Orchestra Framework: Protocol Design for Ad Hoc and Delay Tolerant Networks using Genetic Algorithms

Apoorv Naik

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

Luiz A. DaSilva, Chair
Yaling Yang
Thomas Hou

20 June, 2011
Blacksburg, Virginia

Keywords: Mobile Ad Hoc Networks, Delay Tolerant Networks, Genetic Algorithms.
Copyright 2011, Apoorv Naik
Orchestra Framework: Protocol Design for Ad Hoc and Delay Tolerant Networks using Genetic Algorithms

Apoorv Naik

(Abstract)

Protocol designs targeted at a specific network scenario or performance metric appear promising on paper, but the complexity and cost of implementing and tuning a routing protocol from scratch presents a major bottleneck in the protocol design process. The work in [1] introduces a unique framework called 'Orchestra' to support the testing and development of novel routing designs. The idea of the Orchestra framework is to create generic and reusable routing functional components which can be combined to create unique protocol designs customized for a specific performance metric or network setting. The first contribution of this thesis is the development of a generic, modular, scalable and extensible architecture of the Orchestra framework.

Once the architecture and implementation of the framework is completed, the second contribution of this thesis is the development of functional components and strategies to design and implement routing protocols for delay tolerant networks (DTNs). DTNs are a special type of ad hoc network characterized by intermittent connectivity, long propagation delays and high loss rate. Thus, traditional ad hoc routing approaches cannot be used in DTNs, and special features must be developed for the Orchestra framework to support the design of DTN routing protocols.

The component-based architecture of Orchestra can capture a variety of modules that can be used to assemble a routing protocol. However, manually assembling these components may result in suboptimal designs, because it is difficult to determine what the best combination is for a particular set of performance objectives and network characteristics. The third contribution of the thesis addresses this problem. A genetic algorithm based approach to automate the process of routing protocol design is developed and its performance is evaluated in the context of the Orchestra framework.
Acknowledgement

First and foremost, I would like to express my sincere gratitude towards my advisor Dr. Luiz DaSilva for showing an unwavering trust and confidence in me. It has been a great learning experience working with him. The constructive suggestions and criticisms offered by him have not only helped in shaping up this research but have also helped me grow as an individual, both professionally and personally. I truly appreciate all his support and guidance.

I would like to thank Dr. Yaling Yang for her valuable and insightful suggestions throughout the course of my research work. Her opinions and reviews about the work have always been very helpful and appreciated.

I would also like to thank Dr. Thomas Hou for teaching me the concepts which have immensely helped me in this research work. The classes I have taken with him have been instrumental in providing the strong foundation required for this research.

I would like to specially mention my friends Aditi, Juan, Abdallah, Sarvesh, Sachin, Charan, Mayank, Piyush and Anup for always being a great support.

Last but not the least, I am extremely grateful to my parents, my brother and my family for all their encouragement, support and faith in my abilities. It would not have been possible without them.

Apoorv Naik
This material is based upon the work supported by National Science Foundation under the Grant no. CNS - 0916873. Any opinions, findings, conclusions or recommendations expressed in this material are those of authors and do not necessarily reflect the views of National Science Foundation.
# Contents

1 Introduction .............................................. 1
   1.1 Motivation ........................................... 1
   1.2 Contributions ...................................... 2
   1.3 Thesis Organization ................................. 3

2 Background .............................................. 4
   2.1 Domain Knowledge .................................. 5
      2.1.1 Mobile Ad Hoc Networks (MANETs) ............ 6
      2.1.2 Routing in MANETs .............................. 7
      2.1.3 Delay Tolerant Networks (DTNs) .............. 8
      2.1.4 Routing in DTNs ................................. 9
   2.2 Software Engineering Background ................ 11
      2.2.1 Component-based Software Engineering ........ 11
      2.2.2 Design Patterns in Software Engineering .... 13
   2.3 Operations Research Background .................. 13
      2.3.1 Combinatorial Optimization .................... 14
      2.3.2 Metaheuristic Methods .......................... 14
   2.4 Existing Frameworks for Routing Protocol Design 16
   2.5 Summary .............................................. 17

3 Orchestra Framework - Domain Analysis and System Modeling 18
   3.1 Introduction ......................................... 18
3.2 Orchestra Framework .............................................. 19
3.3 Orchestra Software Design ........................................ 19
3.4 Domain Analysis .................................................... 21
  3.4.1 Commonalities amongst different routing protocols .......... 21
  3.4.2 Flow of Information in Routing Protocols ................... 24
3.5 System Modeling ................................................... 24
3.6 Automated Protocol Design in the Orchestra Framework ....... 27
3.7 Summary .......................................................... 27

4 Orchestra Architecture and Implementation ....................... 29
  4.1 Design Requirements ............................................. 29
  4.2 System Architecture ............................................. 31
  4.3 Architectural Features .......................................... 31
    4.3.1 Interface-based Framework Design ......................... 31
    4.3.2 Additional Layer of Abstraction .......................... 35
    4.3.3 Loose Coupling between Framework and Components ....... 37
  4.4 Orchestra’s Components for DTN Protocols ..................... 38
  4.5 Summary ........................................................ 41

5 Automatic Assembling of Routing Protocols in the Orchestra Framework .... 42
  5.1 Problem Formulation ............................................. 42
  5.2 Proposed Solution - Genetic Algorithms ....................... 44
  5.3 Genetic Algorithms for the Orchestra framework ............... 45
    5.3.1 Representation ............................................. 46
    5.3.2 Initialization .............................................. 46
    5.3.3 Fitness Function and Design Evaluation Mechanism ......... 48
    5.3.4 Selection .................................................. 49
    5.3.5 Crossover ................................................ 49
    5.3.6 Mutation ................................................ 50
List of Figures

2.1 An example of a mobile ad hoc network. ................................. 6
2.2 An example of a store and carry forward routing scheme. .............. 9

3.1 Conventional protocol design versus Orchestra’s protocol design mechanism. 20
3.2 Models of different categories of protocols in Orchestra. ................. 26

4.1 System architecture of the Orchestra framework. ............................ 32
4.2 Interface-based architecture of the Orchestra framework. ................. 34
4.3 Routing strategy layer abstraction in the Orchestra framework. .......... 36
4.4 Routing strategy model in the Orchestra framework. ....................... 36
4.5 Model of Orchestra’s component factory classes. ............................ 38

5.1 Genetic Algorithm (GA) for automatic assembling of routing protocols in the Orchestra framework. ............................................. 47
5.2 Representation of a routing design in genetic format. ....................... 48
5.3 Mechanism to evaluate the fitness of a routing design. ..................... 49
5.4 Crossover operation on routing designs. ...................................... 50
5.5 The mutation operation on a routing design. .................................. 50

6.1 Delivery ratio comparison of the ER scheme in [2] with the ER scheme designed using Orchestra’s functional components. ................... 54
6.2 Latency comparison of the ER scheme in [2] with the ER scheme designed using Orchestra’s functional components. ....................... 54
6.3 Delivery latency comparison of routing designs using various TIC, RME, and PFE components. ................................................. 59
6.4 Delivery ratio comparison of routing designs using various TIC, RME, and PFE components. ................................................................. 60
6.5 Data efficiency comparison of routing designs using various TIC, RME, and PFE components. ................................................................. 60
6.6 Average cost comparison of routing designs using various TIC, RME, and PFE components. ................................................................. 60
6.7 Resource efficiency comparison of routing designs using various TIC, RME, and PFE components. ................................................................. 60
6.8 Delivery latency comparison of routing designs using various BFM and PFE components. ................................................................. 62
6.9 Delivery ratio comparison of routing designs using various BFM and PFE components. ................................................................. 63
6.10 Data efficiency comparison of routing designs using various BFM and PFE components. ................................................................. 63
6.11 Average cost comparison of routing designs using various BFM and PFE components. ................................................................. 63
6.12 Resource efficiency comparison of routing designs using various BFM and PFE components. ................................................................. 63
6.13 Variations in simple GA. ................................................................. 66
6.14 Variations in steady state GA. ................................................................. 66
6.15 Variations in objective value with generations for the simple GA. ................................................................. 67
6.16 Variations in objective value with generations for the steady state GA. ................................................................. 67
6.17 Buffer capacity variations in the simple GA. ................................................................. 68
6.18 Buffer capacity variations in the steady state GA. ................................................................. 68
List of Tables

4.1 DTN components in the Orchestra framework . . . . . . . . . . . 41

6.1 Component’s notations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

6.2 GA parameters used in the experiments . . . . . . . . . . . . . . . . . 65

6.3 Protocols generated by GA for different routing metrics . . . . . . . . 69
Chapter 1

Introduction

Mobile Ad Hoc Networks (MANETs) and Delay Tolerant Networks (DTNs) have broadened the horizon of wireless communication well beyond the cell-based networks with fixed infrastructure. MANETs are self-configuring infrastructureless networks designed to support wireless communication in situations where deploying network infrastructure is not possible. DTNs are a special type of ad hoc network characterized by intermittent connectivity, long propagation delays, and high loss rates. Both MANETs and DTNs use a multihop communication scheme. Thus, a routing protocol is required for both types of networks.

1.1 Motivation

A number of potential routing approaches for MANETs and DTNs are proposed in the literature. Many of these routing approaches are designed to optimize a specific routing metric. However, the cost and complexity of designing and tuning a routing protocol that implements a given approach is usually beyond the reach of an individual protocol designer. As a result, most of these novel routing ideas are never put into practice and a one-protocol-suits-
all approach is adopted. Thus, to tackle the problem of designing and testing a customized routing protocol for a specific routing metric, a mechanism to simplify and speed up the process of protocol design is highly desirable.

Due to the abundance of research work going on in the area of MANET and DTN routing, it is practically not possible for a researcher to keep track of all the novel developments in the field. Thus, a manually designed routing protocol may ignore some useful functionalities which can improve the performance of a routing protocol. But, if a mechanism can be devised to automate the process of routing protocol design utilizing all the available knowledge, this problem can be solved.

