Quo Vadis - a framework for intelligent routing in large communication networks

Armin Robert Mikler
Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/rtd

Part of the Applied Mechanics Commons, Computer Sciences Commons, and the Operational Research Commons

Recommended Citation
http://lib.dr.iastate.edu/rtd/10964

This Dissertation is brought to you for free and open access by Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Quo Vadis - a framework for intelligent routing in large communication networks

by

Armin Robert Mikler

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Department: Computer Science
Major: Computer Science

Approved:
Signature was redacted for privacy.

In Charge of Major Work
Signature was redacted for privacy.

For the Major Department
Signature was redacted for privacy.

For the Graduate College

Members of the Committee:
Signature was redacted for privacy.

Iowa State University
Ames, Iowa
1995

Copyright © Armin Robert Mikler, 1995. All rights reserved.
DEDICATION

To my parents, Erwin and Christel Mikler.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGEMENTS</th>
<th>ix</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 1. BACKGROUND</td>
<td>1</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Routing in Large Networks</td>
<td>3</td>
</tr>
<tr>
<td>Basic Routing Algorithms</td>
<td>3</td>
</tr>
<tr>
<td>Approaches to Reducing Overhead</td>
<td>5</td>
</tr>
<tr>
<td>Overview</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 2. QUO VADIS - A FRAMEWORK FOR INTELLIGENT ROUTING</td>
<td>9</td>
</tr>
<tr>
<td>Background</td>
<td>9</td>
</tr>
<tr>
<td>A Prototype Design of Quo Vadis</td>
<td>13</td>
</tr>
<tr>
<td>Knowledge Representation in Quo Vadis</td>
<td>14</td>
</tr>
<tr>
<td>Routing and Control in Quo Vadis</td>
<td>18</td>
</tr>
<tr>
<td>Summary</td>
<td>21</td>
</tr>
<tr>
<td>CHAPTER 3. A SIMULATOR FOR LARGE COMMUNICATION NETWORKS</td>
<td>23</td>
</tr>
<tr>
<td>Introduction</td>
<td>23</td>
</tr>
<tr>
<td>Framework for Simulating Large Network</td>
<td>24</td>
</tr>
</tbody>
</table>
The Model ........................................... 25
Simulation Methodology ................................. 29
Design and Implementation of a Simulation Toolbox .......... 32
The Object Oriented Approach .......................... 32
The Simulation Testbed .................................. 34
Implementation of the Model ......................... 39
Pre vs. Post Scheduling ................................ 46
A Priori Service Time .................................. 49
Verification of the Simulation Environment ................. 50
Summary .................................................. 55

CHAPTER 4. EXPERIMENTAL STUDY OF QUO VADIS ....... 57
Simulation Experiments and Results ....................... 57
The Effects of α ........................................ 59
The Effects of β ........................................ 62
The Effects of γ ........................................ 68
Summary .................................................. 70

CHAPTER 5. ROUTING DECISIONS BASED ON UTILITIES .... 72
Rewards and Penalties ................................... 72
The Reward Function ................................... 73
Penalty and Payoff ...................................... 74
Routing with Utilities ................................... 76
Summary .................................................. 90

CHAPTER 6. CONCLUSION AND FUTURE WORK ............. 91
Summary .................................................. 91
Future Work .......................................................... 92
  Evaluation of Quo Vadis in Different Network Topologies ........... 93
  Handling Node and Link Failures and Other Pathologies Gracefully . 93
  Evaluation of Quo Vadis in a Dynamic Network Environment .... 94
  Parametric Studies of Knowledge Representation and Decision Functions 95
  Making Quo Vadis Adaptive Using Machine Learning ............... 95
  Refining the Theoretical Framework .............................. 96

BIBLIOGRAPHY ...................................................... 98
LIST OF TABLES

Table 3.1: Expected vs. actual message delay in an M/M/1 queue for different values of $\rho$ ............................................. 52

Table 3.2: Expected vs. actual message delay in an M/D/1 queue for different values of $\rho$ ............................................. 53

Table 4.1: Mean Hop Count ($\bar{h}$) and Mean Message Delay ($\bar{d}$) for different values of $\beta$. ($n > 85700$ messages) ......................... 64

Table 4.2: Points of deflection for different values of $\gamma$ ......................... 70
# LIST OF FIGURES

| Figure 2.1: | Delay vs. utilization | 12 |
| Figure 2.2: | A superimposed coordinate system | 14 |
| Figure 2.3: | Possible position of $n_i, n_k, n_d,$ and $n_p$ in a network | 20 |
| Figure 3.1: | Event-driven simulation cycle | 31 |
| Figure 3.2: | Decomposition tree | 34 |
| Figure 3.3: | Event-class header file | 38 |
| Figure 3.4: | Network-class header file | 41 |
| Figure 3.5: | A canonical node structure | 42 |
| Figure 3.6: | Load distribution in a 1024 node grid network using shortest path routing | 55 |
| Figure 4.1: | 100 node network with its corresponding load landscape | 60 |
| Figure 4.2: | $V^{E}_t$ for $\alpha = 1.0$ and $\alpha = 0.6$ respectively | 61 |
| Figure 4.3: | $V^{E}_t$ for $\alpha = 0.3$ and $\alpha = 0.1$ respectively | 61 |
| Figure 4.4: | Load distribution in a 1024 node grid network using shortest path routing i.e., $\beta = 1.0$ and $\beta = 0.4$ respectively | 63 |
| Figure 4.5: | Effects of different values of $\beta$ on $\bar{d}$ and $\bar{h}$ | 65 |
Figure 4.6: Effects of sudden load increase in node \( n_i \) under Shortest Path Routing .......................... 67

Figure 4.7: Effects of Quo Vadis on sudden load changes in node \( n_i \) .......................... 68

Figure 4.8: 100 node network with its corresponding load landscape .......................... 69

Figure 5.1: \( R^d_i = (m + n) - \frac{D_{i,d}}{m+n} \) ........................................... 74

Figure 5.2: \( x_s < x_h < x_d \land y_s < y_h < y_d \) ........................................... 78

Figure 5.3: \( x_s < x_h < x_d \land y_s < y_h = y_d \) ........................................... 80

Figure 5.4: \( x_s < x_h < x_d \land y_s = y_h = y_d \) ........................................... 81
ACKNOWLEDGEMENTS

I am grateful to all who helped and supported me towards the completion of the Ph.D degree.

In particular, I would like to express my gratitude to my major professor, Dr. Johnny Wong, and Dr. Vasant Honavar for their advise, guidance, patience, and moral support throughout my graduate program. This research would not have been possible without their contribution during countless hours of discussions. I deeply appreciate the team effort that led to the development of Quo Vadis.

I would like to thank my committee members, Dr. Robert Stewart, Dr. David Fernandez-Baca, Dr. James Davis, and Dr. Doug Jacobson, for their suggestions and cooperation.

A special note of thanks is extended to the Department of Computer Science and the Center for Agricultural and Rural Development (CARD) for providing me with financial support during my graduate studies.

For the many discussions and valuable suggestions I would like to thank the members of the Artificial Intelligence Research Group in the Department of Computer Science and especially my office mate, Yong Zhao.

To Todd Campbell and Deepa Parthasarathy, I extend my special thanks for their contribution to the design of a simulation tool, without which this work would
not have been possible.

Without doubt, the most significant contribution to the completion of my Ph.D degree was the encouragement provided by my loving family. I am especially grateful to my parents, Erwin and Christel Mikler, for their support and encouragement to pursue graduate studies at Iowa State University. Their confidence and pride have given me the necessary motivation to proceed towards reaching my personal goals.

For her love, patience, and her moral support, I would like to thank my wife Suhasini. Her efforts and understanding made the combination of Ph.D research and family life possible. I would like to thank Preethu for her understanding when research made it necessary to postpone driving practice. A special THANK YOU goes to my son, Calvin Vijay Mikler for filling my life with joy and laughter, and showing me every day that there is more in life than a Ph.D dissertation.
CHAPTER 1. BACKGROUND

Introduction

Recent advances in computers and communications, along with the ever-increasing need for rapid and reliable information transfer over very long distances has led to unprecedented expansion of such communication infrastructures over the past several years. Such networks contain hundreds if not thousands of interconnected nodes [40]. Traffic management mechanisms must be able to support a cost-effective, responsive, flexible, robust, customer-oriented high speed communication environment while minimizing the overhead associated with management functions. Conventional traffic management mechanisms for routing and congestion control algorithms entail tremendous resource overhead in storage and update of network state information. This will almost certainly result in increased cost and reduced performance with growth in the size of the networks. Thus, a careful reevaluation of conventional traffic management schemes with respect to their efficiency and effectiveness within large, constantly expanding communication environments is needed.

Message routing and congestion control are typical traffic management tasks. These functions are generally thought of being hosted by the layers 2-4 of the Open Systems Interconnection (OSI) protocol stack. The primary objective of routing mechanisms is to propagate messages across the network towards their destinations
while simultaneously trying to optimize one or more performance criteria such as path length or message delay.

The primary objective of congestion control is to prevent uncontrolled influx of messages into a set of network nodes. Without congestion control, network nodes may experience over-utilization which in turn may lead to increased loss of messages due to limited availability of buffers [11, 16]. As a consequence, the quality of service offered by the network will decrease. Furthermore, the loss of messages generally requires their retransmission which in turn reduces the overall network utilization (throughput). Even if network nodes have infinite buffer space available, thereby eliminating message loss, congestion tends to increase the overall delay encountered by messages.

Routing and congestion control are strongly interrelated as routing decisions determine the area through which a message is sent while moving towards its destination. Consequently, routing algorithms must be carefully designed to adapt rapidly to load changes in the network. In addition, routing techniques must minimize the associated resource overhead and should scale well without compromising performance as networks continue to grow in size. Resource overhead to be minimized can be divided into:

- bandwidth requirements;
- storage requirements; and
- computational complexity.

Additional desirable properties of routing and congestion control mechanisms for such communication environments include the ability to:
• route messages anticipating the consequences of routing decisions on the network dynamics (e.g., to pro-actively avoid congestion if possible),

• smoothly trade-off of some subsets of performance measures against others, and

• gracefully adapt without manual intervention to (predictable as well as unpredictable) changes in network dynamics without compromising performance.

Routing in Large Networks

Conventional approaches to routing [4, 25, 39] rely on the timely availability of large amounts of accurate network state information (for example, in the form of distance and routing tables) at each of the switching nodes so that they can make routing decisions designed to optimize (to the extent possible) the desired measures of overall network performance such as delay and throughput [42, 2, 48]. In practice, frequent transmission of such network state information consumes valuable resources such as memory and bandwidth which could otherwise be used for message traffic. Most attempts to reduce the overhead involved in the update of network state information at each switching node lead to a degradation in the accuracy of the information available. As communication networks grow larger, the overhead associated with conventional routing mechanisms becomes prohibitive.

Basic Routing Algorithms

Most conventional routing protocols, such as the routing information protocol (RIP) and open shortest path first (OSPF) have their origin in either one of two basic strategies [36], which are
1. distance vector routing, and

2. link state routing

Distance vector routing is often referred to as the old ARPANET routing algorithm. It is essentially a distributed version of the Bellman-Ford shortest path algorithm [2]. Distance vectors are generally stored in distance tables at each network node. Distance tables thus contain distance estimates to every destination in the network via each neighbor node \( n_k \). The distance vector to a particular destination node is computed by adding the \( \text{distances} \) between nodes along the paths to the destination. A routing table that contains all possible destination nodes is constructed by selecting from the distance tables those routing vectors with minimum distance estimates. Upon receiving a message that is to be routed towards its destination, a network node initiates a table look-up resulting in a node to which the message is to be sent next.

Link state routing is based on the assembly of complete topological information. It is frequently referred to as the new ARPANET routing algorithm as it has replaced the earlier distance vector approach on the ARPANET. Each node measures the distance from itself to all its neighbor nodes and propagates a link state packet (LSP) to all other nodes in the network. This process is generally referred to as flooding. After a node receives a LSP from every node in the network it can construct a spanning tree that is rooted at the node itself. The construction of the spanning tree is based on Dijkstra’s shortest path first (SPF) algorithm [25]. Network nodes must be able to assess the validity of each LSP received to avoid outdated information from corrupting the spanning tree. This is accomplished by employing costly timer, sequence number and aging schemes [36].
Both link state and distance vector routing rely upon complete network state information. That is, each node needs to compile global knowledge of the entire network. While in distance vector routing this knowledge is represented by the set of all distance tables, link state routing relies on information about the state of every link in the network. Clearly, the amount of network state information used by both these routing strategies increases with the size of the network.

The imprecision or uncertainty associated with network state information grows also with the size of the network. This is a direct consequence of the temporal dynamics of the network which causes the network state to change even as the state information is being computed and propagated. The amount of storage required to maintain network state information at each switching node also grows with the size of the network. So does the network bandwidth required to maintain this information up-to-date.

**Approaches to Reducing Overhead**

The immense cost associated with the maintenance and frequent update of network state information prompted the exploration of a number of strategies designed to minimize the resource (e.g., storage and bandwidth) requirements of traffic management in large communication networks. Most of these strategies involve structuring of the network at the logical level, the physical level, or both. Some examples of structuring at the logical level include hierarchical routing [21, 35] and landmark routing [44]. Large networks are organized into a hierarchy of logical units. Switching nodes maintain complete state information only for the nodes within their own logical unit supplemented by a summary of the network state information for other logical units.
Collections of subnetworks connected via a backbone offer an example of structuring of the network at the physical level. While both hierarchical routing and landmark routing do reduce the amount of network state information stored at and transmitted between nodes, they suffer from a number of drawbacks. For instance, it has been shown that the manner in which reduction in network state information is realized in hierarchical and landmark routing results in an increased average path length between source and destination nodes. The existence of an optimal structuring of the network so as to limit the size of routing tables has been shown in [21] and [44]. However, frequent restructuring of hierarchies and landmarks so as to maintain an optimal structure is required in order to provide for acceptable performance in an expanding communication environment. This clearly represents another drawback associated with such techniques.

Hierarchical routing and landmark routing are approaches to reduce the size of routing and distance tables in the underlying distance vector routing algorithm. No such approach is currently available for link state routing as routing tables are computed using a minimum spanning tree that can only be constructed from complete topological information. Instead, approaches such as SPF routing with emergency exits (SPF-EE) [46] are designed to reduce the frequency of link state updates and thus the frequency of recalculating the spanning tree by reducing the degree of oscillation commonly experienced by link state routing.

In a network with \( n \) nodes and \( k \)-connectivity, the space required to store network state information is \( O(k \times n) \) for both, distance vector and link state routing. While there are \( k \times n \) links to be considered in the construction of a spanning tree, distance vector routing must construct \( k \) distance tables each with \( n \) entries. If the network
is structured into a $r$-level hierarchy this requirement can be somewhat reduced [36].

However, the space requirement of a routing strategy is not the only issue to be considered. Maintaining up-to-date knowledge about the network state requires frequent propagation of distance and delay estimates. Thus, all of the above routing mechanisms consume bandwidth proportional to their storage requirement. The precision of information that is ultimately used to construct routing tables clearly depends on the dynamics of the network as well as the update frequency. Even if the time interval $\tau$ between updates is small, a finite amount of time is needed to propagate network state information (or its impact) to every node. Consequently, network state information collected by network nodes almost never represents the state of the network at a time $t$ when a routing decision is made. Some degree of uncertainty is therefore inevitable.

Overview

Chapter 2 describes Quo Vadis, a framework for intelligent traffic management in very large, high-speed communication networks. Quo Vadis draws upon insights from hitherto disparate areas: communication networks, artificial intelligence, machine learning, and optimization in order to strike a balance among various performance criteria. The primary objective of Quo Vadis is to achieve reasonable network performance while minimizing the overhead associated with network traffic management.

Quo Vadis has been implemented within an object-oriented discrete event-driven simulation environment [27, 31], which is presented in Chapter 3. The prototype implementation of Quo Vadis was used to conduct a number of experiments to ex-
explore the behavior of parameterized knowledge representation and heuristic routing mechanisms. The experiments described in Chapter 4 were conducted in a regular $m \times n$ grid network.

Chapter 5 presents a theoretical approach to designing functions for message routing in large communication networks. While the desirable properties of routing mechanisms are still used to guide the design, we draw upon ideas and concepts from the field of utility theory.

