)

*Internet of Things (IoT)* is now a widely known concept that aims at sharing resources via heterogeneous devices by minimizing the communication gap between heterogeneous devices [HeBE12, NiLY13, UBJB12]. To reach this goal, some of the proposed solutions require that the *Original Equipment Manufacturers (OEMs)* agree to produce new products capable of interacting with devices of other OEMs, regardless of their make or model. The idea is that collectively OEMs can agree on an open, universal development framework/standard supported by a thriving technical community and ecosystem.

Other IoT initiatives focus their efforts on developing IoT applications that allow devices to interact, discover, sense and actuate with minimal human interaction. These applications communicate through different technologies such as Wi-Fi, Ethernet, or IEEE 802.15.4 radio; they can form a mesh or utilize the Internet to scale up. The following paragraphs introduce some of the available IoT systems.
AllJoyn [Allj14] is an open-source project (initiated by Qualcomm) which aims to create a platform and services that enable products and applications to inter-operate regardless of their make or operating system. AllJoyn aims to connect devices like televisions, home appliances, home entertainment devices, and automobiles through Wi-Fi to create smart homes or vehicles. Qualcomm along with the AllSeen Alliance are promoting this project by partnering with a consortium of OEMs and Original Design Manufacturers (ODM) like Foxconn, LG, and Xiaomi.

IoT Toolkit [IoTT14] is another open-source project; it aims to create a platform facilitating interoperability between various protocols and cloud services. It provides consistent abstractions for different Machine-to-Machine (M2M) communication platforms. It also provides interoperability between different types of “things” (devices) by developing an API based on resource models that use common concepts across different “things.” The goal for IoT Toolkit developers is to construct the Web standards for IoT in order to broaden its use.

The Thing System [ThTS14] aims to connect smart “things” within a home in order for these “things” to interact autonomously. At the center of this system is the “steward,” which is a software built as server-side Java scripts. The steward runs on different systems ranging from a laptop to single-board computers. The Thing System has an adaptive behavior, i.e., it learns that the home owner likes her coffee at 6 a.m. and she likes to watch the local news channel with the living room blinds open. The Thing System uses three ways to communicate with the devices:

- Built-in support for manufacture-specific protocols to access, e.g., Philips Hue light bulb, the Apple TV, and devices using ZigBee, Z-Wave, INSTEON, or DASH7. In addition, the user can write a “device driver” for her specific communication protocol for the steward to support it.

- It uses Thing Sensor Reporting Protocol to report sensor readings.

- If the device is able not just to sense but also to actuate, the Simple Thing Protocol is used.
In spite of all the current IoT research efforts, there is no single IoT platform that incorporates all functionality proposed by the OVM-based oppnet middleware. The major shortcomings of the current IoT solutions (well visible in the oppnet context) include the following ones:

1) Most if not all IoT systems rely on devices that are pre-configured in a deterministic manner.

2) Some IoT platforms do not fully utilize the ability of some devices to discover other devices, are unable to admit them into their network, are not capable of interacting with them in an ad hoc manner, and cannot help each other without the mediation of a directory, a control center, or another third party.

3) Most of the IoT solutions are designed for home automation (however, with some tweaking many can lend their capabilities to other applications).

4) Some IoT platforms are limited to using only a certain communication protocol, such as Wi-Fi.

5) Some IoT solutions have a built-in support only for manufacturer-specific protocols. In order to extend their functionality to devices from other manufacturers, users have to write new device drivers.

2.1.2 Connectivity-Based Technologies

This subsection reviews some of the connectivity-based technologies related to the OVM-based oppnet middleware and presents some research results for them. The key features distinguishing our oppnet approach from each of the connectivity-based technologies are identified.

2.1.2.1 Opportunistic Networks

*Opportunistic Networks* (e.g., [PePC06, WJMF05]) a proper subset of our opportunistic resource utilization networks (oppnets). They can be viewed as a
generalization of the mobile ad hoc networking (MANET) paradigm, in which the assumption of having complete paths between data senders and receivers is relaxed. This enables stations to communicate in disconnected environments, in which islands of connectivity appear, disappear, and reconfigure dynamically [KGLY07]. The following paragraphs review some of the current systems developed on opportunistic networks.

Seto et al. [Seto10] present a mobile platform for body sensor networking. The platform allows for local processing of data, and uses opportunistic sensing strategies, in which the capabilities of onboard sensors and smartphones may be collected and fused with body sensor data.

Lu et al. [LuLS13] propose a privacy-preserving opportunistic computing framework named SPOC. During a mobile healthcare emergency, SPOC processes computing-intensive personal healthcare information with minimal privacy disclosures—by opportunistically gathering computing and energy resources from smartphones.

Bleda et al. [BMJS14] identify the applications of opportunistic wireless access networks in Assisted Ambient Living (AAL) environments (assisting the disabled users), and discuss how such networks can be used to improve ambient intelligent systems.

The OVM-based oppnet middleware (as defined in Refs. [LKBG06, LGKY10]) is significantly different from systems developed on opportunistic networks (as defined by others, e.g., in Refs. [PePC06, WJMF05]). In the latter, opportunism is limited to communications when devices are within each other’s range. In the “other” opportunistic network systems (e.g., [PePC06]) there is no notion of utilizing resources of “foreign” nodes in a network to perform a task of the network. In contrast, oppnets enable opportunistic use of all kinds of resources, services, or capabilities (incl. hardware, software, human skills, etc.) that happen to be within the oppnet’s reach [LKBG06] through any communication technologies (regardless of the device make or function, cf. Chapter 3). Another distinguishing feature is that an oppnet starts as a relatively small network, known as the seed network (a.k.a. the seed), which can grow into an even large expanded oppnet [LGKY10].

There have been some attempts to incorporate more resources—other than communication capabilities—in opportunistic networks (e.g., [Seto10, LuLS13,
However, the utilized resources are limited to a few kinds of resources (such as sensing or computation), and through a limited set of types of communication technologies.

### 2.1.2.2 Delay-Tolerant Networks

*Delay-Tolerant Networks (DTNs)* can be viewed as the superclass of wireless network systems. The main goal of DTNs is to provide connectivity between regional networks when communication is prone to discontinuation and interruptions. By adapting store-and-forward message switching, DTN can overcome problems causing large delays [SeFM07]. The following are some DTN-based systems.

Husni and Wibow [HuWi12] build a DTN-based email server capable of sending and receiving emails even if the server is not continuously connected to a network. The proposed email server exploits DTN’s ability to overcome problems associated with extreme environments, intermittent connectivity, large or variable delays, and high bit error rates (which are common characteristics for communications with remote areas). The authors propose using facilities and infrastructures such as the *Public Transportation System (PTS)* for building a DTN-based network. A train system was chosen to be the infrastructure for DTN routers using Wi-Fi. A train passing through a village (even not stopping there) can have a mobile Wi-Fi station, which brings to the village new email from the outside world as well as collect local email awaiting for sending to the outside world.

Homing-Pigeon-based Messaging (HoPM) [GLHQ13] is a DTN system that uses a set of messengers (named *pigeons*) that move around the network to deliver messages among multiple *anchor nodes*. Each *source node* (an anchor node or an Internet access point) owns multiple dedicated pigeons. Each pigeon takes a trip disseminating the messages on its way; it starts from its home (i.e., its source node), visits all destination anchor nodes, and finally returns home.

Ribeiro *et al.* [RiPC14] propose the use of DTN to build an underwater monitoring system with sensors distributed over a subsea infrastructure, which is responsible for
operation and transportation of oil products. Underwater acoustic modems are installed on the sensors to transmit data. Platforms and vessels used for logistic support of oil exploration collect data and provide references for positioning the sensors. However, since the vessels may not be within the sensor range at all times, DTN must be used for supporting communication.

The main goal for the DTN-based systems is providing connectivity in the most challenging networking circumstances. On the other hand, the OVM-based oppnet middleware can include not only DTN capabilities, but also all their own unique capabilities, starting with gaining resources via helpers.

### 2.1.2.3 Mesh Networks

*Mesh Networks* [Mesh15, SONW07] have a decentralized topology where each node is connected to at least two other nodes. Such infrastructure eliminates failure in the network due to *Single Point of Failure (SPoF)*, since if a node fails other nodes can still communicate with each other. Mesh network systems can be wired or wireless, a full mesh or a partial mesh where nodes are connected to the nodes they exchange data with on a frequent basis.

*Vision Mesh* [ZhCa10] is a video sensor mesh network platform used for water conservancy engineering. Vision Mesh is composed of a massive number of image or video sensor nodes with which multi-view image or video is simply acquired. OpenCV machine vision lib is integrated into Vision Mesh to enable video and image processing.

*Thingsquare* [Thnq14] is a development system that enables connecting home devices through the home Wi-Fi router. Thingsquare is structured into four major components:

1) *Browser frontend*: allows interacting from a smartphone or the browser frontend with wireless devices through the Thingsquare API.

2) *Thingsquare backend*: connects the devices with the API. The backend can be run in the cloud or behind the users firewall.
3) **Open-source firmware**: contains a single wireless system-on-a-chip that enables connecting the devices to the Thingsquare system.

4) **Wireless mesh network**: allows the devices to automatically discover each other and connect to the Internet via Ethernet or Wi-Fi. Thingsquare network can reach the Internet, but the Internet cannot reach the Thingsquare network. Thingsquare devices can communicate directly with Internet services, via the Thingsquare router. The router translates the low-power IPv6 mesh traffic from within the Thingsquare network to IPv4 that can be routed onto the Internet.

   *Sapphire* [Sapp14] is a new wireless development platform with its own hardware, a lightweight but powerful embedded operating system, and developed network connectivity tools to help users connect “anything to everything.” The Sapphire operating system can create a mesh network, connect with Python, REST, or with any compatible API, provide automatic Sapphire device discovery, and have time synchronization and *Advanced Encryption Standard (AES)* security built-in. The network connectivity tools are Python-based; they connect the system to an extensible central message bus.

Although the topology of mesh network systems provides reliability in communication between nodes, the expenses of constructing such a topology can be too high. Moreover, the interactions among devices in mesh network systems are preconfigured in a deterministic manner; on the other hand, nodes invited by the oppnet middleware are not configured a priori, and there is no determinism in the manner that the joining helper nodes are called to service [KGLY07].

### 2.1.2.4 Self-Organizing Networks

*Self-Organizing Networks (SONs)* are automation technology systems designed to provide cost-efficient deployment, operation and maintenance of mobile network [JPGC14]. SON was among the requirements within the *3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE)* standardization [UTRA15]. SON functions can be categorized into three major sub functions: (i) self-configuration; (ii) self-
optimization; and (iii) self-healing. The following paragraphs present some of the current work on SON.

Tonguz and Viriyasitavat [ToVi13] propose a biologically inspired self-organizing network system whereby certain vehicles serve as Road Side Units (RSUs). Vehicles that act as temporary RSUs can act as a communication bridge for other networks by making occasional brief stops. Authors believe that using vehicles as RSUs could improve message reachability and network connectivity between vehicles while avoiding the cost of deploying RSUs.

Trang [Tran14] proposes Distributed Self-Organizing Network (DISON) which uses a multilevel management schema to provide scalability for large sensor networks. The platform help sensor nodes self adapt to the changes in network resources and application requirements. The author shows how the platform is used to coordinate network resources among groups of adjacent sensor nodes.

Placzek [Plac14] introduces a self-organizing traffic signal system for an urban road network. The main components of this system are agents that control traffic signals at intersections. Agents detect incoming traffic, make decisions, and execute commands to control traffic at the intersection. Such commands will determine autonomously which traffic streams will get green lights. An interval microscopic traffic model is used by the agents to predict and suspend commands whose effects are uncertain.

Although SON systems share with the oppnet middleware the ability of forming, organization, and management of nodes in the absence of a core center [PrBe05], SON systems are not focused on the concept of acquiring resources via helpers, which is the core feature of oppnets developed by the OVM-based oppnet middleware.

2.1.2.5 Mobile Ad Hoc Networks (MANETs)

Mobile Ad Hoc Networks (MANETs) are decentralized network systems comprised of a collection of autonomous mobile nodes communicating over wireless communication channels. A MANET topology can change rapidly over time as nodes can join and leave the network at anytime [SiKV12]. Since the network is decentralized, MANET’s nodes
are responsible for the network’s activities such as discovering the network’s topology, self configuration, and routing. The following paragraphs review some of the current work on MANETs.

**Beddernet** [GoGG11] is a platform-agnostic MANET framework. The Beddernet architecture is designed to work with different networking protocols. In their work, the authors tested Beddernet to work with Bluetooth ad-hoc networks or scatternets. Beddernet middleware has been tested on Java and Android devices.

*Mobility-aware Routing and Interference-aware Topology control (MRIT)* [KhLJ12] is a cross-layer protocol that jointly considers routing and topology control taking mobility and interference into account for MANETs. The main goal of the proposed protocol is to increase the network lifetime, preserve energy, and find stable end-to-end routes for MANETs. The authors evaluate the performance of the MRIT by simulating a set of random MANET environments.

As an attempt to enhance the *Quality of Service (QoS)* communications over MANETs, Obaidat *et al.* propose *QoS-Aware Multipath Routing Protocol (QMRP)* [OASO13]. The proposed protocol considers projected load into route quality computation to provide more accurate measure of the realistic delay as well as maintaining loop freedom of multiple node disjoint paths using neighbor hop list.

A MANET system shares the self-configuration feature with the oppnet middleware as both are decentralized. However, the nodes of the oppnet middleware are capable of realizing more challenging tasks than self-configuration and routing, since helper nodes are responsible for sharing their resources to assist oppnets in achieving their goal. Another distinguishing factor is the commitment of oppnet middleware nodes to keep assisting till the oppnet mission is accomplished, while in MANETs nodes can join and leave a network without any restrictions.

### 2.1.3 Specialized and Other Networks

In this section we discuss *Ambient Networks (ANs)*—a 3G technology, and *Wireless Sensor Networks (WSNs)*—a monitoring and control technology.
2.1.3.1 Ambient Networks (ANs)

*Ambient Networks (ANs)* is a networking paradigm for use beyond 3G systems. It was developed as a part of the European Union sixth framework project [Euro02]. It aims at providing existing and new services over any access technology and any type of network. To attain such services, ANs enable on-demand and transparent cooperation between heterogeneous networks, with little or no pre-configuration or off-line agreement [BeGD08]. The following are some systems that are developed based on ANs.

Vodel *et al.* [VoCH10] present a communication concept that aims at combining the advantages of different communication technologies (such as cognitive radios and ANs) by integrating them into an embedded communication platform. The idea behind the concept is to create a lightweight radio standard that combines the advantages of different communication technologies while overcoming the disadvantages of a single radio standard with its design limitations.

*Context-Specific Overlay Networks Dissemination (CSON-D)* [BaND08] is a context dissemination protocol based on a personalized and customized overlay environment towards ubiquitous networking. The protocol introduces dissemination mechanisms for context-aware services in ANs.

ANs resemble the OVM-based oppnet middleware in its ability to integrate heterogeneous devices to provide various services. However, there are two major differences that distinguish our oppnet middleware from AN systems: (i) AN systems are completely predesigned and all its facilities are built-in, whereas the operations of the oppnet middleware are mostly ad hoc without prior configuration, and (ii) AN systems are global networks intended to replace the Internet, while the OVM-based oppnet middleware is intended to build local/wide area networks to serve a wide range of applications and provide help in real life situations [KGLY07].
2.1.3.2 Wireless Sensor Networks (WSNs)

*Wireless Sensor Networks (WSNs)* are specialized systems that consist of autonomous, resource-constrained devices embedding sensors, processors and transceivers to monitor and control a physical environment [AkVu10]. WSN nodes communicate over wireless communication technologies, usually 2.4 GHz radios based on IEEE 802.15.4 standards.

Capella *et al.* [CBOP14] develop and deploy a WSN for continuous in-line monitoring of nitrates concentration in a river in Eastern Spain. The authors also implement policies to improve the features of the whole system. The improvements include optimizing the times at which measurements are to be carried out, and sampling frequency being altered according to the system evolution, the user preferences and the application features.

As an attempt to increase WSNs lifetime, Cerulli *et al.* [CeDR12] consider that each sensor can be activated in a certain number of alternative power levels, which determine different sensing ranges and power consumptions. The authors present some heuristic approaches and an exact approach based on the column generation technique.

Somov *et al.* [SBSS11] describe the development of a *Wireless Gas Sensor Network (WGSN)* for the detection of combustible or explosive gases. The network consists of a sensor node that attains early gas detection, a relay node, a network coordinator, and a wireless actuator. The authors consider energy saving by employing a pulse heating profile. When an emergency is detected, the network coordinator alarms an operator through the GSM/GPRS or Ethernet network, and may autonomously control the source of gas emission through the wireless actuator.

The major differences between the OVM-based oppnet middleware and WSNs are: (i) the oppnet middleware allows integration of heterogeneous devices that can have powerful computing capabilities (are not just resource-constrained nodes); and (ii) the oppnet middleware allows nodes to communicate over diverse communication channels (not just over a single frequency channel, as is typical for WSNs [KGLY07]).
We must note that a WSN or any subset of WSN nodes can be admitted into an oppnet as helpers—as is done in MicroOppnet (cf. Chapter 4).

2.2 Previous Oppnet Work

Lilien et al. [LKBG06, LGKY10, LiKG06] propose the Oppnet Virtual Machine (OVM) for oppnets and define its basic primitives. After categorizing oppnet nodes into four categories—Control Center (CC) nodes, seed nodes, helpers and lites—separate OVM primitive sets are defined for each node category (for better security and resource utilization).

These original OVM primitives were evaluated for completeness and consistency only conceptually without any actual simulation/emulation or implementation [LGKY10].

In our work reported here, we critically evaluated the original OVM primitives, and decided that the following modifications are needed:

1) Some of the original primitives were eliminated or modified to avoid redundancies.
2) A number of new primitives are added to provide more modular and non-overlapping OVM functionality.
3) The new set of primitives was redesigned using object-oriented principles such as modularity, polymorphism, and inheritance.

Kamal et al. [KGLY07, KLGY08] present OVM as the API framework for oppnets, describing design and implementation of the oppnet testbed MicroOppnet v.2.2. MicroOppnet v.2.2 was developed as a proof of concept for oppnets, not as an OVM validation platform. Hence, despite using the OVM concept, it has a monolithic structure, with code not using OVMs (the code instead of being divided into separate OVM primitives is written without such separation). Moreover, MicroOppnet v.2.2 is rudimentary in its resource utilization opportunism as sensing is the only resource utilized.
In our work we simulate, emulate and implement different components of the OVM-based oppnet middleware. But this time, our code is not monolithic; instead it uses the new set of OVM primitives.
3. OPPNET VIRTUAL MACHINE (OVM)

This chapter starts by defining OVM and giving an overview of the OVM primitives. Then it describes the design of the first-generation OVM, which was proposed earlier [LiKG06, LKBG06, LGKY10]. Next, it presents an overview of the second-generation OVM proposed by us, and lists its improved OVM primitives. The chapter finally shows the motivation behind developing the third-generation OVM, discusses the requirements for the third-generation OVM, describes the object-oriented design principles applied in the design process, and presents the third-generation list of the OVM primitives.

3.1 Definition of OVM

Oppnet Virtual Machine (OVM) is the oppnet middleware that consists of a collection of primitive functions that together should provide a universal middleware for application-level resource sharing (resources include communication which is the basis for opportunistic utilization of all other resources).

OVM primitives can be designed into any system from the beginning, making it an oppnet-based system. Alternatively, in an entirely ad hoc manner, OVM primitives can be downloaded and installed on any device or in any system at any moment to make it oppnet-enabled. An oppnet-based or an oppnet-enabled system is able to share its resources with oppnets.

OVM is envisioned as an open-source middleware and a standard that will assist in development of oppnet-based applications, as well as in contributing to the pervasive computing paradigm by facilitating massive ad hoc resource sharing. We believe that benefits of the oppnet paradigm will result in its widespread use.
3.2 Overview of OVM Primitives

OVM primitives can be viewed as a set of atomic (indivisible) building blocks from which oppnet-enabled communication capabilities can be constructed for any (sufficiently capable) system or device (cf. [BhLi81]). Any system or device can download any subset of OVM primitives that is required to make the system or the device oppnet-enabled.

The set of OVM primitives is developed for four separate node categories to avoid the situation where a node is forced to play a role of a node from another category. This separation will ensure security and save resources.

Freely available OVM primitives will encourage the development and promotion of standard library routines and APIs for a wide spectrum of applications designed as oppnet-based or oppnet-enabled systems.

3.3 First-Generation OVM

In this subsection we discuss the motivation behind designing the first-generation OVM, and present an overview of its functionality.

3.3.1 Motivation and Overview

As we discussed earlier, Lilien et al. [LiKG06, LKBG06, LGKY10] proposed oppnets as a novel paradigm that opportunistically uses all kinds of resources (not only communication) to accomplish oppnets goals. The authors also propose the first-generation OVM as a standard application framework for oppnet applications by different software vendors. The OVM standard is intended to assure interoperability among these different oppnet implementations and third-party oppnet products. OVM is proposed to assist in developing and marketing standard library routines and APIs for all kinds of
applications. The authors also implement the monolithic *MicroOppnet v.2.2* as a testbed for the first-generation OVM. The next chapter explains the implementation details of MicroOppnet v.2.2.

### 3.3.2 OVM Primitives

The OVM primitives were intended for use by all those who want to write programs in C/C++/C# or Java for oppnets and any oppnet-enabled devices. This includes individual application programmers, manufactures of hardware devices, and creators of environments and tools. In order to be attractive to this wide audience, the standard was intended to provide a simple, easy-to-use interface. The standard did not specify program construction tools, debugging facilities, support for task management, or underlying mechanisms for communications. However, oppnet features that were not included in the OVM standard can always be offered as extensions by specific implementations.