1.2 Contributions

In order to address the challenges mentioned above, [1] introduced a unique framework called 'Orchestra - Open Architecture for Checking, Synthesis and Training of Routing Algorithms.' This thesis provides three main contributions in context of the Orchestra framework. The first contribution of this thesis is a major software redesign and architectural modification of the Orchestra framework to make the architecture more generic, modular, scalable, and extensible. The second contribution of this thesis is the extension of the Orchestra framework to support protocol designing for DTNs. The third contribution of this thesis is the modeling of the automated protocol design problem in the Orchestra framework as a combinatorial optimization problem and the development of a GA-based solution for it.
1.3 Thesis Organization

The thesis is organized as follows: Chapter 2 provides the background information related to MANETs, DTNs and routing approaches designed for them. This chapter also gives a brief introduction to component-based software engineering and metaheuristic algorithms, and discusses some related work. Chapter 3 presents the domain analysis for identifying common features present in ad hoc and DTN routing protocols. This chapter also discusses the system modeling of the Orchestra framework. Chapter 4 presents the architectural features and implementation details of the Orchestra framework. DTN-specific components of the Orchestra framework are also discussed in this chapter. The problem formulation of automating the protocol design process as a combinatorial optimization problem and the proposed GA based solutions are the topics of discussion in chapter 5. Chapter 6 presents the evaluation of protocols generated by different functional components of the Orchestra framework and the results of GA-based algorithms used in automated protocol design. The main conclusions and directions for future work are presented in chapter 7.
Chapter 2

Background

In order to understand the system model and architecture of the Orchestra framework, familiarity with Orchestra’s application domain and concepts involved in its development is necessary. This chapter discusses the concepts from ad hoc networking, software engineering, and operations research in the context of the Orchestra framework. It also presents some other protocol design frameworks from the literature. The chapter is organized as follows.

- **Domain knowledge**: The Orchestra framework is targeted at designing MANET and DTN routing protocols. Routing in MANETs and DTNs is different from routing in wired networks. Thus, a review of concepts of MANETs and DTNs, and routing approaches designed for them, is required. This is done in sections 2.1.1 and 2.1.3.

- **Software engineering background**: The goal of the Orchestra framework is to facilitate the development of metric-specific, scenario-dependent routing protocols. However, a routing protocol is a complex distributed software system which operates at all nodes in the network. In order to simplify the process of designing complex software systems, a number of approaches are proposed in the software engineering literature.
One of the well-known approaches is component-based software development. This is the approach we adopt in the redesign of the Orchestra framework. Component-based software engineering is discussed in section 2.2.1. To design a robust, easy-to-maintain source code for the Orchestra framework, software engineering design patterns are used in the development of the Orchestra framework. Software engineering design patterns are discussed in section 2.2.2.

- **Operations research background**: The Orchestra framework enables the automatic generation of routing protocols. In the context of component-based routing protocol design, this implies searching for the best possible component combination, which constitutes a routing protocol, to satisfy a specific performance objective. This is a combinatorial optimization problem. The combinatorial optimization problem is discussed in section 2.3.1. Metaheuristic methods offer an effective solution to tackle combinatorial optimization problems. Metaheuristic methods are discussed in section 2.3.2.

- **Other frameworks for protocol design**: Some other component-based frameworks for protocol design proposed in the literature are presented in section 2.4.

## 2.1 Domain Knowledge

In this section, a review of the concepts of MANETs, DTNs, and routing approaches designed for them is presented.
2.1.1 Mobile Ad Hoc Networks (MANETs)

The recent advances in mobile computing and wireless technologies have paved the way for designing self-configuring infrastructureless networks consisting of mobile nodes. These are known as MANETs. MANETs are useful in situations where deploying a network infrastructure is either not cost effective or not possible. Some situations where MANETs can be used are military scenarios, sensor network applications, etc. Figure 2.1 shows an example of a MANET.

The communication between any two nodes in a MANET, if they are not neighbors, is done in a multihop fashion. Each node in a MANET acts as both a host and a router to forward the data packets from other nodes. However, MANETs are characterized by a dynamically-changing network topology, limited power supply, interference in the medium, bandwidth limitations, etc. All these characteristics make routing in MANETs much more complicated than routing in wired networks. Thus, the routing solutions available for wired networks are
not applicable to MANETs. The routing approaches used in MANETs are discussed in the next section.

2.1.2 Routing in MANETs

Many routing protocols for MANETs are proposed in the literature. These routing protocols can be classified into three broad categories depending upon the type of routing mechanism. These categories are presented next.

Proactive Ad Hoc Routing Protocols

These are table-driven routing protocols which maintain routes to all the possible destinations in the network. In proactive routing protocols, each node periodically broadcasts control messages. The control messages contain information about the network, such as available links, neighbor information, etc. Each node uses this information to compute the routes to all other nodes in the network. Routes are stored in the routing table maintained at each node. The major advantage of a proactive routing approach is that it minimizes the session establishment time. Since routes to all the possible destinations are readily available, no delay is caused in establishing a communication session. However, proactive routing protocols are not suitable for networks with a highly dynamic topology, as frequent global flooding of control messages on link changes will degrade the throughput performance of the network. Some well-known examples of proactive routing protocols are OLSR [3] and DSDV [4].
Reactive Ad Hoc Routing Protocols

To avoid the problem of excessive overhead in the proactive routing protocols, reactive routing protocols implement an on-demand routing scheme. Instead of maintaining paths to all the nodes in the network at all times, routes are computed in reactive routing protocols only when required. Although an additional delay in packet delivery is introduced, the advantages are apparent in terms of reduced network resources consumption. Some well-known protocols belonging to this category are AODV [5] and DSR [6].

Hybrid Routing Protocols

Hybrid routing protocols combine the best features of proactive and reactive routing protocols. Well-known examples of hybrid routing protocols include ZRP [7] and TORA [8]. Although these protocols provide a promising routing solution for MANETs, the cost and complexity of implementing a hybrid routing protocol can be very high as compared to a proactive or a reactive routing protocol.

2.1.3 Delay Tolerant Networks (DTNs)

In the previous section, we presented the routing approaches used in traditional MANETs. The MANET routing protocols are unable to route a packet from source to destination if an end-to-end path does not exist in the network. However, in many practical situations, an end-to-end path may not be always present. Events like link failures, power outages at a node, link obstructions, etc., cause the network to be connected only intermittently. These types of networks with intermittent connectivity are becoming more and more important. Examples of intermittently connected networks include vehicular ad hoc networks (VANETs), military networks with unpredictable mobility, inter-planetary satellite communication networks with
In the absence of continuous network connectivity, the routing scheme for DTNs is based on a ‘store and carry forward (SCF)’ mechanism. In a SCF mechanism, each node in a network buffers the data packet until a forwarding opportunity manifests itself. Figure 2.2 illustrates this scheme. In this example, the data packet needs to be routed from node A to node D in an intermittently connected network. The snapshots of the network at four discrete times are shown in the four blocks.

DTN routing protocols are classified into two categories. These are discussed next.
Replication-Based DTN Routing Protocols

Replication-based DTN routing protocols require minimal knowledge about the network. These protocols flood the network with copies of the packets which need to be routed. A simple version of a replication-based DTN routing protocol is Epidemic Routing [2]. In epidemic routing, each node maintains a vector containing information about the data packets present in its buffer. This is known as a summary vector. On meeting with another node, the summary vectors are exchanged between the two nodes. Each node processes these summary vectors to determine which data packets need to be requested from the other node and exchanges copies of those data packets. In this way, copies of the data packets are spread in the network. The underlying assumption of epidemic routing scheme is that increasing the redundancy of a data packet in a network will improve the probability of its delivery. Epidemic routing is a resource-intensive scheme but allows routing in DTNs without any knowledge of the network topology. Most replication-based routing protocols use the principle of epidemic routing. These protocols try to improve the performance of an epidemic routing scheme by implementing strategies to bound the network resource consumption without compromising the delivery ratio. One such scheme known as ProPhet [9] uses the history of previous encounters between the nodes to make decisions about whether or not to forward the data packet on a transfer opportunity. The MaxProp [10] routing protocol implements a similar mechanism. In MaxProp the packets for transmission are ordered based on this cost. This cost is calculated using information about previous node meetings. Other schemes like the Spray and Wait protocol [11] bound the total number of copies of data packets in the network to a predetermined value. This value is dependent on the network conditions. Some protocols such as RAPID [12] model the DTN routing as a utility-driven resource allocation problem. In RAPID, the utility value of a packet is calculated using a metric (delivery ratio, latency, etc.) dependent function. On a contact opportunity, the packet is replicated only
if the value of its utility function will increase after replication. PREP [13], NECTAR [14], and MV Routing [15] are some other replication-based DTN routing protocols which use the information about previous node encounters in making routing decisions.

Knowledge-based DTN Routing Protocols

Knowledge-based DTN routing protocols, instead of flooding the network with copies of the data packets, use knowledge about the network to make routing decisions. There are numerous knowledge-based protocols proposed in the literature. Protocols like MARP, MDP [16] and CGR [17] require knowledge of future node meetings for making efficient routing decisions. Forwarding algorithms proposed in [18] assume the presence of knowledge oracles in the network. These knowledge oracles provide information required by the algorithms to make forwarding decisions.

2.2 Software Engineering Background

In the previous section, we discussed the application domain of the Orchestra framework. In this section, we will focus on software engineering concepts involved in the design of Orchestra.

2.2.1 Component-based Software Engineering

In recent years, there has been a paradigm shift from hardware-based computing to software-based computing. Some prominent examples from the domain of communications and networking are software-defined radios, software routers, software switches, software firewalls, etc. This paradigm shift has lowered the overall cost of computing, as operations which re-
quired costly hardware devices are now often completely implemented in software. However, in order to realize the true advantages of software-based computing over hardware systems, the software architecture of these systems must be able to adapt as per the application requirements.

Component-based software engineering (CBSE) provides an effective solution to address these challenges. The basic principle of component-based software engineering is to build the software architecture of a system with reusable functional components having well defined interfaces. These components are known as modules which encapsulate a set of related functionalities. The communication between different sets of components is carried out via their interfaces. CBSE provides two important characteristics in a software system. First, it makes the system customizable. Components with well-defined interfaces can be replaced by other components when required. This allows the system to be customized for specific applications. Second, it promotes reusability. The components, with well-defined interfaces, developed for one system can be used by another. A comprehensive review of component-based software engineering is available in [19].

