On a Self-Organizing MANET Event Routing Architecture with Causal Dependency Awareness

Guanhong Pei

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

MASTER of SCIENCE

in

Electrical Engineering

Binoy Ravindran, Chair

Paul Plassmann

Yaling Yang

November 23, 2009

Blacksburg, Virginia

Keywords: Self-organizing, Self-reconfigurable, Event-based Systems, Ad Hoc Networks, MANET, Wireless Networks, Causal Dependency, Publish/Subscribe

On a Self-Organizing MANET Event Routing Architecture with Causal Dependency Awareness

Guanhong Pei

(ABSTRACT)

Publish/subscribe (P/S) is a communication paradigm of growing popularity for information dissemination in large-scale distributed systems. The weak coupling between information producers and consumers in P/S systems is attractive for loosely coupled and dynamic network infrastructures such as ad hoc networks. However, achieving end-to-end timeliness and reliability properties when P/S events are causally dependent is an open problem in ad hoc networks.

In this thesis, we present, evaluate benefits of, and compare with past work, an architecture design that can effectively support timely and reliable delivery of events and causally related events in ad hoc environments, and especially in mobile ad hoc networks (MANETs).

With observations from both realistic application model and simulation experiments, we reveal causal dependencies among events and their significance in a typical use notional system. We also examine and propose engineering methodologies to further tailor an event-based system to facilitate its self-reorganizing capability and self-reconfiguration. Our design features a two-layer structure, including novel distributed algorithms and mechanisms for P/S tree construction and maintenance. The trace-based experimental simulation studies illustrate our design's effectiveness in both cases with and without causal dependencies.

Acknowledgments

I owe much to my advisor, Dr. Binoy Ravindran, for guiding me through the academic and nonacademic endeavors. All his guidance, encouragement, patience and insights have been instrumental in shaping me during my pursue of the degree.

Dr. E. Douglas Jensen has been actively involved with my work. His inputs at various stages of my work have played a vital role.

I would like to thank Dr. Yaling Yang and Dr. Thomas Y. Hou for giving me valuable advice. I would also like to thank Dr. Yaling Yang again and Dr. Paul Plassmann for taking the time to serve on my committee and review my work.

Besides, many thanks go to everyone at our Real-Time Systems Lab, for their help in the process of my work.

Last but not least, I would like to express my gratitude to my family, and my friends, for their constant love, unshakable support and encouragement.