A summary of Quo Vadis together with suggestions for future research are presented in Chapter 6.
CHAPTER 2. QUO VADIS - A FRAMEWORK FOR INTELLIGENT ROUTING

Background

In our view, any intelligent traffic management mechanism must include:

- An effective knowledge representation (KR) mechanism capable of providing sufficiently precise information about the state of the network;

- An efficient knowledge acquisition (KA) technique, that minimizes the overhead that is associated with acquiring network state information.

- Adaptive decision making methods that are designed to optimize the network performance.

The approach adopted by Quo Vadis for traffic management (and routing in particular) in large communication networks is motivated by the following observations:

1. The quality of routing decisions (as measured by suitable metrics such as average delay, average path length, etc.), is a function of the imprecision or uncertainty associated with the network state information upon which such decisions are based (assuming a decision function that makes optimal use of the available information). The imprecision or uncertainty of network state information is a
function of (among other things) network dynamics, frequency of state updates, network delay for control messages, etc. In practice, all routing decisions in a large communication network are based on imprecise, uncertain knowledge of the current network state.

2. As noted in Snyder's proposal for the so-called "traveller architecture" [40], the significance attached to the state (e.g., load) of a node to routing decisions made by another node in the network should be an inverse function of the distance between the two nodes. It follows that switching nodes in large communication networks should be able to make routing decisions based on the network state in their local neighborhood with little overall degradation in the quality of routes. The intuition behind this observation becomes clear if one considers a traveller faced with the task of choosing a route from a current location to a final destination. Such decisions are usually based on the conditions (e.g., traffic density) in the immediate vicinity of his current location. At each step, he is likely to pick a general direction that takes him closer to his destination via a neighboring location that appears to be the best (as measured by the traffic density). A precise knowledge of the current traffic conditions at locations that are sufficiently far from the traveller's current location is of little use to him because the conditions there almost certainly would have changed by the time he gets close to them.

3. The number of routes of comparable length between a source node \( n_s \) with coordinates \((x_s, y_s)\) and a destination node \( n_d \) with coordinates \((x_d, y_d)\) is a non-decreasing function of the distance between the two nodes. For example,
in a regular square grid network, it can be easily shown that the number of possible shortest routes \( P \) between nodes \( n_s \) and \( n_d \) is

\[
P = \left( D_x + D_y \right) \frac{D_x}{D_x}
\]

where

\[
D_x = |x_s - x_d| \quad \text{and} \quad D_y = |y_s - y_d|
\]

It follows that the likelihood of finding alternative paths of comparable length is a non-decreasing function of the distance to the destination. Quo Vadis exploits this fact through its use of a carefully designed knowledge representation mechanism that maintains at all time, at each node, a locally computed *view* of the network state. This view includes precise information about the state of the node (e.g., its load) supplemented by a less precise (spatially and temporally averaged) summary of the state of the entire network as viewed from that node. Thus, each node needs to communicate its state and its view only to a small set of nodes in its immediate neighborhood. Routing decisions made by each node in Quo Vadis are based on the network state as captured by the views of the nodes in its immediate vicinity, and the destination of the message to be routed.

4. The utilization \( \rho \) of network nodes is generally determined by the ratio \( \lambda/\mu \) where \( \lambda \) represents the arrival rate to that node and \( \mu \) designates the rate at which the node can service messages. Hence, high utilization may occur due to a reduced service rate (possibly caused by node failures), or an increased arrival of messages. An increase of a node’s arrival rate can have essentially two causes:
(a) Many network nodes inject messages into the network destined to the same node (or network area).

(b) Routing decisions in neighbor nodes select the same node for a large number of messages. That is, a node is selected as best neighbor as determined by the routing metric used.

Assuming network nodes to be modeled as M/M/1 queues \([17, 37]\), the message delay in each node \(i\), among other things, depends on its utilization \(\rho_i\). The expected delay \(D_i\) is given by

\[
D_i = \frac{1/\mu_i}{1 - \rho_i}
\]  

\(D_i\) grows exponentially as \(\rho_i\) increases (see Figure 2.1). Clearly there exists a tradeoff between utilization and message delay, both of which are important performance measures.

Figure 2.1: Delay vs. utilization
In a uniformly utilized network, the best performance along a particular route can be obtained when the number of intermediate nodes is kept minimal. However, this is not necessarily true for non-uniform utilization as we will show below. This relationship between utilization and delay can be exploited in the design of routing algorithms. Quo Vadis attempts to do precisely this through its use of parameterized heuristic knowledge representation, knowledge acquisition, and decision functions.

A Prototype Design of Quo Vadis

The current design of Quo Vadis [27, 28, 29, 30, 31] consists of two closely coupled modules:

- The knowledge representation module which is primarily responsible for the maintenance and update of network state information as viewed from each node.

- The decision module which implements routing and control algorithms.

Both these modules instantiate a family of parameterized heuristics that follow from the design philosophy of Quo Vadis. Future extensions to this design might include additional modules for adaptation of parameters to particular network dynamics and for learning appropriate classes of routing and congestion control strategies. A detailed description of the design and operation of knowledge representation and routing decision modules in Quo Vadis follows.
Knowledge Representation in Quo Vadis

As noted earlier, the knowledge representation mechanism in Quo Vadis is designed to maintain at all time, at each node, a locally computed view that includes precise information about the node supplemented by a spatially and temporally averaged summary of the state of the network as viewed from that node. This section explains exactly what constitutes such a view and how it is computed by each node based entirely on the information communicated to it by a small set of nodes in its immediate neighborhood. Since the network nodes in Quo Vadis have no knowledge of the network connectivity which is implicitly available in routing tables, it needs an alternative scheme for addressing nodes and for computing their positions relative to each other. This is accomplished by superimposing a 2-dimensional grid on the plane containing the network and identifying each node by its coordinates relative to the grid (see Figure 2.2).

Thus, each node $n_i$ is addressed by its respective coordinates $(x_i, y_i)$. Note that
this does not restrict the allowable network topology in any manner. However, for more complex network topologies it may become necessary for nodes to maintain additional topological information.

Each node $n_j$ maintains a view $V_i(t)$ of the network from its vantage point at time $t$. This view can be decomposed into four components, one for each of the four directions - north, south, east, and west. Thus we have:

$$V_i(t) = [V_i^N(t), V_i^S(t), V_i^E(t), V_i^W(t)].$$

Each component $V_i^d : (d \in \{N, S, E, W\})$ of the view $V_i(t)$ is computed using the corresponding view components $V_k^d(t - \tau)$ (where $\tau$ is the interval between view updates) together with local measurements $\rho_k(t)$ (see below) communicated by each of its neighbors $n_k$ (suitably weighted by a normalized directional gain $g_{i,k}^d$ - see below). This ensures that the contribution of the information provided by the node $n_k$ to the views computed at the node $n_i$ is inversely proportional to the euclidean distance $D_{i,k}$ between the nodes $n_i$ and $n_k$. Also note that the contribution of the node $n_k$ to the view component $V_i^d$ is directly proportional to its relative orientation as viewed from $n_i$ with respect to the direction $d \in \{N, S, E, W\}$. This gain is normalized over the set of all neighbor nodes $H_i = \{n_k \mid n_k \text{ is a neighbor of } n_i\}$. (Note that this definition of directional gain is only one of the alternatives with qualitatively similar properties. Also, different definitions of neighborhood are possible).

Assume that the $x$ and $y$ coordinates increase as one travels further east and north respectively. Let $(x_i, y_i)$ and $(x_k, y_k)$ be the coordinates of nodes $n_i$ and $n_k$ respectively, and the euclidean distance between $n_i$ and $n_k$ be $D_{i,k}$. The directional gain to the south at node $n_i$ for node $n_k$ is given by:
where $\eta$ is to be chosen such that the directional gain appropriately amplifies load information from nodes in direction $d$. The directional gains $G^N_{i,k}$, $G^E_{i,k}$, and $G^W_{i,k}$ for the north, east, and west component of $V^d_{i}(t)$ are given by similar formulae. The normalization factor $G_i^d$ for direction $d$ for gains $G^d_{i,k}$ computed at node $n_i$ is given by:

$$G_i^d = \sum_{n_k \in H_i} G^d_{i,k}$$

The corresponding normalized directional gains are given by:

$$g^d_{i,k} = \frac{G^d_{i,k}}{G_i^d}$$

Now the view component $V^d_{i}(t)$ at node $n_i$ at time $t$ is given by:

$$V^d_{i}(t) = \sum_{n_k \in H_i} g^d_{i,k}(\alpha \cdot \rho_k(t) + (1 - \alpha) \cdot V^d_k(t - \tau)); \quad 0 < \alpha \leq 1$$

where $\tau$ is the time elapsed since the previous view update at the node $n_i$. (It is possible to make the update frequency a function of the local network dynamics. Such an approach is currently under study and will be discussed in a forthcoming paper). The parameter $\alpha$ determines the degree to which the effects of an event (i.e., load change) can impact routing decisions at other network nodes.

The local measurement $\rho_k(t)$ of node $n_k$ has a number of natural interpretations. For example, if each network node is modeled as an M/M/1 queue, $\rho_k(t)$ may
correspond to the utilization or the ratio of the arrival rate $\lambda_k(t)$ to the service rate $\mu_k(t)$ at time $t$. Note that there is nothing in the design of Quo Vadis that forces it to use an $M/M/1$ queue to model each node. A variety of more sophisticated queueing models can be used if necessary. The relative importance attached to the local measurements as opposed to the (spatially and temporally averaged) global view of the network as seen from a node is governed by the parameter $\alpha$. It is a candidate for adaptation to cope with changes in network dynamics. So is the frequency of update of views maintained by nodes in the network (controlled by $\tau$). Note that each node $n_i$ computes its own view $V_i$ only to disseminate it among its neighbors so as to enable them to update their knowledge of the network state. This knowledge is maintained at each node $n_i$ in a knowledge base $S_i(t) = \{ (\rho_k(t), V_k(t)) \mid n_k \in H_i \}$. As explained below, the routing decisions at each node $n_i$ are based on its current knowledge base $S_i(t)$. The performance of Quo Vadis would depend on how well it reflects the actual state of the network.

Suitable mechanisms that adapt parameters such as $\alpha$ and $\tau$ in response to variations in network dynamics and/or changes in performance demands are of interest. It is only in the interest of simplicity of notation that such parameters have been treated as though they were constants in the equations above. Thus it is possible to let them take on different values at different nodes in the network and change their values as a function of spatio-temporal variations in traffic patterns and performance requirements. It is also worth emphasizing that the particular equations for view computation given above represent only one of many possibilities given the overall design philosophy of Quo Vadis.

The size of the knowledge base $S_i(t)$ at node $n_i$ depends solely on the number
of neighbors in its neighborhood \( H_i \) and is independent of the size of the network. Thus if \( M \) is the total number of nodes in the network and \( h \) the average connectivity (i.e., the average cardinality of \( H_i \)), then the storage required at each node in Quo Vadis is \( O(h) \). This constitutes a significant reduction in storage and processing overhead (especially in very large networks where \( M \gg h \)) over conventional routing mechanisms (e.g., those that use global routing tables) which require \( O(M) \) storage at each node.

Routing and Control in Quo Vadis

As pointed out earlier, each node \( n_i \) in Quo Vadis, when it receives (or generates) a message that needs to be sent to a different destination, it makes a routing decision based on the destination of the message and its current knowledge base \( S_i \). This section describes in detail the routing mechanism used in a prototype implementation of Quo Vadis. Consider a message that is on its way from a source \( n_s \) to a destination \( n_d \) through a node \( n_i \). Now \( n_i \) is faced with the task of routing the message along a path that would take it to its destination so as to optimize some desired performance criteria (e.g., average path length, average delay, or other suitable routing metrics). The node \( n_i \) does this by selecting one of the nodes in its neighborhood \( H_i \) that appears to best serve this objective. Choosing the best neighbor is based on the use of an evaluation function (in much the same spirit as the heuristic evaluation functions used in state space search in artificial intelligence problems [Pearl, 1984]). The node \( n_i \) computes the utility \( U_k \) of each node \( n_k \in H_i \) and chooses the one that has the largest utility (it is assumed that during this computation, the view and load values do not change). In the prototype implementation of Quo Vadis, \( U_k \) is a
function of two separate components:

1. the load liability $L_k$ which estimates the load likely to be encountered by the message on its way to its destination $n_d$ if it were to be routed through $n_k$; and

2. the path liability $P_k$ that assigns a value to each neighbor $n_k$ so that neighbors that are closer to the destination of the message being routed reflect lower values of $P_k$.

The overall utility $U_k$ of the node $n_k$ is given by:

$$U_k = -(\beta * P_k + (1 - \beta) * L_k); \ 0 \leq \beta \leq 1 \quad (2.8)$$

where $\beta$ determines the emphasis placed on finding the shortest path to the destination relative to the desire of avoiding heavily loaded paths. Given this general framework for computing the utility of nodes, several different choices exist for the exact form of the expressions used to compute $L_k$ and $P_k$. The particular forms used in the prototype implementation of Quo Vadis are explained below.

The load liability of node $n_k$ is given by:

$$L_k = \gamma * \rho_k(t) + (1 - \gamma) * v_k(t); \ 0 \leq \gamma \leq 1 \quad (2.9)$$

where $v_k(t)$ is the sum of the projections of the appropriate components of the view $V_k$ of the neighbor node $n_k$ onto the vector connecting $n_k$ to the destination node $n_d$.

Depending on $n_d$'s location relative to $n_k$, $v_k(t)$ is composed of two components, namely an east-west component $C_{EW}$ and a north-south component $C_{NS}$. Let $(x_k, y_k)$ and $(x_d, y_d)$ be the coordinates of node $n_k$ and the destination node $n_d$. 
respectively. Let $\theta$ be the angle formed by $n_d, n_k, n_p$, where $n_p$ is a virtual point in the grid with coordinates $(x_d, y_k)$ (see Figure 2.3).

The components of $v_k(t)$ are:

$$
C_{NS} = \begin{cases} 
|\sin \theta \cdot V_k^N| & \text{if } \sin \theta \geq 0 \\
|\sin \theta \cdot V_k^S| & \text{if } \sin \theta < 0 
\end{cases}
$$

$$
C_{EW} = \begin{cases} 
|\cos \theta \cdot V_k^E| & \text{if } \cos \theta \geq 0 \\
|\cos \theta \cdot V_k^W| & \text{if } \cos \theta < 0 
\end{cases}
$$

The projection $v_k(t)$ is then computed as:

$$
v_k(t) = \sqrt{C_{NS}^2 + C_{EW}^2} \quad (2.10)
$$

Thus, if $n_d$ is to the north of $n_k$, then $V_k^N(t)$ (as one would expect logically) should contribute the most to $L_k$. $V_k^E(t)$ or $V_k^W(t)$ contribute to a lesser extent, depending on the relative location of $n_d$. $V_k^S(t)$, in this particular case, does not make any contribution to $L_k$ at all, as the south view of $n_k$ is of little consequence to a message destined to go north through $n_k$. The tunable parameter $\gamma$ determines the
relative emphasis placed on the load (as measured by ρ_k(t)) versus the appropriate projections of V_k(t) (as reflected by u_k(t)).

The path liability of a node n_k with respect to a message passing through n_i on its way to a destination n_d is given by:

\[ P_k = \frac{D_{k,d}}{D_{i,d}} * ρ_i(t) \]  \hspace{1cm} (2.11)

Clearly, choice of a neighbor node that has the smallest \( P_k \) biases Quo Vadis to route messages along paths that cover the largest fraction of the remaining distance to the destination (provided other things being equal).

It is possible to use a variety of other formulations that share the spirit of the examples shown above for the calculation of load and path liabilities. It is also possible to incorporate additional terms suggested by other performance criteria into the calculation of \( U_k \). Routing decisions are based on parameterized heuristics so as to permit a range of tradeoffs through adaptation of tunable parameters to accommodate different (perhaps even conflicting) performance criteria under a range of different network dynamics.

**Summary**

The framework of Quo Vadis consists of functions for the representation of the network state and parameterized decision functions to facilitate the routing mechanism. These functions are designed to reduce the overhead that is generally associated with the acquisition and maintenance of network state information. As network nodes only need to maintain information about the state of nodes in their immediate neighborhood, the size of the knowledge base is small as compared to conventional
routing mechanisms.