Another important advantage of using component-based software engineering is that it lets us use the flow-based programming model [20] for the development of the framework. In flow-based programming, a software system is modeled as a network of ‘blackbox’ modules and an external ‘driver’ module can control the flow of information between these ‘blackbox’ modules.

A routing protocol is complex distributed software running at each node in the network and is a good candidate to be designed using component-based software engineering. The component-based architecture of a routing protocol will allow us to customize the operations of a routing protocol for optimizing a specific routing metric without making any major modifications in the system. The routing components designed for one protocol can be
reused in the design of another routing protocol. The use of a flow-based programming model will let us model different types of routing protocols like proactive routing, reactive routing, knowledge-based DTN routing, etc., in the Orchestra framework.

2.2.2 Design Patterns in Software Engineering

Although a network engineer or a researcher will only implement a novel part of the design as a new component in the Orchestra framework, it is important for them to understand the functioning of other components used and the system as a whole. To facilitate this, the component's development should be based on a set of tested and accepted development paradigms. This will not only provide a medium of communication between researchers while developing modules but will also improve the code readability.

Design patterns provide widely accepted, frequently reused solutions to commonly occurring problems in software design. Strategy pattern, factory method pattern, singleton pattern, and composite design pattern are used in the design of the Orchestra framework. A catalog of widely accepted design patterns is available in [21].

2.3 Operations Research Background

The problem of finding the best component combination to design a routing protocol in the Orchestra framework can be modeled as a combinatorial optimization problem. This is explained in detail in chapter 5. In this section, we will give a brief introduction to combinatorial optimization and metaheuristic methods which are used to tackle combinatorial optimization problems.
2.3.1 Combinatorial Optimization

Combinatorial optimization is an optimization technique in which the variables defining the mathematical model of an optimization problem are discrete. The optimal solution to a problem will be some set of discrete elements. There are many problems in the literature which can be modeled as a combinatorial optimization problem. Some examples include the traveling salesman problem, the vehicular routing problem, the minimum spanning tree of a graph, etc. Often, the combinatorial optimization problems are NP-hard, i.e. no polynomial-time algorithm exists for solving these problems. Metaheuristics provide an effective method of dealing with combinatorial optimization problems. Metaheuristics are discussed in next the section.

2.3.2 Metaheuristic Methods

Metaheuristic methods, unlike heuristic methods, provide a generalized framework to solve a variety of optimization problems, particularly combinatorial optimization problems, which are NP hard in general. These methods do not guarantee the optimality of the obtained solution but if implemented properly can be very effective in obtaining a near optimal solution in reasonable computation times. These methods start with a single feasible solution or a set of feasible solutions to a problem and guide the search based on certain properties of the currently available solution/solutions to reduce the size of the search space. Some of the most used metaheuristic methods are Simulated Annealing (SA) [22], Tabu Search (TS) [23], and GA [24, 25]. These are described as follows.

- **Simulated Annealing**: SA operates on a single solution at a time and looks for a better solution in the neighborhood of the current solution. If a better solution is found it is accepted, but if a better solution is not found it is still accepted with some
non-zero probability value. The acceptance of a solution which is not better than the current solution prevents the search from falling into a local optimum. However, the probability of selecting a worse solution keeps decreasing in every iteration according to a cooling procedure. Thus, the algorithm is guaranteed to converge. The length of the neighborhood and the cooling procedure are defined by the user.

- **Tabu Search**: TS also operates on one solution at a time and searches for a solution in the neighborhood of the current solution. However, it differs from SA in the way in which solutions are selected from the neighborhood. TS maintain a list of recently evaluated solutions, known as tabu, which are stored in the tabu list. In selecting a solution from a neighborhood, TS looks for the best solutions which are not tabu. Even if the solution is not better than the current solution, it is selected with some non-zero probability. This prevents the TS from falling into a local optimum and the tabu list prevents the solutions which were evaluated recent, from being picked again for evaluation. Thus, the search jumps from one region to another in looking for an optimal solution. The specific parameters like the size of a neighborhood, the tabu list, etc. are all problem-specific.

- **Genetic Algorithms**: The GA is a population-based metaheuristic and works on the principle of natural evolution. Instead of working with one solution at a time, the GA operates on a fixed-size set of solutions known as the population. In every iteration, the best performing individuals from the population are selected for generating new solutions for evaluation. These new solutions are generated by crossover and mutation operations. The rationale behind selecting the good individuals for mating is that they will combine to generate even better solutions, thereby increasing the quality of solutions in the population in each iteration. The GA allows us to both explore (global search) and exploit (local search) the solution space in searching for the optimal
solution. The GA does not define a stopping criterion and it must be told when to stop. There can be a number of stopping measures, like the number of generations, the goodness of the solution, the convergence of the population, etc., which can be used to stop the GA.

2.4 Existing Frameworks for Routing Protocol Design

Several component-based architectures for designing routing protocols have been developed by the MANET community. Some of these are discussed in this section.

Click Router [26] is one of the best known examples of an architecture for building MANET routers from discrete functional components. These components are known as elements in Click’s terminology. These elements are basically packet processing modules performing basic routing functions like queuing, scheduling, etc. The elements’ interconnection model in the Click architecture is defined using the Click language. Click however does not support automatic assembling of functional components to create routing protocols.

Another popular component-based architecture for protocol design is Component-Based Routing (CBR) [27]. The CBR architecture is the closest work to ours. It contains various routing functional components implementing different routing functionalities similar to those developed by us. CBR, however, does not mention anything about implementing DTN routing protocols and only supports manual combination of protocol components.

Some other frameworks like PICA [28] and Manetkit [29] focus mostly on the implementation of popular MANET routing protocols (AODV, DSR, DSDV, etc.). Some of these frameworks like the ManetKit do mention a pluggable component architecture but none of them supports automatic assembling of routing protocols.
2.5 Summary

In this chapter, a survey of the concepts which are used in the design of the Orchestra framework were presented. We also discussed some other existing frameworks from the literature. The next chapter focuses on the system modeling of the Orchestra framework utilizing the concepts discussed in this chapter.
Chapter 3

Orchestra Framework - Domain
Analysis and System Modeling

3.1 Introduction

In the last chapter, we presented a survey of various routing approaches and protocols used in mobile ad hoc networks and delay tolerant networks. The majority of these protocols were designed to tackle frequent and continuous changes in network connectivity due to mobility. In addition to mobility, a number of other network and node-dependent parameters like radio transmission range, limited network resources (bandwidth, contact opportunities, etc.), node resources (buffer space, power/battery life, etc.) can have a significant impact on network protocol performance. A number of routing protocols are also designed to address these additional challenges.

Protocol designs targeted at specific network scenarios or performance metrics appear promising on paper, but the complexity and cost of implementing and tuning a routing protocol
from scratch presents a major bottleneck in the protocol design process and is usually beyond the reach of an individual protocol designer. As a result, although not optimal, a one-protocol-suits-all approach is adapted in the majority of cases.

Also, due to limited human experience and capabilities, it is practically impossible to keep track of the plethora of research work going on in the field of network protocol design. Therefore, the manually crafted protocol design fails to utilize all the existing knowledge and may end up being a sub-optimal solution. Thus, a mechanism to automate the protocol design process is highly desirable.

### 3.2 Orchestra Framework

In order to address the problems presented in previous section, [1] proposed a novel framework called Orchestra - Open Architecture for Checking, Synthesis and Training of Routing Algorithms, which provides a flexible architecture for assembling and testing a variety of routing designs. In this work, we have extended the application of the Orchestra framework to the DTN domain and have carried out a major software redesign of Orchestra to make the software architecture model more generic, modular and scalable.

### 3.3 Orchestra Software Design

The Orchestra framework follows a component-based software architecture model [19] (discussed in the previous chapter) where the focus is on automated design of a protocol by combining different functional components. The functional component definitions cannot be arbitrary and should be based on some functionality provided by them in routing protocol design. Once these component definitions are established, we can develop generic interfaces
Figure 3.1: Conventional protocol design versus Orchestra’s protocol design mechanism.

around different classes of components, which then can be seamlessly integrated to create novel protocol designs. Orchestra envisions the collaboration of researchers from all over the world to create a collection of components, each implementing a unique routing functionality. We will call this collection Orchestra’s code database.

A component-based approach simplifies the process of assembling and testing a new routing design, providing network researchers with reusable functional components to be incorporated in their design. A researcher trying to test a particular new functionality only needs to develop a unique component implementing the new functionality following the generic interfaces defined for that class of component. All the other components are provided by Orchestra’s code database.

A component-based approach not only alleviates the difficulties of building every component of a routing protocol from scratch but also provides a platform to develop optimization algorithms for the automatic assembly of routing protocols. Various algorithms can be developed to explore the design space more efficiently as compared to manual assembly, thereby improving the protocol design process. Figure 3.1 shows a comparison between the conventional protocol design process and Orchestra’s protocol design mechanism.
In order to realize the above component-based architecture, we need to define clear and concrete boundaries and definitions of the functional components based on some routing functionality. Thus, an analysis of the features of ad hoc and DTN routing protocols is the topic of the next sections.

3.4 Domain Analysis

One of the important goals of the Orchestra framework is to provide researchers with an ample number of reusable components to be included in their designs. In order to create these reusable components, first we need to identify and catalog the common features of routing protocols belonging to different domains such as proactive ad hoc routing protocols, reactive ad hoc routing protocols, delay tolerant networking protocols, etc. Next, based on these common features, we can provide strong boundaries and definitions to create different classes of components providing the desired functionality. Once we are done creating different components, we can model a complete protocol by combining individual components from different categories. We also need to identify specific requirements of protocols from different domains to implement them in Orchestra.

3.4.1 Commonalities amongst different routing protocols

We studied various proactive ad hoc routing protocols, reactive ad hoc routing protocols, hybrid ad hoc routing protocols and delay tolerant networking protocols discussed in the previous chapter and identified the following features common to most of them.

- **Topology information collection mechanism**: The topology information in a network may include information such as route characteristics, known routes, link infor-
mation, history of previous encounters between nodes, knowledge of future encounters, etc. Most of the ad hoc routing protocols (both proactive and reactive) make use of information like route characteristics and link information to calculate the shortest possible route to the destination. However, in DTNs, as an end to end path may not exist, information such as the history of previous encounters of a node and future knowledge of encounters is used to make a decision about which data packets need to be exchanged between nodes. All the routing protocols we studied contain some mechanism to exchange such topology information.