The first-generation OVM standardization follows the models of *Message Passing Interface (MPI)* [GrLS94] and *Parallel Virtual Machine (PVM)* [SGDM94] used in grid computing. At any moment a node belongs to only one of the four categories: CC nodes, seed nodes, helpers, and lites.

Lilien *et al.*, provided separate primitives for the four node classes to help preventing situations when a node attempts to play a role of a node from another node class. The two main advantages of having distinct primitive classes are better security and resource savings [LGKY10]

A partial list of the original (first-generation) OVM primitives is included in Appendix A.
3.4 Second-Generation OVM

This subsection presents the motivation behind the second-generation OVM designed by us, and lists these OVM primitives.

3.4.1 Motivation and Overview

The motivation behind the second-generation OVM was to evaluate the original first-generation OVM and its primitives for completeness and consistency. We critically evaluated the first-generation OVM and decided the following modifications are needed:

1) Some of the original primitives are eliminated or modified to avoid redundancies.

2) A number of new primitives are added to provide more modular and non-overlapping OVM functionality.

As part of the second-generation OVM we develop the monolithic MicroOppnet v.4, which is a recreation of the monolithic MicroOppnet v.2, to help us understand the functionality of the oppnet operations. The next chapter discusses MicroOppnet v.4 in detail.

3.4.2 OVM Primitives

Tables 3.1 through 3.3 include lists of the second-generation OVM primitives for CC, seed, and regular helper respectively. The primitives for lightweight nodes (lites) are very similar to the primitives for helpers, and are therefore omitted.
Table 3.1. List of second-generation CC primitives.

<table>
<thead>
<tr>
<th>OVM Primitive Name</th>
<th>OVM Primitive Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC_addNode</td>
<td>Add a node to list of children.</td>
<td>A</td>
</tr>
<tr>
<td>CC_initiate</td>
<td>Initiate oppnet</td>
<td>U</td>
</tr>
<tr>
<td>CC_receiveReport</td>
<td>Receive a report from child.</td>
<td>A</td>
</tr>
<tr>
<td>CC_remNode</td>
<td>Remove node from list of children.</td>
<td>A</td>
</tr>
<tr>
<td>CC_sendOppnetTasks</td>
<td>Send the oppnet tasks to children.</td>
<td>M</td>
</tr>
<tr>
<td>CC_terminate</td>
<td>Terminate oppnet</td>
<td>U</td>
</tr>
</tbody>
</table>

Table 3.2. List of second-generation seed primitives.

<table>
<thead>
<tr>
<th>OVM Primitive Name</th>
<th>OVM Primitive Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEED_addNode</td>
<td>Add a node to list of children</td>
<td>A</td>
</tr>
<tr>
<td>SEED_discover</td>
<td>Discover services of a certain device</td>
<td>M</td>
</tr>
<tr>
<td>SEED_evalAdmit</td>
<td>Evaluate a device and admit it into oppnet if the device meets criteria for admittance</td>
<td>M</td>
</tr>
<tr>
<td>SEED_isMember</td>
<td>Check if a device is already an oppnet node (oppnet member)</td>
<td>U</td>
</tr>
<tr>
<td>SEED_listen</td>
<td>Listen to incoming connections, receive and save messages in buffer</td>
<td>M</td>
</tr>
<tr>
<td>SEED_processMsg</td>
<td>Process a message from buffer</td>
<td>U</td>
</tr>
<tr>
<td>SEED_release</td>
<td>Release a child when no longer needed</td>
<td>U</td>
</tr>
<tr>
<td>SEED_remNode</td>
<td>Remove a node from list of children</td>
<td>A</td>
</tr>
<tr>
<td>SEED_report</td>
<td>Report information to an oppnet device</td>
<td>M</td>
</tr>
<tr>
<td>SEED_reqHelp</td>
<td>Request help from candidate helpers</td>
<td>A</td>
</tr>
<tr>
<td>SEED_reqRelease</td>
<td>Request CC to be released from oppnet</td>
<td>M</td>
</tr>
<tr>
<td>SEED_scan</td>
<td>Scan communication spectrum to detect devices that could become candidate helpers</td>
<td>U</td>
</tr>
<tr>
<td>SEED_sendData</td>
<td>Send data (e.g., task list) to another oppnet device</td>
<td>U</td>
</tr>
<tr>
<td>SEED_validate</td>
<td>Validate the credentials of CC, and check the ability to help</td>
<td>M</td>
</tr>
</tbody>
</table>
Table 3.3. List of second-generation helper primitives.

<table>
<thead>
<tr>
<th>OVM Primitive Name</th>
<th>OVM Primitive Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLPR_addNode</td>
<td>Add a node to list of children</td>
<td>A</td>
</tr>
<tr>
<td>HLPR_discover</td>
<td>Discover services of a certain device</td>
<td>M</td>
</tr>
<tr>
<td>HLPR_evalAdmit</td>
<td>Evaluate a device and admit it into oppnet if the device meets criteria for admittance</td>
<td>M</td>
</tr>
<tr>
<td>HLPR_isMember</td>
<td>Check if a device is already an oppnet node (oppnet member)</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_join</td>
<td>Join oppnet</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_listen</td>
<td>Listen to incoming connections, receive and save messages in buffer</td>
<td>M</td>
</tr>
<tr>
<td>HLPR_processMsg</td>
<td>Process a message from buffer</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_release</td>
<td>Release a child when no longer needed</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_remNode</td>
<td>Remove a node from list of children</td>
<td>A</td>
</tr>
<tr>
<td>HLPR_report</td>
<td>Report information to an oppnet device</td>
<td>M</td>
</tr>
<tr>
<td>HLPR_reqHelp</td>
<td>Request help from candidate helpers</td>
<td>A</td>
</tr>
<tr>
<td>HLPR_reqRelease</td>
<td>Request the inviting node to be released from oppnet</td>
<td>M</td>
</tr>
<tr>
<td>HLPR_runApp</td>
<td>Execute application indicated by authorized oppnet seed or helper node</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_scan</td>
<td>Scan communication spectrum to detect devices that could become candidate helpers</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_selectTask</td>
<td>Select a task from the task queue to execute</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_sendData</td>
<td>Send data (e.g., task list) to another oppnet device</td>
<td>U</td>
</tr>
<tr>
<td>HLPR_validate</td>
<td>Validate the credentials of the inviting node, and check the ability to help</td>
<td>M</td>
</tr>
</tbody>
</table>

*The Status column indicates the changes applied to the primitives compared to earlier work [LiKG06, LKBG06, KGLY07, KLGY08, LGKY10]: A = added, M = modified, U = Unmodified.*
3.5 Third-Generation OVM

In this subsection we discuss the third-generation OVM primitives (the latest ones) proposed by us [AlLi15]. We list the requirements for design of this generation of OVM primitives, describe the object-oriented techniques applied to this design, and present the improved set of OVM primitives.

3.5.1 Motivation and Overview

Although the second-generation OVM provides a modified set of OVM primitives with improved functionality, the sequential nature of its OVM primitives does not agree with the object-oriented languages used by most developers. Moreover, the first and second-generation OVM primitives were never implemented nor tested since both MicroOppnet v.2.2 and MicroOppnet v.4 implement the oppnet functionality in a monolithic fashion (despite the fact that a list of OVM primitives was already constructed).

The third-generation OVM incorporates object-oriented techniques into the design of OVM primitives, implements the designed OVM primitives, and tests them using the modular (non-monolithic) MicroOppnet v.5, which is the testbed for the third-generation OVM. (Implementation details for the third-generation OVM and MicroOppnet v.5 are included in the next chapter.)

3.5.2 Requirements

The following are the design requirements for the third-generation OVM:

1) The OVM primitives can be downloaded to any non-oppnet entity (system or device) to make it oppnet-enabled.

2) Any two OVM entities can interact in an ad hoc manner in order to provide needed resources [KGLY07, KGY08, LGY10]. This OVM-based interaction among
entities can support the following required oppnet characteristics, which together
distinguish oppnets from other collaborative distributed systems (incl. other
opportunistic networks):

a) support for the helper paradigm—as the basis for resource acquisition;
b) universality—regardless of the entity make or function, and through any
   communication medium, protocol, etc.;
c) lack of third-party mediators—since interactions among oppnet-based or oppnet-
   enabled entities take place without third parties;
d) ad hoc operation—except predesigned formation of the oppnet seed.

3) OVM will be highly modular. Different entities can use different subsets of the OVM
   primitives depending on their capacity and need.

4) The primitives will be designed in a non-overlapping or minimally overlapping
   manner—so that the OVM is composed of a minimal number of OVM primitives.

5) Object-oriented design principles will be applied to the OVM primitives; they include
   modularity, inheritance, and polymorphism. This will improve the OVM by
   enhancing reusability, extensibility, and readability.

### 3.5.3 OVM Primitives

This subsection presents the third-generation OVM primitives. Table 3.4 lists each
primitive name, function, names the types of nodes implementing that primitive, and the
status of the primitive that indicates the changes applied to the primitives compared to the
original (first-generation primitives).
Table 3.4. List of third-generation OVM primitives.

<table>
<thead>
<tr>
<th>OVM Primitive Name</th>
<th>OVM Primitive Function</th>
<th>Implementing Nodes&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Status&lt;sup&gt;b&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_addNode</td>
<td>Add a node to oppnet</td>
<td>All</td>
<td>A</td>
</tr>
<tr>
<td>NODE_discover</td>
<td>Discover services of a certain device</td>
<td>S, H</td>
<td>M</td>
</tr>
<tr>
<td>NODE_evalAdmit</td>
<td>Evaluate a device and admit it into oppnet if the device meets criteria for admittance</td>
<td>S, H</td>
<td>M</td>
</tr>
<tr>
<td>NODE_initiate</td>
<td>Initiate oppnet</td>
<td>C</td>
<td>U</td>
</tr>
<tr>
<td>NODE_isMember</td>
<td>Checks if a device is already an oppnet node (oppnet member)</td>
<td>S, H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_joinOppnet</td>
<td>Join oppnet</td>
<td>H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_listen</td>
<td>Listen to incoming connections, receive and save messages in buffer</td>
<td>All</td>
<td>M</td>
</tr>
<tr>
<td>NODE_processMsg</td>
<td>Process a message from buffer</td>
<td>S, H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_release</td>
<td>Release a helper when no longer needed</td>
<td>S, H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_remNode</td>
<td>Remove a node to oppnet</td>
<td>All</td>
<td>A</td>
</tr>
<tr>
<td>NODE_report</td>
<td>Report information to an oppnet device</td>
<td>S, H, L</td>
<td>M</td>
</tr>
<tr>
<td>NODE_reqHelp</td>
<td>Request help from candidate helpers</td>
<td>S, H, L</td>
<td>A</td>
</tr>
<tr>
<td>NODE_reqRelease</td>
<td>Request the inviting node to be released</td>
<td>H, L</td>
<td>M</td>
</tr>
<tr>
<td>NODE_runApp</td>
<td>Execute application indicated by authorized oppnet seed or helper node</td>
<td>H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_scan</td>
<td>Scan communication spectrum to detect devices that could become candidate helpers</td>
<td>S, H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_selectTask</td>
<td>Select a task from the task queue to execute</td>
<td>H, L</td>
<td>U</td>
</tr>
<tr>
<td>NODE_sendData</td>
<td>Send data (e.g., task list) to another oppnet device</td>
<td>All</td>
<td>U</td>
</tr>
<tr>
<td>NODE_terminate</td>
<td>Terminate oppnet</td>
<td>C</td>
<td>U</td>
</tr>
<tr>
<td>NODE_validate</td>
<td>Validate the credentials of the inviting node, and check the ability to help</td>
<td>H, L</td>
<td>M</td>
</tr>
</tbody>
</table>

<sup>a</sup> The *Nodes* column indicates the types of oppnet nodes implementing the primitive: C = control center, H = helper, L = lite, S = seed.

<sup>b</sup> The *Status* column indicates the changes applied to the primitives compared to earlier work [LiKG06, LKBG06, KGLY07, KLGY08, LGKY10]: A = added, M = modified, U = unmodified
3.5.4 Applied Object-Oriented Techniques

This subsection describes how each object-oriented (OO) technique is employed in the design of the OVM primitives, and briefly mentions the parts of OVM Primitives in which each design technique is applied.

3.5.4.1 Modularity

The nature of our OVM primitives lends itself to modularity as the primitives are designed with a high level of atomicity to serve as building blocks for wide-range of oppnet applications. There are four dimensions of modularity in the design of OVM primitives:

1) *Oppnet node category:* each one of the node categories (CC, seed, regular helper, and lite) constitutes a module that can be implemented separately using a subset of the OVM primitives. We will see in Subsection 3.5.4.3–Polymorphism, how each primitive can have four different implementations depending on the node category in which it belongs.

2) *Device type:* heterogeneous devices can be used in oppnet applications, such as tablets, laptops, wristbands etc. In order to be oppnet-enabled, each device can install the corresponding module that includes a subset of primitives compatible with the device’s capabilities and resources.

3) *Communication channel:* oppnet devices communicate through several communication technologies, (e.g., Wi-Fi, BT, and ZigBee). Primitives responsible for communication between devices such as `NODE_discover`, `NODE_scan`, `NODE_report` etc., can be implemented differently in a separate module for each communication technologies.

4) *Functionality:* as explained previously, each one of the OVM primitives implements certain functionality with minimal-to-none overlapping with the other primitives. Therefore, every primitive can be implemented as a separate module that implements its own functionality.
3.5.4.2 Inheritance

The new set (third generation) of OVM primitives (cf. Table 3.4) is designed to incorporate the inheritance design pattern. The main class in our design is the Node class which includes all the OVM primitives. By applying inheritance to the Node class, all its methods (i.e., the OVM primitives) can be passed to its children classes. As will be explained in Chapter 4, The Node class can be extended to Seed, Helper where Helper is extended to Regular and Lite. Each one of those subclasses is extended to a device type that fits the specification of that subclass. For example, Regular class is extended to Laptop and Tablet, while Lite is extended to Sensor, Printer etc.

3.5.4.3 Polymorphism

One of the forms of polymorphism is method overriding, which is the ability to provide a specific implementation of a method that is already declared in its superclass under the same name and signature. To illustrates how method overriding is applied in the implementation of the OVM primitives consider the following example, the OVM primitive NODE_scan, is possessed by the parent class Node and all its children and grand children, e.g., Seed, Helper and Regular. However, each one of the subclasses implements the same method differently. Therefore, to refer to NODE_scan of the Helper class, an object H of Helper is instantiated and NODE_scan is referenced as H.Node_scan.

---

5 In object-oriented programming, polymorphism is a principle that allows subclasses and their parent classes to possess variables or methods bearing the same name, but having different implementations (and therefore showing different behaviors) [RaBr11, Jalo08].
3.6 Template for Class Hierarchy of the OVM Primitives

This subsection depicts the template for hierarchy of the classes implementing the OVM primitives. Using the power of object-oriented programming, our OVM primitives’ class hierarchy, depicted in Fig. 3.1, allows for extension to various devices by extending each node type to the desired device and overriding the node type methods to agree with the specifications of the new device.\(^6\)

The OVM_Primitives package in Fig. 3.1 contains the Node class which is the root class. Node class implements the basic functionality of the OVM primitives in Table 3.4. Extended functionality of the OVM primitives is implemented on the children and grandchildren of the Node class. Three classes (namely CC, Seed and Helper) extend the Node class to represent the categories of oppnet nodes. The Helper class is extended to Regular and Lite classes.

Fig. 3.1 depicts how each one of the node types classes (i.e. CC, seed, regular helper and lite) can be extended to several devices. For example, Seed can be extended to Seed_Dev1, …, Seed_DevN, where DevN indicates the N\(^{th}\) seed device. In an actual class hierarchy, three seed devices (laptop, phone and tablet) would be represented as Seed_Laptop, Seed_Phone and Seed_Tablet.

In addition to the need to extend devices for each node type, each device needs to be extended to the classes that implement the primitives responsible for communication through the device’s communication protocols. For example, Seed_Dev1 can be extended to Seed_Dev1_Com1,…, Seed_Dev1_ComN, where ComN indicates the N\(^{th}\) communication protocol supported by Seed_Dev1. Therefore, in an actual class hierarchy, three communication protocols (BT, Wi-Fi, and cellular) supported by a phone seed would be represented as Seed_Phone_BT, Seed_Phone_WiFi, and Seed_Phone_Cell. The next chapter shows how the template depicted in Fig. 3.1 is used in building the class hierarchy of the OVM-based oppnet middleware.

\(^6\) The names of packages and classes take the form of Package_Name, and Class_Name to distinguish them from variable names, which take the form variableName.
Figure 3.1. The template for the class hierarchy of the third-generation OVM primitives.

Legend: \(\text{---}\) contains; \(\rightarrow\) extends
3.7 Chapter Conclusions

In this chapter we defined OVM and its primitives. We then discussed the first, second- and third-generation OVMs and described the motivation behind the design of each OVM generation. We explained how the development of the second-generation OVM helped us in understanding the functionality of OVM and how it could be further improved.

We explained how the design of the third-generation OVM was improved by modifying the functionality of the second-generation OVM, and by incorporating object-oriented design principles. Unlike the first- and second-generation OVMs which were designed but not implemented and tested, the third-generation OVM was implemented and tested, (thus will be discussed in the next chapter).
4. PROOF OF FEASIBILITY OF OVM

In this chapter we start by briefly discussing the implementation of the first-generation OVM and describing its feasibility testbed, MicroOppnet v.2.2. Then, we present the implementation details of the second-generation OVM, and its feasibility testbed, MicroOppnet v.4. Next, the chapter discusses the implementation details of the proposed third-generation OVM, including the communication, hardware and software specifications, and the class hierarchy of the OVM primitives. Finally, the chapter presents MicroOppnet v.5 and its implementation details, including, a description of the scenario used in MicroOppnet v.5; the pseudocode for the seed, helper, and lite nodes in MicroOppnet v.5; and implemented/emulated components of MicroOppnet v.5.

4.1 First-Generation OVM

This section briefly discusses the first-generation OVM [LiGY07, KGLY07, LGKY10] and its testbed, MicroOppnet v.2.2.

4.1.1 Implementation of OVM Primitives

The first-generation OVM defines atomic non-overlapping OVM primitives, dividing the functionality of oppnets into them. However, the oppnet feasibility testbed MicroOppnet v.2.2 (i.e., the OVM simulation code) is monolithic, not using these OVM primitives.

4.1.2 MicroOppnet v.2.2

As explained in Chapter 1, MicroOppnet v.2.2 [LiGY07, KGLY07, LGKY10] is both a small-scale proof of concept and a testbed for resource utilization opportunistic
network. It not only allows opportunistic communications but also opportunistically accesses sensornet (sensor network) nodes to perform sensing. Since sensing is the only resource utilized, MicroOppnet v.2.2 is rudimentary in its resource utilization opportunism. Fig. 4.1 depicts the architecture of MicroOppnet v.2.2.

Figure 4.1. MicroOppnet v.2.2 [LGKY10].

4.2 Second-Generation OVM

This section presents the implementation details of MicroOppnet v.4 testbed, including the second-generation OVM, which was designed as a part of research for this Dissertation. This includes the communication, hardware, and software specifications.

4.2.1 Implementation of OVM Primitives

As discussed in Chapter 3, the second-generation OVM modifies the original first-generation OVM by adding new primitives, modifying the functionality of existing primitives, and removing redundant primitives. However, the second-generation OVM primitives were never implemented. The testbed for the second-generation OVM was implemented in a monolithic fashion, without using the designed OVM primitives within the testbed code.
4.2.2 MicroOppnet v.4

As explained in the previous section, MicroOppnet v.2.2 served as the earliest proofs of concept and testbed for opportunistic resource utilization networks [KLGY08]. For this Dissertation, we first recreated the monolithic MicroOppnet v.2.2 (using new hardware and software) and run it for its original test scenario, to study and monitor the functionality of oppnet operations.

Using the knowledge and insights gained from re-implementing and re-running MicroOppnet v.2.2, we developed MicroOppnet v.4—again in a monolithic fashion, using the same MicroOppnet v.2.2 architecture explained in Chapter 1, and the same MicroOppnet v.2.2 test scenario. However, we used newer software and hardware.

Designing and experimenting with MicroOppnet v.4 software built in a monolithic fashion (that is, without dividing it into OVM primitives) helped us in capturing and understanding all the features and components required for the proper operation of oppnet. MicroOppnet v.4 provided further insights into the design and behavior of oppnets that were later used (cf. Chapter 5) for design and implementation of the modular (non-monolithic) MicroOppnet v.5.

4.2.2.1 Communication Specifications

MicroOppnet v.4 integrates the same communication facilities that were earlier used in MicroOppnet v.2.2 (cf. Subsection 4.1.2).

4.2.2.2 Hardware Specifications

MicroOppnet v.4 was developed using the following (slightly different) devices:

1) Machine A: a laptop equipped with Bluetooth (BT), and run on Ubuntu v.12.
2) Machine B: another laptop equipped with BT and Wi-Fi, and run on Ubuntu v.12.
3) Sensor Network, which includes:
   a) at least 5 Memsic MicaZ motes;
b) two Memsic programming boards MIB 520.

4) Victim’s phone: Samsung Android Lollipop v.5.0 phone equipped with BT.

5) Rescuer's phone: another Samsung Android Lollipop v.5.0 phone equipped with Wi-Fi.

4.2.2.3 Software Specifications

The following software was used in implementing MicroOppnet v.4:

1) The Eclipse (Kepler) development environment and Java 8.

2) TinyOS v.2.2 [Tiny13].

4.2.2.4 Scenario Description

MicroOppnet v.4 uses the same scenario that was used in MicroOppnet v.2.2 (cf. Subsection 1.2.4.2).

4.2.2.5 Implementation of MicroOppnet v.4

MicroOppnet v.4 was implemented as follows:

1) TinyOS v.2.2 was installed on Machines A and B.