In order to understand the behavior of the functions in Quo Vadis, various simulation experiments have been conducted. An object oriented discrete event simulation environment has been designed and implemented to allow for an experimental study of Quo Vadis. The design and implementation of this environment is described in Chapter 3.
CHAPTER 3. A SIMULATOR FOR LARGE COMMUNICATION NETWORKS

Introduction

Simulation is a useful, and often essential tool for the design, implementation, and verification of systems with large number and variety of entities. Their behavior is the result of complex interactions among its entities. This complexity, and the resulting dynamics make an analytical study often impossible. One example of such systems is a high-speed communication network with hundreds or even thousands of nodes and links.

Many network simulation models, such as that of a single network node or a local area network (LAN) only have a small number of entities. Hence, a detailed performance analysis is generally feasible. Larger systems often allow for a decomposition into smaller sub-units, which then can be analyzed independently to determine the performance of the overall system. With the introduction of distributed and/or collaborative computing in a global computing environment, the size and the complexity of the underlying communication network increases. Hence, it may not be feasible to model and analyze the behavior of the overall system based on its sub-units. The system's dynamic behavior emerges from the interaction among individual entities. Their interaction is often designed to optimize global performance criteria.
Various well designed simulation packages with a wide range of features are available [1, 43, 3]. Off the shelf simulation packages generally provide a large set of pre-defined functions implemented in various types of simulation modules that enable users to easily construct a variety of simulation models. In addition, they usually provide a graphical user interfaces, and a wide variety of analysis tools. However, users do not have control over the degree of detail of the individual modules. Hence, the resource demands for very large simulation models may become prohibitive which in turn limits the complexity of a model. Hence, the constraint of having to simulate an environment with hundreds or even thousands of independent objects justified a new, object oriented design. This consideration necessitated the design and implementation of an object oriented toolbox for the simulation of large communication networks.

Framework for Simulating Large Network

The issue of uncertainty in communication networks and the simulation of learning from observation in such an environment to enhance fault management has been discussed in [22]. Our efforts to design an object oriented simulation environment were driven by the need to evaluate heuristic routing strategies for large communication infrastructures [27, 28, 29, 31]. The network to be simulated may consist of hundreds or even thousands of nodes, connected by communication links [40]. Network nodes (routers) and communication links are the basic entities that need to be modeled in simulation. Individual nodes and links are modeled as independent entities together with their associated functions. The behavior of each node is determined by various parameters such as link bandwidth, service rate, the choice of a routing
algorithm, as well as parameters which control the acquisition and representation of the network state.

The Model

Routing [2, 42] is the task of propagating a message from its source node towards its destination node. The routing algorithm used in the network determines how an intermediate node selects one of its neighbors as the next node to which the message should be sent. Routing decisions dictate the path travelled by a message and hence determines the performance of the network as measured by metrics such as path length and total delay.

One of the differences among the various routing algorithms is the complexity of network state information that must be acquired by each network node in order to make appropriate routing decisions. The complexity of network state information is reflected by the resource demand of the routing algorithm in terms of

- memory requirements,
- computational complexity, and
- bandwidth overhead.

While for some routing algorithms such as random routing and hot potato routing induce very little resource overhead, others, such as distance vector and link state routing, have to acquire information about the entire network. There clearly exists a tradeoff between the associated resource overhead and the average network performance. That is, the amount and precision of network state information upon which routing decisions are based will ultimately determine their quality. In the context of
our research, we have designed a set of heuristic knowledge representation and decision functions capable of finding low delay routes while minimizing the associated resource overhead.

The evaluation of adaptive routing algorithms in a dynamic network environment requires simulation experiments with network models of different sizes, topologies, and traffic patterns. The underlying simulation environment must therefore be flexible enough to accommodate the various models. Furthermore, the simulation environment must provide for efficient instantiation of entities, such as network nodes, links, and messages. This enables the user to make appropriate changes without redesigning and reprogramming the entire model and hence allows for the reuse of available modules.

With increasing network size, the resource requirement for the simulation becomes a critical issue to be considered in the design of the simulation environment. In order to accommodate very large network models, simulation experiments should focus on only those functions/features that are deemed essential for the analysis; other functions are to be abstracted and represented implicitly or left out completely in order to limit the required system resources.

The simulation environment used in this research is flexible enough to host a variety of models. Since most changes to a simulation model only impact a small part of the testbed, the testbed design is kept modular, separating different functional units [1, 43]. The simulation environment provides for the performance evaluation of the model in terms of standard performance metrics such as average delay, throughput, and load measures. However, the user can easily implement mechanisms to capture other measurements of interest. The modular design of the testbed makes this task
significantly easier and more efficient.

**Network Nodes** Nodes in our model act as source (i.e., inject new messages into the network) as well as destination (i.e., remove messages from the network). The influx of messages into a network node \( n_i \) is assumed to be Poisson distributed with a mean of \( \lambda \), the arrival rate. This Poisson stream, however, can be broken down into two sub-streams, \( S_1 \) and \( S_2 \) with means of \( \lambda_1 \) and \( \lambda_2 \), respectively \([17, 37]\). \( S_1 \) consists of messages that are generated and injected into the network by \( n_i \). Stream \( S_2 \) consists of messages that were sent to \( n_i \) by nodes in the neighborhood \( H_i \) of \( n_i \). A node \( n_j \) is said to be in \( H_i \) if \( n_i \) and \( n_j \) are connected via a communication link \( l_{i,j} \).

Upon receiving a message \( m_k \), it is added to a central queue in \( n_i \) to await service. The service performed by a node \( n_i \) consists of the removal of \( m_k \) from the queue and its propagation to a node \( n_j \in H_i \) or delivery of \( m_k \) to a user process if \( n_i \) is the destination of \( m_k \). Otherwise, a neighbor node \( n_j \in H_i \) must be selected for the propagation of \( m_k \) towards its destination. In order to make a routing decision (i.e., selecting \( n_j \)) a node must acquire and maintain an adequately precise and up-to-date representation of the state of the network. The type and amount of information upon which routing decisions are based clearly depends on the routing algorithm that is used. For example, the network state may be represented in the form of distance tables or routing tables as in distance vector routing, or in the form of a distance tree as used in link state routing. In our system, the network state is represented by a knowledge base consisting of local load measures that are supplemented by global summary information \([27, 28, 29, 30, 31]\). The functions associated with a
network node are: message generation, message routing, network state acquisition, and network state representation.

Communication Links Two nodes, $n_i$ and $n_j$ communicate via a communication link $l_{i,j}$. Links are assumed to be uni-directional. Various parameters such as bandwidth, cost, error rate, etc., are generally associated with a network link $l_{i,j}$. A link is generally used to communicate both data messages as well as network state information (i.e., control messages).

Within a simulation model, various functions can be associated with a communication link. In our model it is primarily used to simulate the transmission delay between nodes due to bandwidth constraints. In general, functions which model the dropping of messages, corruption of data, or simulated link failures may be associated with a communication link.

Network State The state of the network is determined by the rate at which messages arrive and depart from various queues, as well as the set of messages that are awaiting service. Hence, the state of the network is the collection of all individual node and link states. For large network models, it is not feasible to maintain a centralized complete and up-to-date knowledge of the network state. Our model assumes that network nodes acquire global network state information individually through appropriate update procedures.

Network traffic consists of messages in the network and can generally not be abstracted by a generic representation. Messages must be represented explicitly as they determine the behavior of nodes and links at a particular point in time. Each message is therefore instantiated and represented by a tuple of values, such as, source
address, destination address, message id, as well as information used for performance analysis.

The network model in our simulation environment is an abstraction of the real communication environment at the level of nodes, links, and messages. As the state of an individual node and link is determined by the rate of message arrival and departure, different network load patterns can be simulated by changing the packet generation or service rate in some or all nodes. It should be noted that our simulation testbed can be extended to facilitate the investigation of various functions performed by network nodes and links (such as admission policies, queueing disciplines, etc.).

Simulation Methodology

We have chosen to base the underlying simulation driver on discrete event simulation, although process emulation has been considered [1, 3]. The fact that each active entity in the model would have to be implemented as an independent process renders process emulation non-suitable as the overhead associated with creating large number of processes and communicating among them becomes prohibitive as the model size increases.

Two forms of Discrete Event Simulation (DES), namely time driven and event driven, are deemed to be appropriate for simulating a testbed for routing algorithms. Both approaches are briefly discussed and contrasted below.

Discrete Event Simulation Typically, performance modeling involves the simulation of different system states which are represented by the presence or absence of countable units, such as jobs, requests, processes, messages, users, or errors. A
new state can only be entered through the execution of an event that modifies one or more of these units. As each event, and hence state change, involves a specific number of discrete units, this type of simulation is generally referred to as Discrete Event Simulation (DES) [3, 10].

DES is further divided into two broad classes, namely, time driven simulation and event driven simulation. For time driven simulation, an event \( e \) is selected from an event set \( E \) at every tick of a global clock. \( E \) is said to contain all plausible events that can execute in the current state. The fact that no state change takes place at time \( t \) can be simulated by a null event as part of \( E \). A simulated random walk in a regular grid in which a particle is moved with each clock tick by unit distance in one of four possible directions is an example of a time driven approach.

In event driven simulation, events are scheduled for various (future) instants of time at which they will execute. The system must maintain an event list into which all scheduled events are inserted. Associated with each event \( e \) is a time instant \( t_e \) at which the event is to occur. At all times, a partial order of events on the event list is maintained. That is, event \( e_1 \) will appear before event \( e_2 \) if \( t_{e_1} < t_{e_2} \). If \( t_{e_1} = t_{e_2} \), the order in which \( e_1 \) and \( e_2 \) appear on the event list can be left unspecified under the assumption that the corresponding changes of the network state are not visible instantaneously. In addition, this constrains new events to be scheduled not earlier than the current time \( T \). Upon execution of an event \( e \) at time \( t_e \), the global clock must be advanced to that time \( (T := t_e) \). The execution of event \( e \) may trigger the generation of new events which are then inserted into the event list as discussed above. A typical event driven simulation cycle is shown in Figure 3.1.

The simulation of a multi-agent environment such as a computer system, or a
communication network lends itself to the event driven approach as components tend to schedule their actions at various time instants.

The differences between the two approaches becomes apparent when considering the simulation of a large communication environment with many different agents (components). The set $E$ of plausible events at time $T$ is defined by the applicable actions at all agents to the system state at time $T$. Effectively, a time driven simulation would have to query each entity with every clock tick so as to determine whether or not an event needs to be executed. For most queries, the result is likely to be the null event since the occurrence of an event in the network is determined by parameters such as, traffic patterns and service rates. Time driven simulation of such an environment is thus inefficient and often infeasible. Hence, we have chosen an event driven approach for the simulation of a large communication infrastructure.

Figure 3.1: Event-driven simulation cycle
The details of the design and implementation of our simulation testbed are discussed below.

**Design and Implementation of a Simulation Toolbox**

As mentioned above, the communication environment to be simulated may consist of several hundred network nodes connected by communication links. Messages are packaged into units consisting of actual message content and necessary protocol information and are transmitted on links between network nodes until the destination is reached. The functions of the communication network can therefore be expressed at the abstract level of nodes, links, and messages. In our implementation, nodes and links correspond to the active entities in the simulation model as they generate the necessary events that change the state of the network. The network model is embedded in the simulation testbed which maintains the event list and dispatches events to their corresponding entities for execution. The design and implementation of our simulation environment is described below.

**The Object Oriented Approach**

The decomposition of the simulation model into the functional components above suggests the application of the object oriented programming paradigm. Furthermore, employing the object oriented paradigm to the simulation of large communication networks yields advantages in the design as well as the implementation of both, the simulation testbed and the simulation model. The resulting structured design significantly simplifies the implementation, particularly when functional modules are self-contained and communicate with other modules through well defined interfaces.
Other advantages of employing the object oriented paradigm in the design and implementation of a simulation testbed are

- modularity,
- scalability,
- flexibility, and
- reusability

of the simulation environment.

Object classes together with the concept of inheritance provide for a very flexible framework which can meet the demands of many different simulation enterprises. The users can evolve the degree of detail of the various entities by gradually adding new features to the base classes. This can lead to an ever expanding simulation toolbox from which users can select the functional units necessary to implement their particular simulation environment. For instance, while network nodes are currently designed to support the analysis of a particular class of routing algorithms, the implementation can easily be tailored to support the simulation of other network management functions.

We refer to our design as an object oriented discrete event simulation (OODES). and we have chosen C++ as the implementation language [7, 23]. For the implementation of our OODES environment we distinguish the model under investigation from the simulation testbed which can be considered the host for the model. Clearly, model and testbed are interrelated as the model will determine the type of statistics which is to be collected. However, simple changes should suffice to modify the simulation
testbed to host other models as needed. The interface between the simulation testbed and the model is realized through a status class through which information about the simulation model as well as the state of the simulation itself is made available to both, testbed and model. The structure of the status class is described below. The decomposition of the simulation environment into various functional components is shown in Figure 3.2.

The Simulation Testbed

The simulation testbed basically consists of two modules, namely the simulation status and the simulation driver.

Simulation Status Upon starting the simulation, all definitions necessary to control the simulation are obtained from a parameter file provided by the user. Parameters in this file include all necessary information to dynamically build the network model. Other information provided through the parameter file such as,
simulation time, and random number seeds is used to control the simulation. Various filenames are specified to allow the tracing of individual simulation runs. The status contains all necessary information to enable the simulation driver to execute the same model several times with different random sequences or different parameter settings, thereby providing for an automated execution of multiple experiments.

The most important item hosted by the status class is the event list. It represents the central nervous system of the simulation environment as it contains all events that are scheduled during the time of a simulation run. The list is currently based upon a simple linked-list implementation, however, other implementations (such as hashing) are possible and have been considered. Events on the event list are ordered at all time according to an execution time that is associated with each event. Parallelism in the model is simulated by executing multiple events at the same time. That is, the global simulation time is not being advanced until all events scheduled for a particular instant have been executed. The insertion and removal of events by elements in the model (nodes and links) and the simulation driver are described in detail in the following sections.

Most of the information that is maintained in the status class is used to define the behavior of an individual object within the simulation model. Provided through a parameter file, packet generation rates, service rates, link bandwidths, packet sizes, and thresholds are defined.

Located within the status class is the statistics module (Stats) which provides all necessary functions to simulate various statistical distributions for arrival and service rates as well as destination selection. For the purpose of our research we have implemented the uniform, Poisson, and exponential distribution. Additional distributions
can easily be implemented as required by the respective simulation model. The performance monitor for the simulation is also located in this module. The performance monitor consists of functions that collect individual message statistics and functions that compute the various means, variances, and standard deviations. Since the status class is accessible to all modules, it can easily be extended to provide any function necessary to monitor the performance of the model under test.

**Simulation Driver**   At the heart of the simulation environment is a dispatching function which selects an event $e_x$ with the smallest event time $t$ from an event list $L$. Event $e_x$ is then dispatched to the object (network node, link, or service routine) which originally scheduled this event to be executed at time $t$. As mentioned above, the event-driven simulation strategy mandates events to be executed in strict order of their event time [3]. The event-driven simulation cycle is shown in Figure 3.1.

The implementation of this mechanism draws upon two object classes, namely a linked list and generic events.

The implementation of the event list can be based upon any list mechanism as long as it provides for insertion and ordered removal of elements. The class event-list can hence be derived from a basic list class modifying the associated insert function as required. In order to maintain an ordered event list, the append function used for queueing is replaced by an insert function which places events in their proper position in the event list. The concept of class templates, as available in C++, supports the design of a generic list class. As it is conceivable that different lists can store different types of items, the design of a generic list item has proven to be significant.

It should be noted that the underlying list representation together with the as-
sociated insertion function significantly impacts the overall performance of the simulation. Since the event list is expected to contain a large number of events at all time, search efforts for the appropriate place in the list for an event $e_k$ can be large. Depending on the required performance of the simulator, the implementation of the event list may be based on mechanisms that accelerate the insertion of elements, such as hashing, to be used instead of a linked-list representation.

A generic event in the OODES is represented by a tuple of the form $< T, A, E, P >$ interpreted as follows:

- $T$: represents the time instant (with respect to the global simulation clock) at which the event is to be executed.
- $A$: designates the agent type i.e., network node, link, or service routine for which the event is scheduled and which will ultimately execute the event.
- $E$: identifies the event and thus the action to be performed by agent $A$ upon executing the event.
- $P$: is a pointer to the instance of the agent type $A$ for which the event is scheduled.