- **Mechanism to implement routing and network policies**: In wired networking protocols like BGP, network and routing policies may include traffic engineering policies, scaling policies for congestion control, security policies, etc. [30]. In DTN protocols, there may be policies governing the exchange of information between nodes, for example, priorities of data packets to be forwarded in case of limited link duration, packet scheduling policies, etc. Similarly, in the case of ad hoc routing protocols, routing policies may be used to prioritize one path over another.

- **Mechanism to determine route preferences based on route characteristics and policies**: All ad hoc routing protocols contain some mechanism to calculate the optimum path based on topology information and routing policies. These mechanisms are often based on Dijkstra’s algorithm or the Bellman Ford algorithm. In case of DTN protocols, where information about future contacts between nodes may be available, a modified form of Dijkstra’s algorithm is often used [18]. In replication-based DTN routing, there is no need for any such mechanism, as no end-to-end path exists in the network and routing is based on a ‘store and carry forward’ scheme.

- **Mechanism to efficiently manage limited buffer space in a node**: This mechanism is important in case of replication-based DTN routing schemes, as there is a need
to store a large number of messages before the next transfer opportunity manifests itself. Buffer management policies like FIFO [2], evict most forwarded first [31], evict least probable to reach destination [31], evict oldest [32] and evict youngest [32] are implemented by different DTN routing protocols. In ad hoc routing protocols, a data packet is dropped when no path to the destination exists, so this mechanism may not be relevant.

- **Packet forwarding scheme to select the next hop and packets to be forwarded**: The packet forwarding scheme selects a route on which to forward the data packet. In source routing protocols like DSR [6], where the complete path is contained in the packet header, the packet forwarding mechanism is responsible for making sure that the data packet is forwarded to the correct node. In DTNs, especially replication-based protocols, data packet headers may contain additional information such as the number of replications allowed or the maximum number of hops allowed. The packet forwarding mechanism uses the additional information to implement strategies to minimize the use of network and node resources.

- **Path repair/maintenance mechanism in case of route breaks**: These mechanisms are mostly used in ad hoc routing protocols to conduct local or global route repairs in case of a route break. These mechanisms are responsible for monitoring the status of active paths and conducting repairs when required. As an end-to-end path may not exist in networks where DTN protocols are applicable, no such mechanism is present in DTN protocols.
3.4.2 Flow of Information in Routing Protocols

In order to correctly model any protocol in the Orchestra framework using functional components, we also need to have an understanding of the flow of information between different functional units for different categories of protocols. Based on the same protocols cited in the previous section, we identified that there is a fundamental difference in the flow of information for different categories of protocols. For example, in proactive routing protocols, paths/routes are created and stored in the routing table beforehand and are updated periodically. When a data packet is ready, the information from the routing table is used to forward the data packet to the appropriate neighbor. But, in reactive routing protocols, protocol action is triggered on the arrival of a new data packet and a route is calculated only after this event. Further, in DTN’s, the routing mechanism is ‘store and carry forward’ and protocol actions are triggered when a link is established between two nodes, and not on arrival of the data packet. Thus, there is a need to tackle these inherent differences in protocol mechanisms while designing the Orchestra framework.

3.5 System Modeling

In the previous section, we discussed several common features of ad hoc and DTN protocols. Based on that, we can divide the functional components required to model any protocol in Orchestra framework into six broad categories:

1. Topology Information Collector (TIC)
2. Routing Metric Component (RME)
3. Path Calculation Algorithm (PCA)
4. Buffer Management Component (BFM)

5. Packet Forwarding Engine (PFE)

6. Path Maintenance Component (PME)

Also, in order to model different varieties of protocols (proactive routing, reactive routing, DTN routing, hybrid routing, etc.) in the Orchestra framework, we introduce another layer of abstraction called the **routing strategy**. This layer of abstraction lets us model the flow of information as required by a particular type of protocol.

Figure 3.2 shows models of different categories of protocols in the Orchestra framework.

To provide a neighbor discovery mechanism for DTN protocols, we have integrated IMEP (Internet MANET Encapsulation Protocol) [33] with the Orchestra framework. IMEP works at the network layer in conjunction with other routing protocols to provide numerous functionalities, which may include network layer address resolution, link/connection status sensing, broadcast reliability, multipoint relaying, authentication, and message aggregation. We have utilized the link/connection status sensing functionality of IMEP protocols to facilitate neighbor discovery for DTN protocols. The integration of the IMEP protocol not only supports a neighbor discovery mechanism for DTN protocols but also provides other functionalities (as described above) which may be of importance in some routing protocols and thus broadens the scope of Orchestra as a generic framework for protocol design.

We will discuss the architecture of the Orchestra framework and implementation details in the next chapter.
Figure 3.2: Models of different categories of protocols in Orchestra.
3.6 Automated Protocol Design in the Orchestra Framework

The second major goal of the Orchestra framework is to automate the process of protocol design. Once we have modeled the system as comprising different functional components, the next step is to determine what possible combination of components yields the best routing protocol for a specific network scenario and a particular performance metric.

The performance metric for evaluation can be a network- and application-dependent metric, for example, latency, throughput, delivery ratio, a measure of network/node resources, or a weighted combination of any of these metrics. Determining the optimal combination is not a trivial task, especially given the large number of components (with their tunable parameters) envisioned in the Orchestra framework.

This problem of determining the best component combination from a finite set of discrete components belongs to a category of problems known as combinatorial optimization problems [34]. In the previous chapter, we discussed some metaheuristic methods like SA, TS and GA to tackle combinatorial optimization problems.

Any of these metaheuristics can be used in the Orchestra framework for finding an optimal combination of components. In this work, we have used a genetic algorithm-based approach to automate the process of protocol design. Chapter 5 deals with this topic in more detail.

3.7 Summary

In this chapter, we studied the common features of ad hoc and DTN routing protocols and created a component-based system model of the Orchestra framework. The features of the
Orchestra framework were also introduced throughout the chapter. In the next chapter, we will discuss in detail the architectural features and implementation details of the Orchestra framework and present the DTN-specific components built into Orchestra.
Chapter 4

Orchestra Architecture and Implementation

In this chapter, we will discuss the more specific design requirements, architecture and implementation details of the Orchestra framework. The chapter is organized as follows. First, we analyze the essential design requirements for the development of a generic framework like Orchestra. Then, we present Orchestra’s system architecture. This is followed by a more detailed discussion of the architectural features and design decisions involved in the development of the Orchestra framework. In the last section, we present DTN-specific components of the Orchestra framework.

4.1 Design Requirements

In order to create a generic framework like Orchestra, several design requirements need to be met. These requirements are discussed below.
• **Modularity:** The Orchestra framework aims at automating the process of routing protocol design. A number of algorithms will be used in Orchestra for creating new protocol designs by combining the components available in its code database. In order to explore a large design space, it is imperative that there should be no/minimum dependency amongst different sets of components. This will allow Orchestra’s design algorithms to seamlessly combine various available components to produce and test novel routing protocols. Thus, modular design of components is a major requirement.

• **Extensibility:** The Orchestra framework is intended to provide a platform for researchers to not only test their routing designs but also to save them from the burden of doing redundant work, by providing them with a number of reusable protocol components. These reusable protocol components are classified according to their routing functionality and are designed based on our knowledge of routing functions. However, there is a good possibility that the given set of components may not be able to represent all the routing functionalities to be created in the future and a need to add new classes of components to the Orchestra framework may arise. Thus, the system should be designed such that it supports the addition of new sets of components, i.e. the system should be extensible.

• **Scalability:** The Orchestra framework is conceived as a collaborative framework for developing novel routing designs where researchers and network engineers from all over the world would submit components to Orchestra’s component database. Such a system is expected to grow significantly with the submission of more and more component code. Thus, the Orchestra framework should be designed in way to incorporate this growth in graceful manner, i.e. a major modification in code/system should not be required upon the addition of new components. The system should be scalable.
4.2 System Architecture

The Orchestra framework is designed and prototyped in the ns2 simulator. In ns2, the protocols are implemented as agents operating in all nodes in the network. According to [35], agents represent endpoints where network layer packets are constructed or consumed, and are used in the implementation of protocols at the network layer. The Orchestra framework is implemented as a routing agent operating at the network layer in ns2. The Orchestra agent defines an interface to the transport layer and the data link layer. As discussed in section 3.5, we have also integrated the IMEP protocol layer to the Orchestra framework to provide the optional link sensing capabilities to realize DTN protocols in the framework. Figure 4.1 shows the system architecture of the Orchestra framework. The architectural features of the Orchestra framework are discussed in the next section and details of the optimization algorithms are presented in chapter 4.

4.3 Architectural Features

The Orchestra framework is developed as a completely object-oriented software system. In the following sections, we will present the architectural features of the Orchestra framework and discuss how those features address one or more of the above mentioned design requirements.

4.3.1 Interface-based Framework Design

The Orchestra framework is required to be a modular framework capable of accepting functional components developed by third parties (protocol designers, network researchers, etc.). In such a collaborative development paradigm, in order to produce a flexible, maintainable
Figure 4.1: System architecture of the Orchestra framework.
and reusable source code, there is a need to minimize the dependencies between different parts of the software system [36]. The interface-based development model offers an effective solution for this scenario and we have used it for the development of the Orchestra framework.

Interface-based development is based on the principle of separating the public interface of the software component from its concrete implementation [37]. Interfaces are implemented as abstract classes only containing the signature of interactions between different functional components of the system. The implementation of a functional component can be provided by a third party/user by coding to the interface defined for that class of component. Since the interaction between the components is only through their interfaces, dependency between different functional components of the system is greatly reduced and all components can be developed individually, making the architecture more flexible and easier to maintain.

As mentioned in section 3.5, we have divided the routing functionalities into six component classes, namely Topology Information Collector (TIC), Routing Metric Component (RME), Path Calculation Algorithm (PCA), Buffer Management Component (BFM), Packet Forwarding Engine (PFE), and Path Maintenance Component (PME). Each of these functional components has its interfaces defined as abstract classes from which classes implementing the unique functionality can be inherited. A researcher implementing a novel routing functionality just has to write a concrete component class implementing the interface defined in the Orchestra framework for that routing functionality. Figure 4.2 shows the interface-based architecture of the Orchestra framework.