2) Modified TestSerial.Java from the TestSerial application in TinyOS v.2.2 and installed on Machine A as SenderTestSerial.java. TestSerial.java was modified to read the help file sent by the victims phone to machine A, and send the contents of the file to BS1 through the serial port.

3) BaseStation, an application of TinyOS v.2.2 written in nesC, installed on 2 MicaZ motes as BS1 and BS2 following the TinyOS v.2.2 tutorial [Tiny13].

4) Modified BaseStation application to EditedBaseStation to be able to receive and send via radio. EditedBaseStation is then installed on 3 MicaZ motes as mote1, mote2, mote3 respectively.
5) Modified TestSerial.java from the TestSerial application in TinyOS v.2.2 installed on Machine B as ReceiverTestSerial.java. Upon the reception of the help file's contents, the ReceiverTestSerial.java writes the contents into a file and then uploaded through FTP to a server.

In case the motes are not available, they can be emulated using Cooja tool of Contiki [Cont13]. The steps to implementing MicroOppnet v.4 in this case are the following:

1) TinyOS v.2.2, and Cooja tool of Contiki should be installed on Machine A.
2) Install SenderTestSerial.java (from steps above) on Machine A.
3) In the Cooja tool, create 5 Sky motes named 1–5.
4) Compile BaseStation. Change the extension of the main.exe file to main.sky and install it on motes 1 and 2.
5) Compile EditedBaseStation. Change the extension of the main.exe file to main.sky and install it on motes 3–5.
6) Install ReceiverTestSerial.java on Machine A

Information about source code for MicroOppnet v.4 is available in Appendix B. More implementation and running details are available at the Readme file in the source code package. Information about source code for MicroOppnet v.4 is available in Appendix B.

4.2.2.6 Running MicroOppnet v.4

During running MicroOppnet v.4 the following events occur in this order:

1) A help text file is sent from the Victims phone to Machine A through BT.
2) SenderTestSerial.java is run on Machine A, which reads the contents of the help message and sends it to BS1 through the serial port.
3) BS1 sends the received message to the radio.
4) Mote1 receives the message from BS1 and forwards it to Mote2.
5) Upon receiving the message from Mote1, Mote2 forwards it to Mote3.
6) Mote3 forwards the message received from Mote2 to BS2 through the radio.

7) BS2 receives the message from Mote3 and forwards it through the serial port to Machine A.

8) ReceiverTestSerial.java runs on Machine B, which receives the message sent to it by BS2, writes it to a file, and then uploads it through FTP to the database server.

More implementation and running details are available at the Readme file in the source code package. Information about source code for MicroOppnet v.4 is available in Appendix B.

4.2.3 Challenges and Lessons Learned

In re-implementing MicroOppnet v.2.2, we faced many difficulties:

1) Some of software used in the original MicroOppnet v.2.2 (such as TinyOS v.1) is now outdated, and therefore not available for the newer machines we are using. Instead, we used TinyOS v.2.2, the latest version of TinyOS at the time of the re-implementation.

2) Crossbow Mica2 motes that served as sensors in the original MicroOppnet v.2.2 were not available to us; we used Memsic MicaZ motes instead.

3) Nokia phones that served as Victim and Responder in MicroOppnet v.2.2 were not available to us; we used 2 Samsung Android Lollipop smartphones instead.

Design of MicroOppnet v.4 required a good understanding of the oppnet features and operation. In turn, implementing and then running MicroOppnet v.4 provided verification of the design decisions, and provided further insights into the behavior of oppnets. In particular, the experiments revealed what additions or modifications are needed for the third-generation OVM.
4.3 Third-Generation OVM

In this section we discuss the implementation details of the third-generation OVM. We show the class hierarchy of the OVM primitives, and discuss the message and task specifications. We then present the MicroOppnet v.5 testbed for the third-generation OVM, and describe the scenario used for it. By presenting the pseudocode for each type of oppnet node, we show how the OVM primitives are used. We conclude the section by discussing the challenges and lessons learned.

4.3.1 Implementation of the Third-Generation OVM

The third-generation OVM project contains two main code packages:

1) OVM_Primitives, which contain the classes implementing the third-generation OVM primitives. See Subsection 4.3.2.

2) MicroOppnet_v5, which contains the classes that implements the MicroOppnet v.5 testbed for the third-generation OVM. These classes contain the methods using the third-generation OVM primitives. MicroOppnet v.5 and its implementation are explained later in Subsection 4.3.3.

Information about source code for the third-generation OVM and its testbed (MicroOppnet v.5) is available in Appendix B.

4.3.2 Implementation of OVM Primitives

In this subsection we present the implementation details for the OVM primitives. Implementation details include the communication, hardware and software specifications. Class hierarchy of OVM primitives is depicted and explained. We discuss the message specifications, task specifications and the implemented requirements are discussed.
4.3.2.1 Communication Specifications

Five communication technologies are used by the OVM primitives:

1) BT at 2.4 GHz;
2) Cellular 3G communication;
3) IEEE 802.15.4 at 916/433 MHz;
4) Serial Communication through UART;
   Wireless Internet standards 802.11b and 802.11g, both at 2.4 GHz.

4.3.2.2 Hardware Specifications

As explained previously, OVM primitives can be downloaded in an ad hoc way by any entity (system or device) to make it oppnet-enabled. This means that during design and implementation of the OVM primitives, entity specifications of entities have to be considered. For example, we might require different implementation of the same primitive depending on the type of the device running an OVM primitive (e.g., regular or lite).

While OVM primitives should be available for any entity, our implementation of the OVM primitives considers only the devices as specified below (and any entity that is fully compatible with the specifications of these devices). Later in this section we will discuss how our design of the class hierarchy for the OVM primitives allows for extensions to other entities.
The following devices were used to implement the third-generation OVM primitives:

1) Three laptops running Ubuntu v.12 equipped with BT and wireless network.
2) Two Samsung Android Lollipop v.5.0 smartphones equipped with BT and cellular network.
3) At least 6 Sky motes\(^7\) equipped with IEEE 802.15.4 at 916/433 MHz.

### 4.3.2.3 Software Specifications

The following programs were used for implementing/emulating the OVM primitives:

1) The Eclipse (Kepler) development environment for implementing laptop primitives written in Java.
2) Android Studio v.1.0 for implementing phone primitives written in Java for Android.
3) TinyOS v.2.2 for implementing primitives written in nesC for embedded devices such as Sky motes.
4) The Cooja tool of Contiki [Cont13] for emulating: (i) embedded devices such as Sky motes and (ii) the IEEE 802.15.4 links between various devices.

### 4.3.2.4 OVM Primitives Class Hierarchy/Nodes Specifications

This subsection depicts the hierarchy of the classes implementing the OVM primitives in our implementation of the OVM-based oppnet middleware. While the design of the third-generation OVM primitives (cf. Section 3.6 in Chapter 3) is not limited to certain devices, the implementation of the OVM primitives considers only the devices discussed Subsection 4.3.2.2–Hardware Specifications (and any entities whose specifications are compatible with the specifications of our devices). Nonetheless, thanks to the power of object-oriented programming, our OVM primitives’ class hierarchy

\(^7\) We started with 5 mica motes to develop MicroOppnet v.4. However, at the time of implementing the third-generation OVM primitives, we resorted to emulate the motes in Cooja, because we did not have enough number of motes available to us.
allows for extension to various devices by extending each node type to the desired device and overriding the node type methods to agree with the specifications of the new device.\(^8\)

The OVM Primitives package in Fig. 4.2 contains three packages: OVM_Primitives_Java, OVM_Primitives_Android, and OVM_Primitives_NesC. OVM_Primitives_Java includes the Java classes that implement the primitives for PC devices.

Following the template for the OVM primitives class hierarchy described in Chapter 3, OVM_Primitives_Java in Fig. 4.3 includes the Node class which implements (in Java) the basic functionality of the OVM primitives listed in Chapter 3, Table 3.4. Extended functionality of the OVM primitives is implemented on the children and grandchildren of the Node class. Three classes (namely CC, Seed and Helper) extend the Node class to represent the categories of oppnet nodes. The Helper class is extended to Regular class. The CC class is extended to CC_PC, which implements the OVM primitives that should run on a PC which acts as a CC node. CC_PC is extended to CC_PC_BT, CC_PC_WiFi and CC_PC_UART to implement the OVM primitives that run on PCs acting as CC nodes and requiring BT, Wi-Fi, and UART connectivity, respectively. The hierarchy of the Seed class is similar to the CC class hierarchy. Regular is extended to Regular_PC which is extended to Regular_PC_BT, Regular_PC_WiFi and Regular_PC_UART, which implement the OVM primitives that should run on PCs acting as regular helper nodes requiring BT, Wi-Fi and UART connectivity, respectively.

OVM_Primitives_Android in Fig. 4.4 includes the Node class which implements the basic functionality of the OVM primitives written in Java for Android devices. Node is extended to Helper which is extended to Regular that is finally extended to Regular_Phone. Two classes (namely Regular_Phone_BT and Regular_Phone_Cell) extend the Regular_Phone class. The two children implement the OVM primitives that should run on phones acting as regular helper nodes and requiring BT and cellular connectivity, respectively. Note that in our case, we only

---

\(^8\) The names of packages and classes take the form of Package_Name, and Class_Name to distinguish them from variable names, which take the form variableName.
implemented a single helper for the OVM_Primitives_Android package which is Regular_Phone; however, the class hierarchy depicted in Fig. 4.4 allows for extending the Node class to CC and Seed and allows extending Helper to Lite. Such extended classes can implement OVM primitives that run on a wide range of Android devices.

OVM_Primitives_NesC package in Fig. 4.5 includes the classes written in NesC language for lite nodes. NesC is a component-based language and not an object-oriented language. Therefore, we could not apply the object-oriented design of the OVM primitives explained in Chapter 3 on the design of the NesC primitives. OVM_Primitives_NesC package includes two packages namely Lite_Mote_ZigBee and Lite_BaseStation. Lite_Mote_ZigBee includes the programs that run on motes that communicate only through ZigBee which are MoteP and MoteC. Lite_BaseStation includes BaseStationC and BaseStationP which are the programs that run on motes that communicate through UART and ZigBee. As will be discussed later, Lite_BaseStation will be used to emulate the ZigBee port of some of the devices.

Figure 4.2. Packages containing the full OVM primitive hierarchy.
Figure 4.3. The class hierarchy for OVM primitives in the Java package.
Figure 4.4. The class hierarchy for the OVM primitives in the Android package.

Figure 4.5. The class hierarchy for the OVM primitives in the NesC package.
4.3.2.5 Message Specifications

This subsection describes the two main types of messages exchanged between the nodes in our simulation scenario (described in Section 4.3.3.7).

4.3.2.5.1 Oppnet Control Messages

Oppnet Control Messages are text messages that are exchanged between oppnet nodes and trigger an action by the receiving node. Oppnet Control Messages can be of five types:

1) **Help Request**: is a text message created by the inviting node and sent to candidate helper(s) to request help for oppnet, e.g., “Help requested by oppnet <oppnet_id>.”

2) **Invitation Acceptance**: is a text message created by the candidate helper and sent to the inviting node to accept invitation to oppnet, e.g., “Joining oppnet <oppnet_id>.”

3) **Admittance Notification**: is a text message created by the inviting node and sent to candidate helper(s) that passed the evaluation for admittance into oppnet and now admitted into oppnet, e.g., “Admitted into oppnet <oppnet_id>.”

4) **Release Request**: is a text message created by the admitted helper and sent to the inviting node to request release from oppnet, e.g., “Release request from oppnet <oppnet_id>.” We allow a helper to resign only after getting permission from its inviting node (parent). We allow no unilateral quitting by a helper before it completes its assigned job.

5) **Release Notification**: is a text message created by the inviting node and sent to its admitted helper(s) (children), to release the child from oppnet, e.g., “Released by oppnet <oppnet_id>.”
4.3.2.5.2 Data Messages

Data Messages are messages that contain data objects created initially by the CC node and sent from each inviting node (parent) to its admitted helpers (direct children). There are two main data messages:

1) Object messages, which contain the information collected or created by the CC and passed from an inviting parent to all its direct children.

2) Oppnet task messages, which contain the list of oppnet tasks created by the CC and passed from each inviting node to all its direct children.

4.3.2.6 Task Specifications

During the development of the OVM primitives we tried to capture the generic needs of a wide range of applications, without actually specifying a very broad range of possible tasks.

The list of tasks with which we actually experimented includes the following tasks: (i) send a task via Wi-Fi/BT/ZigBee/Cellular; (ii) forward a report via Wi-Fi/BT/ZigBee/Cellular; (iii) log readings to the DB server; (iv) retrieve readings from the DB server; and (v) display a message on a local monitor. (Chapter 5 will explain why these tasks were used.)

4.3.2.7 Implemented Requirements

Recall the third-generation requirements we specified in Chapter 3. The following requirements are used for the third-generation OVM (some with indicated limitations):

1) The OVM primitives can be downloaded by and installed in an ad hoc way on any entity (system or device) to make the entity oppnet-enabled. This requirement is currently limited to the devices specified in Subsection 4.3.2.2–Hardware Specifications.
2) Any two OVM components or oppnet-enabled devices can interact in an ad hoc manner [KGLY07, KLGY08, LGKY10]. This OVM-based interaction between entities can support the following required oppnet characteristics, which together distinguish oppnets from other available collaborative distributed systems:

a) Support for the helper paradigm—as the basis for resource acquisition.

b) Universality—regardless of the system or device make or function, and through any communication medium, protocol, etc. *This requirement is limited to communication technologies specified in Subsection 4.3.2.1–Communication Specifications and the devices we specified in Subsection 4.3.2.2–Hardware Specifications.*

c) Lack of third-party mediators—since interactions among oppnet-enabled entities take place without (trusted) third parties.

d) Ad hoc operation—except predesigned formation of the oppnet seed. During the testing of our implementation, we specified the id’s of our devices for the scanning of ad hoc devices, to avoid inviting outside devices.

3) OVM shall be highly modular. Different systems or devices can use different subsets of the OVM primitives depending on their capacity and need. *This requirement is currently limited to the devices we specified in Subsection 4.3.2.2–Hardware Specifications.*

4) The primitives shall be designed in a non-overlapping or minimally overlapping manner so that the OVM is composed of a minimal set of OVM primitives.

5) Object-oriented design patterns shall be applied to the OVM primitives including modularity, inheritance, and polymorphism. Such techniques would improve the OVM primitives by enhancing reusability, extensibility, and readability.

4.3.3 MicroOppnet v.5

*MicroOppnet v.5* is the testbed for the third-generation OVM, and tests the OVM for developing a healthcare application for monitoring medical conditions and wellness.
4.3.3.1 Communication Specifications

The communication technologies used in MicroOppnet v.5 are the same as the communication technologies explained in Subsection 4.3.2.1.

4.3.3.2 Hardware Specifications

The hardware devices used in implementing/emulating MicroOppnet v.5 are the same as the hardware devices explained in Subsection 4.3.2.2.

4.3.3.3 Software Specifications

The following software tools were used for implementing/emulating MicroOppnet v.5:

1) Eclipse Kepler for implementing the java main class that is run by laptop devices;

2) Android Studio v.1.0 for implementing the java main class that is run by the smartphone;

3) TinyOS v.2.2 for implementing the nesC main classes that are run by embedded devices such as Sky motes;

4) Cooja tool of Contiki for emulating the embedded devices.

4.3.3.4 Node Specifications

MicroOppnet v.5 has 9 nodes. Table 4.1 lists the specifications for each node.
Table 4.1. Nodes in MicroOppnet v.5.

<table>
<thead>
<tr>
<th>No.</th>
<th>Node Class (nodeClass)</th>
<th>Node Name (nodeName)</th>
<th>Helper Category (helperCategory)</th>
<th>Communication Protocol (commProtocol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Seed and CC</td>
<td>Wristband</td>
<td>N/A</td>
<td>{BT}</td>
</tr>
<tr>
<td>1</td>
<td>Regular</td>
<td>Tablet1</td>
<td>adhoc</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>2</td>
<td>Lite</td>
<td>Printer</td>
<td>adhoc</td>
<td>{ZigBee}</td>
</tr>
<tr>
<td>3</td>
<td>Lite</td>
<td>Streetlight</td>
<td>adhoc</td>
<td>{ZigBee}</td>
</tr>
<tr>
<td>4</td>
<td>Lite</td>
<td>ParkingMeter</td>
<td>adhoc</td>
<td>{ZigBee}</td>
</tr>
<tr>
<td>5</td>
<td>Lite</td>
<td>DigitalTV</td>
<td>reservist</td>
<td>{ZigBee}</td>
</tr>
<tr>
<td>6</td>
<td>Regular</td>
<td>Tablet2</td>
<td>adhoc</td>
<td>{Wi-Fi, BT, ZigBee}</td>
</tr>
<tr>
<td>7</td>
<td>Regular</td>
<td>Smartphone</td>
<td>adhoc</td>
<td>{Wi-Fi, BT, Cellular}</td>
</tr>
<tr>
<td>8</td>
<td>Regular</td>
<td>DBserver</td>
<td>adhoc</td>
<td>{Wi-Fi}</td>
</tr>
</tbody>
</table>

4.3.3.5 Message Specifications

There are two main types of messages exchanged between the nodes in our healthcare scenario (presented in Section 4.3.3.7).

4.3.3.5.1 Oppnet Control Messages

*Oppnet Control Messages* in MicroOppnet v.5 are text messages exchanged between oppnet nodes that trigger an action in the receiving node. The specifications of the control messages are as explained in Subsection 4.3.2.5.1.

4.3.3.5.2 Data Messages

Recall that *Data Messages* contain data objects created initially by the CC node and sent from each inviting node (parent) to its admitted helpers (direct children). The two main data messages in MicroOppnet v.5 are:
1) *Help object messages*, where each such message contains a help object; the helpObject is an object created by the CC/seed (the Wristband in this scenario) and passed from each inviting node to all its direct children.

The helpObject is composed of the following:

a) A help text message (helpTextMsg): indicating that a person has collapsed and needs an urgent help.

b) The person’s profile (personProfile): containing the current person’s location, name, age, weight, allergies, etc.

c) The person’s contact information (personContact): for her primary healthcare facility.

d) The person’s recent vitals (personVitals): such as temperature, blood pressure, etc.

2) *Oppnet Task Messages*, where each such message contains the list of oppnet tasks created by the CC/seed (the Wristband in this scenario) and passed from each inviting node to all its direct children.

4.3.3.6 Task Specifications

For the scenario used for the MicroOppnet (presented in Section 4.3.3.7), the tasks are:

1) Task 1: Create the helpObject and oppnetTasks. Done by the seed/CC: Wristband.

2) Task 2: Send/Forward helpObject (stored locally in main memory) to a remote helper. Done by the seed/CC: Wristband; or by the regular helpers: DigitalTV, Tablet1, Tablet2; or by lite helpers: ParkingMeter, Printer, Streetlight.

3) Task 3: Send/Forward oppnetTasks (stored locally in main memory) to remote helpers. Done by the seed/CC: Wristband; or by the regular helpers: DigitalTV, Tablet1, Tablet2; or by lite helpers: ParkingMeter, Printer, Streetlight.
4) Task 4: Display on a local monitor the text help message stored locally in the helpObject in main memory. Done by regular helpers: DigitalTV, Smartphone, Tablet1, and Tablet2.

5) Task 5: Print on a local printer the help text message stored locally in the helpObject in main memory. Done by lite helper: Printer.


7) Task 7: Read/process the logging request and log the helpObject to the database server. Done by regular helper DBserver.

8) Task 8: Deliver helpTextMsg to the (remote) 9-1-1 service via SMS (this is the final task of the scenario, realizing its goal). Done by regular helper Smartphone.

Tasks can be prioritized. For this scenario, Tasks 1, 2 and 3 are urgent, Tasks 4–7 are unhurried, and Task 8 is the goal.

4.3.3.7 Scenario Description

When a person loses consciousness outdoors, his Body Sensor Network (BSN) Wristband detects an emergency situation (the signs include: low temperature, low blood pressure, collapsing). Fig. 4.6 illustrates how an oppnet is used in wellness monitoring and detecting emergencies.

The Wristband (which is the CC—and thus a seed node—in this case) initiates an oppnet expansion; it prepares the helpObject and the oppnetTasks.

When Tablet1 receives the invitation, it decides to join oppnet based on: (i) its willingness to help; and (ii) its ability to achieve the required tasks. Tablet1 willingness to help can be a result of two situations: (i) it is idle and has nothing else to do (ii) it is busy, but the tasks that the tablet is asked to do by oppnet are far more important (e.g., life or death situations). Tablet1 joins the oppnet by sending a join acknowledgement to the inviting node (Wristband).
The inviting node then evaluates the joining node; it admits the new node as Helper H1 into the oppnet if it meets the criteria. The inviting node then adds Tablet1 to its list of children and sends to it the oppnet task message containing oppnetTasks, and the help object message containing helpObject.

Upon receiving the messages, Helper H1 checks if it can perform the most urgent task which is informing 9-1-1. However, it (Tablet1) is incapable of contacting 9-1-1; thus, it tries to perform at least one of the five unhurried tasks (see the Task Specification subsection).

Helper H1, in possession of a passer-by is not connected to a Wi-Fi network. So, it is only able to display the help message to attract attention of its owner. At the same time, it looks for other BT devices. Since it finds no BT devices by either directory look up or discovery, it searches for ZigBee devices. It finds three ad hoc candidate helpers: Streetlight (H2), ParkingMeter (H3), and Printer (H4). Helper H1 then sends requests for help to all three candidate helpers. The helpers validate the credentials of the inviting node and check their ability and their willingness to help. Eventually, the three helpers accept the invitation to join the oppnet. However, after evaluation, Helper H1 chooses the printer—since it has more capabilities for displaying the help message. The printer becomes Helper H4 (a “lite” helper).
Upon receiving its tasks, Helper H4 prints the help message, and searches for ZigBee devices. It finds a reservist: DigitalTV (H5) in a nearby house. The digital TV set becomes Helper H5 (a “lite” helper). It plays the video message (received from CC via H1 and H4) to attract people’s attention.