Associated with the Event-Class is a dispatch function which uses the event-type $A$ to appropriately cast the pointer to the executing entity. It then dispatches the event to that entity by calling an `execute` function which must be provided by every simulation entity. The class definition for generic events is shown in Figure 3.3. Examples of typical event types in a network simulation are packet generation, dequeue and service function in nodes, and packet transfer in links. In addition, there
#ifndef INC_EVENT
#define INC_EVENT

class Event
{
friend ostream& operator<< (ostream &os, Event &e);

public :
Event(void);
Event(Event&);
void Set(TIME time, EVENT_TYPE eventType, EVENT_NUMBER event, void *object);
TIME GetEventTime(void){return time;}
void Dispatch(void);
void Display(void);
int operator< (Event &e);

private :
TIME time;
EVENT_TYPE eventType; // Node event | Link event ...
EVENT_NUMBER event; // Service event | Generate pkt....
void *object; // points to a node|link .. based on eventType.
};

#endif // INC_EVENT

Figure 3.3: Event-class header file
can be a variety of service event such as check pointing or the invocation of a user interface that can be scheduled at various instants of time.

Implementation of the Model

The instantiation of the simulation model entails the representation of each individual entity whose behavior is to be simulated. A network environment is constructed with nodes and links as the active entities. Active entities in our model act on messages, which are considered passive. The representation of the network model based upon the implementation of nodes and links is discussed below.

The Network Class  The network class constitutes the module which defines the topology of the communication environment under test. Nodes and links are represented by corresponding data structures. Upon instantiation, the network-object reads two datafiles supplied by the user.

The first file to be read defines the set of network nodes by providing various parameters for each individual node. The minimum information necessary to define the existence of a particular node is its location in terms of x & y coordinates in a virtual two-dimensional grid together with a node-id. Upon reading the information for a particular node, a node-object is instantiated with the corresponding parameter values. The new node-object is then stored in a node-list, thus making this node available to the simulation environment.

The second file read upon network instantiation contains a description of the communication links between nodes. For the purpose of our research, a link is defined simply by a tuple < n_i, n_j >, where n_i and n_j represent the source and destination,
respectively. Upon reading link-information from the file, a link class object is instantiated and stored in a link-list. Link-specific parameters, such as bandwidth, are assumed to be uniform throughout the simulation environment. For other simulation models it may be necessary to supply this information as part of the tuple read from the datafile.

The network class provides functions for accessing individual network nodes and links. As these functions are public they can be invoked from anywhere within the simulation. Pointers to the nodes and links of the network are stored in *nodeList and *linkList, respectively. Both are declared as private data members of the network class so as to restrict access to the respective access functions. Both, *nodeList and *linkList are based upon a commonly used implementation of a linked list. Access to individual list members is defined in the list class. The header file for the network class is shown in Figure 3.4.

**Network Nodes** As our simulation aims at the investigation of routing algorithms which are executed in the network nodes, this module displays the highest degree of complexity. In addition to the basic functional components of a network node the node class must provide the various structures and mechanisms necessary for the various routing algorithms under investigation.

The canonical structure of a network node is based upon a single queue, a service function, and a message generator (see Figure 3.5). Arriving messages are added to the queue and upon invocation by the event dispatcher, the service function removes the first element (message) from the queue and performs service in the form of routing. A generate-message event causes the message generator to create a message to a
```cpp
#ifndef INC_NETWORK
#define INC_NETWORK

class Network
{
  public:

    Network(Status *statusPtr);
    ~Network();
    void PrintNet();

    Node *GetNodeById(int nid) {return (Node *)(*nodeList)[nid];}
    Link *GetLinkById(int lid) {return (Link *)(*linkList)[lid];}
    void ExecuteEvent(EVENT_NUMBER num, List *el);
    void Checkpoint(void);

  //private:

    int NodeCnt;
    int LinkCnt;
    Status *status;
    List *nodeList;
    List *linkList;

    int ReadNodeList(void);
    int ReadLinkList(void);
    void InitNet(void);

}; //Network

#endif // INC_NETWORK
```

Figure 3.4: Network-class header file
randomly chosen destination node. This message is then added to the node’s queue. The structure of messages used in the simulator is discussed below. Message generation and service represent the fundamental events that are scheduled and executed by a network node. The rates at which messages are generated and serviced by the node is determined by the frequency at which these events are scheduled. This, in turn, is defined by the generation and service rates that are set upon instantiating a network node. The values for these rates are provided by the status class as described above.

Upon generating a new message, node \( n_i \) randomly determines the time \( \delta t \) until the next generation. A new generate message event is then scheduled at \( T + \delta t \), where \( T \) represents the current simulation time. The time interval \( \delta t \) is exponentially distributed with a mean of \( 1/\lambda_g \) [12, 17, 37, 38]. Hence, the generation of messages
is Poisson distributed with mean $\lambda_g$:

$$P_n(\Delta t) = \frac{(\lambda g \Delta t)^n}{n!} e^{-\lambda g \Delta t} \quad \Delta t \geq 0, \quad n = 0, 1, 2, \ldots$$ (3.1)

A node's service rate $\mu$ (in $k$ bits/sec) remains constant for all messages that are serviced at this node. The time spent to service an individual message depends therefore on the size of the message. In order to simulate exponential service times, messages sizes are exponentially distributed. Thus, a network nodes displays the behavior of a single $M/M/1$ queue. Our approach to simulate an exponentially distributed service rate by the means of exponentially distributed packet size is motivated by the need to preserve the sequence of random numbers used for the simulation. This approach will be further discussed below.

Upon servicing a message, node $n_i$ determines the time instant at which the next message is to be removed from its queue. In the current implementation, post-scheduling is used for scheduling the next service in a node $n_i$. The concepts of pre- and post-scheduling are discussed below.

Communication links to and from neighbor nodes are accessible through a list of pointers, each of which uniquely identifies a particular communication link. A node $n_i$ can communicate with a neighbor node $n_j$ through the corresponding outgoing or incoming links. Depending on the routing mechanism used, different types of network state information may be propagated along communication links in addition to the actual messages.

To determine the current load $\rho$ (or queue utilization), a node must be able to measure the current arrival rate $\lambda$ to the queue. Using its service rate $\mu$, the node
can derive the current load as:

\[ \rho = \frac{\lambda}{\mu} \]  

Clearly, different routing algorithms require different data structures in order to represent the state of the network upon which routing decisions can then be based. While random routing does not require any acquisition and propagation of network state information, other routing algorithms may require data structures that grow linearly with the size of the network. Distance vector and link state routing are examples of the latter.

One of the routing strategies used in our research is minimum distance routing. The network is based on a regular grid topology which enables nodes to determine their distance to the destination. A routing decision made by node \( n_i \) with respect to a message with destination node \( n_d \) would result in the selection of a neighbor node \( n_j \) which minimizes the remaining distance to \( n_d \). Different distance metrics can lead to different routing decisions. Hence, we have implemented both, euclidean as well as non-euclidean distance measures. Depending on the simulated network topology, some distance measures may be more applicable than others and can be implemented as necessary. An example of a non-euclidean distance measure is the Manhattan or city block distance

\[ M_{i,j} = |x_i - x_j| + |y_i - y_j| \]  

which is suitable for a regular grid topology.

The implementation of node class provides various functions to access private data members, including the necessary set- and get-functions to modify the rates at which node events are scheduled. These functions are used to dynamically alter the behavior of network nodes as a simulation experiment is conducted.
Communication Links  Like network nodes, communication links are also modeled as a single queue. The service provided by a communication links consists of the propagation of messages from its internal queue to the corresponding network node. A node $n_i$ can forward a message via link $l_{i,j}$ to neighbor node $n_j$. If no other messages are currently queued, the time until $l_{i,j}$ delivers the message to $n_j$ is determined by its corresponding link-bandwidth. If required, the queue in a link $l_{i,j}$ can be implemented so as to limit the available buffer space to a constant $k$. The $k + 1$ message appended by $n_j$ is then dropped. As the link bandwidth is assumed to be constant in bits/sec, the actual time spent to service a message depends on its size.

Link failure can easily be simulated by either setting the bandwidth to 0, or by removing the link entry in the corresponding node. In the first case, all messages will be lost (i.e., queued indefinitely). In the second case, link $l_{i,j}$ will not be considered for routing, thereby removing node $n_j$ from $n_i$’s neighbor set $H_i$. In the current implementation of our model we have chosen the second approach. The burden to deal with link failure is thus given to the network nodes.

As for service events in network nodes, post-scheduling (see Section 3.4) is used to schedule the removal of messages from the link queue.

Messages  A message in our network model can potentially contain any number of information fields needed to execute the simulation and allow for an efficient statistical evaluation of the model. The values of these fields can either be determined at the time of message creation or be acquired and altered as the message propagates among nodes towards its destination. Clearly, messages used in the simulation model
are an abstraction and should provide for the analysis of the model under test. The structure of the message class used for simulating traffic in a network environment does not necessarily coincide with the various protocol fields in actual messages.

The minimum information that should be available in a message is:

- Its identifier (ID), used to refer to different messages in the system;
- Sender and Receiver ID, used to make routing decisions;
- The message size, used to determine the service time at various queues;

Depending on the statistical variables to be analyzed other measures, such as hop-count and queueing time, may be accumulated as the message moves through the network.

Pre vs. Post Scheduling

Whenever a message arrives at a network node or link, it is appended to the central queue of the corresponding entity. In general, it is assumed that the message remains in the queue until service is completed. This implies that the message is serviced while it allocates the first position in the queue. Clearly, the completion of service, signified by the removal (or departure) of the message from the queue, constitutes an event in our OODES that must be scheduled. This *departure event* can be scheduled either upon message arrival or when the message enters service (i.e. enters the first position in the queue). We refer to the two strategies as *pre*- and *post-scheduling* respectively.

Let $t_s^M$ be the service time for a message $M$. For pre-scheduling, the completion of service is scheduled as soon as the message is appended to the queue.
Pre-scheduling requires queues to maintain information about the earliest possible time of service for a newly arriving message. Let $T$ be the current simulation time and let $t_{queue}$ be the earliest time an arriving message can be serviced. The completion and thus the removal of this message is then scheduled as an event for time $t_e$ where

$$t_e = t^M + \max(t_{queue}, T)$$  \hspace{1cm} (3.4)$$

A node or link may determine not to schedule the event $e$ for message $M$ if $t_e \geq T_{max}$ (where $T_{max}$ is the maximum simulation duration), as the event $e$ is not going to be dispatched during the lifetime of the simulation. For the same reason, message $M$ can be removed from the queue and thus memory resources are freed.

Pre-scheduling may be necessary when the corresponding server does not serve customers in strict first in first out (FIFO) order but instead interleaves the service on different customers. An example of such a system is a car repair service where customers are given a date and time when repair work on their car will be completed. The service provider, however, may choose to work on various cars in parallel, interrupt work, or postpone service on a car until the latest possible time. The earliest possible time for uninterrupted service $t_{queue}$ might hence depend on a complex set of parameters $\vec{Y}$, representing the state of the system and possibly external events; i.e.,

$$t_{queue} = F(\vec{Y})$$  \hspace{1cm} (3.5)$$

Post-scheduling on the other hand does not require the service completion to be scheduled upon message arrival at a queue. Instead, the completion event is scheduled at the time when message $M$ moves to the first position in the queue ready to be
serviced. With the respective service time $t^M_3$ the completion event is scheduled at:

$$t_e = t^M_3 + T$$  \hspace{1cm} (3.6)

Post-scheduling clearly reduces the number of events on the event list since it contains at most one completion event for any queue in the network. However, knowledge about whether or not an arriving message will ultimately be serviced during the simulation is not available. Hence, resources cannot be freed as with pre-scheduling. However, all messages in a queue can be deleted as soon as a completion event is scheduled at $t_e \geq T_{max}$.

We have chosen the post-scheduling approach in our simulation environment, as the primary objective is to simulate a large network environment with hundreds or even thousands of queues. The following example demonstrates the advantage of using post-scheduling vs. pre-scheduling in such an environment.

**Example** Consider an 1024-node network arranged as a $m \times n$ grid with $n = 32$ and $m = 32$. With a regular grid topology, this network contains

$$2 \times [(m - 1) \times n + m \times (n - 1)] = 3968$$ \hspace{1cm} (3.7)

unidirectional communication links. Assuming that each of the network nodes and each of the links are modeled as single M/M/1 queues, this model thus contains 4992 queues.

If we assume an average queue utilization $\rho$ of 0.66, the average number of messages in each of the queues is

$$\frac{\rho}{1 - \rho} \approx 2$$ \hspace{1cm} (3.8)
If post-scheduling is in effect, there is at most one completion event for each queue on the event list. Thus, at most 4992 events need to be scheduled in addition to events such as message generation and state computation events.

With pre-scheduling, one completion event must be scheduled for each message in the system. Hence, in average there are 9984 completion events on the event list. As \( \rho \) grows larger, the total number of events generated by all entities clearly becomes prohibitive, as the overhead associated with creating events and searching for their appropriate place in the event list degrades the performance of the simulation testbed.

**A Priori Service Time**

Our simulation testbed has been designed to execute on a single PC or workstation. In such a computing environment there is generally only a single random number stream \( R A N D \) available. Message inter-arrival rates, service times, and message destinations are generally based on random variate generations with individual calls to \( R A N D \). In general, the number of calls to \( R A N D \) for an individual message to be created and to propagate through the network until it is delivered to its destination node depends on the number of queues encountered. This, however, is a function of the routing algorithm used. As our research requires the trace of an individual message and the comparison of results with various parameter settings, it is imperative to maintain the same sequence of random numbers obtained from \( R A N D \). In general, this cannot be achieved if random variates are generated to determine the service time of a message \( M \) at every queue visited by \( M \) while propagating through the network.

In order to preserve a random sequence across various experiments, we have
chosen to simulate exponential service at every queue, by assigning a deterministic service rate in \( k \text{ bits/sec} \), and to vary message sizes according to an exponentially distributed random variate. However, the disadvantage of this approach is the possible magnification of error in the approximation of the exponentially distributed random sequence. This error may nevertheless be acceptable if the experiment does not rely on the precision of this approximation.

A second possible approach to preserve the random sequence is to pre-assign each message an exponentially distributed service time upon message generation. This approach, however, deprives the user of the possibility to model queues with different service rates.

In either approach, all necessary calls to \( \text{RAND} \) are thereby moved to the time of message creation thus preserving the same random sequence among experiments with different parameter settings.

**Verification of the Simulation Environment**

Various experiments have been conducted in order to verify the behavior of our implementation of the OODES and to identify its limitations. Since network nodes and links are based upon the implementation of a single queue, it is imperative to verify the correctness of these basic elements. Among other experiments, we have simulated the behavior of single \( M/M/1 \) queue and a single \( M/D/1 \) queue [2, 12, 17, 37].

A 1024-node network model is used to investigate the limitations and intricacies of simulating a large communication environment. These experiments as well as their results are described below.
Simulating M/M/1 and M/D/1 Queues  The simulation of a single queue in our OODES is achieved by simulating a communication network containing a single node. The message generator in this node can provide the necessary Poisson arrival of messages. The server can be manipulated to either determine the service time by using an exponentially distributed random variate (for M/M/1) on a per message basis, or by using an apriori assigned service time (for M/D/1) for all messages serviced.

Let $T_m$ be the time at which message $M$ enters the queue and let $T_M$ be the time at which service on $m$ is completed and $M$ is removed from the queue. The total delay experienced by $M$ is then given by:

$$D_M = T_M - T_m$$  \hspace{1cm} (3.9)

Upon completing service on $M$, $D_M$ is recorded by the corresponding function in the STATS-class of our OODES. After $T_{\text{max}}$ seconds of simulation time, the average delay $\overline{D}$ over all messages that are recorded during the interval $[0.1 \ T_{\text{max}}, 0.9 \ T_{\text{max}}]$ is reported. From queueing theory it is known that the mean delay depends on the utilization $\rho$ of the queue, i.e., $\overline{D} = f(\rho)$. From equation 3.2 it is apparent that the load or utilization $\rho$ depends on the mean message arrival rate of $\lambda$ and the mean service time, $1/\mu$.