Contents

1 Introduction **1**

1.1 P/S and Ad Hoc Networks 2

1.2 Event Causal Dependencies 6

2 Related Work **9**

2.1 Tree-based vs. Non-tree Approaches 10

2.2 Existing MANET Tree-based P/S Architecture 11

3 Hierarchical Tree-based Model **14**

3.1 Graph Theory Preliminaries 15

3.2 Path Overhead Ratio 18

4 Basic Architecture Design **21**

4.1 Overview 21

4.2 Inner-tree Level Interconnection 22

4.2.1 Inner-tree Subscription and Publication 22

4.2.2 Parent Evaluation Metrics (PEM) 23

4.3 Inter-tree Level Interconnection 24

4.3.1 Root Node Selection and Tree Merging 24

4.3.2 Inter-tree communication 27

5 Self-reconfiguring Architectural Facilitation for Event Causal Dependencies 30

5.1 Causal Graph Construction 30

5.2 Self-reconfiguration with Awareness of Event Causal Dependencies 32

6 Experimental Evaluation 36

6.1 Simulation Environment Setup 36

6.2 Architecture Evaluation and Analysis 37

6.2.1 Influence of D_T 37

6.2.2 Tree Merging and New Root Selection 39

6.2.3 Event Delivery with Awareness of Causal Dependencies 40

6.2.4 Timely and Reliable Event Delivery 43

7 Conclusion and Discussion 48

Bibliography 51

A Confidence Interval Tables 59

List of Figures

1.1 Example Scenario 3

1.2 Example Causal Graph 7

3.1 Pentagon, Octagon — Invalid Model Example 16

3.2 Network Model Structures 17

3.3 Advantage over Single-tree Model 19

4.1 Architecture Overview 21

4.2 Tree Merging Algorithm 26

4.3 New Root Node Selection Algorithm 27

5.1 Event Causal Graph 31

6.1 Timeliness Performance under Different D_T w/o Tree Merging/New Root Selection 38

6.2 Effectiveness of Tree Merging and New Root Selection 39

6.3 Improvement from Causal Event Delivery Architectural Support 41

6.4 Timeliness Gain 42

6.5 Comparison of Timely Delivery 44

6.6 Comparison of Delivery Ratio under Node Failure 45

6.7 Network Traffic Load with Low P/S Load 46

6.8 Network Traffic Load with High P/S Load 47

List of Tables

6.1 Simulation Settings 37

A.1 Confidence interval for Figure 6.1. 59

A.2 Confidence interval for Figure 6.2. 60

A.3 Confidence interval for Figure 6.3(a). 60

A.4 Confidence interval for Figure 6.3(b). 60

A.5 Confidence interval for Figure 6.5. 61

A.6 Confidence interval for Figure 6.6. 61

Chapter 1

Introduction

The publish/subscribe paradigm [26] communicates on the basis of either the message content or the message source being of interest to destinations – as opposed to the source specifying the recipient(s). Publish/subscribe communication has been used in computing systems (notably for defense and industrial automation) since at least the early 1970's (e.g., [15]). Only since the middle of this decade has it gained rapidly increasing interest – for example, the Object Management Group has developed a publish/subscribe standard named the Data Distribution Service [38].

Publish/Subscribe (P/S) systems can be considered to be a form of event-based systems, in the sense that the information injected to and propagated through the system can be treated as *events*. An entity in the system can act either or both as information producers (*publishers*) and consumers (*subscribers*). The subscribers declare their interests by *subscriptions* to certain events, most commonly specified by the content or the topic of the events (with different expressive power), and publishers produce events of information to the system. The *event routing* mechanism implemented in the P/S system (usually middleware) then takes charge of the event delivery according to the subscriptions. In short, a P/S system

is an information dispatching system which does not explicitly involve the use of network addresses.

P/S systems are usually thought of in the context of disseminating data, but they can also be used for control information. Indeed, the data/control dichotomy in general is often an over-simplification, since the same information can be data at one time and place, and control at another time and place. This fact plays a role in our motivation for the work described in this thesis.

1.1 P/S and Ad Hoc Networks

Many problems pertaining to P/S in a wired network system have been proposed and solved. With increasing popularity and availability of wireless devices, extending the applications of P/S into wireless environments becomes a pressing need. The advantages of this extension are due to the natures of both P/S systems and wireless ad hoc networks.

In essence, a wireless ad hoc network is formed by wireless devices communicating without a fixed network infrastructure, either because it is difficult to build a wired infrastructure due to environmental, economical, or resource constraints, or because the existing wired infrastructure is not functioning due to damage from war or natural disaster. Devices discover and "talk" to others within range to form a network. Connections may be multi-hop. New properties such as broadcasting and signal conflicts arise from the use of wireless MACs.

The weak coupling between information producers' and consumers' identities in a P/S system provides P/S the advantages of asynchronous, multipoint communication, hence freeing the communicating entities from temporal and spacial requirements for connection. Therefore P/S is appealing in loosely coupled network contexts such as ad hoc networks, and especially

mobile ad hoc networks (MANETs) [16] because of the ease with which components can be added, removed or changed at runtime.

A MANET, as a subclass of ad hoc networks, is a collection of (usually) mobile wireless devices with dynamically changing membership and multi-hop topologies composed of wireless links. By its nature, a MANET is a self-organizing adaptive network, and thus needs to be formed and maintained in a distributed manner without centralized support or fixed infrastructures [47].