Since the goal of notifying 9-1-1 is not realized yet, Helper H5 searches for ZigBee devices, and discovers Tablet2 (H6) in a nearby house. In a due process, this tablet becomes Helper H6.

Helper H6 can connect to Wi-Fi; hence, it is able to upload the helpObject to the healthcare link (H7a) identified in the help object message. Helper H6 searches for BT devices and finds Smartphone (H7b). The smartphone becomes Helper H7b.

Helper H7b deliver an SMS message containing the text help message to 9-1-1.

Since the goal (notifying 9-1-1) is now achieved, Helper H7b sends a release request to its inviting node, Helper H6, indicating completion of it task and the overall goal. Helper H6 (Tablet2) removes Helper H7b from the list of its children and then sends a release message to it.

The process of releasing helpers is repeated, until the CC (Wristband) receives the last release request from Helper H1. The CC then terminates oppnet.

We must note that if we were luckier, the goal (notifying 9-1-1) could have been achieved earlier, maybe even by Helper H1 (if it had cellphone or other capabilities to contact 9-1-1).

It should also be noted that in our scenario the devices were asked for help without specifying what kind of help was needed. This implies that a helper unable to perform all requested tasks must forward them to next-level helpers. OVM primitives allow also for an alternative approach, in which candidate helpers are asked by the inviting node if they can perform the required tasks; the response becomes a criterion for selecting helpers from a set of candidate helpers.
4.3.3.8 Components and Pseudocode for MicroOppnet v.5 Using OVM Primitives

This subsection describes the class hierarchy for the MicroOppnet_v5 package, and presents the pseudocode for its components.

Each component of MicroOppnet_v5 package, depicted in Fig. 4.7 Each class contains a main method that use the OVM primitives implemented in OVM_Primitives package (illustrated in Subsection 4.3.2.4). WristBand_Main contains a main method that implements the seed pseudocode explained next. Tablet1_Main, Tablet2_Main, Smartphone_Main and DBserver_Main contain main methods implementing the helper pseudocode explained in this section. All_Lites_Main contains the nesC classes implementing the lite pseudo code that should run on Printer, ParkingMeter, StreetLight and DigitalTV.

Our pseudocode is an object-oriented (OO) pseudocode based on the Java code syntax, which allows using some desirable OO capabilities in the pseudocode (as will be explained). To the best of our knowledge we are the first to use OO pseudocode.
4.3.8.1 Seed Pseudocode

Fig. 4.8 shows the pseudocode for the seed node. In our scenario, the seed pseudocode is implemented in Wristband_Main.

Since this single seed node is also a CC, the pseudocode for it includes—either directly in the seed pseudocode, or indirectly in the methods NODE_searchHelpers() and NODE_releaseAllChildren()—primitives defined: for CC but not seed nodes (NODE_initiate, NODE_terminate); or for seed nodes but not CC (NODE_scan, NODE_discover, NODE_isMember, NODE_reqHelp, NODE_evalAdmit, NODE_report, NODE_release); or for both CC and seed nodes (NODE_listen, NODE_addNode, NODE_sendData, NODE_remNode).
In Line 5 of the Seed pseudocode we declare an object s of the class Seed which extends the class Node. Both classes (illustrated in Fig. 4.3) implement the oppnet primitives defined in Chapter 3, Table 3.4. In Line 7, the module passes the object s as an argument for the called method NODE_searchHelpers().

Use of the class Node in the name NODE_searchHelpers() requires an explanation. As mentioned earlier, we use polymorphic pseudocode. The method NODE_searchHelpers() and OVM primitives are polymorphic, i.e., they are implemented in the parent class Node as well as all its children classes, namely CC, Seed, Helper, and Lite.

As an example of using polymorphism, suppose that the method NODE_searchHelpers() is polymorphic in the sense that it can be called by either the class Node or any of its derived (children) classes.

The method defines an object parameter of the class Node to use it for calling any of the OVM primitives. Therefore, when the method is called within (for example) a Seed module, it is passed an object of the Seed class; this indicates that the Seed version of the OVM primitive needs be executed.

The method NODE_searchHelpers(), shown in Fig. 4.9, searches either for ad hoc helpers, or looks up the appropriate reservist directories [2]. The Boolean variable

Figure 4.8. Seed pseudocode.
canForward indicates the availability of forwarding services in the given device. If the device canForward, the node—the Seed node $s$ for the call we consider—invites it to join the oppnet. If the device agrees to join the oppnet, the inviting node (here node $s$) checks if it meets the criteria for admittance (e.g., if it has the “competence” to do the required tasks); if the check result is positive the inviting node (here node $s$) admits the invited helper node into the oppnet.

```java
1: public List NODE_searchHelpers(Node n)
2: {
3:     devices = n.NODE_scan(communicationProtocolChannel);
4:     foreach device in devices do {
5:         canForward = n.NODE_discover(forwardServices);
6:         if (canForward)
7:             candidateNode = device;
8:         if (!n.NODE_isMember(candidateNode)) {
9:             n.NODE_reqHelp(candidateNode);
10:             joined = n.NODE_listen(“Join”);
11:         } // end if (!n.NODE_is_member(…))
12:     }
13:     if (joined) {
14:         admitted = n.NODE_evalAdmit(candidateNode);
15:         if (admitted) {
16:             /* Add the admitted node to its list of children and send oppnet tasks and helpObject to it */
17:             admittedNode = candidateNode;
18:             children.add(admittedNode);
19:             n.NODE_addNode(admittedNode);
20:             n.NODE_sendData(oppnetTasks, admittedNode);
21:             n.NODE_report(helpObject, admittedNode);
22:         } // end if (admitted)
23:     } // end if (joined)
24: }
25: // end method NODE_searchHelpers
```

Figure 4.9. Pseudocode for NODE_searchHelpers().

Back in Fig. 4.8, in Line 7, the set of children returned by the NODE_searchHelpers() is assigned to children. After the oppnet’s goal is achieved, in Line 13 the seed $s$ calls NODE_releaseAllChildren(), which is shown in Fig. 4.10.
In `NODE_releaseAllChildren()`, the inviting node (the Seed node in this case) listens for incoming release requests. For each release request coming from a child, the inviting node removes the child from its list of admitted nodes, and sends it a release notification.

```java
public void NODE_releaseAllChildren()
{
    /* When goal achieved, release all admitted nodes. */
    while (children.size() > 0)
    {
        reqRLS = NODE_listen("Request Release");
        if (reqRLS){
            NODE_release(admittedNode);
            children = NODE_remNode(admittedNode);
        }
    }
}
```

Figure 4.10. Pseudocode for `NODE_releaseAllChildren()`.

### 4.3.3.8.2 Helper Pseudocode

Fig. 4.11 shows the pseudocode for helper nodes; in our scenario, this is the code for `Tablet1_Main`, `Tablet2_Main`, `DBserver_Main` and `Smartphone_Main`. In Line 12, the method `NODE_selectPerformTasks()` is called.

In the method `NODE_selectPerformTasks(oppnetTasks)`, shown in Fig. 4.12, the calling node iterates through the `oppnetTasks` sent to it in order to select the task it is willing to do. The Boolean variable `doneGoal` is used in Line 9 to test whether the node was able to accomplish the goal; if it was, the Boolean variable `done` becomes true—indicating that the oppnet goal was achieved, and the helper needs no additional helpers.

Returning to the helper pseudocode (Fig. 4.11), we see in Line 16 declaration and creation of an object `h` of the class `Helper`, derived from the class `Node`. Both classes `Helper` and `Node` (shown in Fig. 4.3) implement the oppnet primitives defined in Table 3.4 in Chapter 3. In Line 18, the helper object `h` is passed as an argument for the method `NODE_searchHelpers()`.
/* Helper receives an invitation to join oppnet. */
invited = NODE_listen("Request Help, Invite to Oppnet");
if (invited) {
    /* Helper accepts the invitation if the helper is willing and can help. */
    NODE_validate(invitingNode);
    NODE_joinOppnet(invitingNode);
    NODE_listen(oppnetTasksMsg);
    oppnetTasks = NODE_processMsg(oppnetTasksMsg);
    NODE_listen(helpObject);
    helpObject = NODE_processMsg(helpObjectMsg);
    done = NODE_selectPerformTasks(oppnetTasks);
    /* While goal is not achieved, search for other candidate helpers. */
    Helper h = new Helper();
    while (!done) {
        children = NODE_searchHelpers(h);
        done = NODE_listen("Done");
    } // end while
    /* When goal achieved, release all helpers. */
    NODE_releaseAllChildren();
    /* When all helper’s children are released, the helper sends a release request to its inviting node (parent). */
    NODE_reqRelease(invitingNode);
    released = NODE_listen("Released");
    if (released) {
        /* Device returns to its original operations. */
    } // end if (released)
} // end if (invited)

Figure 4.11. Helper pseudocode.

4.3.3.8.3 Lite Pseudocode

Lite pseudocode is implemented in Printer_Main, Streetlight_Main, ParkingMeter_Main, and DigitalTV_Main. Pseudocode for lites is very similar to the helper pseudocode; hence it is omitted for the sake of brevity.
4.3.3.9 Implemented and Emulated Components of MicroOppnet v.5

In our development of MicroOppnet v.5, we have a limited number of devices and technologies available to us. Therefore, some of the devices and technologies used in the scenario were implemented with the real corresponding devices and technologies while some were emulated using emulation programs (as indicated in Subsection 4.3.2.3–Software Specifications). Table 4.2 lists the implemented and emulated devices in MicroOppnet v.5, and Table 4.3 lists the implemented and emulated links in MicroOppnet v.5.
Table 4.2. Implemented/emulated devices in MicroOppnet v.5.

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Implemented / Emulated</th>
<th>Implementation / Emulation Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wristband</td>
<td>Emulated</td>
<td>Laptop #1 running Ubuntu v.12 equipped with BT</td>
</tr>
<tr>
<td>Tablet1</td>
<td>Emulated</td>
<td>Laptop #2 running Ubuntu v.12 equipped with BT</td>
</tr>
<tr>
<td>Printer</td>
<td>Emulated</td>
<td>Sky mote # 3 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>Streetlight</td>
<td>Emulated</td>
<td>Sky mote # 4 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>ParkingMeter</td>
<td>Emulated</td>
<td>Sky mote # 5 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>DigitalTV</td>
<td>Emulated</td>
<td>Sky mote # 6 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>Tablet2</td>
<td>Emulated</td>
<td>Laptop #2 running Ubuntu v.12 equipped with BT and Wi-Fi</td>
</tr>
<tr>
<td>Smartphone</td>
<td>Implemented</td>
<td>Samsung Android Lollipop v.5.0 smartphone equipped with BT and Cellular connection</td>
</tr>
<tr>
<td>DBserver</td>
<td>Implemented</td>
<td>Laptop #3 running Ubuntu v.12 equipped with Wi-Fi</td>
</tr>
</tbody>
</table>

Table 4.3. Implemented/emulated links in MicroOppnet v.5.

<table>
<thead>
<tr>
<th>Link</th>
<th>Implemented / Emulated</th>
<th>Implementation / Emulation Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT link between Wristband and Tablet1</td>
<td>Implemented</td>
<td>BT stack in Wristband (laptop#1) and Tablet1 (laptop#2).</td>
</tr>
<tr>
<td>ZigBee port of Tablet1</td>
<td>Emulated</td>
<td>Sky mote #1 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>ZigBee link between Tablet1 and Printer</td>
<td>Emulated</td>
<td>IEEE 802.15.4 protocol emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12</td>
</tr>
<tr>
<td>ZigBee link between Tablet1 and Streetlight</td>
<td>Emulated</td>
<td></td>
</tr>
<tr>
<td>ZigBee link between Tablet1 and ParkingMeter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZigBee link between ParkingMeter and DigitalTV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZigBee link between DigitalTV and Tablet2</td>
<td>Emulated</td>
<td>Sky mote#2 emulated using Cooja emulator which runs on laptop #2 running Ubuntu v.12.</td>
</tr>
<tr>
<td>Wi-Fi link between Tablet2 and DBserver</td>
<td>Implemented</td>
<td>Socket connection in Tablet2 (laptop#2) and DB server (laptop#3).</td>
</tr>
<tr>
<td>BT link between Tablet2 and Smartphone</td>
<td>Implemented</td>
<td>BT stack in Tablet2 (laptop#2) and Smartphone (android smartphone#1).</td>
</tr>
<tr>
<td>Cellular link between Smartphone and 9-1-1</td>
<td>Implemented</td>
<td>SMS connection in Smartphone (android smartphone #1) and 9-1-1 (android smartphone #2).</td>
</tr>
</tbody>
</table>
4.3.3.10 Implementation of MicroOppnet v.5

MicroOppnet v.5 is implemented as follows:

1) Implement Wristband_Main.java which uses the Seed pseudocode explained in Subsection 4.3.3.8.1 using Java 8 in laptop#1.

2) Implement Tablet1_Main.java which uses Helper pseudocode explained in Subsection 4.3.3.8.2 using Java 8 in laptop#2.

3) Implement MicroOppnet_v5_NesC which includes the Lite_BaseStation_Main application and Lite_Mote_ZigBee_Main application. Lite_BaseStation_Main is the nesC application that emulates the ZigBee port of Tablet1 and Tablet2 nodes. Lite_Mote_ZigBee_Main is the nesC application that should reside in the lite nodes which are Printer, Streetlight, and ParkingMeter.

4) Compile the nesC files under Lite_BaseStation_Main. Change the extension of the main.exe file to main.sky.

5) Compile the nesC files under Lites_Mote_ZigBee_Main. Change the extension of the main.exe file to main.sky.

6) Create a Cooja emulation file in laptop#2 call it Cooja_Emulation.csc.

7) Inside Cooja_Emulation.csc, create Skymote#1 named 1 to emulate the ZigBee port of Tablet1. Create Skymote#2 named 2 in Cooja to emulate the ZigBee port of Tablet2. Install main.sky of the Lite_BaseStation_Main on motes 1 and 2.

8) Create 4 Sky motes 3–6 in Cooja to emulate the Printer, Streetlight, ParkingMeter, and DigitalTV devices. Install main.sky of the Lite_Mote_Zigbee_Main on motes 3–6.

9) Implement Tablet2_Main.java which uses Helper pseudocode using Java 8 in laptop#2.
10) Implement `DBserver_Main.java` which uses Helper pseudocode using Java 8 in laptop#3.

11) Implement `MicroOppnet_v5_Android` application which uses Helper pseudocode using Java for android in smartphone#1. Compile `MicroOppnet_v5_Android` application. A file with an apk extension should be generated. Install the apk file in one of the Android Lollipop v.5.0 smartphones which is smartphone#1.

More implementation details are available in the Readme file under the source code package. Information about the source code is available in Appendix B.

### 4.3.3.11 Running MicroOppnet v.5

In order to run MicroOppnet v.5, the following steps must be followed in this order:

1) In laptop#2, run the following:
   a) Run `Tablet1_Main.java` which will start by listening to incoming BT communication.
   b) Run the Cooja tool; start `Cooja_Emulation.csc` which will run the six sky motes which will start listening for incoming ZigBee communication.
   c) Run `Tablet2_Main.java` which will start by listening to incoming ZigBee communication.

2) In laptop#3, run `DBserver_Main.java` which will start listening to Wi-Fi communication.

3) In smartphone#1 run the `Phone.apk`, which will start listening to incoming BT communication.

4) In laptop#1, run `Wristband_Main.java` which will start oppnet and communicate with `Tablet1_Main.java`.

5) After performing steps 1–4, the communication among nodes and oppnet operations take place according to the scenario described in Section 4.3.3.7 and smartphone#2 should get an SMS message (containing the `helpTextMsg`).
It is important to note that in order to rerun MicrOppnet v.5, several parameters should be changed to agree with addresses/names of used devices. Such parameters includes, but are not limited to the phone number of smartphone#2 (the phone that acts as 9-1-1) which is specified in MicroOppnet_v5_Android application, and the IP address of the DB server (which is specified in DBserver_Main.java).

More implementation and running details are available in the Readme file under the source code package. Information about the source code is available in Appendix B.

4.3.4 Challenges, Lessons Learned, and Limitations

The main challenge faced during the development of the third-generation OVM was the limited types and number of devices available for building the testbed. Therefore, we resorted to emulating a number of components (cf. Subsections 4.3.3.9).

The other challenge was choosing a proper emulation tool. We started with the TinyOS Simulation Tool (TOSSIM) [Tiny13]; however, it did not provide the expected facilities. Then, we experimented with Cooja (a simulator/emulator for Contiki), which uses the C language. Since we used nesC for writing the code of some of the primitives, we had to compile the nesC code and run the compiled files in Cooja.

As with any large project, the development of the third-generation OVM went through many iterations of design, pseudocode development, code development, testing, evaluation and correction. During this process, we were able to evaluate our design/implementation choices, and improve both design and implementation. These choices included but were not limited to the selection of devices for running the primitives, the choice of data structures holding the oppnet messages, and the choice of emulation tools.

We recognize that many more even critical extensions or improvements had to stay beyond the scope of this Dissertation. Maybe the most critical limitation (due to the limited “womanpower”) was excluding from the development any privacy and security considerations. Removing these limitations remains as future work.
4.4 Chapter Conclusions

We briefly discussed the implementation details for MicroOppnet v.2.2, the testbed for the first-generation OVM.

We then presented the implementation details for the MicroOppnet v.4, the testbed for the second-generation OVM. The shown implementation details for MicroOppnet v.4 include its hardware and software specifications, its execution steps, and the challenges and lessons learned during the development of MicroOppnet v.4.

Finally, the chapter discussed the implementation of the proposed third-generation OVM. The discussion included the communication, hardware and software specifications, and the class hierarchy for the OVM primitives. The chapter finally presented the implementation of the MicroOppnet v.5 testbed including the description of its scenario; the pseudocode for its seed, helper, and lite nodes; listing its implemented/emulated components; the challenges faced and the lessons learned during the development of MicroOppnet v.5; and limitations of this work waiting for future work.
5. PERFORMANCE EVALUATION OF THE OVM PRIMITIVES AND COMPARISON OF THE MONOLITHIC VS. MODULAR OVM

5.1 Simulation Scenario

We simulate the healthcare scenario [AlLi15] using Java and the simulation package SimJava2 [SiJa15].

The goal of the simulation is to compare the overhead required by the following two oppnet implementations to achieve the goal presented by the scenario:

1) Monolithic implementation—without the OVM primitives.

2) Modular (OVM-based) implementation—using OVM primitives.

5.2 Simulation Design

This Section presents the simulation design including: simulation general assumptions, simulation helper assumptions, assumptions for simulated nodes, simulation message assumptions, simulation technology/standard related assumptions, simulation task assumptions/setup, simulation input parameters, simulation random variables, and simulation measures.
5.2.1 Simulation General Assumptions

1) Simulation area is a 600 m by 600 m square (3 × 3 = 9 Chicago blocks\(^9\)).

2) There are 9 types of nodes; their specifications are given in Table 5.1.

3) In general, there are three modes of requesting candidate helpers for help:
   a) *Single-helper mode*: an oppnet node asks a *single* candidate helper to join oppnet.
   b) *Many-helper mode*: an oppnet node asks *k* out of *n* candidate helpers to join oppnet.
   c) *All-helper mode*: an oppnet node asks *n* (all) candidate helpers to join oppnet.

4) In our simulation we only implement 3.a and 3.c as the two extreme modes.

5) The *workload ratio* for a node is the ratio of the actual/current workload to workload capacity for this node. We assume that any candidate helper which has the workload ratio <95% is willing to help.\(^{10}\) This means that the helper has at least 5% of spare capacity.

6) If an oppnet node runs more than one application, we assume that:
   a) If the node is a regular helper, then multiple applications are run on it *in parallel*—since we assume that such nodes have fewer parallel applications than processors or cores.
   b) If the node is a lite, then multiple applications are run on it *sequentially*—since we assume that such nodes have a single-core processor.

5.2.2 Assumptions for Simulated Nodes

The specifications for 9 types of nodes (cf. item 2 in Subsection 5.2.1) are listed in Table 5.1.

---

\(^9\) Chicago was planned on a grid system with eight blocks to each mile on the city street grid [GaBl15]. Therefore, each block is 200 m by 200 m.

\(^{10}\) This is an assumption since, in general, any ad hoc helper may refuse to help; only reservist helpers have to help.
Table 5.1. Assumed helper categories and communication protocols for simulated nodes.

<table>
<thead>
<tr>
<th>Node Type (nodeType)</th>
<th>Node Class (nodeClass)</th>
<th>Entity Name (entityName)</th>
<th>Helper Category (helperCategory)</th>
<th>Communication Protocol (commProtocol)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Seed and CC</td>
<td>Wristband</td>
<td>N/A</td>
<td>{BT}</td>
</tr>
<tr>
<td>1</td>
<td>Regular</td>
<td>Tablet1</td>
<td>adhoc</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>2</td>
<td>Lite</td>
<td>Printer</td>
<td>adhoc</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>3</td>
<td>Lite</td>
<td>Streetlight</td>
<td>adhoc</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>4</td>
<td>Lite</td>
<td>ParkingMeter</td>
<td>adhoc</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>5</td>
<td>Regular</td>
<td>DigitalTV</td>
<td>reservist</td>
<td>{BT, ZigBee}</td>
</tr>
<tr>
<td>6</td>
<td>Regular</td>
<td>Tablet2</td>
<td>adhoc</td>
<td>{WiFi, BT, ZigBee}</td>
</tr>
<tr>
<td>7</td>
<td>Regular</td>
<td>Smartphone</td>
<td>adhoc</td>
<td>{WiFi, BT, Cellular}</td>
</tr>
<tr>
<td>8</td>
<td>Regular</td>
<td>DBserver</td>
<td>adhoc</td>
<td>{WiFi, BT}</td>
</tr>
</tbody>
</table>

5.2.3 Simulation Helper Assumptions

1) After checking the current directory for reservist helpers and scanning for ad hoc helpers, the inviting node uses the following criteria for inviting helpers:

   a) Helper’s communication protocol: the helper should be equipped with at least one of the communication protocol (cf. Table 5.1) available to the inviting node;

   b) Helper’s location: (i) within the simulation area, and (ii) within the communication range of the inviting node.\textsuperscript{11}

5.2.4 Simulation Message Assumptions

This subsection starts by defining the types of simulated messages and their payloads. We then show how size of each message payload was calculated for our simulation. We then explain the calculations of time needed for reading message payloads.