Interface-based development not only helps us reduce the dependencies between different software components but also lets us utilize the concept of runtime polymorphism. Runtime polymorphism allows the late binding of component objects, facilitating the substitution of one component for another at runtime. This enables Orchestra’s optimization algorithms
Figure 4.2: Interface-based architecture of the Orchestra framework.
to replace one component for another while searching for the optimal solution (component combination).

4.3.2 Additional Layer of Abstraction

As mentioned in section 3.5, we have designed an additional layer of abstraction called the Routing Strategy Layer in the Orchestra framework. This abstraction layer serves two purposes. First, it lets us model the flow of information between different components of a routing protocol based on different routing strategies like proactive routing, replication-based DTN routing, etc. Secondly, it makes the Orchestra framework extensible. Since all the interactions between different functional components of a routing protocol are done via the Routing Strategy layer, if a novel routing functionality cannot be mapped onto one of the existing component classes there is always a provision for introducing new types of components into the system. Figure 4.3 shows the Routing Strategy layer abstraction in the Orchestra framework.

The Routing Strategy layer has its own inheritance hierarchy defined in the Orchestra framework. The abstract base class called 'RoutingStrategy' defines the public interface, which can be inherited by classes implementing specific routing behavior. The design of this abstraction layer is based on Strategy Design Pattern [21], in which different routing behaviors are encapsulated as objects and can be interchanged at runtime without requiring any modifications to the framework. This mechanism enables us to explore a variety of routing strategies for a specific network scenario. For example, modeling a routing behavior as a Routing Strategy object can help us compare an ad hoc routing strategy and a DTN-based strategy for a network setting, without the need of recompiling and making manual changes in the framework source code. Figure 4.4 shows this design scheme in more detail.
Figure 4.3: Routing strategy layer abstraction in the Orchestra framework.

Figure 4.4: Routing strategy model in the Orchestra framework.
4.3.3 Loose Coupling between Framework and Components

The Orchestra framework, to provide modularity, is designed to only use the interfaces defined for routing components while assembling a routing protocol. The interface for all the routing components is defined in abstract classes which cannot be instantiated. Thus, the Orchestra framework only has the knowledge of when to instantiate a specific component class, but the actual information about the components class type is not known to Orchestra at compilation time. This information is provided at runtime by the optimization algorithm operating outside the Orchestra routing agent or manually by using configuration files. Thus, we eliminate the need for binding component specific classes to the Orchestra framework.

In order to provide this loose coupling between framework classes and component-specific classes, we have introduced Factory classes into the framework design. These classes encapsulate the knowledge of which component subclasses to create and decouple them from the framework classes. They provide the parameterized interface for creating component objects at runtime. This design is based on the Factory Method design pattern [21]. Figure 4.5 illustrates this feature in more detail.

This decoupling of framework and component classes serves another important purpose. It makes the framework design scalable. In order to add a new routing component, the only thing required is to register the component with a factory class and optimization framework using a unique component ID. There is no need to make any changes to framework classes. During runtime, the optimization algorithm can just pass the component IDs to the Orchestra framework, which can call its factory classes to instantiate the required component objects.
4.4 Orchestra’s Components for DTN Protocols

In this section, we will present the components we specifically designed to implement DTN routing protocols in the Orchestra framework. We will limit our discussion to replication-based DTN routing protocols, as they are the most widely used protocols in delay tolerant networking. The replication-based DTN protocols can be implemented by using only four functional components, namely TIC, RME, BFM, and PFE. The PCA and PME components are not required in implementing these protocols because the routing mechanism used is ‘store and carry forward’ which does not depend upon the existence of an end-to-end path between source and destination.

We will discuss each functional component categorized according to the routing functionality provided by it.

- Topology Information Collector Components: We have developed three TIC compo-
nents, namely TIC\_IMEPLinkInfo, TIC\_IMEPProHist, and TIC\_IMEPMaxHist, for the implementation of DTN routing protocols. Each of these components uses the link sensing service provided by the IMEP routing protocol. The component TIC\_IMEPLinkInfo is the simplest of the three TIC components and only provides information about the presence of a link between two nodes in the network. This type of functionality is sufficient for implementing DTN routing protocols like Epidemic Routing [2], SWIM [38], MRP [39], Spray and Wait [11], etc. The components TIC\_IMEPProHist and TIC\_IMEPProHist, in addition to providing link information, process the link availability information to calculate the future probability of meeting between two nodes in the network. The component TIC\_IMEPProHist processes this information as described in [9], while the component TIC\_IMEPMaxHist uses the approach described in [10]. The probabilistic information about future node meetings is used in a number of DTN routing protocols, including ProPhet [9], MaxProp [10], and MV [15].

• Routine Metric Components: Two DTN-specific RME components are currently implemented in the Orchestra framework. The component RME\_DelayMinimize, as the name suggests, tries to minimize the average delivery latency of a data packet by following a policy of aggressively replicating the data packets at each transfer opportunity. The underlying assumption is that the greater the number of nodes carrying a copy of the packet, the sooner it will be delivered to the destination. On the other hand, the component RME\_NetResMin implements the policy of minimizing network resource consumption. One very important network resource is the amount of bandwidth available in the system. Each transmission of a packet requires some amount of network bandwidth. This component makes a decision about intelligently replicating the data packets to minimize the usage of network bandwidth without compromising delivery ratio.
• Buffer Management Components: The Buffer Management component implements various strategies to efficiently manage the limited buffer space at a node. These components are responsible for making a decision about which data packets need to be dropped when the limited buffer space of a node is exhausted. The Orchestra framework currently supports four BFM components: BFM_FIFO, BFM_EMFW, BFM_ELFW, and BFM_DLE. The component BFM_FIFO implements a simple first-in-first-out buffer management strategy used in [2, 9]. The components BFM_EMFW and BFM_EMLW use the hop count information available in the packet header. The component BFM_EMFW prioritizes the packets with a lower hop count value to stay in the buffer while BFM_ELFW follows the opposite approach. The component BFM_DLE orders the packets to be dropped based on their likelihood of getting delivered. The greater the chances of a packet being delivered, the higher its priority of staying in the buffer is. The decisions about packets delivery predictability can be based on a number of parameters like future probability of node meeting, user defined utility functions, etc.

• Packet Forwarding Components: Currently, the Orchestra framework contains two PFE components. These are PFE_HopbyHop and PFE_LimitCopies. The component PFE_HopbyHop implements a simple scheme of forwarding the data packet hop by hop until the hop count value is exhausted. This component is sufficient for many protocols like Epidemic Routing, ProPhet, and MaxProp. In DTN protocols, a packet’s copies are created at each transfer opportunity. Some protocols like Spray and Wait try to limit the number of copies of a packet in a network to achieve a certain performance objective. The component PFE_LimitCopies, in addition to a hop by hop forwarding scheme, also implements the functionality to limit the packet copies in a network to a predetermined value.
Table 4.1: DTN components in the Orchestra framework

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Parameters</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIC</td>
<td>IMEPLinkInfo</td>
<td></td>
<td>IMEP based link information exchange</td>
</tr>
<tr>
<td>TIC</td>
<td>IMEPProHist</td>
<td>$\alpha$, $\beta$, $\gamma$</td>
<td>IMEP based link information exchange + ProPhet based processing</td>
</tr>
<tr>
<td>TIC</td>
<td>IMEPMaxHist</td>
<td>$n$</td>
<td>IMEP based link information exchange + MaxProp based processing</td>
</tr>
<tr>
<td>RME</td>
<td>DelayMinimize</td>
<td></td>
<td>Simple DTN routing policy</td>
</tr>
<tr>
<td>RME</td>
<td>NetResMin</td>
<td>$p$</td>
<td>Routing Policy to minimize network resource consumption</td>
</tr>
<tr>
<td>BFM</td>
<td>FIFO</td>
<td>$b$</td>
<td>First in first out buffer management policy</td>
</tr>
<tr>
<td>BFM</td>
<td>EWLF</td>
<td>$b$</td>
<td>Evict least forwarded packet from the buffer</td>
</tr>
<tr>
<td>BFM</td>
<td>EWMF</td>
<td>$b$</td>
<td>Evict most forwarded packet from the buffer</td>
</tr>
<tr>
<td>BFM</td>
<td>EDLE</td>
<td>$b$</td>
<td>Evict least probable to reach destination</td>
</tr>
<tr>
<td>PFE</td>
<td>HopbyHop</td>
<td>$h$</td>
<td>Simple hop by hop packet forwarding</td>
</tr>
<tr>
<td>PFE</td>
<td>LimitCopies</td>
<td>$c$</td>
<td>Packet forwarding scheme limiting number of packet’s copies</td>
</tr>
</tbody>
</table>

Table 4.1 summarizes the functional components developed in the Orchestra framework for implementing DTN protocols.

4.5 Summary

In this chapter, we presented the architectural and implementation details of the Orchestra framework. We used various software engineering concepts like the design pattern, runtime polymorphism, etc., to make the design of the Orchestra framework scalable, modular and extensible. We also described the DTN-specific components of the Orchestra framework. In the next chapter, we will present our approach to automate the process of routing protocol design.
Chapter 5

Automatic Assembling of Routing
Protocols in the Orchestra Framework

In the previous chapters, we have established that a routing protocol can be assembled by combining reusable functional components designed in the Orchestra framework. In this chapter, we will present the approach of automating the protocol design process. We will start with formulating the problem of protocol design from functional components as a combinatorial optimization problem. Then, we will discuss our proposed GA-based solution. This is followed by the details of mapping the problem to a genetic algorithm and discussion GA operations in the context of our problem.

5.1 Problem Formulation

Each feasible combination of functional components with specific parameter settings is a unique routing protocol design. So, in designing a routing protocol for a given performance metric, the automation algorithm will have to evaluate numerous routing protocol designs.
Given the large number of components envisioned in the Orchestra framework, finding the optimal component combination may be a complex problem. A more detailed discussion of the problem follows.