The P/S scheme is adopted in various types of ad hoc networks with different configurations. In a sensor network context, for instance, sensor nodes may host temperature sensors in a plant and subscribe to all the application events querying about temperature data of that location; an actuator or monitor may only be interested in receiving all the messages concerning a temperature greater than 1,000 degrees centigrade, to activate an alarm.

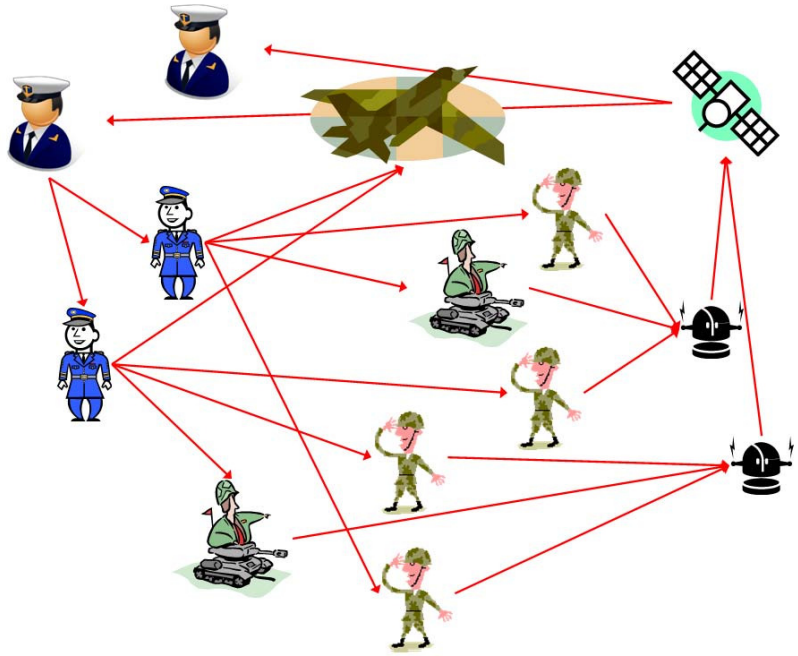


Figure 1.1: Example Scenario

An interesting notional scenario for the use of P/S in a MANET is illustrated by Figure 1.1.

It begins when a commander either makes (i.e., invokes) or publishes a service request to "agents" to find out something about the situation in a particular combat region. Note that although making a request for information by publishing it is a departure from the conventional P/S mindset of publishing only "data", it can be a natural and effective technique. Typically there is a mission-critical time constraint for him to obtain that information — the information's utility [29] to the mission degrades after a certain amount of time following his service request.

To obtain the information: two forms of imagery are obtained by subscription from surveillance platforms, and fused by recipient agents; re-scoped down to the geographical region of interest (perhaps by publishing it to a service which does that); re-sized to fit on soldiers' PDA screen (perhaps by publishing the re-scoped information to a service which re-sizes it); published to, and annotated by, one or more soldiers in that region; then sent by those soldiers to a ground-to-space relay and from there to a satellite and then to the commander making the original service request (and probably other interested commanders). Most, if not all, of these communications can effectively be in the form of publishes and subscribes.

More application examples, in the settings of sensor networks, unmanned aerial vehicles, etc., can be found in [7, 21, 28, 33, 43].

Currently, the P/S mindset, and hence P/S concepts and techniques, are focused on minimizing latency (sometimes called delay) and thus maximizing throughput as the primary performance metric. This is in contrast to the real-time mindset, and hence concepts and techniques, that are focused on satisfying time constraints, conventionally by minimizing response times (or, increasingly, focused on maximizing accrued utility [14]). Although throughput and response time are conjugate metrics, latency plays a fundamental role in both. To compare our P/S research results with those of others, we necessarily employ that communities' performance metric. Time constraints such as those in our motivating notional

scenario, and latencies, can be mapped between each other.