\textsuperscript{11} As discussed in Section 1.2.2 “Basic Oppnet Operations,” if the helper is able to perform all tasks assigned to by the inviting node alone, then it does so. Once the tasks are completed, the helper reports task completion to the inviting node. Consequently, the inviting node (or the CC) can release the helper. On the other hand, if the helper is unable to execute all the tasks from the list of assigned tasks without a further help, the helper can look for its own (next-level) helpers and assign to them the “remaining” tasks—the ones it cannot complete itself. (This includes the special case when a helper is unable to perform any tasks, but is able to locate next-level helpers, to which it forwards the whole task list.)
5.2.4.1 Type of Simulated Messages

This subsection defines the oppnet messages used in our simulation.

5.2.4.1.1 Oppnet Control Messages

As explained previously in Chapter 4, Oppnet Control Messages are text messages that are exchanged between oppnet nodes, and trigger an action by the receiving node. The specifications of the oppnet control messages for simulation are as explained in Chapter 4.

5.2.4.1.2 Data Messages

As explained in Chapter 4, Data Messages are messages that contain data objects created initially by the CC node and sent from each inviting node (parent) to its admitted helpers (direct children). The two main data messages for this simulation are Help Object Messages and Oppnet Task Messages.

A. Help Object Messages

The Help Object Messages are messages containing helpObject which is an object created by the seed Wristband and passed from each inviting node to all its direct children. The helpObject is composed of the following:

1) A help text message (helpTextMsg): indicating that a person has collapsed and needs an urgent help.

2) A person’s profile (personProfile): containing the current person’s location, name, age, weight, allergies, etc.

3) A person’s contact information (personContact): for her primary healthcare facility.

4) A person’s recent vitals (personVitals): such as temperature, blood pressure, etc.
B. Oppnet Task Messages

As explained in Chapter 4, Oppnet Task Messages contain the list of oppnet tasks created by the CC/seed (Wristband in this simulation scenario), and passed from each inviting node to all its direct children.

5.2.4.2 Size of Message Payload

This subsection shows how the size of each message payload is calculated.

5.2.4.2.1 Sizes of Used Data Types

The following are the sizes of the Java data types used to represent each message payload:

1) The size of a character is 2 B.

2) The size of an integer is 4 B.

3) The size of an empty object header is 8 B.

4) An empty array (of any type) in Java contains the following [ChCo15, JaMe15]:
   a) The array object header is 8 B.
   b) The integer length of the array is 4 B.

   Therefore, an empty array uses 12 B.

5) The Java string contains the following [ChCo15, JaMe15]:
   a) A string object header 8 B.
   b) A char array—thus a separate object—containing zero or more actual characters
      i) Char array object header is 8 B.
      ii) Char array length is 4 B.
      iii) Char array value (depends on the number of characters, each char uses 2 B).
   c) A pointer (reference) to the char array 4 B.
d) An integer offset into the array at which the string starts 4 B.

e) An integer length of the string 4 B.

f) An integer for the cached value of the hash code 4 B.

g) Padding to bring the size of the string to a multiple of 8 B.

Therefore, an empty string uses up to 40 B. Table 5.2 illustrates Java string size calculation.

Table 5.2. Java String Size.

<table>
<thead>
<tr>
<th>Field</th>
<th>Header</th>
<th>Char array</th>
<th>Pointer</th>
<th>Offset</th>
<th>Length</th>
<th>Hash code</th>
<th>Padding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>Number of chars * 2</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Size [B]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Round up to multiple of 8</td>
<td></td>
</tr>
</tbody>
</table>

5.2.4.2.2 Payload Size of Oppnet Control Messages

The payload of each oppnet control message is represented as a string. To show how sizes of the payloads of oppnet control messages were calculated, consider as an example the admission notification message “Admitted into oppnet”. In his case the string contains 20 characters, so the char array requires 12 B plus 40 B for 20 chars (2B per char), that is 52 B. Since 52 is not a multiple of 8, we need to round up to the next multiple of 8, which is 56. Therefore, in total our text will use 56 B + 24 B (which is string header 8 B + pointer to char array 4 B + offset 4 B + length of the string 4 B + hash code 4 B) = 80 B.

Payloads sizes of oppnet control messages are summarized in Table 5.3.

---

12 We need to add two comments. First, we use a simplifying assumption that a single oppnet is in a given computing environment. In general an oppnet identifier will be needed in this and other messages (so any candidate helper can distinguish multiple oppnets contacting it). Second, we keep messages human-readable. If necessary to shorten the messages, computer-readable messages could be much more succinct.
5.2.4.2.3 Payload Size of Data Messages

This subsection describes how the payload size of data messages is calculated.

A. Help Object Messages

To calculate the size of a Help Object Message payload \((\text{helpObject})\), we consider the following:

1) Number of characters in the string variables is determined by the values used in our simulation.

2) The size of the empty object which is 8 B for the object header [JaMe15].

3) The size of the string used for the help text message \((\text{helpTextMsg})\) is calculated based on our calculations of string size in Subsection 5.2.4.2.1–Sizes of Used Data Types. We assume that the help text message includes 31 characters. The string object itself still requires 24 B. But now the char array requires 12 B of header plus 31 char * 2 B/char = 62 B for the 31 chars. Since 12 B + 62 B = 74 B is not a multiple of 8, we also need to round up to the next multiple of 8 (80). Therefore, in total \(\text{helpTextMsg}\) will use 80 B + 24 B = 104 B.

4) We assume that a person profile includes 137 characters. The size of the string used for the person profile \((\text{personProfile})\) is 312 B. (Calculations are analogous to those in Item 3 above.)

5) We assume that person contact includes 102 characters. The size of the string used for the person contacts \((\text{personContact})\) is 240 B. (Calculations are analogous to those in Item 3 above.)

6) We assume that person vitals include 114 characters. The size of the string used for the person vitals \((\text{personVitals})\) is 264 B. (Calculations are analogous to those in Item 3 above.)
7) Therefore, the total size of helpObject = 8 B for empty object header + 104 B for helpTextMsg + 312 B for personProfile + 240 B for personContact + 264 B for personVitals = 920 B. See No. 2 in Table 5.3 for Size of helpObject.

Table 5.3. Size of message payloads.

<table>
<thead>
<tr>
<th>No.</th>
<th>Message Name and Definition</th>
<th>Message Payload</th>
<th>Size of Message Payload[B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oppnet Control Message</td>
<td>Help request</td>
<td>128 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Join request</td>
<td>64 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Admit notification</td>
<td>80 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Release request</td>
<td>96 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Release notification</td>
<td>72 B</td>
</tr>
<tr>
<td>2</td>
<td>Data Message</td>
<td>Help Object Message</td>
<td>The help object (helpObject)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oppnet Task Message</td>
<td>The list of oppnet tasks (oppnetTasks)</td>
</tr>
</tbody>
</table>

B. Oppnet Task Messages

To calculate the size of an Oppnet Task Message payload (oppnetTasks), we consider the following:

1) The size of the oppnetTasks empty array is 12 B.

2) The total size of the strings in oppnetTasks is 544 B. (Based on our calculations of string size in Subsection 5.2.4.2.1. and number of characters in each task used in our simulation.

3) Therefore the total size of oppnetTasks = 12 B (for array size) + 544 B = 556 B.

4) See No. 2 in Table 5.3 for Size of oppnetTasks.

5.2.4.3 Message Payload Reading Time

Message that arrives includes both control fields and payload fields. We consider that all fields (not just payload fields) are read at the receiving node. Control fields would be read by OS, and payload fields by the application. Based on the values of control fields,
OS would deliver the payload to a proper application. However, message headers is done by a kernel (of the OS) and kernel execution is orders of execution faster, since kernel code responsible for sending/receiving messages is implemented in an assembly language (not in Java –as assumed for the application-level oppnet code). Therefore, in calculation of the message reading time we only consider the payload fields.

*Message payload reading time* is the execution time for reading the payload of an oppnet message read at the receiving node, we calculated the payload message reading time for oppnet control messages and for data messages (i.e., `helpObject` and `oppnetTasks`).

In general, we calculate *execution time* \( T \) for a given code [CyIn15] as follows:

\[
T = \frac{(CPI \times IC)}{PF} \quad (1)
\]

where \( CPI \) denotes *cycles per instruction*, \( IC \) is *instruction count* (the number of instructions being executed), and \( PF \) is *processor frequency*.

We use Formula 1 for calculating message content payload reading time. The following assumptions are made:

1) CPI = 1 (this assumption is based on the average CPI for benchmark programs [CyIn15]).
2) IC for reading oppnet message payload [instr.] =
   \[
   = (\text{size of message payload}^{13} [\text{B}]) \times (\text{IC per byte of transferred data} [\text{instr./B}])
   \]

For example, to calculate reading time for `helpObject` (data message payload) by a Tablet (Nexus 9, with a 2.3 GHz processor) the following calculations are performed:

3) IC for reading `helpObject` = (size of `helpObject` [B]) \times (IC per byte of transferred data [instr./B]) = 920 B \times 2 instr./B \text{ }^{14} = 1840 \text{ instr.}

4) \( T = (CPI \times IC) / PF = \)

---

\^{13} \text{ See Table 5-III for payload size for each message.}

\^{14} \text{ Reading 1 B of data requires 1 } LDRB \text{ instruction and 1 } STRB \text{ instruction. } LDRB \text{ is a data transfer instruction used to load a byte of data from memory to a CPU register for reading. } STRB \text{ is a data transfer instruction (following LDRB) to transfer a byte of data from a CPU register back to memory.}
\[
= \frac{(1 \text{ [instr.]} \times 1840 \text{ instr.})}{2.3 \text{ GHz}} = \frac{1840}{2.3 \times 10^9} [1/\text{s}]
\]
\[
= 0.8 \times 10^{-3} \text{ ms}
\]

### 5.2.5 Simulation Technology/Standard-Related Assumptions

Table 5.4 summarizes the message payload reading times for each of the used technologies.
Table 5.4. Message payload reading times for used technologies.

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter*</th>
<th>Standard/Technology</th>
<th>Exact Value [ms]</th>
<th>Approx. Value for Simulation [ms]</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Message Payload Reading Time for Regular Helpers</td>
<td>Tablet (Nexus 9, 2.3 GHz)</td>
<td>$0.8 \times 10^{-3}$</td>
<td>0</td>
<td>See Table Notes 1, 2, and 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Digital TV (Samsung 2013, 1.35 GHz)</td>
<td>$1.4 \times 10^{-3}$</td>
<td>0</td>
<td>See Table Notes 1, 2, and 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Smartphone (Nexus 5, 2.26 GHz)</td>
<td>$0.82 \times 10^{-3}$</td>
<td>0</td>
<td>See Table Notes 1, 2, and 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DB Server (HP ProLiant DL560 Gen8, 2.2 GHz)</td>
<td>$0.84 \times 10^{-3}$</td>
<td>0</td>
<td>See Table Notes 1, 2, and 3</td>
</tr>
<tr>
<td>2</td>
<td>Message Payload Reading Time for Lites</td>
<td>Printer (HP Officejet Pro 8630, 600 MHz)</td>
<td>$3.07 \times 10^{-3}$</td>
<td>0</td>
<td>See Table Notes 1, 2, and 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Embedded systems such as the ones in parking meters, street lights, sensors, etc.</td>
<td>29–43</td>
<td>29–43</td>
<td>Cf. Ref. [LMSL10]. See Table Note 3</td>
</tr>
<tr>
<td>3</td>
<td>Message Payload Reading Time for Seed</td>
<td>Embedded systems such as the ones in wristbands</td>
<td>29–43</td>
<td>29–43</td>
<td>Cf. Ref. [LMSL10]. See Table Note 3</td>
</tr>
</tbody>
</table>

* Parameters shown in Table 5.4 are not variables in the simulation, since they are not used directly in the simulation; they are instead used to obtain the values for parameters shown in Table 5.7 (listing parameters/variables of the simulation program).

Table Notes:

1) Calculated using Formula 1 in Subsection 5.2.4.3.

2) The message payload reading times are for reading helpObject, which is the largest message payload 920 B among all payloads. Since the values for reading helpObject are minuscule, we set zero as the values for time for reading oppnetTasks and the payloads of oppnet control messages, because they are even smaller.

3) Some parameters have a range of values while others have single values; the reasons for such a difference are:

   a) Some parameters (such as message payload reading time for embedded systems) were taken from references specified in the last section; such references indicate the values of the best case and worst case; hence the range of values.

   b) Other parameters—specifically message payload reading time for regular helpers and message payload reading time for printer—were calculated using Formula 1, which calculates the message reading time for processors specific to those devices, hence the single value.
5.2.5.1 Delivery Time

In our simulation, we consider *packet delivery time*—defined as the time period from the moment when the *first* bit of a packet leaves the *transmitter* (the transmitting node, the sender, the sending node) until the moment when the *last* bit of this packet is received by the *receiver* (the receiving node) [TrTi15]. We use the following formula to calculate packet delivery time [TrTi15]:

\[ \text{packet delivery time} = \text{packet transmission time} + \text{packet propagation time} \] (2)

*Packet transmission time* is defined as the time period from the moment when the *first* bit of the packet leaves the *transmitter* until the moment when the *last* bit of the packet leaves the *transmitter* [TrTi15]. We use the following formula to calculate packet transmission time [TrTi15]:

\[ \text{packet transmission time} = \frac{\text{packet size}}{\text{bit rate}} \] (3)

*Packet propagation time* is defined as the time period from the moment when the *last* bit of the message leaves the *transmitter* until the moment when the *last* bit of the message is received by the receiver [TrTi15]. We assume that a wireless communication link has the bandwidth large enough not to slow down transmission of the message bits, so that the propagation time for the bits is on the order of the speed of light (cf. Ref. [TrTi15]).
Table 5.5. Delivery parameters of used communication technologies.

<table>
<thead>
<tr>
<th>No.</th>
<th>Standard/Technology</th>
<th>Parameter*</th>
<th>Exact and Simulation Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ZigBee</td>
<td>Delivery time [ms] for oppnet control message</td>
<td>4.928</td>
<td>See No. 2 in Subsection 5.2.5.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for help object message containing helpObject</td>
<td>34.84</td>
<td>See No. 3 in Subsection 5.2.5.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for oppnet task message containing oppnetTasks</td>
<td>21.12</td>
<td>See No. 4 in Subsection 5.2.5.1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transmission Range [m]</td>
<td>100</td>
<td>Cf. Ref. [Zigb15]</td>
</tr>
<tr>
<td>2</td>
<td>BT</td>
<td>Delivery Time [ms] for oppnet control message</td>
<td>0.383</td>
<td>See No. 1 in Subsection 5.2.5.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for help object message containing helpObject</td>
<td>2.582</td>
<td>See No. 2 in Subsection 5.2.5.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for oppnet task message containing oppnetTasks</td>
<td>1.566</td>
<td>See No. 3 in Subsection 5.2.5.1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transmission Range [m]</td>
<td>100</td>
<td>Cf. Refs. [Blut15, BIWi15, BlSp15]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum Payload Size [B]</td>
<td>343</td>
<td>Cf. Ref. [Blut15c]</td>
</tr>
<tr>
<td>3</td>
<td>Wi-Fi</td>
<td>Delivery Time [ms] for oppnet control message</td>
<td>0.1312</td>
<td>See No. 1 in Subsection 5.2.5.1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for help object message containing helpObject</td>
<td>0.765</td>
<td>See No. 2 in Subsection 5.2.5.1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Delivery Time [ms] for oppnet task message containing oppnetTasks</td>
<td>0.4736</td>
<td>See No. 3 in Subsection 5.2.5.1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transmission Range [m]</td>
<td>∞</td>
<td>If a device has a Wi-Fi connection, it can communicate with other devices through the Internet, no matter how far they are. Hence the infinite transmission range.</td>
</tr>
<tr>
<td>4</td>
<td>Cellular</td>
<td>Delivery Time [ms] for SMS help message delivered to 9-1-1</td>
<td>4060 ms</td>
<td>Cf. Ref. [SaBB15]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transmission Range [m]</td>
<td>∞</td>
<td>If a device has a cellular connection, it can communicate with other devices through the cellular network, no matter how far they are. Hence the infinite transmission.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Maximum Payload Size [B]</td>
<td>140 8-bit characters (160 7-bit characters)</td>
<td>Cf. Ref. [ShMS15]</td>
</tr>
</tbody>
</table>

* Parameters shown in Table 5.5 are not variables in the simulation, since they are not used directly in the simulation; they are instead used to get the values of the parameters in Table 5.7 (listing parameters/variables of the simulation program).
5.2.5.1.1 ZigBee

To calculate delivery time for messages sent via ZigBee we make the following assumptions [KnBa15]:

1) ZigBee 802.15.4, 2.4 GHz Series 2 modules;
2) Data rate of 250 Kbps;
3) Maximum payload of 72 B;
4) Packet header of 13 B.

Based on 1–4 above and Formulas 2 and 3 in Subsection 5.2.5.1:

5) For a packet with max. payload size 72 B, ZigBee delivery time [ms] = 
   \[= \frac{(13 \text{ B header } + 72 \text{ B}) \times 8 \text{ b/B}}{(250 \text{ Kbps} \times 10^3)} \approx 2.720 \text{ ms}\]

6) To deliver the largest oppnet control message (i.e., help request) of size 128 B, we need 1 packet with maximum payload 72 B and 1 packet with 56 B payload. Therefore, ZigBee delivery time [ms] for the oppnet control message containing help request can be calculated as:

   \[(\text{no. packets with 72 B payload size [packet]} \times \text{delivery time per 72 B packet [ms/packet]}) + \]
   \[+ (\text{no. packets with 56 B payload size [packet]} \times \text{delivery time per 56 B packet [ms/packet]}) = \]
   \[= (1 \text{ packet } \times 2.720 \text{ ms/packet}) + (1 \text{ packet } \times 2.208 \text{ ms/packet}) \approx 4.928 \text{ ms}\]

7) To deliver a help object message containing helpObject of size 920 B, we need 12 packets with maximum payload 72 B and 1 packet with 56 B payload size. Therefore, ZigBee delivery time [ms] for the help object message containing helpObject can be calculated as:

   \[(\text{number of packets sent with payload 72 [packet]} \times \text{delivery time per packet with payload of size 72 B [ms/packet]}) + \]
   \[+ (\text{number of packets sent with payload of size 56 B [packet]}}]
× delivery time per packet with payload of size 56 B [ms/packet]) =
= (12 packet × 2.720 ms/packet) + (1 packet × 2.2 ms/per packet)
≈ 34.84 ms

8) Calculation of ZigBee delivery time for the oppnet task message containing oppnetTasks (of size 556 B; cf. Table 5.5) is analogous to Item 3; it is 21.12 ms.

Row No.1 in Table 5.5 summarizes ZigBee delivery time for 3 different oppnet messages.

5.2.5.1.2 BT

To calculate delivery time for messages sent via BT we make the following assumptions (Ref. [BlSp15]):

1) Bluetooth v.4.0 at 2.4 GHz;
2) Data rate of 3 Mbps;
3) Maximum payload of 343 B;
4) Access code of 72 b, and header of 54 b;

Based on Nos. 1–4 of this subsection and Formulas 2 and 3 in Subsection 5.2.5.1:

5) BT delivery time of the largest oppnet control message (i.e., help request) of size 128 B sent in a single packet =
= ((128 B × 8 b/B) + (72 b access code + 54 b header)) / (3 Mbps × 10^6 b/s)
≈ 0.383 ms

6) To deliver a help object message containing helpObject of size 920 B, we need 2 packets with maximum payload 343 B and 1 packet with 234 B payload. Therefore, the following calculations are needed:

a) BT delivery time per packet with payload of size 343 B [ms/packet]) =
= ((343 B × 8 b/B) + (72 b access code + 54 b header)) / (3 Mbps × 10^6 b/s)
≈ 0.956 ms

b) BT delivery time per packet with payload of size 234 B [ms/packet]) =
\[\begin{align*}
&= ((234 \text{ B} \times 8 \text{ b/B}) + (72 \text{ b access code} + 54 \text{ b header})) / (3 \text{ Mbps} \times 10^6 \text{ b/s}) \\
&\approx 0.67 \text{ ms}
\end{align*}\]

c) BT delivery time [ms] for a help object containing helpObject can be calculated as:

\[\begin{align*}
\text{(number of packets sent with payload of size 343 B [packet] \times delivery time per packet with payload of size 343 B [ms/packet])} \\
+ \text{(number of packets sent with payload of size 236 B [packet] \times delivery time per packet with payload of size 236 B [ms/packet])}
\end{align*}\]
\[= (2 \text{ packet} \times 0.956 \text{ ms/packet}) + (1 \text{ packet} \times 0.67 \text{ ms/packet})
\approx 2.582 \text{ ms}
\]

7) BT Delivery time for data message containing oppnetTasks is analogous to No. 2.

Row No.2 in Table 5.5 summarizes BT delivery time for oppnet messages.

5.2.5.1.3 Wi-Fi

To calculate delivery time for messages sent via Wi-Fi we make the following assumptions (cf. Refs. [TrTi15, Ieee15]):

1) Wireless Internet standards 802.11b and 802.11g, both at 2.4 GHz;

2) Data rate of 10 Mbps [TrTi15] (a conservative assumption);

3) Maximum payload of 1500 B;

4) MAC header of 32 B. Frame Check Sequence (FCS) of 4 B.