Let there be $R$ TIC components, $S$ RME components, $T$ PCA components, $U$ BFM components, $V$ PFE components, and $W$ PME components in the Orchestra framework. Then, a unique protocol design comprising different component combinations and implementing a specific routing strategy can be represented as:

$$\Psi(\Upsilon, \Gamma_r, \Delta_s, \Theta_t, \Phi_u, \Omega_v, \Pi_w),$$

where $\Upsilon$ represents the routing strategy (proactive ad hoc, reactive ad hoc, replication based DTN, hybrid, forwarding based DTN, etc.); $\Gamma_r, r \in \{1, ..., R\}$ represents a TIC component; $\Delta_s, s \in \{1, ..., S\}$ represents a RME component; $\Theta_t, t \in \{1, ..., T\}$ represents a PCA component; $\Phi_u, u \in \{1, ..., U\}$ represents a BFM component; $\Omega_v, v \in \{1, ..., V\}$ represents a PFE component; $\Pi_w, w \in \{1, ..., W\}$ represents a PME component, and $i, j, k, l, m, n \in \{0, 1\}$ are binary variables which represent whether or not a particular type of component is present in a routing design.

Please note that the same types of components with different parameter values are considered as two different components in the above problem formulation.

In any networking application, there can be different performance metrics, for example, delivery latency, delivery ratio, throughput, or a weighted combination of any of these metrics. These performance metrics depend on various factors such as available bandwidth, network coding scheme, routing protocol used, etc. If all other factors are kept constant, the performance metric becomes completely dependent on the routing protocol used in the network.

Let us represent a performance metric for a specific network application as $PM$. Assuming
that all other factors are kept constant, the $PM$ of a network can be represented as

$$PM = f(\Psi(\Upsilon, i\Gamma_r, j\Delta_s, k\Theta_l, l\Phi_u, m\Omega_v, n\Pi_w)),$$

where function $f(\Psi)$ depends on the type of performance metric we are trying to evaluate.

Now, the fitness or suitability of any routing protocol for a specific network setting can be evaluated by measuring its ability to minimize or maximize the value of $PM$. Thus, we can formulate the problems as:

$$\text{Min or Max } PM = \text{Min or Max } f(\Psi(\Upsilon, i\Gamma_r, j\Delta_s, k\Theta_l, l\Phi_u, m\Omega_v, n\Pi_w))$$

The problem described above depends on the number of discrete variables representing different functional components of the Orchestra framework and belongs to a category of problems known as combinatorial optimization problems.

### 5.2 Proposed Solution - Genetic Algorithms

In the previous section, we formulated the problem of finding the optimal component combination as a combinatorial optimization problem which is NP hard in general. The GA provides a very effective way of tackling such problems. The GA is a population-based metaheuristic which works on the principle of 'survival-of-the-fittest.' We introduced GA in section 2.3.2. The steps of a GA are as follows.

1. **Initialization** - Generate a population of $n$ chromosomes (solutions).
2. **Evaluation** - Evaluate the fitness of each chromosome (solution) in the population.
3. **Testing** - If the stopping criteria is met, stop the algorithm. Else continue.
4. Generate new population using the following steps:
(a) **Selection** - Select two chromosomes (solutions) from the population. The probability of a selection of an individual is directly proportional to its fitness evaluated in step 2.

(b) **Crossover** - Cross over the two chromosomes (solutions) selected in the previous step to generate new chromosomes (solutions). The chances of crossover depend on the crossover probability.

(c) **Mutation** - Mutate (vary the parameter settings) the new chromosomes generated in the previous step. The chances of mutation depend on the mutation probability.

(d) **Replace** - Replace the chromosomes (solutions) with low fitness value in the population with newly generated chromosomes (solutions).

5. **Loop** - Go to step 2.

The population size (n), stopping criteria, crossover operation, crossover probability, mutation operation and mutation probability are user-defined entities and depend on the type of problem we are trying to optimize. We will discuss the details of GA used in the Orchestra framework in the next section.

### 5.3 Genetic Algorithms for the Orchestra framework

In this section we will discuss the implementation details of the genetic algorithm in the context of the Orchestra framework. According to [40], in order to solve a problem using the GA, three things are required.

1. Genetic representation of the solution domain.

2. Definition of genetic operators (mutation and crossover) in the solution domain.

3. A fitness function for the evaluation of individual solutions.

Once the above requirements are met, we can use the GA to find the best possible combination of functional components to create a customized routing protocol. Figure 5.1 shows
the mapping of a routing design problem to the GA framework and the following sections discuss the details of various GA operations in the context of our problem.

5.3.1 Representation

In order to use a GA in solving the problem of automated protocol design, we need to define a representation of our solution in genetic format. Each component type (TIC, RME, PCA, BFM, PFE, and PME) in the Orchestra framework represents a genotype and the different functional components (TIC.IMEPLinkInfo, RME.NetResMin, BFM.FIFO, PFE.HopbyHop, etc.) implementing the specific routing functionality represent a gene in the GA. The possible solutions in a GA are known as chromosomes (or genome) consisting of a combination of different types of genes. Thus, genes (components) can be combined to create a chromosome (routing design) which represents a solution to our problem. Figure 5.2 shows the representation of a routing design from functional components in genetic format.

5.3.2 Initialization

The initialization phase involves generating some random feasible solutions to our problem from which the optimal design can evolve. Although we have designed generic interfaces for functional components, not all the functional component combinations result in a logical routing protocol design. However, the compatibility analysis of different functional components is not a part of this work and is an open issue which needs to be addressed in the future. In this work, we assume that all the possible component combinations represent a feasible solution to the routing design problem. The incompatible designs are tackled with the help of a fitness function (discussed in the next section) by assigning a negative fitness score to incompatible routing designs. Also, in order for the GA to explore the complete
Figure 5.1: GA for automatic assembling of routing protocols in the Orchestra framework.

Fitness evaluation of each solution (components combination) in the population done either in a simulation environment or on a real-network test bed. The performance metric for evaluation can be network and application dependent. (example: latency, throughput, delivery ratio or a weighted combination of any of these metrics).

Stop, if the desired performance objective is met or sufficient convergence is achieved.

Otherwise, selection of a better individual from the population (based on fitness value calculated in the evaluation step) to apply genetic operators to produce a new population of solutions for evaluation.

The mutation operation involves a variation of component parameters to achieve randomized self-adaptation of individuals to avoid falling in a local optimum. (Examples of component parameters can be frequency of update messages in TIC, packet hop limit in PFE, etc.)
search space without falling into a local optimum, we have to make sure that during the initialization phase, all the components get represented in the initial population of feasible solutions. The GA framework [41] used by us lets us satisfy this requirement.

### 5.3.3 Fitness Function and Design Evaluation Mechanism

The fitness function used for the evaluation of a routing design depends on the type of routing metric we are trying to optimize. For example, in case of DTN, if the metric in consideration is packet delivery ratio, then our fitness function will assign a higher score to a routing design (chromosome) producing a higher delivery ratio. Similarly, if the performance objective is to lower the packet delivery latency, a routing design with a lower average packet delivery latency will score higher than one with high delivery latency.

It may be mathematically intractable to model the fitness function for evaluating the performance of a routing design, so we have used simulation as the evaluation mechanism. For each routing design (chromosome) which needs to be evaluated, we perform ns2 simulations and pass the results back to the GA. Figure 5.3 illustrates this scheme.
5.3.4 Selection

A simple tournament selection scheme [42] is used for selecting the individuals for mating. In this scheme, some random individual (chromosomes) from the population of solutions are first chosen for selection. Now out of these individuals, selection is done based on the fitness value of the solution. This process is repeated a number of times in every iteration. There are many other selection schemes like roulette wheel selection [42], ranking based selection [42], etc., which could be used in place of tournament selection but we have not explored them in this work.

5.3.5 Crossover

The crossover operation is used in the GA to create new members (new routing designs) by recombining genes (components) of the members selected for mating in the selection step. The underlying assumption in performing the crossover is that a solution created by combining genes (components) from good solutions (routing designs) will create an even better solution (routing design). We have used a n-point crossover mechanism in which the number of points where crossover can take place is fixed. Figure 5.4 shows the fixed crossover
5.3.6 Mutation

The mutation operation is used to explore the genetic diversity of possible routing designs. The mutation operator on a routing design is defined to randomly switch either a component type, or the parameter values, or both. This operation prevents the GA from falling for a local optimum solution in the design space. The chances of mutation are controlled by a parameter known as mutation probability. The value of the mutation probability should not be high, otherwise the GA will become a randomized search operation. Figure 5.5 illustrates the mutation operation.
5.4 Types of Genetic Algorithms used in the Orchestra Framework

Two types of GA are used for automated protocol design in the Orchestra framework. These are presented below:

1. **Simple Genetic Algorithm**: This is the most basic form of GA, which was proposed in [24]. In a simple GA, the population between two generations is not overlapped. An entirely new population of solutions for evaluation is created in every generation. In the context of our problem, this means that for each new generation of a population, new routing designs created after crossover and mutation operations replace the previous routing designs.

2. **Steady State Genetic Algorithm** [43]: In a steady state GA, the populations between different generations of GA are overlapped. This implies that for each new generation only some part of the population of routing designs is new.

The evaluation of our GA-based approach to automated protocol design problem is performed in the next chapter. We will also present a comparison between the two types of GA in the context of the protocol design problem and present the results of routing protocols designed by the GA for various performance metrics.
Chapter 6

Evaluation and Analysis

The Orchestra framework is an open-ended protocol design framework which is expected to expand as more and more functional components are added by researchers and protocol designers. Thus, we cannot say, at any point in time, that the framework is complete. However, in order to prove the feasibility of the Orchestra framework, in this chapter we will perform an evaluation of protocols designed using a combination of different functional components and will show that a variation in even one or two functional components can significantly improve the performance of a routing protocol. We will limit our discussion to the replication-based DTN routing protocols but the concept can be readily verified for other protocol domains. We will start with the design of an Epidemic Routing (ER) [2] scheme, which is the basis of most replication-based DTN routing protocols. Then, we will discuss the different performance metrics used in the evaluation of DTN routing protocols. This is followed by a comparison of various protocol designs assembled using different components combinations in the Orchestra framework. In the last section, we will evaluate the GA-based approach, presented in the last chapter, for automating the routing protocol design process.
6.1 Case Study - Epidemic Routing

We discussed the details of replication-based DTN routing protocols and epidemic routing in chapter 2. Epidemic routing [2] is a replication-based DTN routing scheme in which every node keeps an index of all the messages stored in its buffer. This is known as a summary vector. When two nodes meet, they exchange summary vectors. Each node determines if the other currently stores messages not present in its own buffer and requests only those messages from the other node. In this manner, a message is spread across the network and is eventually delivered to the destination.