The potential advantages of the P/S interaction model atop ad hoc networks are not fully realized by the state of the art predominant industrial and academic solutions, such as TIB/RV [37], SCRIBE [12], SIENA [10], REDS [22], Kyra [5], IBM's Gryphon [2], IBM's WS-Notification [27], and RTI's DDS [11] based on OMG's DDS [38].

In the context of P/S in ad hoc networks, many of the previous efforts have focused primarily on throughput and overhead. Timeliness and reliability, which are important for many ad hoc-based applications, have not been well addressed. Furthermore, as one of the key issues in ad hoc networks, full mobility support also needs to be accommodated. Some previous efforts have been made [7, 25, 36] assuming either that only a subset of nodes in the network can roam and act as clients or that the clients are always one hop away from the fixed infrastructure. They therefore focus only on a restricted subset of the problem space. A more general and common scenario in mobile ad hoc networks is that every node in the network is potentially mobile, can join, leave the system, and have access to the publish/subscribe service (e.g., by running P/S middleware), while also acting as a broker for message forwarding and matching to make the service available.

In addition, MANETs are subject to significantly greater run-time uncertainties and resource constraints than are traditional fixed-infrastructure networks, including:

(1) frequent link breakages and temporary network disconnections;

(2) temporary node unavailability and node joins or departures at unpredictable times; and

(3) mobility-induced resource constraints on the overall architecture, such as limits on bandwidth, latency, and energy consumption.

1.2 Event Causal Dependencies

To garner the most timeliness and reliability from the P/S system in real-world ad hoc applications, we explore architectural real-time support for *event causal dependencies*, an important property of many emerging ad hoc-based applications that are suited for P/S-style communication. This refers to the existence of multiple publication and subscription hops that are causally related (e.g., topic-wise), resulting in an event causal chain or event causal graph. We illustrate this above with a motivating scenario from the military domain (similar scenarios can be found in other application domains such as automotive [34] and e-health [32] systems, etc.). To better understand causal dependency, we should also note that logical implication is not equivalent to event causal dependency in a P/S system because of the lack of temporal relationships [41].

The conventional ideas about timely and reliable P/S event delivery are always on a *single publish-hop* basis, by which we mean the travel of an event from its publisher to its subscriber(s), perhaps transparently through network devices such as routers. Thus that neglects a common phenomenon of *event causal dependencies*, which refers to the existence of *multi-publish-hop* — i.e., an event causal chain or even an event causal graph constructed based on the event causal dependencies. To better understand these concepts, let us consider again the example scenario in a P/S system in MANETs.

Figure 1.2 shows the underlying causal relationship graph for the Figure 1.1 scenario. The timeliness and reliability of the event delivery from the causal event initiator (commander in Figure 1.1 above) to the last publisher in the whole chain (satellite in Figure 1.1), is mission critical and is challenging due to the MANET dynamics.

This scenario raises fundamental yet unsolved problems:

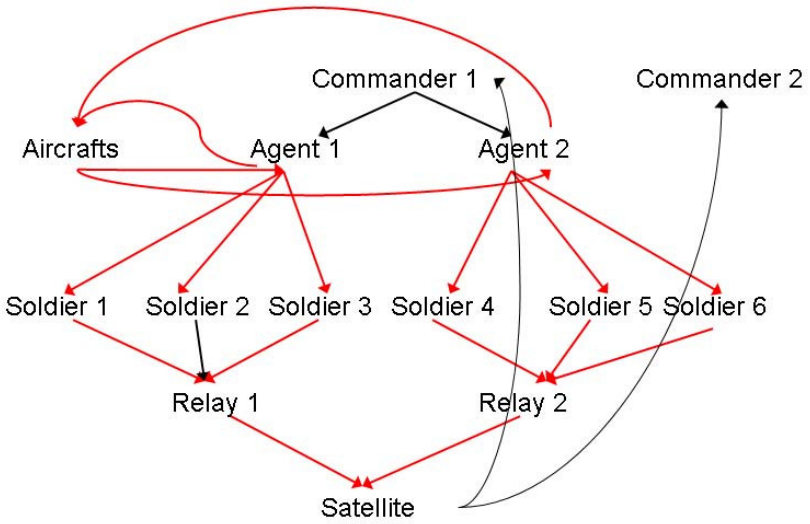


Figure 1.2: Example Causal Graph

(1) What interconnection architecture is appropriate for P/S service in MANETs that can handle causal dependencies?