Based on Nos. 1–4 of this subsection and Formulas 2 and 3 in Subsection 5.2.5.1:

5) Wi-Fi delivery time of the largest oppnet control message (i.e., help request) of size 128 B sent in a single packet =

\[\begin{align*}
&= ((128 \text{ B} + 32 \text{ B MAC header} + 4 \text{ B FCS}) \times 8 \text{ b/B}) / (10 \text{ Mbps} \times 10^6 \text{ b/s}) \\
&\approx 0.1312 \text{ ms}
\end{align*}\]
6) Wi-Fi delivery time for a help object message containing `helpObject` of size 920 B sent in a single packet =
   
   \[= \frac{(920 \text{ B} + 32 \text{ B MAC header} + 4 \text{ B FCS}) \times 8 \text{ b/B}}{(10 \text{ Mbps} \times 10^6 \text{ b/s})}\]
   
   \[\approx 0.765 \text{ ms}\]

7) Wi-Fi Delivery time for an oppnet task message containing `oppnetTasks` is analogous to No. 2.

   Row No.3 in Table 5.5 summarizes Wi-Fi delivery time for oppnet messages.

### 5.2.5.1.4 Cellular

The following assumptions are made:

1) 3G Cellular is used;

2) Maximum payload is 140 8-bit characters (160 7-bit characters) [ShMS15].

   Cellular delivery time of a SMS containing `helpTextMsg` delivered to 9-1-1 is 4,060 ms [SaBB15]. See Row No. 4 in Table 5.5.

### 5.2.6 Simulation Task Assumptions/Setup

This subsection presents the task assumptions. Assumptions include the task name, the node executing this task and communication technologies used, the task running time, and a description of the task running time.

There are three types of tasks in this simulation: (i) *unhurried tasks*: do not need immediate execution and are not critical to the success of oppnet; (ii) *urgent tasks*: require immediate execution, and are necessary for the success of the oppnet; and (iii) a *goal task*: requires immediate execution, and its successful execution means that the oppnet succeeded.
### Table 5.6. Tasks and their running times.

<table>
<thead>
<tr>
<th>No.</th>
<th>Task Description</th>
<th>Node(s) executing this task and communication technologies used</th>
<th>Task Running Time [ms]</th>
<th>Task Running Time Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Task 1 (urgent): Create helpObject and oppnetTasks</td>
<td>Wristband (seed/CC)</td>
<td>29–43</td>
<td>Time to create helpObject and oppnetTasks. Cf. Ref. [LMSL10].</td>
</tr>
<tr>
<td>2</td>
<td>Task 2A (urgent): Send helpObject (stored locally in main memory) to a remote helper</td>
<td>Wristband (seed/CC) – BT</td>
<td>(29–43) + 2.582</td>
<td>Time to prepare helpObject for sending it to a helper plus the time to deliver helpObject to the helper, which is listed in Table 5.5. See Table Note 1.</td>
</tr>
<tr>
<td></td>
<td>Task 2B (urgent): Forward helpObject (stored locally in main memory) to a remote helper</td>
<td>DigitalTV, Table1 (regular helpers) – BT</td>
<td>0 + 2.582</td>
<td>Time to read the helpObject which is listed in Table 5.4 plus the time to deliver helpObject to the helper, which is listed in Table 5.5. See Table Note 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tablet2 (regular helper) – Wi-Fi</td>
<td>0 + 0.765</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParkingMeter, Printer, Streetlight (lites) – ZigBee</td>
<td>(29–43) + 34.84</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Task 3A (urgent): Send oppnetTasks (stored locally in main memory) to remote helpers</td>
<td>Wristband (seed/CC) – BT</td>
<td>(29–43) + 1.566</td>
<td>Time to prepare oppnetTasks for sending it to a helper plus the time to deliver oppnetTasks to the helper, which is listed in Table 5.5.</td>
</tr>
<tr>
<td></td>
<td>Task 3B (urgent): Forward oppnetTasks (stored locally in main memory) to remote helpers</td>
<td>DigitalTV, Table1 (regular helpers) – BT</td>
<td>0 + 1.566</td>
<td>Time to read oppnetTasks which is listed in Table 5.4 plus the time to deliver oppnetTasks to the helper, which is listed in Table 5.5.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tablet2 (regular helper) – Wi-Fi</td>
<td>0 + 0.4736</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParkingMeter, Printer, Streetlight (lites) – ZigBee</td>
<td>(29–43) + 21.12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Task 4 (unhurried): Display on a local monitor the text help message stored locally in the helpObject in main memory</td>
<td>DigitalTV, Smartphone, Table1, and Table2. (regular helpers), their local monitor screen</td>
<td>5,000–11,000</td>
<td>Time to display a text message helpTextMsg stored locally in the helpObject in memory. (e.g., on a tablet screen.) Cf. Ref. [YCGK10].</td>
</tr>
</tbody>
</table>
Table 5.6 – continued

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Device</th>
<th>Time</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Task 5 (unhurried): Print on a local printer the help text message stored locally in the helpObject in main memory</td>
<td>Printer (a lite helper), local</td>
<td>12,000–13,000</td>
<td>Time to print the help message stored locally in the helpObject in memory. Cf. Ref. [HPSt15].</td>
</tr>
<tr>
<td>6</td>
<td>Task 6 (unhurried): Upload helpObject to the remote healthcare database server</td>
<td>Tablet2 (a regular helper)--Wi-Fi</td>
<td>0 + 0.765</td>
<td>Time to prepare an upload of the helpObject to the remote healthcare DB server plus the time to deliver the helpObject to the server, which is mentioned in No. 3 of Table 5.5. See Table Note 1. Cf. Ref. [YCGK10].</td>
</tr>
<tr>
<td>7</td>
<td>Task 7 (unhurried): Read/process the logging request and log the helpObject to the database server</td>
<td>DBserver, (regular helper)</td>
<td>10–30</td>
<td>Time to Read/process the logging request and log the helpObject to the database server. Cf. Ref. [KwMo06].</td>
</tr>
<tr>
<td>8</td>
<td>Task 8 (urgent and goal): Deliver helpTextMsg to the (remote) 9-1-1 service via SMS (this is the final step of the scenario, realizing its goal). Done by regular helper Smartphone</td>
<td>Smartphone (regular helper)--Cellular</td>
<td>0 + 4,060</td>
<td>Time to prepare an SMS that contains the helpTextMsg plus the time to deliver the SMS to 9-1-1 which is mentioned in No. 4 of Table 5.5. See Table Note 1. Cf. Refs. [OxBl14, SaBB15].</td>
</tr>
</tbody>
</table>

Table Notes:
1) See Subsection 5.2.5.1–Delivery Time.
### 5.2.7 Simulation Input Parameters

Table 5.7 describes input parameters and their value ranges.

Table 5.7. Input parameters.

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter</th>
<th>Value Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Input Parameters for the Simulation Area</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Minimum and Maximum X/Y Coordinate (0\text{–areaMaxX/areaMaxY})</td>
<td>(0\text{ m – 600 m})</td>
<td>Minimum and maximum values for the X/Y coordinate of the simulation area.</td>
</tr>
<tr>
<td></td>
<td><strong>Input Parameters for the Node Set</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Minimum and Maximum Number of Candidate Helpers Requested for Help</td>
<td>(1 – k) (where\ k\ is the number of candidate helpers within the range of the requesting oppnet node)</td>
<td>There are three modes of requesting candidate helpers for help:   1) Single-helper mode.  2) Many-helper mode.  3) All-helper mode.</td>
</tr>
<tr>
<td></td>
<td>(\text{requestedHelpersMin–requestedHelpersMax})</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Input Parameters for Individual Nodes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Approximate Message Payload Reading Time for Regular Helpers</td>
<td>(0\text{ ms})</td>
<td>Cf. No. 1 in Table 5.4</td>
</tr>
<tr>
<td></td>
<td>(\text{approxMsgPayloadReadTimeRegHlpr})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Range of Values for Message Payload Reading Time for Lites</td>
<td>(29\text{ ms–43 ms})</td>
<td>Cf. No. 2 in Table 5.4</td>
</tr>
<tr>
<td></td>
<td>(\text{msgPayloadReadTimeLiteMin–msgPayloadReadTimeLiteMax})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Range of Values for Message Payload Reading Time for Seed Nodes</td>
<td>(29\text{ ms–43 ms})</td>
<td>Cf. No. 3 in Table 5.4</td>
</tr>
<tr>
<td></td>
<td>(\text{msgPayloadReadTimeSeedMin–msgPayloadReadTimeSeedMax})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Range of Values for Task 1 Runtime (\text{task1RuntimeMin–task1RuntimeMax})</td>
<td>(29\text{ ms–43 ms})</td>
<td>Cf. No. 1 in Table 5.6</td>
</tr>
<tr>
<td>7</td>
<td>Range of Values for Task 2 Runtime for Seed using BT</td>
<td>((29\text{ ms–43 ms}) + 2.582\text{ ms})</td>
<td>Cf. No. 2 in Table 5.6</td>
</tr>
<tr>
<td></td>
<td>(\text{task2RuntimeSeedBTMin–task2RuntimeSeedBTMax})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Range of Values for Task 3 Runtime for Seed Using BT</td>
<td>((29\text{ ms–43 ms}) + 1.566\text{ ms})</td>
<td>Cf. No. 3 in Table 5.6</td>
</tr>
<tr>
<td></td>
<td>(\text{task3RuntimeSeedBTMin–task3RuntimeSeedBTMax})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Range of Values for Task 4 Runtime (\text{task4RuntimeMin–task4RuntimeMax})</td>
<td>(5,000\text{ ms–11,000 ms})</td>
<td>Cf. No. 4 in Table 5.6</td>
</tr>
<tr>
<td>10</td>
<td>Range of Values for Task 5 Runtime. (\text{task5RuntimeMin–task5RuntimeMax})</td>
<td>(12,000\text{ ms–13,000 ms})</td>
<td>Cf. No. 5 in Table 5.6</td>
</tr>
</tbody>
</table>
Table 5.7 – continued

<table>
<thead>
<tr>
<th></th>
<th>Approximate Task 6 Runtime (approxTask6Runtime)</th>
<th>0 + 0.765 ms</th>
<th>Cf. No. 6 in Table 5.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Range of Values for Task 7 Runtime. (task7RuntimeMin–task7RuntimeMax)</td>
<td>10 ms–30 ms</td>
<td>Cf. No. 7 in Table 5.6</td>
</tr>
<tr>
<td>13</td>
<td>Approximate Task 8 Runtime (approxTask8RunTime)</td>
<td>0 + 4.060 ms</td>
<td>Cf. No. 8 in Table 5.6</td>
</tr>
<tr>
<td>14</td>
<td>Range of Values for Workload Ratio for a Candidate Helper (helperWorkloadRatioMin–helperWorkloadRatioMax)</td>
<td>0%–100%</td>
<td>The range of workload ratios to be simulated.</td>
</tr>
</tbody>
</table>

**Input Parameters for Links**

<table>
<thead>
<tr>
<th></th>
<th>Actual ZigBee Delivery Time for an Oppnet Control Message (actualZigBeeDelTimeCtrlMsg)</th>
<th>4.928</th>
<th>Cf. No. 1 in Table 5.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Actual ZigBee Delivery Time of a Help Object Message Containing helpObject (actualZigBeeDelTimeHelpObjectMsg)</td>
<td>34.84</td>
<td>Cf. No. 1 in Table 5.5</td>
</tr>
<tr>
<td>17</td>
<td>Actual ZigBee Delivery Time of an Oppnet Task Message Containing oppnetTasks (actualZigBeeDelTimeOppnetTaskMsg)</td>
<td>21.12</td>
<td>Cf. No. 1 in Table 5.5</td>
</tr>
<tr>
<td>18</td>
<td>Actual BT Delivery Time of an Oppnet Control Message (actualBDelTimeCtrlMsg)</td>
<td>0.383</td>
<td>Cf. No. 2 in Table 5.5</td>
</tr>
<tr>
<td>19</td>
<td>Actual BT Delivery Time of a Help Object Message Containing helpObject (actualBDelTimeHelpObjectMsg)</td>
<td>2.582</td>
<td>Cf. No. 2 in Table 5.5</td>
</tr>
<tr>
<td>20</td>
<td>Actual BT Delivery Time of an Oppnet Task Message Containing oppnetTasks (actualBDelTimeOppnetTaskMsg)</td>
<td>1.566</td>
<td>Cf. No. 2 in Table 5.5</td>
</tr>
<tr>
<td>21</td>
<td>Actual Wi-Fi Delivery Time of an Oppnet Control Message (actualBDelTimeCtrlMsg)</td>
<td>0.0164</td>
<td>Cf. No. 3 in Table 5.5</td>
</tr>
<tr>
<td>22</td>
<td>Actual Wi-Fi Delivery Time of a Help Object Message Containing helpObject (actualWiFiDelTimeHelpObjectMsg)</td>
<td>0.7696</td>
<td>Cf. No. 3 in Table 5.5</td>
</tr>
<tr>
<td>23</td>
<td>Actual Wi-Fi Delivery Time of an Oppnet Task Message Containing oppnetTasks (actualWiFiDelTimeOppnetTaskMsg)</td>
<td>0.4736</td>
<td>Cf. No. 3 in Table 5.5</td>
</tr>
<tr>
<td>24</td>
<td>Actual Cellular Delivery Time of an SMS Containing helpTextMsg (actualCellDelTimeHelpTextMsg)</td>
<td>4060 ms</td>
<td>Cf. No. 4 in Table 5.5</td>
</tr>
</tbody>
</table>
Table 5.8. Other simulation parameters.

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Iterations (iterationsNum)</td>
<td>30(^{15})</td>
<td>Number of simulation iterations</td>
</tr>
</tbody>
</table>

Notes for Section 5.2.7:

1) Some parameters have a range of values while others have single values; the reasons for such a difference are as follows:

a) The min and max range of values are the parameters in which the random variables in Table 5.9 generate their values from using the uniform distribution.

b) Min and max range of values were chosen based on values of Table 5.4, Table 5.5, and Table 5.6. More information is specified in the Description column of Table 5.7.

\(^{15}\) Most statisticians agree that the minimum sample size for obtaining valid results is 30, cf. Refs. [Bart08, BeBT03].
5.2.8 Random Variables for the Simulation

In this subsection we discuss random variables for the simulation. The *Description* column of Table 5.9 explains how a random number between minimum and maximum is generated according to the indicated distribution.

Table 5.9. Random variables for the simulation.

<table>
<thead>
<tr>
<th>No.</th>
<th>Random Variable</th>
<th>Value Range</th>
<th>Distribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Number of Helpers (numOfHelpers)</td>
<td>20 – 60</td>
<td>Uniform</td>
<td>Number of helpers within the simulation area. See Table Note 1.</td>
</tr>
<tr>
<td>2</td>
<td>Node Location (nodeLocation = &lt;Xvalue, Yvalue&gt;)</td>
<td>0 – areaMaxX/areaMaxY</td>
<td>Uniform</td>
<td>The position of each node within the simulation area (cf. No. 1 in Table 5.7).</td>
</tr>
<tr>
<td>3</td>
<td>Helper Type (helperType)</td>
<td>1 – 8</td>
<td>Uniform</td>
<td>The type of each helper node within the simulation area (cf. Table 5.1).</td>
</tr>
<tr>
<td>4</td>
<td>Helper Workload Ratio (HelperWorkloadRatio)</td>
<td>helperWorkloadRatioMin – helperWorkloadRatioMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 14 in Table 5.7).</td>
</tr>
<tr>
<td>5</td>
<td>Message Payload Reading Time for Lites (MsgPayloadReadTimeLite)</td>
<td>msgPayloadReadTimeLiteMin – msgPayloadReadTimeLiteMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 4 in Table 5.7).</td>
</tr>
<tr>
<td>6</td>
<td>Message Payload Reading Time for Seed Nodes (MsgPayloadReadTimeSeed)</td>
<td>msgPayloadReadTimeSeedMin – msgPayloadReadTimeSeedMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 5 in Table 5.7).</td>
</tr>
<tr>
<td>7</td>
<td>Task1 Runtime (Task1Runtime)</td>
<td>task1RuntimeMin – task1RuntimeMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 6 in Table 5.7).</td>
</tr>
<tr>
<td>8</td>
<td>Task2 Runtime for Seed using BT (Task2RuntimeSeedBT)</td>
<td>task2RuntimeSeedBTMin – task2RuntimeSeedBTMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 7 in Table 5.7).</td>
</tr>
<tr>
<td>9</td>
<td>Task3 Runtime for Seed Using BT (Task3RuntimeSeedBT)</td>
<td>task3RuntimeSeedBTMin – task3RuntimeSeedBTMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 8 in Table 5.7).</td>
</tr>
</tbody>
</table>
Table 5.9 – continued

<table>
<thead>
<tr>
<th></th>
<th>Task 4 Runtime (Task4Runtime)</th>
<th>task4RuntimeMin - task4RuntimeMax</th>
<th>Uniform</th>
<th>A random number is generated from the value range (cf. No. 9 in Table 5.7).</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Task 5 Runtime (Task5Runtime)</td>
<td>task5RuntimeMin - task5RuntimeMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 10 in Table 5.7).</td>
</tr>
<tr>
<td>14</td>
<td>Task 7 Runtime (Task7Runtime)</td>
<td>task7RuntimeMin - task7RuntimeMax</td>
<td>Uniform</td>
<td>A random number is generated from the value range (cf. No. 12 in Table 5.7).</td>
</tr>
</tbody>
</table>

**Table Notes:**

1) Value range is chosen arbitrarily. However, the reason for the high maximum range is the possible density of helpers, especially the helpers available in skyscrapers in the chosen simulation area (i.e., Chicago).

**Notes for Section 5.2.8:**

1) Notice that the runtime for some tasks such as Task 6 Runtime (Task6Runtime) and Task 8 Runtime (Task8Runtime) are not considered random variables but are considered (constant) input parameters of the simulation, cf. Table 5.7. The reason for that is the single fixed value of these tasks’ runtimes.

2) As specified in the Description column of Table 5.9, a random number is generated between minimum and maximum according to the uniform distribution.
5.2.9 Simulation Measures

Table 5.10 describes measures used in the simulation, divided into three categories: time, resource usage and success rate.

Table 5.10. Simulation measures.

<table>
<thead>
<tr>
<th>No.</th>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Time Measures</strong></td>
<td></td>
</tr>
<tr>
<td>1A</td>
<td>Average Helper Integration Time Per Iteration for All InvitingNodes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(avgHlprIntegTimePerIterAllNodes)</td>
<td>We calculate this measure in the following steps (the last step shows the final result):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1) Summation of all helpers integration times per inviting node per iteration, which is total time for an oppnet node to discover one or more candidate helpers and integrate them into oppnet. For example, suppose that Node N1 integrated three helpers H1, H2, and H3. Integration time for H1 is 4; integration time for H2 is 5 and Integration time for H3 is 12; therefore, sumHlprIntegTimePerNodePerIter = (4 + 5 + 12) = 21. sumHlprIntegTimePerNodePerIter(inv, iter) = [integrationTimeForHelper(1, inv, iter) + ... + integrationTimeForHelper(nrHlp, inv, iter)] where inv identifies the inviting node, iter is iteration number, and nrHlp is the total number of integrated helpers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Average time for all inviting nodes to discover candidate helpers and integrate them into oppnet, per simulation iteration avgHlprIntegTimePerIterAllNodes(iter) = [sumHlprIntegTimePerNodePerIter(1, iter) + ... + sumHlprIntegTimePerNodePerIter(nrInv, iter)] / nrAllHelprs(iter) where nrInv is the number of inviting nodes, iter is iteration number, nrAllHelprs is number of all helpers integrated in an iteration.</td>
</tr>
<tr>
<td>1B</td>
<td>Average Helper Integration Time for All Iterations and for All InvitingNodes (avgHlprIntegTimeAll)</td>
<td>Average value of avgHlprIntegTimePerIterAllNodes for all successful simulation iterations avgHlprIntegTimeAll = [avgHlprIntegTimePerIterAllNodes(1) + ... + avgHlprIntegTimePerIterAllNodes(ssi)] / ssi where ssi is the number of successful simulation iterations.</td>
</tr>
</tbody>
</table>
Table 5.10 – continued

| 2A | Average Task Runtime for All Node Per Iteration (avgTaskRuntimeAllNodePerIter) | We calculate this measure in the following steps (the last step shows the final result):
|    |                                                                            |
|    | 1) Summation of all tasks runtimes for all tasks done by a node per iteration |
|    | sumTaskRuntimePerNodePerIter(nd, iter) = |
|    | [(taskRuntime( 1, nd, iter) + ... + |
|    | + [(taskRuntime(nrTasks, nd, iter)) |
|    | where nd identifies the node, iter is iteration number, and |
|    | nrTasks is the total number of tasks done by nd. |
|    | 2) Average task runtime for all nodes performing tasks |
|    | avgTaskRuntimeAllNodePerIter(iter) = |
|    | [sumTaskRuntimePerNodePerIter(nd, iter) + ... + |
|    | sumTaskRuntimePerNodePerIter(nrNd, iter)] / |
|    | nrAllTasks(iter) |
|    | where nrNd is the number of nodes performing tasks, iter is |
|    | iteration number, nrAllTasks is number of all tasks done in an |
|    | iteration. |

| 2B | Average Task Runtime for All Iterations and for All Nodes (avgTaskRuntimeAll) | Average value of avgTaskRuntimeAllNodePerIter for all successful simulation iterations |
|    |                                                                            |
|    | avgTaskRuntimeAll = |
|    | [avgTaskRuntimeAllNodePerIter (1) + ... + |
|    | avgTaskRuntimeAllNodePerIter (ssi)] / ssi |
|    | where ssi is the number of successful simulation iterations. |

| 3A | Average Time for Oppnet to Succeed Per Iterations (OppnetTimetoSucceedPerIter) | Time from the moment when the seed initiates oppnet activities till the moment its goal is realized. |
|    |                                                                            |
|    | For example, in this scenario, it is time from the moment when WristBand initiates oppnet till the moment when Smartphone delivers helpTextMsg to 9-1-1. See Table Note 1. |