The operations of ER can be mapped onto the components of the Orchestra framework as follows: The presence of a link between two nodes is indicated by TIC.IMEPLinkInfo in the TIC component. RME_DelayMinimize, one of the components of RME, is used to determine which messages need to be requested from the other nodes. The BFM_FIFO component is used for buffer management and PFE_HopByHop is used as the packet forwarding scheme. The flow of information between different functional components is controlled by the DTNRoutingStrategy layer.

The ER protocol designed in the Orchestra framework is tested under the same simulation settings used in [2]. Figures 6.1 and 6.2 show the variations in delivery ratio and packet delivery latency for ER implementation in the Orchestra framework and ER as given in [2].

It can be seen from the two figures that the ER implementation in the Orchestra framework behaves similarly to the original ER implementation. For both implementations, the delivery ratio increases with an increase in the buffer capacity of the node. The delivery latency also varies with buffer capacity in a similar fashion for both implementations. However, a

---

1Data from [2] is used in the graphs.
difference in specific values, particularly latency, is observed. This might be due to differences in the implementation code.

### 6.2 Performance Metrics for DTN protocols

In the previous section, the details of designing a simple ER protocol from functional components of the Orchestra framework were presented. The ER scheme is designed to minimize the delivery latency of a data packet and might be the most appropriate choice of routing protocol when there is no limit on the amount of resources available in the network and at a node. However, in practical situations this might not be the case. The amount of network resources, like the network bandwidth, and node resources, like buffer capacity, is limited. In many applications, for example, wild life tracking applications like the Zebranet [44], meteorological applications for recording data such as temperature, relative-humidity using mobile sensor networks, etc., delivery latency is not the most important metric to optimize. Instead, the focus is on making the efficient use of available resources like network bandwidth, buffer
capacity of a node, contact opportunities, etc. The bottomline is that there can be different performance metrics which need to be optimized depending on the type of DTN application. Some of these important performance metrics for DTNs are as follows.

- **Delivery Ratio**: It is defined as the ratio of the total number of unique messages/bundles successfully received at the destination nodes to the total number of messages/bundles transmitted by the source nodes.

- **Data Efficiency** [45]: It is defined as the the ratio of the total number of unique messages/bundles received at the destination nodes to the total traffic generated, including copies, in the network.

- **Average Delivery Latency**: It is defined as ratio of total delay of received messages/bundles to the total number of received messages/bundles.

- **Average Cost**: Average cost is the average number of forwards (hops) per message before it is received at the destination.

- **Routing Efficiency** [46]: Routing efficiency is defined as the ratio of the delivery ratio to the average cost.

In the next section, these performance metrics will be used to evaluate the protocol designs assembled using different functional components.
6.3 Comparison of Routing Designs Assembled from Different Functional Components

In order to show the advantages of designing a routing protocol from functional components, the performance of routing designs assembled from different functional components is compared with the ER protocol’s performance. It has been already discussed in section 6.1 how the ER protocol can be designed from functional components available in the Orchestra framework. In this section, first the details of our experimental design are presented. This is followed by the comparison of protocols generated by variations in the TIC, RME, and PFE components for different performance metrics. Then, an evaluation of the BFM policies with different PFE components is carried out by keeping the TIC and RME components unchanged.

6.3.1 Experiment Design

The different DTN protocol designs are evaluated using ns2 simulations. Each simulated node in our implementation contains an Orchestra agent responsible for carrying out the routing layer operations. This routing agent comprises different functional components and implements a DTN routing strategy similar to ER. The detailed design of the Orchestra agent is discussed in chapter 4.

In these simulations, we model 20 mobile nodes moving in a 1500 m x 300m area, with node speed ranging between 0-10 m/s. The communication range of each node is set to 50m. This implies that each node’s radio coverage area is only about 0.02 times ($\pi r^2/1500 \times 300$ m$^2$) the total simulation area. This is a good setup for testing DTN routing protocols, as most
of the ad hoc routing approaches will fail to establish a source to destination path with such low node densities.

For the data traffic, messages of size 1KB are generated using the CBR agent in ns2. The communication pattern used in our simulations is as follows: Every node in the simulation acts as transmitter and receiver. Each of the 20 nodes generates 1 message per second for all the other 19 nodes. Thus, a total of 380 messages are generated. After, the first 380 messages are generated, the same process is repeated two more times to generate a total of 1140 (380+380+380) messages in the system.

It is assumed that the messages are uniformly distributed amongst different nodes in the simulation area. Also, each node keeps only one unique copy of a message at any time. So, the average number of messages required to be stored at any node during the simulation is 60 (1140/20). Thus, in these simulations, the buffer capacity of nodes is varied from 10KB (small buffer space) to 120KB (practically unlimited buffer space) to capture effects of the available buffer space on routing protocol’s performance.

We run the simulations for 5000 seconds.

Table 6.1 lists the notations used to represent different functional components of the Orchestra framework in the experiments.

### 6.3.2 Protocol designs with Variations in TIC, RME, and PFE Components

There can be many different component combinations which constitute a routing protocol. However, it is not possible to evaluate each possible component combination manually. In order to show the advantages of a component-based approach in designing a customized
Table 6.1: Component’s notations

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMEPLinkInfo</td>
<td>Topology Information Collector</td>
<td>T1</td>
</tr>
<tr>
<td>IMEPProHist</td>
<td>Topology Information Collector</td>
<td>T2</td>
</tr>
<tr>
<td>IMEPMaxHist</td>
<td>Topology Information Collector</td>
<td>T3</td>
</tr>
<tr>
<td>DelayMinimize</td>
<td>Routing Metric Engine</td>
<td>R1</td>
</tr>
<tr>
<td>NetResMin</td>
<td>Routing Metric Engine</td>
<td>R2</td>
</tr>
<tr>
<td>FIFO</td>
<td>Buffer Manager Component</td>
<td>B1</td>
</tr>
<tr>
<td>EWMF</td>
<td>Buffer Manager Component</td>
<td>B2</td>
</tr>
<tr>
<td>EWLF</td>
<td>Buffer Manager Component</td>
<td>B3</td>
</tr>
<tr>
<td>EDLE</td>
<td>Buffer Manager Component</td>
<td>B4</td>
</tr>
<tr>
<td>HopbyHop</td>
<td>Packet Forwarding Engine</td>
<td>P1</td>
</tr>
<tr>
<td>LimitCopies</td>
<td>Packet Forwarding Engine</td>
<td>P2</td>
</tr>
</tbody>
</table>

A routing protocol for a given routing metric, a study of some protocol designs assembled using different combinations of routing components is carried out. The details of why the assembled routing design behaves in a particular way are not presented, as this would require a sophisticated protocol analysis and verification mechanism which is out of the scope of this work. The purpose of this study is to establish the fact that different component combinations result in different routing protocols suitable for a specific performance metric.

The analysis is divided into two sections. In this section, the study of protocol designs obtained by variations in the TIC, RME, and PFE components is performed and their performance for a specific routing metric is evaluated. In the next section, the routing protocol’s performance is analyzed with variations in BFM and PFE components.

The component R2 makes use of the information about the future probability of meeting between two nodes in the network, which can be provided by components T2 and T3 but not T1. Also, the component R1 does not make use of any such information. Thus, although valid, a routing design involving components T1 and R2 or T2/T3 and R1 together performs similarly to routing designs using components T1 and R1. Thus, we have not included the
designs having these component combinations in our study. The BFM component (B1) is kept fixed for the designs considered in this section.

Figure 6.3 shows the performance of different routing designs assembled from combinations of TIC, RME and PFE components, with delivery latency as the performance metric. As shown in the figure, the component combination T1R1B1P2 produces the protocol design with minimum delivery latency. On the other hand, we can see from figure 6.4 that the optimal protocol design for delivery ratio as the performance metric is given by the combination T3R3B1P2 when the buffer capacity of the node is more than 60KB. Both T2R2B1P2 and T3R3B1P2 perform equally well for a buffer size less than 60KB. Again, for the performance metric of data efficiency, T2R2B1P2 and T3R2B1P2 perform more or less equally, with T3R2B1P2 performing slightly better for buffer sizes less than 60KB. This is shown in Figure 6.5. Finally, as shown in Figures 6.6 and 6.7, the optimal routing protocol design for performance metrics average cost and resource efficiency is given by the combination T2R2B1P2.

It is observed that for all the DTN metrics used in the evaluation, we achieve a signifi-
Figure 6.4: Delivery ratio comparison of routing designs using various TIC, RME, and PFE components.

Figure 6.5: Data efficiency comparison of routing designs using various TIC, RME, and PFE components.

Figure 6.6: Average cost comparison of routing designs using various TIC, RME, and PFE components.

Figure 6.7: Resource efficiency comparison of routing designs using various TIC, RME, and PFE components.
6.3.3 Protocol designs with Variations in BFM and PFE Components

To compare the protocols designs with different combinations of BFM and PFE components, the TIC and RME components are fixed to T1 and R1, respectively. Figure 6.8 shows that the combinations T1R1B1P2 and T1R1B4P2 perform equally well when the performance metric is delivery latency. The combination T1R1B2P2 performs best for the performance metric delivery ratio. This is shown in Figure 6.9. The most data-efficient combination is T1R1B3P2, which is shown in Figure 6.10. For performance metrics average cost and resource efficiency, the choice of components to assemble a protocol is T1R1B2P2, as shown in Figures 6.11 and 6.12, respectively.

Please note that the component B4 makes use of the information about future probability of node meeting to prioritize the data packets in the buffer. This information can only be provided by components T2 and T3. However, the TIC component used in this comparison is only T1. Thus, in the absence of information about future probability of node meeting, the component B4 operates similarly to component B1. Thus, the designs involving BFM components B1 and B4 perform equally in this experiment. However, when the component B4 is used in conjunction with component T2/T3, we expect significant differences in the performance of designs using B4 from the designs using B1.