Using the theoretical results from queueing theory we compute the mean delay for M/M/1 and M/D/1 as:

$$E[D] = \frac{1/\mu}{1 - \rho} \quad (M/M/1)$$ \hspace{1cm} (3.10)

$$E[D] = 1/\mu + \frac{\rho/\mu}{2*(1 - \rho)} \quad (M/D/1)$$ \hspace{1cm} (3.11)
Table 3.1: Expected vs. actual message delay in an M/M/1 queue for different values of $\rho$

| $\rho$     | $E[D]$     | $\bar{D}$   | $\frac{|E[D] - \bar{D}|}{E[D]}$ |
|----------|-----------|-------------|---------------------------------|
| 0.100000 | 0.111111  | 0.110758    | 0.003174                        |
| 0.150000 | 0.117647  | 0.117730    | 0.000702                        |
| 0.200000 | 0.125000  | 0.125196    | 0.001565                        |
| 0.250000 | 0.133333  | 0.133354    | 0.000152                        |
| 0.300000 | 0.142857  | 0.143110    | 0.001769                        |
| 0.350000 | 0.153846  | 0.154131    | 0.001852                        |
| 0.400000 | 0.166667  | 0.167745    | 0.006472                        |
| 0.450000 | 0.181818  | 0.182129    | 0.001711                        |
| 0.500000 | 0.200000  | 0.200482    | 0.002409                        |
| 0.550000 | 0.222222  | 0.222451    | 0.001028                        |
| 0.600000 | 0.250000  | 0.249536    | 0.001858                        |
| 0.650000 | 0.285714  | 0.283663    | 0.007179                        |
| 0.700000 | 0.333333  | 0.333133    | 0.000600                        |
| 0.750000 | 0.400000  | 0.400757    | 0.001892                        |
| 0.800000 | 0.500000  | 0.494867    | 0.010267                        |
| 0.850000 | 0.666667  | 0.651828    | 0.022258                        |
| 0.900000 | 1.000000  | 0.993463    | 0.006537                        |
| 0.950000 | 2.000000  | 2.018213    | 0.009106                        |

Our experiments involve the simulation of a single queue for 3600 seconds. The mean service rate is set to $10 \text{msgs/sec}$. In order to alter the queue utilization, $\rho$, the message generation rate $\lambda$ (effectively the arrival rate) is gradually increased from $1.0 \text{msgs/sec}$ to $9.5 \text{msgs/sec}$ in steps of $0.5 \text{msgs/sec}$. The mean message delay, $\bar{D}$ is computed for the various values of $\rho$, and compared to the theoretical derived expected message delay, $E[D]$. For each $\rho$, $\bar{D}$ is based on 50 model executions, each using a different random sequence. Tables 3.1 and 3.2 summarize and contrast the results of a single M/M/1 queue and a single M/D/1 queue, respectively.
Table 3.2: Expected vs. actual message delay in an M/D/1 queue for different values of $\rho$

| $\rho$     | $E[D]$  | $\overline{D}$ | $\frac{|E[D] - \overline{D}|}{E[D]}$ |
|------------|---------|----------------|--------------------------------------|
| 0.100000   | 0.105556| 0.105550       | 0.000053                             |
| 0.150000   | 0.108824| 0.108790       | 0.000309                             |
| 0.200000   | 0.112500| 0.112466       | 0.000305                             |
| 0.250000   | 0.116667| 0.116682       | 0.000133                             |
| 0.300000   | 0.121429| 0.121420       | 0.000074                             |
| 0.350000   | 0.126923| 0.127021       | 0.000768                             |
| 0.400000   | 0.133333| 0.133403       | 0.000525                             |
| 0.450000   | 0.140909| 0.140854       | 0.000394                             |
| 0.500000   | 0.150000| 0.150190       | 0.001269                             |
| 0.550000   | 0.161111| 0.161318       | 0.001282                             |
| 0.600000   | 0.175000| 0.175446       | 0.002550                             |
| 0.650000   | 0.192857| 0.192996       | 0.000720                             |
| 0.700000   | 0.216667| 0.216208       | 0.002117                             |
| 0.750000   | 0.250000| 0.248646       | 0.005417                             |
| 0.800000   | 0.300000| 0.298541       | 0.004862                             |
| 0.850000   | 0.383333| 0.378362       | 0.012969                             |
| 0.900000   | 0.550000| 0.544446       | 0.010098                             |
| 0.950000   | 1.050000| 1.035018       | 0.014268                             |

The model is simulated for 3600 seconds for different values of $\rho$. The mean service rate for each queue remained at $\mu = 10 \text{msgs/s}$ throughout the experiment. It follows that the number of messages from which $\overline{D}$ is derived increases as $\rho$ increases. The number of messages recorded in the 80%-interval $[0.1T_{max}, 0.9T_{max}]$ is at least 2800. For both, M/M/1 and M/D/1 queues, the actual message delay is within 2.5% relative error,

$$\frac{|E[D] - \overline{D}|}{E[D]},$$

(3.12)

despite the fact that the exponential random variate only approximates the true exponential distribution. The behavior of an individual queue is thus deemed sufficiently
precise to be used for the construction of larger network model.

**Simulation of a Large Network** To assess the performance of the OODES when executing large, complex models, we have simulated 300 seconds of traffic in a 1024-node network [31]. The nodes are arranged in a $32 \times 32$ grid topology. In this model, each node is implemented as a single M/M/1 queue. Each node generates messages at a rate of $0.3 \text{msgs/s}$. The destination for each message is chosen at random, self-traffic, however, does not occur. The mean service rate for each node remains static at $20 \text{msgs/s}$ throughout the course of the simulation. It is further assumed that communication links between nodes have sufficient bandwidth so as to regard transmission delays as negligible. Message delays are thus assumed to be caused solely by queueing delays encountered in network nodes.

In order to propagate a message from its source to its destination, each node follows the same routing algorithm. The algorithm will select among all neighbor nodes the one which minimizes the remaining euclidean distance to the destination. The load distribution in the network is therefore a function of the routing algorithm used. The respective load landscape is shown in Figure 3.6.

Depending on the system load, the simulation of 300 seconds of network traffic as described above is executed in approximately 4400 seconds on a HP 712/80 workstation. The utilization of individual queues clearly depends on the routing algorithm used in the model. Hence, it is difficult to predict whether or not there will be queues that experience infinite queueing (i.e., $\rho > 1.0$). It may indeed be necessary to rely on the result of short time sample execution of the model or intermittent sampling of the nodes' utilization. The consequence of infinite queueing in multiple nodes can
result in an unbounded resource requirement which may be impossible to be satisfied during the simulation time. However, it may be acceptable to permit a small set of nodes to enter the state of infinite queueing if the resource demand is bounded by the simulation time.

Summary

In the context of simulating large communication networks there clearly exists a tradeoff between the size of the model that is to be simulated and the resource demand to execute a simulation experiment. As the size of the network under test grows, the resource requirements may become prohibitive. Hence it is imperative to enable the user to determine the degree of detail at which the communication network is modelled.

Of-the-shelf simulation packages are generally very versatile and allow users to model a wide variety of different communication environments. However, user gen-
erally cannot control the degree of detail at which the individual modules have been implemented. As the simulation models increase in size, it may become necessary to trade off a detailed implementation for a better performance of the simulation environment. This may be particularly important if the model size is bounded by the resources that are available at the underlying computing environment.

An object oriented approach to designing and implementing a simulation toolbox can provide the necessary control of detail. The resulting modularity together with the concept of inheritance allows users to tailor the individual building blocks according to the individual need of the model under test. Such an approach demands more insight into simulation strategies such as, event scheduling.

The object oriented toolbox has been used to conduct the simulation experiments necessary to gain insight into the behavior of Quo Vadis. The simulation experiments as well as their results are described in Chapter 4.
CHAPTER 4. EXPERIMENTAL STUDY OF QUO VADIS

Simulation Experiments and Results

A prototype implementation of Quo Vadis was used to conduct a number of experiments to explore the effects of the various parameters used in Quo Vadis. These experiments were conducted in simple regular $m \times n$ grid networks. We anticipate that more general network topologies might present several additional specific issues that may need to be addressed by Quo Vadis. However, our primary objective in this paper was to study and understand the behavior of Quo Vadis within a relatively simple setting through a set of carefully designed experiments.

Quo Vadis has been implemented within an object-oriented discrete event-driven simulation environment [27, 31]. Each network node is represented as a single $M/M/1$ queue with infinite buffer space, guaranteeing that every message in the network will ultimately be delivered to its respective destination node. Upon arriving at a particular node, the message is added to the queue, awaiting service by the routing mechanism. The queuing discipline is strictly First-In-First-Out (FIFO), so that a message is stalled until all messages that arrived earlier at this node are serviced. Service consists of two possible actions:

1. If the routing mechanism determines that the message has reached its destination, it is passed on to the higher protocol layers. Within the simulation, this
entails the removal of the message from the network and the recording of its contribution to statistics for delay and hop-count.

2. If it is determined that the message needs to be passed on to other network nodes to further propagate towards its destination, the routing mechanism employs the heuristic decision mechanism (described above) to select a best next node.

The update of routing information is assumed to take place via a separate channel hence bypassing the FIFO queuing used for messages. Effectively, this could have been implemented through priority queuing, giving state change information the highest priority.

A message in the network is represented by general protocol information such as creation time, source node, destination node, hop-count and message ID together with a field that represents the simulated message size, i.e., the number of data bytes in the message. Additional protocol information may have to be associated with each message in order to enable nodes to adapt decision parameters and to perform well in various network topologies. This is currently being investigated and will be described in a forthcoming paper.

In order to study the effects of parameters $\alpha$, $\beta$, and $\gamma$, for each of the experiments an $m \times n$ grid network was simulated for $T$ seconds (real time).

To avoid biasing the results by the transient behavior of the networks at the beginning and the end of the simulation, statistics were recorded for only those messages that reached their destination during the time interval $(0.1T, 0.9T)$. Clearly, $T$ must be chosen such that a sufficiently large number of messages can be recorded, thus yielding a good approximation of the various means computed.
The Effects of $\alpha$

The parameter $\alpha$ determines how the composite load landscape of the network is reflected by the nodes' individual views. Therefore, both the distance over which a specific load condition can have impact on routing decisions as well as the degree of this impact are governed by $\alpha$.

As all parameters in Quo Vadis are tightly coupled a demonstration of the effects of $\alpha$ with respect to the view computation required the isolation of the knowledge representation from the overall routing mechanism. For this simulation experiment, a $10 \times 10$ grid network was set in a particular state corresponding to a pre-determined load distribution. The underlying motivation of this approach is to statically model various load conditions and to determine their impact on the view $V_i(t)$ as acquired by node $n_i$. In order to eliminate the effects of routing decisions on the load distribution in the network, nodes generated only self-traffic at a constant rate. Thus, messages did not have to be routed among network nodes but could be delivered to the node itself at a node's service rate. As a consequence, the values for parameters $\beta$ and $\gamma$ were rendered irrelevant for this experiment. The network together with its corresponding load distribution is shown in Figure 4.1.

Adverse load conditions were simulated by increasing the message generation rate at a single node (or a small number of nodes). Since no messages were sent across the network, the only information communicated among network nodes was view and load information. The views into each of the four directions (East, West, North, and South) as acquired after $T$ seconds of simulation by each individual node were then analyzed. This experiment was repeated for different values of $\alpha$ (0.1 through 1.0 in steps of 0.1). Figures 4.2 and 4.3 show the East-Views, $V_i^E$, as
acquired at every node $n_i$ in the network after $T$ seconds of simulation for different values of $\alpha$. It should be noted that Figures 4.2 and 4.3 do not display view values for nodes \{9, 19, 29, 39, 49, 59, 69, 79, 89, 99\}, as the East-Views in these nodes are undefined.

From equation 2.7 it is apparent that for $\alpha = 1.0$, a node $n_i$ computes its east-view $V_i^E$ solely as the weighted average of local load values $\rho_j$ obtained from neighbor nodes $n_j \in H_i$. The views, $V_j^E$, computed in neighbors $n_j$ do not contribute to $V_i^E$. Depending on the value of $\eta$ in equation 2.4, $V_i^E$ is a more or less precise image of $\rho_j$ computed in $n_j$ if $n_j$ is east of $n_i$ and $n_j \in H_i$. For smaller values of $\alpha$ (i.e., $\alpha = 0.6$), equation 2.7 takes the view $V_j^E$ of neighbors into account thus computing $V_i^E$ as an average of view and load measures of nodes in an extended neighborhood. That is, network nodes $n_k \notin H_i$ affect the magnitude of $V_i^E$. These effects are clearly displayed in Figure 4.2.

As $\alpha \to 0$ a load condition in a single node $n_k$ affects the view in a much larger
Figure 4.2: $V_i^E$ for $\alpha = 1.0$ and $\alpha = 0.6$ respectively

Figure 4.3: $V_i^E$ for $\alpha = 0.3$ and $\alpha = 0.1$ respectively
set of nodes. However, the magnitude of impact on the view $V_i^E$ is significantly reduced. Figure 4.3 shows the change of magnitude as a function of distance from $n_k$.

How views $V_i^d$, can be used to optimize performance in an anticipatory fashion is further highlighted in the study of effects of parameter $\gamma$.

**The Effects of $\beta$**

For the study of the effects of $\beta$ on the selection of routes, Quo Vadis was simulated in a 1024-node grid network for 300 seconds. Each of the $N = m \times n$ network nodes created messages at the same rate, i.e., $0.3 \text{msgs/s}$. The destination nodes for messages are chosen at random at message creation. Every node in the network has equal probability of being selected as destination node for a particular message. Self-traffic, however, does not occur. It is further assumed that links have sufficient bandwidth so that transmission delays are negligible. Message delays are thus assumed to be caused solely by queueing delays encountered in network nodes.

The following simulation results clearly demonstrate the success of Quo Vadis in selecting routes so as to reactively as well as pro-actively avoid highly utilized network areas. This behavior is governed primarily by the setting of the parameter $\beta$ in equation 2.8. To isolate the effect of $\beta$ on the performance of Quo Vadis, other parameters - namely, $\alpha$ and $\gamma$ - were maintained constant at $\alpha = \gamma = 0.5$.

**Shortest Path versus Quo Vadis Routing** From equation 2.8 it is apparent that choosing parameter $\beta = 1.0$ forces Quo Vadis to select routes so as to minimize the remaining distance to the destination node. This is equivalent to what is generally
Figure 4.4: Load distribution in a 1024 node grid network using shortest path routing i.e., $\beta = 1.0$ and $\beta = 0.4$ respectively

referred to as shortest path routing. In a grid topology, the number of shortest paths between a node $n_i$ and the destination node $n_d$ depends on their relative hop-distance. As one might expect, not all nodes in the grid network experience the same amount of traffic. In fact, nodes in the center of the grid network have to route a larger number of messages on average as compared to nodes at the fringes of the grid. This is due to the fact that a larger number of shortest paths between randomly chosen source-destination pairs pass through nodes in the center of the grid. The corresponding load-graph is shown in Figure 4.4. It clearly displays an increased load in nodes closer to the center of the grid and less load in those nodes at the grid’s edges.

As the message delay in a network node increases exponentially with its load, it follows that nodes in the center of the grid contribute most to the overall message delay along path traversed by the message. Thus, load at these nodes impacts the total message delay to a much higher degree than nodes at the fringes of the grid. This effect is amplified as the average network load increases. Quo Vadis delays the
Table 4.1: Mean Hop Count ($\bar{h}$) and Mean Message Delay ($\bar{d}$) for different values of $\beta$.

\[(n > 85700 \text{ messages})\]

<table>
<thead>
<tr>
<th>$\beta$</th>
<th>$\bar{h}$</th>
<th>$\bar{d}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>23.07</td>
<td>2.43</td>
</tr>
<tr>
<td>0.4</td>
<td>22.76</td>
<td>2.41</td>
</tr>
<tr>
<td>0.5</td>
<td>22.44</td>
<td>2.36</td>
</tr>
<tr>
<td>0.6</td>
<td>22.15</td>
<td>2.34</td>
</tr>
<tr>
<td>0.7</td>
<td>21.89</td>
<td>2.33</td>
</tr>
<tr>
<td>0.8</td>
<td>21.58</td>
<td>2.35</td>
</tr>
<tr>
<td>0.9</td>
<td>21.33</td>
<td>2.51</td>
</tr>
<tr>
<td>1.0</td>
<td>21.29</td>
<td>2.79</td>
</tr>
</tbody>
</table>

onset as well as reduces the impact of this effect given an appropriate setting of $\beta$.

While a shortest path routing algorithm makes a random decision among neighbors with equal path utility (equation 2.11), Quo Vadis takes network load into account and biases the selection towards neighbors with better load utility (equation 2.9). The price paid for the ability to circumvent a highly utilized network area is an increase in mean path length $\bar{h}$.

The means of path length and message delay for different values of $\beta$ are summarized in the Table 4.1.