<p>| 3B | Average Time for Oppnet to Succeed for All Iterations (avgOppnetTimetoSucceedAll) | avgOppnetTimetoSucceedAll = |
|    |                                                                            |
|    | [OppnetTimetoSucceedPerIter(1) + ... + |
|    | OppnetTimetoSucceedPerIter(ssi)] / ssi |
|    | where ssi is the number of successful simulation iterations. |</p>
<table>
<thead>
<tr>
<th></th>
<th>Resource Usage Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>4A</strong></td>
<td>Average Number of Helpers Requested for Help Per Iteration (avgHelperRequestedPerIter)</td>
</tr>
</tbody>
</table>
|     | \[
|     | \text{avgHelperRequestedPerIter}(\text{iter}) = \frac{[\text{numberHelpersReqByNode}(1, \text{iter}) + \ldots + \text{numberHelpersReqByNode}(\text{nrInv}, \text{iter})]}{\text{nrInv}(\text{iter})} \]
|     | where the first argument of \text{numberHelpersReqByNode} identifies a node, and \text{nrInv} is the number of inviting nodes, \text{iter} is iteration number. |
| **4B** | Average Number of Helpers Requested for Help for All Iterations (avgHelperRequestedAll) |
|     | \[
|     | \text{avgHelperRequestedAll} = \frac{[\text{avgHelperRequestedPerIter}(1) + \ldots + \text{avgHelperRequestedPerIter}(\text{ssi})]}{\text{ssi}} \]
|     | where \text{ssi} is the number of successful simulation iterations. |
| **5A** | Average Number of Helpers That Agreed to Join Oppnet Per Iteration (avgJoinedHelpersPerIter) |
|     | \[
|     | \text{avgJoinedHelpersPerIter}(\text{iter}) = \frac{[\text{number of helpers accepted to join oppnet by node}(1, \text{iter}) + \ldots + \text{number of helpers accepted to join oppnet by node}(\text{nrInv}, \text{iter})]}{\text{nrInv}(\text{iter})} \]
|     | where \text{nrInv} is the number of inviting nodes, \text{iter} is iteration number. |
| **5B** | Average Number of Helpers That Agreed to Join Oppnet for All Iterations (avgJoinedHelpersAll) |
|     | \[
|     | \text{avgJoinedHelpersAll} = \frac{[\text{avgJoinedHelpersPerIter}(1) + \ldots + \text{avgJoinedHelpersPerIter}(\text{ssi})]}{\text{ssi}} \]
|     | where \text{ssi} is the number of successful simulation iterations. |
| **6A** | Average Number of Helpers That Refused to help Oppnet Per Iteration (avgRefusedHelpersPerIter) |
|     | \[
|     | \text{avgRefusedHelpersPerIter}(\text{iter}) = \frac{[\text{number of helpers refused to join oppnet by node}(1, \text{iter}) + \ldots + \text{number of helpers refused to join oppnet by node}(\text{nrInv}, \text{iter})]}{\text{nrInv}(\text{iter})} \]
|     | where \text{nrInv} is the number of inviting nodes, \text{iter} is iteration number. |
| **6B** | Average Number of Helpers Refusing to help Oppnet for All Iterations (avgRefusedHelpersAll) |
|     | \[
|     | \text{avgRefusedHelpersAll} = \frac{[\text{avgRefusedHelpersPerIter}(1) + \ldots + \text{avgRefusedHelpersPerIter}(\text{ssi})]}{\text{ssi}} \]
|     | where \text{ssi} is the number of successful simulation iterations, \text{iter} is iteration number. |
| **7A** | Average Number of Helpers Admitted into Oppnet Per Iteration (avgAdmittedHelpersPerIter) |
|     | \[
|     | \text{avgAdmittedHelpersPerIter}(\text{iter}) = \frac{[\text{number of helpers admitted to oppnet by node}(1, \text{iter}) + \ldots + \text{number of helpers admitted to oppnet by node}(\text{nrInv}, \text{iter})]}{\text{nrInv}(\text{iter})} \]
|     | where \text{nrInv} is the number of inviting nodes, \text{iter} is iteration number. |
Table 5.10 – continued

<table>
<thead>
<tr>
<th>7B</th>
<th>Average Number of Helpers Admitted into Oppnet for All Iterations (avgAdmittedHelpersAll)</th>
</tr>
</thead>
</table>
|    | \[
|    | \text{avgAdmittedHelpersAll} = \\
|    | \left[ \text{avgAdmittedHelpersPerIter(1)} + \ldots + \\
|    | \text{avgAdmittedHelpersPerIter(ssi)} \right] / \text{ssi} \\
|    | \text{where ssi is the number of successful simulation iterations.} \\
|    | \] |

<table>
<thead>
<tr>
<th>8</th>
<th>Average Number of Helpers That Performed Urgent Tasks for All Iterations (avgHelperUrgentTaskAll)</th>
</tr>
</thead>
</table>
|    | We calculate this measure in the following steps (the last step shows the final result): \\
|    | 1) Summation of all helpers performing urgent tasks per iteration \\
|    | \text{sumHelperUrgentTaskPerIter(iter)} = \\
|    | \left[ \text{number of helpers performing urgent tasks requested by} \\
|    | \text{node(1, iter)} + \ldots + \text{number of helpers performing urgent} \\
|    | \text{tasks requested by node (nrInv, iter)} \right] \\
|    | \text{where nrInv is the number of inviting nodes, iter is iteration} \\
|    | \text{number.} \\
|    | 2) Average of all helpers performing urgent tasks for all iterations \\
|    | \text{avgHelperUrgentTaskAll} = \\
|    | \left[ \text{sumHelperUrgentTaskPerIter(1)} + \ldots + \\
|    | \text{sumHelperUrgentTaskPerIter(ssi)} \right] / \text{ssi} \\
|    | \text{where ssi is the number of successful simulation iterations.} |

<table>
<thead>
<tr>
<th>9</th>
<th>Average Number of Helpers That Performed Unhurried Tasks for All Iterations (avgHelperUnhurriedTaskAll)</th>
</tr>
</thead>
</table>
|    | We calculate this measure in the following steps (the last step shows the final result): \\
|    | 1) Summation of all helpers performing unhurried tasks per iteration \\
|    | \text{sumHelperUnhurriedTaskPerIter(iter)} = \\
|    | \left[ \text{number of helpers performing unhurried tasks requested by} \\
|    | \text{node(1, iter)} + \ldots + \text{number of helpers performing} \\
|    | \text{unhurried tasks requested by node (nrInv, iter)} \right] \\
|    | \text{where nrInv is the number of inviting nodes, iter is iteration} \\
|    | \text{number.} \\
|    | 2) Average of all helpers performing unhurried tasks for all iterations \\
|    | \text{avgHelperUnhurriedTaskAll} = \\
|    | \left[ \text{sumHelperUnhurriedTaskPerIter(1)} + \ldots + \\
|    | \text{sumHelperUnhurriedTaskPerIter(ssi)} \right] / \text{ssi} \\
|    | \text{where ssi is the number of successful simulation iterations.} |

<table>
<thead>
<tr>
<th>10</th>
<th>Success Rate for Achieving Oppnet Goal (successRate)</th>
</tr>
</thead>
</table>
|    | \text{successRate} = \text{ssi/tsi} \\
|    | \text{where ssi is the number of successful simulation iterations} \\
|    | \text{and tsi is the total number of simulation iterations.} |

Table Notes:

1) A simulation iteration is successful when the oppnet goal is achieved, which in this case is delivering helpObject to 9-1-1 via SMS.

2) Oppnet may fail to succeed in achieving its goal for the following reasons:
   a) The number of helpers is not sufficient (i.e., too few and/or scattered in "unreachable" locations within the simulation area), which causes interruptions in receiving or forwarding the message or a scarce of applications and resources to achieve the task.
   b) The number of helpers is sufficient but the helpers are grouped too close to each other but either far from the seed, or another relay node, which causes the message to be interrupted.
5.2.10 Simulation Code

The source code for the simulations described in this chapter is available online, as specified in Appendix B.

5.3 Simulation Results and Their Discussions

As discussed at the beginning of the chapter, our goal is to compare the monolithic oppnet middleware (which does not use OVM primitives) and the modular (OVM-based, non-monolithic) middleware that uses the third-generation OVM primitives. These two implementations of middleware were simulated in SimJava using the healthcare scenario described in Chapter 4.

Both middleware implementations (i.e., monolithic and modular) are considered in the single-helper mode and the all-helper mode (as discussed in Subsection 5.2.1). Therefore, in this section, the simulation results for four different categories of implementation are analyzed and compared:

1) **Monolithic single-helper mode (MSM).**
2) **Modular (OVM-based) single-helper mode (OSM)\(^{16}\).**
3) **Monolithic all-helper mode (MAM).**
4) **Modular (OVM-based) all-helper mode (OAM).**

The same set of simulation measures is used for each implementation category, so that we can compare them. We analyze the effectiveness of our modular oppnet middleware (both single-helper mode and all-helper mode) against the monolithic oppnet middleware (both single-helper and all-helper mode). We also compare the single-helper mode (both monolithic and modular) with the all-helper mode.

The results in figures in this section show average values (colored bars) and standard deviation values (black lines).

\(^{16}\) To avoid conflict of acronyms, secondary name of the implementation is used in the acronym; hence “O” rather than “M” in the acronym.
The graphs in this section depict only the successful iterations for each of the implementations (18 successful iterations for MSM and OSM, and 22 successful iterations for MAM and OAM). The reason for the difference in the number of successful iterations: each inviting node in the all-helper mode can admit all the candidate helpers in its range, while in the single-helper mode only one helper is admitted by an inviting node, which decreases the chances for an oppnet to succeed.

The graphs presented here depict the measures for all successful simulation iterations. Each one of the graphs presented here corresponds to a graph in Appendix C that depicts the same measure but per successful simulation iteration. For example, Fig. 5.1 depicts the average helper integration time for all inviting nodes for all successful iterations. Fig.5.1 corresponds to Fig. C.1 which displays the average helper integration time for all inviting nodes per successful iteration.

5.3.1 Average Helper Integration Time for All Inviting Nodes for All Iterations

Fig. 5.1 depicts the average helper integration time for all inviting nodes for all iterations for each of the four implementations listed above.
The graph first shows the average helper integration time for MSM and OSM implementations. MSM is slightly slower, by 1%, in integrating helpers than OSM. This is surprising since we expected MSM to be faster—due to the overhead introduced by module interfaces in OSM. We believe that the difference between the expected slightly better MSM performance and the actual slightly better OSM behavior might be within the range of statistical accuracy. The ranges of standard deviation seem to support this belief.

The average helper integration time for OAM is slightly higher (only by 1.6%) than for MAM. This is expected, considering the overhead of using the OVM primitives in implementing OAM.

We observe that the average helper integration times for the all-helper mode (for both MAM and OAM) are higher (by over 180%) than the average helper integration times for the single-helper mode (for both MSM and OSM). The reason: each inviting node in MAM and OAM invites all helpers in its range; some of the helpers have slow message reading times which contribute to the higher average integration time.

Figure 5.1. Average helper integration time for all inviting nodes for all iterations (the standard deviation indicated as well).
The ranges of standard deviations for MAM and OAM are high; the reason is that each inviting node in the all-helper mode invites all helpers in its range, whether slow (such as lites, which take a longer time to integrate into the oppnet), or fast (such as regular helpers which integrate faster into the oppnet).

5.3.2 Average Task Runtime for All Iterations

The average task runtime by all helpers for all iterations for our four simulation implementations is displayed in Fig. 5.2.

MSM falls behind OSM on the average task runtime by 1.8%, which is within the statistical error range.

The average task runtime for OAM is slightly slower (by 0.27%) than for MAM. which is a small overhead considering the use of all the OVM-primitives in implementing the modular oppnet middleware.

![Figure 5.2. Average task runtime for all nodes for all iterations.](image-url)
We also observe that the average task runtime for the single-helper modes (both MSM and OSM) is lower (by over 21.4% than the average task runtime for the all-helper modes (both MAM and OAM). We believe that this could be the effect of having higher chances of using help from slow helpers incl. lites) for MAM and OAM.

5.3.3 Average Time for Oppnet to Succeed for All Iterations

Fig. 5.3 shows the average time for oppnet to succeed, which is the average time from the start of the oppnet until the oppnet succeeds in achieving its goal (i.e., delivering the help text message to 9-1-1).

MSM and OSM have the same average time to succeed. The average times to succeed for MAM and OAM are nearly identical (MAM falls behind OAM by barely 0.01%).

The time for oppnet to succeed in the single-helper mode (both MSM and OSM) is higher than in the all-helper mode (both MAM and OAM) by 19.1%. The reason is that in the all-helper mode the inviting node requests help from all helpers in its range, which increases the chance of finding a helper that can do the most urgent task. On the other hand, in the single-helper mode, each inviting node requests help from only a single helper; if that helper is unable to achieve the goal, it forwards the message to another helper, until a helper capable of doing the most urgent task is found. Such a sequence of helpers requesting help increases the time for oppnet to succeed.
Figure 5.3. Average time for oppnet to succeed for all iterations.

The standard deviation values in all implementations are high (higher than the mean in MSM and OSM) which results in a lower minimum value for the standard deviation line. For example, minimum values of the standard deviations for MSM and OSM go slightly below zero. The reason for that: in the single-helper mode, each inviting node requests help from only a single helper; in some iterations, the inviting node can find a helper that can achieve the goal, which results in a short time to succeed. However, in other iterations, the inviting node will find only a helper that is unable to achieve the goal; therefore, the helper will forward the message to another helper, until a helper capable of doing the most urgent task is found. Such a sequence of helpers requesting help increases the time for oppnet to succeed.

When we first generated the results of simulation for our four oppnet implementations, the time for oppnet to succeed in the in the monolithic middleware (both MSM and MAM) was faster (by around 99%) than the time for the modular middleware (both OSM and OAM). Investigating this, we discovered that when NODE_selectTask and NODE_runApp are implementing separately, they cause a large overhead. First, in the NODE_selectTask the helper has to loop through all the oppnet tasks to select the task(s) it can perform. Then, the tasks the helper can perform are
passed to the NODE_runApp, where another loop goes through all the tasks the helper can perform, in order to choose the right application(s) to run the tasks. To avoid such overhead, we called Node_runApp within Node_selectTask, which requires only a single loop to select the task the helper can perform and to immediately run the application executing that task. With the change, the overhead was minimized.

Now, MSM takes the same time to succeed as OSM, and MAM (8,305.2) is negligibly slower)—within the statistical error range—than OAM (8,304.4).

5.3.4 Average Number of Helpers Requested for Help by All Inviting Nodes for All Iterations

Fig. 5.4 depicts the average number of helpers requested for help by all inviting nodes for all iterations for our four oppnet implementations.

![Bar chart showing average number of helpers requested for help by all inviting nodes for all iterations.](image)

- Monolithic single-helper mode (MSM)
- OVM-based single-helper mode (OSM)
- Monolithic all-helper mode (MAM)
- OVM-based all-helper mode (OAM)

Figure 5.4. Average number of helpers requested for help by all inviting nodes for all iterations.

The graph shows first the average number of helpers requested for help for MSM and OSM. Both implementations use deterministically a single helper, so there is no additional overhead for OSM.
The average number of helpers requested for help for OAM is slightly higher (only by 3.4%) than for MAM. The reason for that is that OAM takes a longer time to integrate helpers than MAM (see Subsection 5.3.1). During that time, the inviting node keeps requesting more helpers for help, since it has not received an invitation acceptance from a helper yet. However, this overhead (i.e., the 3.4%) is small considering the use of all the OVM-primitives in implementing the modular oppnet middleware.

We observe that the average number of helpers requested for help in the single-helper modes (both MSM and OSM) is lower (around 66.6%) than in the all-helper modes (both MAM and OAM). The reason for this is that in the single-helper mode only one helper is requested for help per inviting node, while in the all-helper mode each inviting node can request for help all the candidate helpers in its range.

Notice that the range of the standard deviations is very high in MAM and OAM which means that the variance in the number of requested helpers among nodes is high. The reason for that: due to the large simulation area (600m x 600 m) and the high range for number of helpers (20–60 helpers, cf. Table 5.9), the helpers can be scattered through the simulation area; as a result, some nodes can find many helpers in their ranges, while other helpers find only a few helpers in their ranges, which results in requesting fewer helpers. On the other hand, the standard deviation for the single-helper mode is zero, since in every case exactly one helper is requested to help an inviting node.

### 5.3.5 Average Number of Admitted Helpers by All Inviting Nodes for All Iterations

Fig. 5.5 depicts the average number of admitted helpers by all inviting nodes for all iterations for our four oppnet implementations.

The graph first shows the average number of admitted helpers for MSM and OSM. Both implementations invite a single helper (a deterministic value), which means that there is no overhead for OSM.

The average for OAM (2.9) is slightly higher (only by 3.6%) than for MAM (2.8). The reason is that OAM takes a longer time to integrate helpers than MAM (see
Subsection 5.3.1). During that time, the inviting node keeps requesting more helpers for help and admits them into oppnet. However, this overhead (i.e., 3.6%) is small considering the use of OVM primitives in implementing OAM.

![Bar chart showing the average number of admitted helpers by all inviting nodes for all iterations.](image)

Notice that the average number of admitted helpers for the single-helper mode (both MSM and OSM) is lower by around a 64% than for the all-helper mode (both MAM and OAM). The reason for this is that in the single-helper mode only one helper is admitted by its inviting node, while in the all-helper mode each inviting node can admit all the candidate helpers in its range.

Notice that the range of the standard deviations is very high in MAM and OAM which means that the variance in the number of admitted helpers among nodes is high. The reason for that: due to the large simulation area (600m x 600 m) and the high range for number of helpers (20–60 helpers, cf. Table 5.9), the helpers can be scattered through the simulation area; as a result, some nodes can find many helpers in their ranges, while other helpers find only a few helpers their ranges, which results in admitting fewer helpers. On the other hand, the standard deviation for the single-helper mode is zero, since here in every case exactly one helper is admitted by an inviting node.
5.3.6 Average Number of Helpers *Refusing* to Help by All Inviting Nodes for All Iterations

Fig. 5.6 depicts the average number of helpers *refusing* to help for all inviting nodes for all iterations for our four implementations.

![Graph showing average number of helpers refusing to help](image)

**Figure 5.6.** Average number of helpers *refusing* to help by all inviting nodes for all iterations.

The graph first shows that no helper refuses to accept an invitation for MSM and OSM. This is because only successful oppnets are considered in the shown statistics. For the single-helper mode, an oppnet is successful only when the single candidate helper does not refuse. Also, the inviting node was never refused when it requested help because its workload ratio is less than 95% (cf. Section 5.2.1) the inviting node was lucky the first time it requested help to find a helper that is willing to help (because its workload ration is less than 95%).

The average for OAM is higher (by 12.5%) than for MAM. The reason for this is that the number of helpers requested for help for OAM is higher than for MAM (see Subsection 5.3.3). As a result, more helpers refuse to help for OAM than for MAM.
The standard deviation in MAM and OAM is higher than the mean which causes the minimum value of the standard deviation to reach significantly below zero. The reason for the high variance between values in MAM and OAM: due to the large simulation area (600m x 600 m) and the high range for number of helpers (20–60 helpers, cf. Table 5.9), the helpers can be scattered through the simulation area; as a result, some nodes can find many helpers in their ranges, while other helpers find only a few helpers their ranges, which results in requesting fewer helpers and therefore fewer helpers refusing to help. On the other hand, the standard deviation for the single-helper mode is zero, since here in every case exactly one helper is requested to help an inviting node.

5.3.7 Average Number of Helpers that Performed Urgent Tasks for All Iterations

Fig. 5.7 shows the average number of helpers that performed urgent tasks among all iterations for our four oppnet implementations.

The average number of helpers performing urgent tasks (cf. Subsection 5.2.6) for OSM (3.1) is higher by 10.7% than for MSM (2.8). On the other hand, the average number of helpers performing urgent tasks for OAM (4.5) is slightly lower (by 2.2%) than for MAM (4.6). This is an unexpected result for which we still lack an explanation.

We observe that the average number of helpers performing urgent tasks in the all-helper mode (both MAM and OAM) is higher than in the single-helper mode (both MSM and OSM) by over 45%. This is due to the fact that with the larger number of helpers involved in the all-helper mode there are higher the chances of finding helpers that can perform urgent tasks.
Figure 5.7. Average number of helpers that performed *urgent* tasks for all iterations.

Notice that the standard deviation ranges are high in all four implementations (especially in the all-helper mode implementations). The reason for that is the wide range of devices used in the simulation; some of such devices are capable of performing urgent tasks and some are not. As a result, in some of the iterations, the inviting node can find many helpers in its range that can perform urgent tasks, while in other iterations only a few helpers are capable of performing urgent tasks.

### 5.3.8 Average Number of Helpers that Performed *Unhurried* Tasks for All Iterations

Fig. 5.8 shows the average number of helpers that performed *unhurried* tasks among all iterations for our four implementations.

The average number of helpers performing unhurried tasks for OSM is higher by 25% than for MSM. On the other hand, the average number of helpers performing unhurried tasks for OAM (4.3) is slightly lower (by 6.5%) than for MAM (4.6). Until now we do not have an explanation for this phenomenon.
We observe that the average number of helpers performing unhurried tasks for the single-helper mode (both MSM and OSM) is lower than for the all-helper modes (both MAM and OAM). This is because all-helper mode admits more helpers, which increases the chances of finding helpers that can perform unhurried tasks.