From the last two sections, it can be seen that protocols generated from different component combinations perform significantly better than the ER routing protocol. This further substantiates the fact that designing a protocol from interchangeable functional components
Figure 6.8: Delivery latency comparison of routing designs using various BFM and PFE components.

gives us the freedom to tailor a protocol for a specific routing metric and is much better than the one-protocol-suits-all approach of protocol design.

## 6.4 Performance Evaluation of GA for Automated Protocol Design

In the previous sections, it was established that a significant improvement in routing performance can be achieved by designing protocols customized for a specific routing metric. Also, it was shown in chapter 5 that designing the optimal protocol from functional components for a specific network setting and performance metric is a combinatorial optimization problem. We proposed a GA based solution for this. We also discussed the two varieties of GA, namely Simple GA and Steady State GA, used in the Orchestra framework. In this section, an evaluation of the performance of the GA-based approach to protocol design is carried out. First, the details of the experiment design are presented. Then, a comparison is shown.
Figure 6.9: Delivery ratio comparison of routing designs using various BFM and PFE components.

Figure 6.10: Data efficiency comparison of routing designs using various BFM and PFE components.

Figure 6.11: Average cost comparison of routing designs using various BFM and PFE components.

Figure 6.12: Resource efficiency comparison of routing designs using various BFM and PFE components.
between a Simple GA and Steady State GA. Finally, the results of the optimal protocols, for different routing metrics, produced by the GA are presented.

### 6.4.1 Experiment Design

The Orchestra framework is still in its early stages of development and does not contain many protocol components. So, there cannot be many component combinations and an exhaustive search will do the task as efficiently as any other search algorithm. Thus, in order to show the advantages of using a GA based approach in searching for the component combination which constitutes the optimal routing protocol design, we need to devise a way to expand our search space. So, in our evaluations, we have considered the buffer capacity of a node to be a variable parameter. This will greatly expand our search space and will let us realize the advantages of GA over an exhaustive search. The use of buffer capacity as a variable parameter serves another important purpose. It lets us verify the effectiveness of the GA in the context of the Orchestra framework. The GA, if implemented correctly and effectively in searching for the optimal combination, will drive the search into the region of higher buffer capacity for performance metrics delivery ratio, data efficiency, and routing efficiency. This can be observed from the graphs shown in the previous section, where a higher value of delivery ratio, data efficiency and routing efficiency were observed at a higher buffer capacity. Besides the buffer capacity, the other variables used in our search are TIC, RME, BFM, and PFE. We are still in the process of developing a uniform interface for parameterizing the TIC and RME components, so that they can be represented uniformly as a chromosome in the GA. Thus, the parameters of TIC and RME components are hard coded at the moment.

The GALib [41] library is used for implementing the GA for the Orchestra framework. The
Table 6.2: GA parameters used in the experiments

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genome</td>
<td>Bin2DecGenome</td>
</tr>
<tr>
<td>Scaling</td>
<td>Linear</td>
</tr>
<tr>
<td>Population size</td>
<td>30</td>
</tr>
<tr>
<td>Crossover Probability</td>
<td>0.6</td>
</tr>
<tr>
<td>Mutation Probability</td>
<td>0.05</td>
</tr>
<tr>
<td>Stopping Criteria</td>
<td>97% of the optimal</td>
</tr>
<tr>
<td>Number of Generations to check for Convergence</td>
<td>30</td>
</tr>
</tbody>
</table>

representation of our solution is represented as a Binary to Dec genome in GAlib. The other parameter settings are listed in Table 6.2.

6.4.2 Comparison of Simple GA and Steady State GA used in the Orchestra framework

As discussed in section 5.4, two types are GA are used in the Orchestra framework for automating the process of routing protocol design. In order to compare their performance, both algorithms were used for finding the optimal component combination to constitute a routing protocol, with resource efficiency as the performance metric. The simulation settings used for the evaluation of any design assembled during the algorithm run are the same as those used in section 6.3.

For resource efficiency as the performance metric, the optimal component combination found by the simple GA algorithm is T3R2B2P2 with buffer capacity of 118KB and the resource efficiency value of 0.56939. The number of iterations taken by simple GA algorithm was 928. The optimal component combination found by the Steady State GA is T3R2B2P2 with buffer size of 114KB and the resource efficiency value of 0.563816. The number of iterations
taken by the Steady State GA was 331. Figures 6.13 and 6.14 show the comparison of the performance of both GA algorithms used in the Orchestra framework.

If we would have resorted to an exhaustive search for finding the most optimal component combination, the number iterations taken would have been 5670 ( 3(TIC) x 2(RME) x 4(BFM) x 2(PFE) x 120(Buffer capacity)) even without varying the parameter settings. On the other hand, we can get the nearly optimal solution in 928 iterations with the Simple GA and 331 iterations with the Steady State GA. This is a significant performance improvement.

It is also observed that although the Simple GA finds a better solution with resource efficiency value of 0.56939 as compared to the solution found by the Steady State GA, with resource efficiency value of 0.563816, the difference in the performance metric value is negligible. However, the number of iterations taken by the Simple GA is much more than that of the Steady State GA.

The Steady State GA converges to a search space in the neighborhood of the optimal solution much faster than the Simple GA. This is shown in Figures 6.15 and 6.16. Thus, if the stopping criterion of search is set to a limited amount of time, the likelihood of finding a
better solution is much higher with the Steady State GA as compared to the Simple GA in case of the protocol design problem addressed by the Orchestra framework.

It was stated in Section 6.4.1 that the correct implementation of the GA will drive the search into the region with higher buffer capacity for the performance metric of resource efficiency. This can be verified from Figures 6.17 and 6.18. Thus, we can conclude that the protocol design assembled by the GA used in the Orchestra framework is indeed a very good, if not optimal, routing design.

6.4.3 Protocols Generated by GA for different Routing Metrics

In the previous section, the two varieties of GA were compared in the context of our problem. It was found that a steady state GA converges much faster than a simple GA for our problem of automated protocol design. Thus, only the steady state GA was used in finding the optimal component combinations for other routing metrics. The GA parameter settings and the ns2
Figure 6.17: Buffer capacity variations in the simple GA.

Figure 6.18: Buffer capacity variations in the steady state GA.

Simulations settings used in obtaining the results presented in this section were the same as those in the previous sections. Table 6.3 summarizes our results.
Table 6.3: Protocols generated by GA for different routing metrics

<table>
<thead>
<tr>
<th>Routing Metric</th>
<th>Metric Value</th>
<th>TIC</th>
<th>RME</th>
<th>BFM</th>
<th>PFE</th>
<th>Buffer Size</th>
<th>No. of Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivery Ratio (Max)</td>
<td>90.78%</td>
<td>T3</td>
<td>R2</td>
<td>B1</td>
<td>P2</td>
<td>120</td>
<td>469</td>
</tr>
<tr>
<td>Data Efficiency (Max)</td>
<td>0.18428</td>
<td>T3</td>
<td>R2</td>
<td>B3</td>
<td>P2</td>
<td>47</td>
<td>392</td>
</tr>
<tr>
<td>Delivery Latency (Min)</td>
<td>476.719 s</td>
<td>T1</td>
<td>R1</td>
<td>B1</td>
<td>P2</td>
<td>45</td>
<td>523</td>
</tr>
<tr>
<td>Average Cost (Min)</td>
<td>1.63</td>
<td>T2</td>
<td>R2</td>
<td>B2</td>
<td>P2</td>
<td>11</td>
<td>599</td>
</tr>
<tr>
<td>Resource Efficiency (Max)</td>
<td>0.563816</td>
<td>T3</td>
<td>R2</td>
<td>B2</td>
<td>P2</td>
<td>114</td>
<td>331</td>
</tr>
</tbody>
</table>
Chapter 7

Conclusion

7.1 Summary of Results

In this work, first a major software redesign of the Orchestra framework was carried out. The development of the framework architecture was based on various software engineering design practices. Although the development and modeling of the architecture took a substantial amount of time, once the architecture was developed, the time taken for implementing and testing different protocol designs was greatly reduced. As an example of developing protocols from functional components, the ER protocol was modeled in the Orchestra framework. This was discussed in section 6.1. It was observed that the implementation of the ER in the Orchestra framework behaved similarly to the original implementation given in [2]. This proved the feasibility of the approach of designing a routing protocol from functional components.

Next, in order to substantiate our claim that different component combinations produce different routing protocols which are optimal for a specific routing metric, in section 6.3,
manual testing of some DTN routing strategies assembled using the valid combinations of DTN functional components available in the Orchestra framework was performed. Significant performance improvement was obtained, for all the five DTN metrics defined in section 6.2, with protocols generated from different component combinations when compared to a simple ER protocol. We also obtained different optimal component combinations for different routing metrics. This justified our motivation of designing customized routing protocols for a specific routing metric.

In the last part of this work, an evaluation of GA based approach for automated protocol designing, presented in chapter 5, was performed. The GA was successful in finding a near optimal, if not optimal, solution to the problem. The results of the best protocols designed by GA for different routing metrics were summarized in section 6.4.3. The comparison between two varieties of GA was also performed and it can be concluded that the steady state GA is much more suitable than the simple GA for our routing design problem. The comparative analysis was presented in section 6.4.2.

7.2 Future Work

In this work, we have successfully shown that a routing protocol can be designed using reusable functional components present in the Orchestra framework. However, given the generic interfaces developed across different components numerous combinations of components are possible. However, not all of these combinations produce a logical routing protocol design. So, in order to detect and eliminate these component combinations from the search space, a protocol analysis and verification mechanism is a highly desirable feature in the Orchestra framework. The presence of such a mechanism can significantly improve the performance of GAs used in the Orchestra framework. Also, the Orchestra framework is
currently implemented in the ns2 simulator and relies on simulation results in evaluating a routing design assembled using functional components. So, a logical next step would be to test the feasibility of the concept on a real network testbed.
Appendix A

Acronyms

MANET  Mobile Ad Hoc Network
DTN  Delay Tolerant Network
SA  Simulated Annealing
TS  Tabu Search
GA  Genetic Algorithm
TIC  Topology Information Collector
RME  Routing Metric Component
PCA  Path Calculation Algorithm
BFM  Buffer Management Component
PFE  Packet Forwarding Engine
PME  Path Maintenance Component
ER  Epidemic Routing
Bibliography