Figure 4.5 shows the corresponding graphs for the $\bar{d}$ and $\bar{h}$. Figure 4.5 indicates the existence of an optimal value for $\beta$, $\beta^*$ that minimizes the mean message delay. An increase in the mean delay is observed for $\beta < \beta^*$ as the routing decisions are dominated by the load liability $L_k$. For $\beta \ll \beta^*$ the performance can approach that of random routing. For $\beta > \beta^*$, Quo Vadis approaches shortest path routing thereby causing an increased mean message delay as discussed above.

The load distribution in the network using Quo Vadis routing with different
values of $\beta$ is shown in Figure 3.6.

Clearly, a load sensitive setting of $\beta$ results in a more balanced distribution of load, thus preventing a single network area from becoming overutilized. If load vigilance is high (i.e., small $\beta$), routing decisions may result in extended path length. However, this does not necessarily lead to an increase in total message delay along the path if the message is routed through a lightly loaded area. The exponential increase in delay with increasing load justifies such a tradeoff. The following example clarifies this point:

Let $\mu = 10 \text{msgs/s}$ and consider two paths $P_1$ and $P_2$ with path lengths 5 and 3 respectively. Further assume the loads along $P_1$ to be

$$\rho_{1-5} = (0.3, 0.3, 0.2, 0.3, 0.4)$$

and loads along $P_2$ to be

$$\rho_{1-3} = (0.3, 0.8, 0.4).$$

While the total load along $P_1$ and $P_2$ are the same, equation 2.2 yields total delays
of 0.720 s and 0.810 s along $P_1$ and $P_2$ respectively. Though longer, $P_1$ clearly is a better choice when delay is to be minimized.

If routing decisions result in path $P_2$, the message not only experiences a larger delay, but in addition would make things worse for messages that cannot avoid intersecting $P_2$ on their way to their destination.

Routing in the Presence of Hotspots  Hot spot refers to a single node or a small group of nodes in the network that experience a sudden increase in utilization. Such hotspots may be due (among other things) to:

- localized increases in arrival rate, or
- localized node or link failures.

One of the desirable properties of a routing mechanism is its ability to react to such load changes. A good routing algorithm should attempt to route messages around the hotspot, thereby reducing the message delay, perhaps at the expense of increasing the total length of the route.

The ability to adapt to such localized load changes quickly has been deliberately designed into Quo Vadis. Nodes in the neighborhood of a suddenly over-utilized node start to divert traffic as soon as the load increase is made known to them. High load in an affected node (as in highly loaded network areas) has a repulsive effect on traffic and routing decisions are automatically biased towards avoiding that node. Again, the extent of this bias is determined by $\beta$. Such dispersion of traffic is accomplished with minimal impact on nodes that are sufficiently distant from those that are affected by local increases in load.
While the increase in a node's load should clearly repel messages from being routed though it, a sudden load decrease should be utilized by nodes in the neighborhood in their effort to distribute network load uniformly.

Sudden load changes have been simulated by increasing and decreasing a node's service rate. The effects of such a change when shortest path routing is in place is shown in Figure 4.6. The effects of adaptive measures taken by Quo Vadis are shown in Figures 4.7.

Shortest path routing (i.e., $\beta = 1.0$) does not attempt to reduce the influx of traffic into the affected area in order to normalize the load conditions at the hotspot. Quo Vadis, however, balances load conditions in the network in a relatively short time. This is accomplished by the dispersion of traffic which would otherwise have been routed through the hotspot area. The relationship between the time needed for the normalization of load conditions and parameters $\alpha$, $\beta$, and $\gamma$ is currently being investigated.

![Figure 4.6: Effects of sudden load increase in node $n_i$ under Shortest Path Routing](image)
Figure 4.7: Effects of Quo Vadis on sudden load changes in node $n_i$

The Effects of $\gamma$

In equation 2.9, $\gamma$ defines the significance of load measures $\rho_k$ versus $v_k$, the projections of a node's view $V_k(t)$ with respect to a particular destination. The underlying motivation is to enable network nodes to make routing decisions in either reactive or anticipatory fashion. For $\gamma = 1$, only $\rho_k$ determines the load liability of $n_k$, thereby enabling $n_i$ to route messages so as to circumvent the neighbor node $n_k \in H_i$ with the highest utilization, thus reacting to adverse load conditions in the immediate neighborhood. On the other hand, small values of $\gamma$ (i.e., $\gamma \to 0$) node $n_i$ will base its evaluation of neighbors $n_k$ on a load summary as represented by $V_k(t)$ with respect to the relative location of the destination. Hence, adverse load conditions on the path towards the destination can be sensed by $n_i$ so as to adjust the routing decision. As for the evaluation for $\alpha$, the isolation of the effects of $\gamma$ required the
network to remain in a pre-determined state. The corresponding network and load graph are shown in Figure 4.8.

In addition, nodes 40 and 49 have been selected to serve as source and destination nodes for a single message which is traced on its journey through the network. The purpose of the trace is to identify all nodes that are visited by the message thus revealing the routing decisions made by intermediate nodes. This experiment is repeated for various values of $\gamma$. Since $\beta$ controls the significance of the load liability, it has been chosen so as to amplify the effects of $\gamma$, i.e., $\beta$ was maintained constant at 0.2. The value of $\alpha$ was set to 0.3, thus making the effects of adverse load condition visible at distant nodes.

The different routes traveled by a test message are presented in Table 4.2 for various values of $\gamma$. Clearly, the shortest path between source node 40 and destination node 49 is given by (40, 41, 42, 43, 44, 45, 46, 47, 48, 49). However, the high utilization of node 48 forces the route to deflect. The nodes at which deflection occurs are
Table 4.2: Points of deflection for different values of $\gamma$

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>(40,30,31,32,33,34,35,36,37,38,39,48,49)</td>
</tr>
<tr>
<td>0.1</td>
<td>(40,41,31,32,33,34,35,36,37,38,39,49)</td>
</tr>
<tr>
<td>0.2</td>
<td>(40,41,31,32,33,34,35,36,37,38,39,49)</td>
</tr>
<tr>
<td>0.3</td>
<td>(40,41,42,32,33,34,35,36,37,38,39,49)</td>
</tr>
<tr>
<td>0.4</td>
<td>(40,41,42,43,33,34,35,36,37,38,39,49)</td>
</tr>
<tr>
<td>0.5</td>
<td>(40,41,42,43,33,34,35,36,37,38,39,49)</td>
</tr>
<tr>
<td>0.6</td>
<td>(40,41,42,43,44,45,55,56,57,58,59,49)</td>
</tr>
<tr>
<td>0.7</td>
<td>(40,41,42,43,44,45,46,47,57,58,59,49)</td>
</tr>
<tr>
<td>0.8</td>
<td>(40,41,42,43,44,45,46,47,57,58,59,49)</td>
</tr>
<tr>
<td>0.9</td>
<td>(40,41,42,43,44,45,46,47,57,58,59,49)</td>
</tr>
<tr>
<td>1.0</td>
<td>(40,41,42,43,44,45,46,47,57,58,59,49)</td>
</tr>
</tbody>
</table>

printed in bold. Table 4.2 shows that for large values of $\gamma$ deflection takes place only when adverse load conditions are encountered in the immediate neighborhood; (i.e.; $n_{47}$ deflects as $n_{48} \in H_{47}$ experiences a high utilization.) Small values of $\gamma$, force equation 2.9 to attach a higher significance to the view projection $v_k(t)$, which reflects the adverse load conditions at node 48. As a consequence, nodes can take anticipatory action and deflect earlier.

Summary

The functional framework of Quo Vadis, as introduced in Chapter 2 aims at modeling the behavior of a traveler. This traveler is faced with the task of having to choose among different available routes at each intermediate step of the journey. Choices are to be made such that the resulting route from the beginning to the end of the journey displays certain qualities as defined by a performance metric.

The experiments presented in this chapter were designed to study the impli-
cations of different parameter settings on the behavior of Quo Vadis. The results
demonstrate that Quo Vadis is successful in meeting its primary design objectives,
at least in a restricted grid topology.

Chapter 5 presents a theoretical approach to designing a set of functions for
making routing decision in large networks. This approach draws upon concepts and
ideas from the field of utility theory.
CHAPTER 5.  ROUTING DECISIONS BASED ON UTILITIES

Rewards and Penalties

The problem of routing messages in a large communication network is made difficult by the fact that any routing algorithm has to deal with inherent uncertainty or imprecision of network state information. The approach taken by Quo Vadis is to rely on locally available information which is supplemented by global summary information. This available network state information is used to compute an estimate of a node's promise to be used as intermediate node to forward a message $M$ towards its destination. A network node $n_i$ makes a routing decision for $M$ by selecting a neighbor with the largest promise with respect to $M$'s destination. The objective is to design a set of functions which would result in "good" routing decisions, hence producing routes which meet given performance criteria, despite incomplete and uncertain knowledge of the network state.

For simplicity, we will limit the scope of the design of these functions to a regular grid network topology. The grid consists of $n \times m = N$ network nodes, with adjacent nodes located unit distance from each other. Furthermore, we assume that the quality of a route is determined by its ability to circumvent network areas under heavy load which is reflected by the cumulative delay.
The Reward Function

The goal of the routing algorithm is to propagate a message $M$ along a route which ultimately reaches $M$'s destination. Hence, routing algorithms must have some form of guidance information available that the emerging route upon which a message $M$ is propagated approaches the destination node.

Assuming that $M$ receives some type of reward upon reaching its destination, we can define a partial reward for every node along each route which serves as intermediate reward for making a particular routing decision. The amount of partial reward in a node $n_i$ is an inverse function of distance from the destination $n_d$. Any routing algorithm which maximizes the partial reward at every intermediate node on the route will guide a message towards its destination.

Let $R^d$ be the reward landscape for the network with respect to destination a node $n_d$. Let $D_{i,d}$ denote the Manhattan (or city block) distance between an individual node $n_i$ and $n_d$. We define the partial reward for node $n_i$ as

$$R^d_i = f_R(D_{i,d})$$

such that $D_{i,d} \leq D_{j,d} \iff R^d_i \leq R^d_j \forall i, j$. There are many possible choices for the reward function $f_R(.)$, e.g., linear, polynomial, or exponential. An example of $f_R(.)$ is given by

$$R^d_i = (m + n) - \frac{D_{i,d}}{m + n}$$

where $n$ and $m$ are the dimensions of the grid network. The corresponding reward landscape for an $11 \times 11$ grid network with destination node $n_d = (10, 5)$ is shown in Figure 5.1.

The cumulative reward $R^P$ that is obtained along a path $P$ leading from $M$'s
source $n_s$ to $M$'s destination $n_d$ is

$$R^d_i = (m + n) - \frac{D_{i,d}}{m+n}$$

Penalty and Payoff

At each intermediate node $n_i$ along a path $P$, the message $M$ will incur a penalty or cost, $C_i$. $C_i$ is assumed to be non-negative and bounded by some constant $\xi$, such that $\forall i \ 0 \leq C_i \leq \xi$. $C_i$ is assumed to be constant in the time interval in which a routing decision is being made. Considering cumulative delay as the performance criterion that is to be minimized by the routing algorithm, $C_i$ corresponds to the message delay in $n_i$. Hence, $\xi$ is to chosen to correspond to the maximum tolerable delay. A node $n_i$ can maintain the bound $\xi$ by discarding a fraction of its message.
influx which otherwise would yield $C_i > \xi$. If cumulative load is the criterion to be minimized, $C_i$ is bounded by the maximum utilization $\rho \leq 1.0$.

The total cost incurred along path $P$ is defined as

$$C^P = \sum_{n_i \in P} C_i$$  \hspace{1cm} (5.4)

Given node $n_i$'s partial reward $R^d_i$ and penalty $C_i$ we define the partial payoff which a message $M$ receives upon visiting $n_i$ on a path towards its destination node $n_d$ as

$$Z^d_i = R^d_i - C_i$$  \hspace{1cm} (5.5)

Correspondingly, the total payoff along path $P$ is defined as

$$Z^P = R^P - C^P$$  \hspace{1cm} (5.6)

Let $\Pi$ represent a minimum cost path from a source $n_s$ to a destination $N_d$. The cost $C^\Pi$ along this path is given by:

$$C^\Pi = \min_{\forall \Pi} \{ C^P \}$$  \hspace{1cm} (5.7)

Definition 1 If $\forall i, C_i = \kappa$ ($0 \leq \kappa \leq \xi$), we refer to the network as a uniform cost network.

Lemma 1 In a uniform cost network a routing algorithm which propagates a message $M$ such that $Z^d_i$ is maximized at every intermediate step will yield an optimal path $\Pi$ with cost $C^\Pi$.

Proof of Lemma 1

Since $\forall i, C_i = \kappa$, the partial reward $Z^d_i$ in equation 5.5 can hence be rewritten
as \( Z_i^d = R_i^d - \kappa \). Maximizing \( Z_i^d \) at each intermediate step along route \( P \) is hence equivalent to maximizing \( R_i^d \). Let \( \Lambda_P \) be the number of nodes on path \( P \). As message \( M \) is propagated along a \( P \) such that \( R_i^d \) is maximized at every intermediate step, the property of the reward function guarantees that \( M \) is propagated along a shortest path \( P \) i.e., \( \Lambda_P \leq \Lambda_{P'} \forall P'. \) Hence equation 5.4 can be written as \( C^P = \kappa \Lambda_P \). Since \( P \) is a minimum hop path, it follows that \( C^P = C_{II} \). □

Routing with Utilities

Utility is a measure that quantifies a decision maker’s preference for one choice over another. While the numerical value of payoff often serves as utility and allows for the preference-ordering of choices, different utility functions must be defined if the payoff is not monetary. The utility function may combine different types of payoffs, incorporate the notion of uncertainty, and weight the various components so as to reflect the bias (or strategy) of the decision maker \([8, 9, 45]\).

Definition 2 The utility \( U_i^d \) of node \( n_i \) with respect to destination \( n_d \) is its promise to a neighbor node \( n_j \) to route a message \( M \) to \( n_d \) on a minimum cost path.

Thus far, the partial payoff, \( Z_i^d \), has been implicitly used as utility to guide a message \( M \) on a minimum cost path towards its destination, i.e., \( U_i^d = Z_i^d \). However, the above result only holds if a uniform cost network is assumed. The uniform cost assumption renders the cost component in the payoff function inconsequential to the routing decision. A node \( n_i \) preference-orders its neighbor nodes according to their respective payoffs. Node \( n_i \) is indifferent to the choice of one neighbor, \( n_k \), over a second neighbor, \( n_l \), if \( Z_k^d = Z_l^d \). Hence, the a random choice is made between \( n_k \).
and \( n_l \). We denote the indifference between two nodes as \( n_k \sim n_l \). In all other cases, \( n_k \) is preferred to \( n_l \) or \( n_l \) is preferred to \( n_k \), denoted as \( n_k > n_l \) and \( n_l > n_k \), respectively.

**Definition 3** A hotspot as a single network node with a utilization that is significantly larger than the utilization in nodes in its vicinity. Hence, a hotspot incurs a comparatively higher cost \( C_h \) on a message \( M \) if routed through it than other nodes.

**Definition 4** The neighborhood \( H_i \) of a node \( n_i \) is the set of nodes \( n_j \), such that there exists a communication link \( l_{i,j} \) from \( n_i \) to \( n_j \).

**Routing in a Uniform Cost Network with a Single Hotspot** As the uniform cost assumption is relaxed, such that a single hotspot \( n_h \) with cost \( C_h \gg C_j \ \forall j \neq h \) is introduced into the network, using partial payoff as utility to route \( M \) may prove too naive as it may unnecessarily result in sub-optimal routes. Consider a grid network with node coordinates increasing as a message \( M \) travels east and south.

Assume \( C_i = C_j \ \forall i,j \neq h \). Let \( x_s, y_s, x_d, \) and \( y_d \) be the x- and y-coordinates of \( M \)'s source and destination, respectively. Let \( x_h \) and \( y_h \) be the x- and y-coordinates of a hotspot with one of the 2 configurations:

1. \( x_s \leq x_h \leq x_d \land y_s \leq y_h \leq y_d \) or
2. \( x_s \geq x_h \geq x_d \land y_s \geq y_h \geq y_d \)

That is, the probability of a shortest path from \( n_s \) to \( n_d \) to go through the hotspot \( n_h \) is non-zero. Hence, there exists a node \( n_i \) with \( n_h \in H_i \) that must decide how to route \( M \) so as to minimize the total cost. In the following we distinguish 3 canonical
Figure 5.2: $x_s < x_h < x_d \land y_s < y_h < y_d$

scenarios. The arguments are based on configuration 1 above, however, equivalent arguments hold for configuration 2.