![Average number of helpers that performed unhurried tasks for all iterations.](image)

Notice that the standard deviation ranges are high in all four implementations (especially in the all-helper mode implementations). The reason for that is the wide range of devices used in the simulation; some of such devices are capable of performing unhurried tasks and some are not. As a result, in some of the iterations, the inviting node can find many helpers in its range that can perform unhurried tasks, while in other iterations only a few helpers are capable of performing unhurried tasks.
5.3.9 Success Rates

The success rates for our four implementations are depicted in Fig. 5.9. Recall that the success rate equals the number of successful iterations—in which an oppnet achieves its goal, that is, delivers the help message to 9-1-1—divided by the total number of iterations.

The success rates for MSM and OSM are exactly the same, 60%. Similarly, the success rates for MAM and OAM are exactly the same, 73.3%.

Notice that the success rate for the single-helper mode (both MSM and OSM) is lower (by around 18%) than for the all-helper mode (both MAM and OAM). The reason for this is that for the all-helper mode each inviting node can admit all the candidate helpers in its range, while in the single-helper mode only one helper is admitted by an inviting node, which decreases the chances for oppnet to succeed.

We should note that during the simulation oppnet, we encountered an emergent inexplicable behavior. Using the parameters we discussed earlier in this chapter, the
simulation success rate should have been 70% for MSM and OSM, and 83.3% for MAM and OAM. However, our simulation implementation for the oppnet middleware (both monolithic and modular) misses 14% of the successful iterations for the single-helper mode, and 12% of the successful iterations for the all-helper mode. We have no explanation yet and are still investigating this issue.

5.4 Chapter Conclusions

This chapter started with the presentation of the simulation scenario and the simulation design. The latter included simulation assumptions, simulation parameters, random simulation variables, and simulation measures.

We then described the simulation results and discussed them. Results show that the modular middleware has low overhead considering the use of the OVM primitives.

Results of the experiment show that the modular oppnet middleware has almost identical results as the monolithic oppnet middleware in the measures used to test the effectiveness of the modular middleware namely: average helper integration time, average time for oppnet to succeed, average task runtime, average number of helpers (requested/admitted/refusing to help), and average number of helpers that performed (urgent/unhurried) tasks. Surprisingly, in some cases, the modular middleware has a slightly better performance than the monolithic middleware; we believe that such slight difference is within the statistical error range. The modular middleware achieves significantly high oppnet success rates (as high as for the monolithic middleware).

We can summarize the results in some detail as follows. The average helper integration time for MSM is slower by 1% than for OSM, which is within the statistical error range. The average helper integration time for OAM is longer by 1.6% than for MAM, which is expected considering the overhead of using the OVM primitives in implementing OAM.
The average task runtime for MSM is slower by 1.8% than for OSM, which is within the statistical error range. The average task runtime for OAM is slower by merely 0.27% than for OAM, which is a negligible overhead considering the use of OVM primitives in implementing OAM.

The values of average time for oppnet to succeed for the modular oppnet middleware (OSM and OAM) are almost identical to the values of the monolithic oppnet middleware (MSM and MAM).

The average number of helpers requested for help is identical in MSM and OSM. On the other hand, the average number of helpers requested in OAM is higher by only 3.4% than in MAM—due to the longer integration time in OAM. However, this overhead is small considering the use of all the OVM primitives in implementing the modular middleware.

The average number of helpers admitted into an oppnet is identical in MSM and OSM. On the other hand, the average number of helpers admitted into an oppnet in OAM is higher by only 3.6% than in MAM—due to the longer integration time in OAM. It is a small overhead considering the use of all the OVM primitives in implementing the modular middleware.

The average number of helpers refusing to help is identical in MSM and OSM. On the other hand, the average number of helpers requested in OAM is higher by 12.5% than in MAM—due to the longer integration time in OAM. The reason is that more helpers are requested for help in OAM than in MAM which results in more helpers refusing to help in OAM.

The average number of helpers performing urgent tasks in OSM is higher by 10.7% than in MSM. On the other hand, the average number of helpers performing urgent tasks in OAM is lower by only 2.2% than in MAM. This is an unexpected result for which we lack an explanation.

The average number of helpers performing unhurried tasks in OSM is higher by 25% than in MSM. On the other hand, the average number of helpers performing unhurried
tasks in OAM is lower by 6.5% than in MAM. We do not have an explanation for this phenomenon.

The success rates for MSM and OSM are exactly the same: 60%. Similarly, the success rates for MAM and OAM are exactly the same: 73.3%.

The above results of the simulation experiments show that the use of the modular (OVM-based) middleware is efficient and does not cause high overhead. Most importantly, the use of the OVM primitives does not reduce the success rates of oppnet middleware (it stays as high as for the monolithic middleware).
6. CONCLUSIONS, SIGNIFICANCE, EXPECTED IMPACTS, AND FUTURE WORK

This chapter presents the conclusions, the significance of our work and its expected impacts. We discuss the future work and delineate the areas that need more investigation.

6.1 Conclusions

We developed and validated the improved second version of the OVM primitives that is intended to support a broad and diverse set of applications. As a proof of feasibility, we used this set of OVM primitives to develop modular (non-monolithic) oppnet middleware that implements a healthcare and wellness monitoring scenario.

The OVM primitives can be downloaded by any computational entity (system or device)—making the entity oppnet-enabled. OVM ensures that oppnet-based and oppnet-enabled entities can interact and share resources in an opportunistic and ad hoc manner.

The OVM-based interactions among oppnet-based and oppnet-enabled entities realize the following desirable oppnet characteristics, which distinguish oppnets from other opportunistic, collaborative, or pervasive systems: (i) support for the helper paradigm—as the basis for resource acquisition; (ii) universality—regardless of the system or device make or function, and through any communication technology, protocols, etc.; (iii) lack of third-party mediators—since interactions among oppnet-based or oppnet-enabled entities take place without third parties; and (iv) ad hoc operation—except the need for a predesigned formation of the so called oppnet seed from which the expanded oppnet grows.

The features that distinguish our oppnet middleware from the previous oppnet work are: (i) implementation of the object-oriented OVM primitives in a modular fashion; (ii)
utilization of a wider number of resources in the experiments; and (iv) ability to use of more kinds of communication technologies.

We should mention an innovation we introduced in our pseudocode specifications of the algorithms. Our pseudocode is an object-oriented pseudocode, based on the Java code syntax, which allows using some desirable object-oriented (OO) capabilities (like polymorphism) in the pseudocode. To the best of our knowledge we are the first to use OO pseudocode.

In order to test the effectiveness of the modular middleware in ad hoc resource sharing, we used a healthcare and wellness monitoring scenario to simulate both monolithic oppnet middleware and modular oppnet middleware, each in single-helper mode and all-helper mode. This resulted in four different categories of oppnet implementations: (i) MSM—monolithic single-helper mode; (ii) OSM—modular (OVM-based) single-helper mode; (iii) MAM—monolithic all-helper mode; and (iv) OAM—modular all-helper mode.

We used a number of criteria to analyze and compare the simulation results for these four different categories of oppnet implementation. Summarizing briefly, we found out that: (i) the average helper integration time and the average task runtime for MSM are slower by 1% and 1.8%, respectively, than for OSM; (ii) the average helper integration time and the average task runtime for OAM are longer by 1.6% and 0.27%, respectively, than for MAM; (iii) the values of average time for oppnet to succeed for the modular oppnet middleware (OSM and OAM) are almost identical to the values of the monolithic oppnet middleware (MSM and MAM); (iv) the average numbers of helpers requested to help, admitted into oppnet, and refusing to help are identical in MSM and OSM, while the average numbers of helpers requested to help, admitted into oppnet, and refusing to help in OAM are higher by only 3.4%, 3.6% and 12%, respectively, than in MAM; (v) the average numbers of helpers performing urgent tasks and helpers performing unhurried tasks in OSM are higher by 10.7% and 25%, respectively, than in MSM, while the average numbers of helpers performing urgent tasks and helpers performing unhurried tasks in OAM are lower by only 2.2% and 6.5%, respectively, than in MAM;
and (vi) the success rates for MSM and OSM are exactly the same: 60%, and the success rates for MAM and OAM are exactly the same: 73.3%.

The use of the modular oppnet middleware in diverse applications looks very promising. The above results show that the use of the modular middleware does not introduce significant overhead: it remains negligible or small. Most importantly, the use of the OVM primitives does not reduce the success rates of oppnet middleware (it stays as high as for the monolithic middleware).

Thus, the obtained results provide a strong evidence supporting our research hypothesis:

\textit{OVM-based oppnet middleware can provide efficient ad hoc application-level resource sharing among diverse systems and devices.}

\section*{6.2 Significance}

OVM-based oppnet middleware is envisioned as an open-source middleware and a standard that will assist in development of oppnet-based applications, as well as in contributing to the pervasive computing paradigm by facilitating massive ad hoc resource sharing via a helper mechanism; it allows for oppnet-enabling entities in the totally ad hoc way (by downloading and installing the needed OVM primitives). We hope that benefits of the oppnet paradigm will result in its widespread use.

\section*{6.3 Expected Impacts}

Our work is expected to have the following positive impacts:

1) The modular oppnet middleware will realize the main goal of opportunistic resource utilization networks which is helper-based sharing of resources among nodes.
Resources are not limited to communication, but include also computation, sensing, actuating, storage, etc.

2) The open-source OVM primitives allow different entities to lend their resources to help others in diverse applications, including healthcare, border security, weather prediction, and emergency response. As the world is moving towards the concept of the Internet of Things, our development of an open-source solution that breaks the barriers among entities regardless of their function or manufacturer will facilitate smart interaction among these entities.

3) The modular oppnet middleware will facilitate implementations of oppnet-based and oppnet-enabled entities from a diverse set of vendors or developers.

4) OVM primitives will allow open-source sharing, or possibly marketing, standard library routines and APIs that can be used for implementing all kinds of oppnet-based and oppnet-enabled applications.

6.4 Future Work

The future work on the OVM-based oppnet middleware includes the following:

1) Investigating the reasons of unexpected phenomena of some simulation results such as better performance of the modular implementation than the monolithic implementation under some measures. These phenomena were indicated in the discussion of the results and in conclusions of Chapter 5.

2) Integrating privacy and security techniques into the development of the OVM primitives. Privacy methodologies would protect sensitive data (such as the helpObject containing personal information) transmitted among nodes from being exposed to unauthorized entities. Security techniques will protect oppnet applications and oppnet nodes from tacks of malicious nodes.
3) Implementation of software for actual devices instead of emulation, as was done in this Dissertation for some devices (e.g., wristband, printer, etc.), and integration of more communication protocols, such as Li-Fi and WiMAX.

4) Applying optimization algorithms to further enhance performance of the OVM primitives. Optimized OVM primitives will facilitate meeting even more stringent QoS goals, enhance software performance. Realizing QoS goals for the OVM primitives is especially critical in life-or-death situations, such as emergency response [Lili07] where inefficiencies and error margins must be very low.
REFERENCES


Kraków, Poland, Sept. 2006, pp.722-726.


[WJMF05] Y. Wang, S. Jain, M. Martonosi and K. Fall, “Erasure-Coding Based Routing for Opportunistic Networks,” ACM Conf. of the SIG on Data Communication (SIGCOMM),


APPENDIX A: FIRST-GENERATION OVM PRIMITIVES

In this appendix we present the partial lists of the first-generation OVM primitives, as defined in [KGLY07, KLGY08, LGKY10]. The first-generation OVM primitives are discussed in Subsection 3.3.2.

Table A.1. Partial list of OVM primitives for CC nodes.

<table>
<thead>
<tr>
<th>Name of the Primitive</th>
<th>Functions of the Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL_initiate</td>
<td>Initiate Oppnet</td>
</tr>
<tr>
<td>CTRL_terminate</td>
<td>Terminate Oppnet</td>
</tr>
<tr>
<td>CTRL_command</td>
<td>Send commend to seed nodes</td>
</tr>
</tbody>
</table>

Table A.2. Partial list of OVM primitives for seed nodes.

<table>
<thead>
<tr>
<th>Name of the Primitive</th>
<th>Functions of the Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEED_scan</td>
<td>Scan communication spectrum to detect devices that could become candidate helpers</td>
</tr>
<tr>
<td>SEED_discover</td>
<td>Discover candidate helpers with a specific communication mechanism</td>
</tr>
<tr>
<td>SEED_listen</td>
<td>Receive and save messages in buffer</td>
</tr>
<tr>
<td>SEED_validate</td>
<td>Verify the received command</td>
</tr>
<tr>
<td>SEED_isMember</td>
<td>Checks if a device is already an Oppnet node (Oppnet member)</td>
</tr>
<tr>
<td>SEED_evaluateAdmit</td>
<td>Evaluate a device and admit it into Oppnet if the device meets criteria for admittance</td>
</tr>
<tr>
<td>SEED_sendTask</td>
<td>Send a task to other Oppnet device</td>
</tr>
<tr>
<td>SEED_delegateTask</td>
<td>Delegate a task that requires a permission from the delegating entity</td>
</tr>
<tr>
<td>SEED_release</td>
<td>Release a helper when no longer needed</td>
</tr>
<tr>
<td>SEED_processMsg</td>
<td>Process a message from buffer</td>
</tr>
<tr>
<td>SEED_report</td>
<td>Report information to control center/coordinator</td>
</tr>
<tr>
<td>SEED_update</td>
<td>Update a device in the Oppnet with new expectations</td>
</tr>
<tr>
<td>SEED_receiveTask</td>
<td>Receive task from control center or another seed</td>
</tr>
<tr>
<td>SEED_wait</td>
<td>Wait for a certain amount of time before take another action</td>
</tr>
<tr>
<td>SEED_barrier</td>
<td>Block the caller until all devices specified in the input parameter have called it</td>
</tr>
</tbody>
</table>
Table A.3. Partial list of OVM primitives for helpers.

<table>
<thead>
<tr>
<th>Name of the Primitive</th>
<th>Functions of the Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLPR_isMember</td>
<td>Test if a helper is already a member of Oppnet</td>
</tr>
<tr>
<td>HLPR_joinOppnet</td>
<td>Join Oppnet</td>
</tr>
<tr>
<td>HLPR_scan</td>
<td>Scan communication spectrum to detect devices that could become candidate helpers (regular or lites)</td>
</tr>
<tr>
<td>HLPR_discover</td>
<td>Discover candidate helpers with a specified communication mechanism</td>
</tr>
<tr>
<td>HLPR_validate</td>
<td>Verify the received command</td>
</tr>
<tr>
<td>HLPR_switchMode</td>
<td>Switch between helpers’ regular application and Oppnet application</td>
</tr>
<tr>
<td>HLPR_report</td>
<td>Send information/data to specified device</td>
</tr>
<tr>
<td>HLPR_selectTask</td>
<td>Select a task from the task queue to execute</td>
</tr>
<tr>
<td>HLPR_listen</td>
<td>Receive message and save it</td>
</tr>
<tr>
<td>HLPR_evaluateAdmit</td>
<td>Evaluate a candidate helper and admit it into Oppnet if it meets criteria defined by Oppnet</td>
</tr>
<tr>
<td>HLPR_runApplication</td>
<td>Execute application indicated by authorized Oppnet seed or helper node</td>
</tr>
<tr>
<td>HLPR_release</td>
<td>Release a helper (unless delegated a release task, a helper H can release only helpers admitted by H)</td>
</tr>
<tr>
<td>HLPR_processMsg</td>
<td>Process a message from buffer</td>
</tr>
<tr>
<td>HLPR_sendData</td>
<td>Send information/data to specified authorized Oppnet node</td>
</tr>
<tr>
<td>HLPR_leave</td>
<td>Inform a seed that the caller will quit Oppnet</td>
</tr>
<tr>
<td>HLPR_strongTask</td>
<td>Respond to the request sent from device and express the willingness to join Oppnet. By accepting this task, the device will abort previous task</td>
</tr>
<tr>
<td>HLPR_weakTask</td>
<td>Respond to the request sent from device and express the willingness to join Oppnet. By accepting this task, the device will put the task in a queue</td>
</tr>
<tr>
<td>HLPR_assignStrongTask</td>
<td>Assign tasks to a device. If accepted, the task will interrupt the previous task at the device</td>
</tr>
<tr>
<td>HLPR_assignWeakTask</td>
<td>Assign tasks to a device. If accepted, the task will be queued</td>
</tr>
</tbody>
</table>
Table A.4. Partial list of OVM primitives for lites (lightweight helpers).

<table>
<thead>
<tr>
<th>Name of the Primitive</th>
<th>Functions of the Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>LITE_isMember</td>
<td>Test if a lite is already a member of Oppnet</td>
</tr>
<tr>
<td>LITE_joinOppnet</td>
<td>Join Oppnet</td>
</tr>
<tr>
<td>LITE_validate</td>
<td>Verify the received command</td>
</tr>
<tr>
<td>LITE_switchMode</td>
<td>Switch between lites’ regular application and Oppnet application</td>
</tr>
<tr>
<td>LITE_report</td>
<td>Send information/data to specified device</td>
</tr>
<tr>
<td>LITE_selectTask</td>
<td>Select a task from the task queue to execute</td>
</tr>
<tr>
<td>LITE_listen</td>
<td>Receive message and save it</td>
</tr>
<tr>
<td>LITE_runApplication</td>
<td>Execute application indicated by authorized Oppnet seed or helper node</td>
</tr>
<tr>
<td>LITE_processMsg</td>
<td>Process a message from buffer</td>
</tr>
<tr>
<td>LITE_sendData</td>
<td>Send information/data to specified authorized Oppnet node</td>
</tr>
<tr>
<td>LITE_leave</td>
<td>Inform a seed that the caller will quit Oppnet</td>
</tr>
<tr>
<td>LITE_strongTask</td>
<td>Respond to the request sent from device and express the willingness to join Oppnet. By accepting this task, the device will abort previous task</td>
</tr>
<tr>
<td>LITE_weakTask</td>
<td>Respond to the request sent from device and express the willingness to join Oppnet. By accepting this task, the device will put the task in a queue</td>
</tr>
</tbody>
</table>
APPENDIX B: SOURCE CODE INFORMATION

The source code for all the middleware (implemented or emulated in Chapter 4 and simulated in Chapter 5 of this Dissertation) can be found at:

https://github.com/malduailij/Oppnet-OVM

In particular:

- Implementation and emulation code for MicroOppnet v.4 (the testbed for the second-generation OVM, cf. Section 4.2) can be downloaded from the following link:
  https://github.com/malduailij/Oppnet-OVM/tree/master/MicroOppnetv.4

- Implementation and emulation code for MicroOppnet v.5 (the testbed for the third-generation OVM, cf. Section 4.3) can be downloaded from the following link:
  https://github.com/malduailij/Oppnet-OVM/tree/master/OVM_ThirdGeneration

- Simulation code for the monolithic oppnet middleware, single-helper mode (MSM) using the healthcare scenario (cf. Chapter 5) can be downloaded from the following link:
  https://github.com/malduailij/Oppnet-OVM/tree/master/Simulation/Monolithic_Healthcare_Simulation_SingleHelperMode

- Simulation code for the monolithic oppnet middleware, all-helper mode (MAM) using the healthcare scenario (cf. Chapter 5) can be downloaded from the following link:
  https://github.com/malduailij/Oppnet-OVM/tree/master/Simulation/Monolithic_Healthcare_Simulation_AllHelperMode
• Simulation code for the modular (OVM-based) oppnet middleware single-helper mode (OSM) using the healthcare scenario (cf. Chapter 5) can be downloaded from the following link:

https://github.com/malduailij/Oppnet-OVM/tree/master/Simulation/OVM_Based_Healthcare_Simulation_SingleHelperMode

• Simulation code for the modular (OVM-based) oppnet middleware all-helper mode (OAM) using the healthcare scenario (cf. Chapter 5) can be downloaded from the following link:

https://github.com/malduailij/Oppnet-OVM/tree/master/Simulation/OVM_Based_Healthcare_Simulation_AllHelperMode
APPENDIX C: DETAILS OF SIMULATION RESULTS

In this appendix we present more details of the simulation results discussed in Chapter 5. Specifically, the graphs presented here depict the measures per simulation iteration. Each of the per-iteration graphs presented here corresponds to a graph in Chapter 5 that depicts a measure for all iterations. For example, Fig. C.1 depicts the average helper integration time for all inviting nodes per iteration, which corresponds to Fig 5.1 that displays the average helper integration time for all inviting nodes for all iterations.

Each graph depicts only the successful simulation iterations. As mentioned at the beginning of Section 5.3, the number of successful simulation iterations for MSM and OSM is 18, and the number of successful iterations for MAM and OAM is 22. Hence the curves for MSM and OSM stop at 18, and the curves for MAM and OAM stop at 22.

Figure C.1. Average helper integration time for all inviting nodes per iteration.
In Fig. C.1, the MSM (blue) curve is completely covered by the OSM (red) curve.

![Figure C.1](image1)

Figure C.2. Average task runtime for all nodes per iteration.

In Fig. C.2, the MSM (blue) curve is in the most part covered by the OSM (red) curve.

![Figure C.2](image2)

In Fig. C.2, the MSM (blue) curve is completely covered by the OSM (red) curve.

![Figure C.3](image3)

Figure C.3. Time for oppnet to succeed per iteration.

In Fig. C.3, the MSM (blue) curve is completely covered by the OSM (red) curve.
Figure C.4. Average number of helpers requested for help by all inviting nodes per iteration.

In Fig. C.4, the MSM (blue) curve is completely covered by the OSM (red) curve.

Figure C.5. Average number of admitted helpers by all inviting nodes per iteration.

In Fig. C.5, the MSM (blue) curve is completely covered by the OSM (red) curve.
In Fig. C.6, the MSM (blue) curve is completely covered by the OSM (red) curve.

Figure C.7. Number of helpers that performed urgent tasks per iteration.

In Fig. C.7, the MSM (blue) curve is in the most part covered by the OSM (red) curve.
Figure C.8. Number of helpers that performed *unhurried* tasks per iteration.

In Fig. C.8, the MSM (blue) curve is in the most part covered by the OSM (red) curve.