Case 1

Case 1 is shown in Figure 5.2. Here, the hotspot $n_h$ does not share any of the $x$- and $y$-coordinates of either $n_s$ and $n_d$, i.e., $x_s < x_h < x_d \land y_s < y_h < y_d$. In this configuration of nodes $n_d$, $n_h$, and $n_d$ the number of shortest paths from $n_s$ to $n_h$, $P\Delta_{s,h}$, is given by:

$$P\Delta_{s,h} = \frac{(x_h - x_s) + (y_h - y_s)}{(x_h - x_s)}$$

(5.8)

Each of these paths would require a node $n_i$ with $n_h \in H_i$ to route $M$ so as to circumvent $n_h$. Here, the utility function $U_i^d = Z_i^d$ does suffice to route $M$ on a minimum cost path to its destination $n_d$.

**Definition 5** Let $U^0$ be the utility function defined by $U_i^d = Z_i^d$. 
Lemma 2 In a uniform cost network with a single hotspot \( n_h \) located such that \( x_s < x_h < x_d \) \& \( y_s < y_h < y_d \), a routing algorithm which propagates a message \( M \) such that \( U^0 \) is maximized at every intermediate step will yield an optimal path \( \Pi \) with cost \( C_{\Pi} \).

Proof of Lemma 2

Clearly, the only nodes at which a decision has to be made to circumvent \( n_h \) are at nodes \( n_i \) or \( n_j \) with coordinates \( (x_h - 1, y_h) \) and \( (x_h, y_h - 1) \), respectively. Since \( x_h < x_d \) \& \( y_h < y_d \), there exist nodes \( n_k \) and \( n_l \) with coordinates \( (x_h - 1, y_h + 1) \) and \( (x_h + 1, y_h - 1) \), respectively, that lie on a minimum hop path from \( n_s \) to \( n_d \). Since \( C_k = C_l << C_h \) it follows that \( Z^d_k = Z^d_l >> Z^d_h \). Hence, a routing decision in \( n_i \) or \( n_j \) that maximizes the partial payoff will choose \( n_k \) or \( n_l \) to propagate \( M \) towards \( n_d \). Since \( C_i = \kappa \) \forall i \neq h \), and \( M \) is propagated along a minimum hop path, Lemma 1 guarantees that \( M \) is routed along an optimal path \( \Pi \). \( \square \)

Case 2

Here, \( n_s, n_d, \) and \( n_h \) are placed such that \( x_s < x_h < x_d \) \& \( y_s < y_h = y_d \) or \( x_s < x_h = x_d \) \& \( y_s < y_h < y_d \) (see Figure 5.3). Assuming the former, there exists a node \( n_i \) with coordinates \( (x_i, y_i) \) with \( x_s < x_i < x_h \) \& \( y_i = y_h = y_d \) from which the number of minimum hop routes is given by:

\[
P_{i,d} = \frac{(x_i - x_d) + (y_i - y_d)}{(x_i - x_d)} = 1
\]

(5.9)

Since in a uniform cost network \( n_k \sim n_l \) \forall k, l \neq h \) the naive utility function \( U^0 \) may guide message \( M \) through \( n_i \), thereby committing to a path \( P \) with cost \( C_P > C_{\Pi} \). Assuming that \( M \) is only routed along minimum hop routes, the additional cost
(\(C_P - C_{\Pi}\)) is inflicted on \(M\) by \(n_h\). If \(M\) is permitted to deflect from a minimum hop route, the additional cost \((C_P - C_{\Pi})\) is either inflicted by \(n_h\) itself or due to the extended length of \(P\) due to circumventing \(n_h\).

**Case 3**

This scenario consists of all placements of \(n_s, n_d,\) and \(n_h\) such that \(x_s = x_h = x_d \land y_s \leq y_h \leq y_d\) or \(x_s \leq x_h \leq x_d \land y_s = y_h = y_d\) (see Figure 5.4). Since there is only a single optimal path \(\Pi\) from \(n_s\) to \(n_d\), i.e., \(P\Delta_{s,d} = 1\), message \(M\) must either visit \(n_h\) or deflect from the minimum hop path in order to circumvent \(n_h\). \(U^0\), however, does not use sufficient information to guarantee an optimal routing decision. Hence, \(M\) may be along a path \(P\) for which \(C_P > C_{\Pi}\).

**Assumption 1** In the following we assume that a node \(n_i\) upon receiving a message \(M\) from a neighbor node \(n_j\) will refrain from propagating \(M\) back to \(n_j\).
Lemma 3 In a uniform cost network with a single hotspot $n_h$, a routing algorithm based on $U^0$ will deflect a message $M$ at most once in order to circumvent $n_h$.

Proof of Lemma 3

Consider a node $n_i$ with coordinates $(x_i, y_i)$ such that $x_s < x_i = x_h - 1 < x_d \land y_s < y_i = y_h = y_d$. (an equivalent argument can be made for $x_s = x_i = x_h < x_d \land y_s < y_i = y_h - 1 < y_d$) Node $n_i$ can deflect $M$ to a node $n_j$ with coordinates $(x_j, y_j)$, such that $x_s < x_j = x_h - 1 < x_h < x_d \land y_s < y_j = y_h \pm 1$. Clearly, $P \triangle j,h = 2$. Since $x_h < x_d$, $P \triangle j,d > 2$. Hence there must exist a node $n_k$ with $n_k \in H_j$ which lies on a minimum hop path $P$ from $n_j$ to $n_d$ such that $n_h \not\in P$. The property of the reward function 5.1 guarantees the $R^d_i = R^d_k > R^d_j$. In a grid topology, $\forall n_l \ n_l \neq i, k$ such that $n_l \in H_j \land R^d_l \geq R^d_k$. Since $C_j = C_k = C_i$, $Z^d_k = Z^d_k < Z^d_k$. By assumption 1, $M$ will not be routed back to $n_i$ but instead through $n_k$, on path $P$ from $n_j$ to $n_d$. Since $n_h \not\in P$, Lemma 1 guarantees that $M$ is propagated along $P$ without further deflection. $\Box$
Theorem 1  In a uniform cost network with a single hotspot \( n_h \) with \( C_h > C_i \) \( \forall i \neq h \), a routing algorithm which propagates a message \( M \) such that \( U^0 \) is maximized at every intermediate step will yield a path \( P \) with cost \( C^P \) such that \( C^P - C^\Pi \leq \max((C_h - C_i),2C_i) \).

Proof of Theorem 1

In case 1, Lemma 2 guarantees that a routing algorithm based on \( U^0 \) will find a minimum cost path if \( n_s, n_h, \) and \( n_d \) are placed such that \( x_s < x_h < x_d \) and \( y_s < y_h < y_d \). Hence, \( C^P = C^\Pi \) and thus \( C^P - C^\Pi = 0 < \max((C_h - C_i),2C_i) \).

Case 2 involves a node \( n_i \) with coordinates \((x_i,y_i)\) such that \( x_s < x_i = x_h - 1 < x_d \) and \( y_s < y_i = y_d \) or \( x_s = x_i = x_h < x_d \) and \( y_s < y_i = y_h - 1 < y_d \), which must decide whether to route message \( M \) through \( n_h \) or to deflect from a minimum hop path. Routing through \( n_h \) will yield \( C^P \) which is sub-optimal by \( C_h - C_i \), i.e., \( C^P - C^\Pi = C_h - C_i \leq \max((C_h - C_i),2C_i) \). If \( n_i \) chooses to deflect \( M \) so as to circumvent \( n_h \), \( M \) is propagated along a path \( P' \). Let \( \Lambda_P \) be the length of the minimum hop path \( P \) from \( n_i \) to \( n_d \) via \( n_h \). Let \( \Lambda_{P'} \) be the length of path \( P' \). Deflecting from path \( P \) in a grid topology yields a path \( P' \) with \( \Lambda_{P'} = \Lambda_P + 2 \). Since Lemma 3 guarantees that \( M \) is deflected at most once, \( C^{P'} = C^\Pi + 2C_i \). Hence \( C^{P'} - C^\Pi = 2C_i \leq \max((C_h - C_i),2C_i) \).

In case 3, the minimum cost \( C^\Pi \) for a path between \( n_s \) and \( n_d \) is given by \( \Pi = \Lambda_P C_i + \min(C_h - C_i,2C_i) \). Hence, \( C^{P'} - C^\Pi \leq \max((C_h - C_i),2C_i) \).

If \( n_h \) coincides with either \( n_s \) or \( n_d \), the hotspot cannot be circumvented and \( C^P = C^\Pi \). Clearly, \( C^P = C^\Pi \) and \( 0 < \max((C_h - C_i),2C_i) \). Therefore, \( C^P - C^\Pi \leq \max((C_h - C_i),2C_i) \forall P \). \( \qed \)
The Probability of Sub-Optimality  Let the probabilities of a sub-optimal path due to cases 1, 2, and 3 be $\phi_1$, $\phi_2$, and $\phi_3$, respectively. Suboptimal paths due to routing decisions that are based on $U^0$ only emerge from cases 2 and 3 above, hence, $\phi_1 = 0$. The probability of sub-optimal routes $P_{-\Pi}$ in an N-node $m \times n$ grid network is therefore given by:

$$pr(P_{-\Pi}) = \phi_1 + \phi_2 + \phi_3$$ \hspace{1cm} (5.10)

The probability $\phi_2$ is computed as:

$$\phi_2 = \sum_{W} \sum_{H_x} pr(P_{-\Pi,x}) + \sum_{W} \sum_{H_y} pr(P_{-\Pi,y})$$ \hspace{1cm} (5.11)

where $pr(P_{-\Pi,x})$ and $pr(P_{-\Pi,y})$ are the probabilities of sub-optimal paths when $x_h = x_d$ and $y_h = y_d$, respectively. Here, $W$ be the set of all possible placements of $n_s$ and $n_d$ such that $|x_s - x_d| > 0$ and $|y_s - y_d| > 0$. $H_x$ be the set of hotspot placements $n_h$ such that $x_h$ lies strictly between $x_s$ and $x_d$ and $y_h = y_d$. $H_y$ be the set of hotspot placements $n_h$ such that $y_h$ lies strictly between $y_s$ and $y_d$ and $x_h = x_d$. Let $n_i$ be a node with coordinates $(x_i, y_i)$ such that $x_i = x_h - 1$ and $y_i = y_h$ in the case where $x_s < x_h < x_d$ ($x_i = x_h + 1$ and $y_i = y_h$ if $x_s < x_h > x_d$), with $n_h \in H_x$ (or $n_h \in H_y$ correspondingly). Define the number of minimum hop paths from $n_s$ to $n_i$ as $P_{\Delta,s,i}$. Correspondingly, define the number of minimum hop paths from $n_s$ to $n_d$ to be $P_{\Delta,s,d}$. The probabilities $pr(P_{-\Pi,x})$ and $pr(P_{-\Pi,y})$ are given by:

$$\frac{P_{\Delta,s,i}}{P_{\Delta,s,d} \cdot N^3}$$ \hspace{1cm} (5.12)

where $1/N^3$ is the probability of a particular configuration of $n_s$, $n_d$, and $n_h$ in the $m \times n$ grid.
The number of possible sub-optimal routing decisions in case 3 depends on the relative cost difference $C_h - C_i$. Hence, it can only be bounded by the number of possible case 3 node placements in an $n \times m$ network. The resulting probability of sub-optimality due to case 3, $\phi_3$, is given by:

$$\phi_3 \leq \frac{2n\binom{m}{3} + 2m\binom{n}{3}}{N^3}$$

(5.13)

**Corollary 1** The expected penalty for choosing sub-optimal routes in a $m \times n$ uniform cost network with a single hotspot due to routing decisions based on $U^0$ is bounded by $pr(P_{-\Pi}) \max((C_h - C_i), 2C_i)$.

**Extending the Utility Function** So far, routing decisions in node $N_k$ are based on the simple utility function $U_i^d = Z_i^d$ which preference-orders nodes $n_i$ in $H_k$ according to the payoff $Z_i^d$ that can be attained. $Z_i^d$, however, is determined solely from local information in $n_i$ such as the reward $R_i^d$ and the cost $C_i$.

Sub-optimal routes, as described above, are primarily due to the lack of information about the remaining path $P$ from $n_i$ to a message $M$'s destination $n_d$. Both, scenario 2 and 3 are configurations of node placements that can lead to a sub-optimal path $P$ from $n_s$ to $n_d$, that is $C_P - \Pi > 0$.

This warrants an extension to $U^0$ such that the expected cost along the remaining portion of $P$ at every intermediate step is taken into consideration. In an intermediate step, we can define utility function $U^1$, such that the expected penalty due to sub-optimal routing is bounded by $pr(P_{-\Pi}) \min((C_h - C_i), 2C_i)$.

**Definition 6** Let $U^1$ be a utility function defined by
\[ U^1 = \begin{cases} 
R_j^d & \text{if } (C_j > C_i) \land (C_j - C_i < 2C_i) \land \exists k (R_k^d = R_j^d) \land (n_j \neq n_d) \\
Z_j^d & \text{otherwise} 
\end{cases} \] (5.14)

\( U^1 \) clearly relies on the fact that messages are to be routed in a uniform cost network with single hotspot. While \( U^1 \) does not eliminate the possibility of sub-optimal routing decisions, it optimizes the routing decision in a node \( n_i \) with hotspot \( n_h \in H_i \). With \( U^1 \), \( M \) is guaranteed to be propagated along an optimal path \( P \) from \( n_i \) to \( n_d \). Using \( U^1 \), \( n_i \) can decide whether or not to propagate \( M \) through \( n_h \) or to circumvent \( n_h \) by forwarding \( M \) through a neighbor node \( n_k \), i.e.,

- \( C_h - C_i > 2C_i \iff n_k \succ n_h \) and
- \( C_h - C_i < 2C_i \iff n_h \succ n_k \).

Hence, sub-optimality due to case 3 scenarios are eliminated, that is, in equation 5.13, \( \phi_3 = 0 \).

**Corollary 2** The expected penalty for choosing sub-optimal routes in a \( m \times n \) uniform cost network with a single hotspot due to routing decisions based on \( U^1 \) is bounded by \( \phi_2 \min((C_h - C_i), 2C_i) \).

**Eliminating Sub-Optimality** The fact that \( \phi_2 > 0 \) is due to the lack of information in a node \( n_i \) about the cost along paths from \( n_i \) to \( n_d \). Any routing decision in case 2 that propagates \( M \) to a node \( n_k \) with \((x_k < x_h < xd \land y_k = y_h = y_d) \) or \((x_k = x_h = xd \land y_k < y_h < y_d) \) will yield a sub-optimal path.
Node $n_k$ is selected by $n_i$ as the result of a random experiment which is required since in a uniform cost network $\forall j, k \neq h n_k \sim n_j$. Note that case 2 encompasses all placements of $n_s$, $n_h$, and $n_d$, such that $\forall \{n_i \mid x_i \neq x_d \wedge y_i \neq y_d\} \exists k, l$, such that $(n_k \in \Pi) \vee (n_l \in \Pi)$. Hence, an estimate of the cost along paths from $n_k$ to $n_d$ as a component of the utility function could perform as a tie-breaker, allowing for preference-ordering $n_k$ and $n_l$, i.e.,

- $(n_k \in \Pi) \wedge (n_l \not\in \Pi) \implies n_k \succ n_l$ and
- $(n_k \not\in \Pi) \wedge (n_l \in \Pi) \implies n_l \succ n_k$.

**Definition 7** A cost estimator $E^d_k(.)$ is a function which estimates the expected cost $E^d_k$ along paths to $n_d$ through node $n_k$, such that an appropriate preference ordering is maintained, i.e.,

- $(n_k \in \Pi) \wedge (n_l \not\in \Pi) \implies E^d_k < E^d_l$ and
- $(n_k \not\in \Pi) \wedge (n_l \in \Pi) \implies E^d_l < E^d_k$.

**Definition 8** Assuming the existence of an estimator that provides a sufficiently precise estimate of the expected minimum cost along paths, we define a utility $U^2$ as

$$
U^2 = \begin{cases} 
U^1 & \text{if } x_s = x_d \vee y_s = y_d \\
R^d_i - C_i - E^d_i & \text{otherwise}
\end{cases}
$$

(5.15)

**Assumption 2** It is assumed that $R^d_i > (C_i + E^d_i)$ everywhere in the network, i.e., $\forall i \ U^2_i \geq 0$.

**Assumption 3** In the following we assume that the routing algorithm will only consider minimum hop paths from $n_s$ to $n_d$, i.e., $\forall j, k$ such that $R^d_j < R^d_k$, $(R^d_j - C_j - E^d_j) < (R^d_k - C_k - E^d_k)$ everywhere in the network.
Clearly, the value returned by $E^d_k(\cdot)$ must be based on some knowledge of the current cost distribution in the network. As $E^d_k$ is computed at the time a message $M$ is to be routed, this knowledge must be assimilated at a different time scale, preferably in a continuous fashion so as to reflect changes in network load. However, the representation of network cost distribution cannot be specific to a particular destination node $n_d$ since $n_d$ is only specified at the time when a routing decision is to be made for $M$. Hence, we define a view, $V_k$, which is maintained in every node in the network. This view can be decomposed into four components, one for each of the four directions - north, south, east, and west. Thus we have: $V_k = [V^N_k, V^S_k, V^E_k, V^W_k]$.

Each component $V^\delta_k : (\delta \in \{N,S,E,W\})$ represents a weighted average of costs $C_i$ along the minimum hop path from $n_k$ to the $\delta$-border of the grid network. Consider two nodes, $n_i$ and $n_k$, located such that $n_k \in H_i$ and $n_k$ is east (to the right) of $n_i$, i.e., $x_i < x_k \land y_i = y_k$. $V^E_i$ is given by:

$$V^E_i = \frac{C_k + V^E_k}{2}$$ (5.16)

$V^N_i, V^S_i,$ and $V^W_i$ are computed correspondingly. Assuming that $n_d$ is located such that $x_s < x_d \land y_s < y_d$. Let $D^x_i = |x_i - x_d|$ and $D^y_i = |y_i - y_d|$ denote the distance from $n_i$ to $n_d$ in $x$ and $y$ direction, respectively. Based on $V^E_i$, $E^d_i(\cdot)$ is given by:

$$E^d_i(\cdot) = \frac{D^x_i V^E_i + D^y_i V^S_i}{D^x_i^2 + D^y_i^2}$$ (5.17)

The estimator defined by equation 5.17 is one of many alternatives to estimate the cost into a particular direction. As $E^d_i(\cdot)$ is based on the view $V^E_i$, it is assumed that the value of $V^\delta_k : (\delta \in \{N,S,E,W\})$ has converged at the time of computing $E^d_i(\cdot)$. 
Since $\forall i$, $C_i \leq \xi$, equation 5.16 guarantees that $V_0^\delta \leq \xi$. Hence, by equation 5.17 $E_1^d(\cdot) \leq \xi$. Clearly, $(C_i + E_1^d) \leq 2\xi$. Assumption 2 can be satisfied if $\forall i$, $R_i^d \geq 2\xi$. Any reward function that yields $|R_i^d - R_k^d| \geq 2\xi, \forall k \in H_i$ will hence satisfy assumption 3.

**Theorem 2** In a uniform cost network with a single hotspot $n_h$ with $C_h > C_i \forall i \neq h$, a routing algorithm which propagates a message $M$ such that $U^2$ is maximized at every intermediate step will yield a path $\Pi$.

**Proof of Theorem 2**

Consider the placement of $n_s$ and $n_d$, such that $x_s < x_d \land y_s < y_d$ (other placements can be proven correspondingly). For nodes $n_i, n_j, \text{ and } n_k$ for which $x_i, x_j, x_k < x_h \land y_i, y_j, y_k < y_h$ and $n_j, n_k \in H_i \land R_i^d < R_j^d = R_k^d, n_i \sim n_k$ with respect to routing decisions made by $n_i$. Hence, a message will be propagated until a routing decision has to be made which involves a node $n_k$ with coordinates $x_k = x_h \land y_k < y_h$ or $x_k < x_h \land y_k = y_h$. At this point, the utility of $n_k$ is reduced due to $E_i^d$ and node $n_j$ with coordinates $x_j < x_h \land y_j < y_h$ or $x_k < x_h \land y_k < y_h$ will be preferred over $n_k$. Therefore, $M$ will be propagated to a node $n_l$ with coordinates $x_l = x_h - 1 \land y_l = y_h - 1$. We can now show that $M$ will always circumvent $n_h$ and is propagated along $\Pi$.

For case 1 scenarios, we have $x_h < x_d \land y_h < y_d$. Consider the two possible routing decisions $n_j$ and $n_k$ with coordinates $x_j = x_h - 1 \land y_j = y_h$ and $x_k = x_h \land y_k = y_h - 1$. Since both, $n_j$ and $n_k$ offer a minimum cost path to $n_d$, either decision will yield a path $\Pi$. As $C_h >> C_i$ for $n_l \in H_k$ or $n_l \in H_j$ and $x_h < x_d \land y_h < y_d$, $M$ will circumvent $n_h$ while approaching $n_d$. Hence, for case 1 scenarios, $U^2$ guarantees that $M$ is propagated along $\Pi$. 

Consider the case 3 scenario where routing decisions in node \( n_j \) involve nodes \( n_j \) and \( n_k \) with coordinates \( x_j = x_h - 1 \) \( y_j = y_d \) and \( x_k = x_h \) \( y_k = y_h - 1 \). Clearly, a routing decision that would yield \( n_j \) will commit to a suboptimal path \( P \) since \( x_j = x_h - 1 < x_d \) \( y_j = y_h = y_d \). We can now prove that routing decision based on \( u^2 \) will yield \( n_k \) and hence circumvent \( n_h \). (Other cases can be proven equivalently.) Clearly, \( C_j = C_k = C_i \) (where \( C_i \) denotes the uniform cost in the network \( \forall i \neq h \)). Since all nodes east of \( n_h \) have cost \( C_i \), equation 5.16 yields \( V_h^E = C_i \). Hence, the east view computed in \( n_j \) is \( V_j^E = (C_h + C_i)/2 \). Correspondingly, the south view computed in \( n_k \) is \( V_k^S = (C_h + C_i)/2 \). As \( n_h \) does not impact the south and east views of \( n_j \) and \( n_k \), respectively, we have \( V_j^S = V_k^E = C_i \). Note that \( n_j \) and \( n_k \) have the same distance from \( n_d \), i.e., \( R_j^d = R_k^d \). Therefore the order of preference between \( n_j \) and \( n_k \) in \( n_i \) is determined by \( E_j^d(.) \) and \( E_k^d(.) \). Since \( E_j^d(.) \) represents the expected cost, preference is given to \( n_k \) if \( (E_j^d(.) - E_k^d(.) > 0 \). \( E_j^d(.) \) and \( E_k^d(.) \) are given by:

\[
E_j^d(.) = \frac{D_j^x C_i + D_j^y (C_i + C_h)}{D_j^x + D_j^y} \\
E_k^d(.) = \frac{D_k^x C_i + D_k^y (C_i + C_h)}{D_k^x + D_k^y} \\
E_j^d(.) - E_k^d(.) \text{ is then given by:}
\]

\[
\frac{D_j^x C_i + D_j^y (C_i + C_h)}{D_j^x + D_j^y} - \frac{(D_k^x + 1)(C_i + C_h) + (D_k^y - 1)C_i}{D_j^x + D_j^y}
\]

Since \( D_j^x + D_j^y = (D_k^x + D_k^y) \) it suffices to consider the difference

\[
D_j^x (\frac{C_i + C_h}{2}) + D_j^y C_i - (D_k^x + 1)(\frac{C_i + C_h}{2}) + (D_k^y - 1)C_i
\]
which simplifies to
\[ r = \left( \frac{C_h - C_i}{2} \right)(D_k^x - D_k^y + 1) \]
Now, \( r > 0 \Rightarrow E_{ij}(. > E_{ik}(.) \) and \( n_k \) should be preferred over \( n_j \). This is the case when \( (D_k^x - D_k^y + 1) > 0 \).

To show that in the case 3 scenario, which was introduced above, \( n_h \) is circumvented by routing \( M \) through \( n_k \), \( (D_k^x - D_k^y + 1) > 0 \) needs to be evaluated. Since \( x_k < x_d, D_k^x \geq 1. \) As \( y_h = y_d \) and \( y_k = y_h - 1 \), we must have \( D_k^y = 1 \). Therefore, \( (D_k^x - D_k^y + 1) > 0 \) and a routing decision based on \( U^2 \) will yield \( n_k \) as the preferred node to propagate \( M \) towards \( n_d \).

For case 2 scenarios, \( U^2 \) uses \( U^1 \) which, by corollary 2, will yield an optimal path \( \Pi \). This proves theorem 2.\( \square \)

Summary

In this chapter, we have developed the notion of a utility function that can facilitate the routing algorithm in network nodes. It has been formally shown that a utility function which utilizes information about cost along routes will be able to route a message along a minimum cost path in a uniform cost network with a single hotspot. Furthermore, we have given an example of such a utility function.

Different utility functions may have to be developed to facilitate routing in non-uniform cost networks. The implications of different network topologies and their impact on the design of decision functions will have to be investigated.
CHAPTER 6. CONCLUSION AND FUTURE WORK

Summary

Quo Vadis attempts to reduce the resource requirement for storage, acquisition, and use of network state information while achieving the desired performance (as defined by the criteria such as average message delay).

The size of the knowledge base $S_i(t)$ at node $n_i$ depends solely on the number of neighbors in its neighborhood $H_i$ and is independent of the size of the network. Thus if $M$ is the total number of nodes in the network and $h$ the average connectivity (i.e., the average cardinality of $H_i$), then the storage required at each node in Quo Vadis is $O(h)$. This constitutes a significant reduction in storage and processing overhead (especially in very large networks where $M \gg h$) over conventional routing mechanisms (e.g., those that use global routing tables) which require $O(M)$ storage at each node.

Since Quo Vadis propagates only local measurements $\mu_j(t)$ and the view vector $V_j(t)$ between neighboring nodes $n_j$ and $n_i$, the bandwidth requirement is small compared to conventional routing mechanisms. As explained in previous sections, Quo Vadis does not attempt to construct a precise picture of the network state as imprecision increases with distance and uncertainty of routing decisions is inevitable. Instead, it uses a coordinate system that provides for directional orientation together
with a summary of network state information. This allows Quo Vadis to avoid the costly validity check of information as required by routing methods that use the link state protocol.

The experimental results presented in this work clearly demonstrate that Quo Vadis is largely successful in meeting its primary design objectives, at least when it is used within the relatively simple regular grid network. Particularly noteworthy is the ability of Quo Vadis to pro-actively as well as reactively avoid congestion in the network while simultaneously minimizing message delay.

A theoretical framework has been developed which facilitates the design and analysis of decision functions for Quo Vadis. This framework is based on concepts and ideas from the field of utility theory. We have shown the existence of utility functions which guarantee minimum cost routes in uniform cost grid networks with single hotspot. Clearly, this framework will have to be extended for communication networks with complex cost dynamics and different topologies.

**Future Work**

The results obtained to date from experimental studies of Quo Vadis underline the advantage of viewing routing as a distributed, heuristic multi-criterion optimization task with adaptive properties so as to respond quickly to various forms of network dynamics. However, there is need to further understand, analyze, refine, and adapt this approach to a much broader class of network topologies and network dynamics that are representative of real-world communication environments.
Evaluation of Quo Vadis in Different Network Topologies

As pointed out earlier, the results obtained with Quo Vadis on regular grid networks are promising. While the experience gained from such initial experiments has been extremely valuable in understanding Quo Vadis, it is clear that regular grids do not sufficiently challenge the parameterized knowledge representation and routing mechanisms. Furthermore, routing in such regular networks could be accomplished without the use of explicit network state information [47]. It is imperative to gain a deeper understanding of the behavior of Quo Vadis in a broad range of network topologies with different types and different degrees of regularity. Carefully crafted experiments designed to evaluate the performance of Quo Vadis with different topologies must be conducted. The results of such experiments would help shed light on the strengths and limitations of different forms of knowledge representation and routing decision functions as they relate to the impact of topological properties of the network on the overall performance (as measured by standard performance metrics such as average delay, throughput, etc.)

Handling Node and Link Failures and Other Pathologies Gracefully

Extensive research by other researchers on both link state and distance vector routing algorithms have identified many issues that need to be considered in the design of new routing mechanisms. Examples of such design issues are bandwidth and storage overhead, performance in the presence of failure, message looping and bouncing [26, 15, 47]. The current design of Quo Vadis aims at reducing resource overhead. Issues such as message looping, message bouncing, as well as mechanisms to deal with node and link failures are currently under study.
Unexpected node and link failures during network operation can lead to bouncing and looping of messages [15, 26]. If this is not carefully controlled, it can have a severe adverse impact on the overall network performance (e.g., reduced network utilization, increased average message delay), especially in connectionless network environments. Graceful handling of network node and link failures is especially important in traffic management frameworks like Quo Vadis that do not maintain explicit global connectivity information of the sort available in routing tables. This makes it necessary for Quo Vadis to contain mechanisms that enable it to detect and handle the possibility of messages being routed into a dead end. (In fact, such a situation can occur even in the absence of network failures, simply as a consequence of certain topological properties of the network). The design of mechanisms that would extend Quo Vadis as necessary to equip it to deal with such pathological conditions would clearly enhance the overall performance of Quo Vadis. The resulting system would be carefully evaluated through a series of experiments. In addition, theoretical bounds e.g., on the expected time for detection of and escape from a loop, would be established.

Evaluation of Quo Vadis in a Dynamic Network Environment

The preliminary experiments on Quo Vadis described in Chapter 4 were conducted in an environment with static (average) load conditions at each network node. Although we anticipate similar results in an environment that exhibits much more complex dynamics, additional systematic experiments are needed to verify this hypothesis. Hence, future research must evaluate the performance of Quo Vadis under a wide range of spatio-temporal network dynamics. The effects of complex network dynamics can be simulated in a number of ways e.g., varying time interval between
Parametric Studies of Knowledge Representation and Decision Functions

An important objective is to identify suitable parameterized heuristics for knowledge representation and decision functions that are particularly well-matched to specific overall performance measures for different classes of network topologies and dynamics. This entails systematic studies of the performance of knowledge representation and decision functions in Quo Vadis as a function of the tunable parameters \((\alpha, \beta, \gamma, \text{and } \tau)\). While for a static network environment an optimum set of parameters may be found, it is necessary to investigate methods in adjusting these parameters for a dynamic environment.

In addition, precise quantification of the relationship between the resource requirements of Quo Vadis (as a function of parameters such as frequency of update of views etc.) and the relevant measures of network performance are clearly necessary.

Making Quo Vadis Adaptive Using Machine Learning

As indicated by the experiments described earlier, the optimal setting of various parameters used by Quo Vadis is a function of network dynamics as well as the desired performance objectives. For instance, under strongly fluctuating conditions it might be reasonable to emphasize local information such as the load at a neighboring node over global information about network state summarized by the views. It is also reasonable to expect that the optimal values for various parameters would depend on network topology, frequency and patterns of node and link failures. etc. This underlines the importance of investigating a range of mechanisms that adapt
the tunable parameters used to maintain the desired overall performance. It is anticipated that the results of parametric studies of knowledge representation and decision functions outlined above would provide the empirical knowledge necessary for pursuing an important long-term goal, namely, the design of completely autonomous self-managing, intelligent, low-overhead, robust and adaptive traffic management mechanisms for very large high speed communication networks of the future. Admittedly, this presents a number of major challenges to the state-of-art machine learning and adaptive control techniques. In view of this, mechanisms that dynamically adapt the tunable parameters \((\alpha, \beta, \gamma, \tau)\) used by Quo Vadis at each node in response to changes in network dynamics are of interest and must be explored. In particular, variations of techniques drawn from adaptive control and machine learning [14], especially reinforcement learning will be investigated. Suitable error measures and feedback mechanisms that would enable the nodes in Quo Vadis to tune their parameters in such a way as to maintain the desired network performance under changing conditions will be defined. (See [22, 24] for preliminary work on using machine learning for adaptive communication network management by other investigators). This would lay the groundwork for further research on adaptive autonomous self-managing communication networks.

Refining the Theoretical Framework

We have introduced the concept of utility in the context of routing messages in large communication networks. The existence of utility functions that yield minimum cost paths has been rigorously proven for a uniform cost grid network with a single hotspot. Nevertheless, the implications of different network topologies with complex
cost distributions over the network and their impact on the design of decision function must be investigated. The possibilities to trade off optimality for simplicity will have to be studied for Quo Vadis to perform in a variety of communication environments.
BIBLIOGRAPHY