(2) How to self-organize mobile nodes into an interconnection topology with support for reliable and timely delivery of causally-dependent messages?

(3) How to reliably and timely route causally-dependent messages?

In this thesis, we address these questions. Without loss of the potential to be adopted in generic ad hoc networks, we consider a MANET, where every node in the network has mobility and can access a P/S service. Based on analysis, we present an event routing architecture design called SOMER (Self-Organizing MANET Event Routing) that supports timely and reliable data delivery across causally-dependent event chains. Key aspects of SOMER include algorithms and mechanisms for P/S tree construction and maintenance with self-organizing and self-configurable capabilities. Our simulation-based experimental results show that SOMER achieves 30%–100% timeliness improvement and up to 30% reliability improvement over previous solutions.

To the best of our knowledge, this is the first work to consider causally-dependent networked P/S systems. Thus, our contribution is the SOMER architecture.

The remainder of this thesis is organized as follows. Chapter 2 discusses related work and identifies SOMER's unique aspects. Then we illustrate the advantages of a hierarchical tree-based interconnection through an analytical model in Chapter 3. Chapter 4 presents SOMER's basic design. In Chapter 5, we discuss how we design architectural support for causality awareness. Chapter 6 reports our experimental results, including comparison with past work. Chapter 7 concludes the thesis with discussion and future work.

Chapter 2

Related Work

Typically, in MANETs, a structured interconnection topology is used; most of the recent representative work [6, 28, 35, 39, 42] presents algorithms for constructing and maintaining an event routing tree as the interconnection topology under a filter-based routing scheme. There are also structure-less approaches—e.g., Baldoni et al [1] employ a form of informed flooding-based event routing using Euclidean distances; the adoption of rendezvous-based protocols first described in SCRIBE [12], has been studied in a MANET setting by Carvalho et al [8], and also in an Internet setting by Bharambe et al [3]; Costa et al [18] propose a gossip-based protocol used in VANETs (Vehicular Ad Hoc Networks).

It is also interesting to note various approaches used in P/S systems atop wired networks. For example, a distributed hash table configuration is used to facilitate the interconnection topology [17, 44, 48, 50]. Some approaches do not maintain any deterministic data structure on the topology at a peer. In this case, event routing is neither filtering-based nor rendezvous based, like in [20, 23]; the authors employ probabilistic flooding and gossiping, respectively. Chakraborty and Finin [13] propose a service/control information dissemination architecture/protocol in a MANET pervasive computing environment, utilizing bounded

advertisement, peer-to-peer dynamic caching and logical group-based selective forwarding of discovery requests.

2.1 Tree-based vs. Non-tree Approaches

We refer to a *tree-based interconnection topology* as an approach to interconnect mobile nodes with an acyclic structure. We refer to *non-tree approaches* as the ones that use either rendezvous-based or flooding-/gossip-based approaches. The fundamental difference between the tree-based topology and the non-tree approaches is that the interconnection topology of non-tree approaches tends to weakly match the underlying physical topology, because of the loose geographical coupling among the nodes. The impact on the message delivery is that latency can be very high and sometimes unpredictable, as information might pass across many nodes, some of which might be slow or have long physical paths between them in the underlying network. Non-tree approaches may find their suitable applications in scenarios such as delay tolerant networks [19], and thus deviate from part of our goal as to accommodate timing properties of the system. Also, in the case of large periods of message multicast or flooding in peer-to-peer networks it may suffer congestions and inefficient use of bandwidth with non-tree-based connections.

By contrast, a tree-based topology can more efficiently and effectively meet the end-to-end timeliness requirement [9, 30]. However, in order to meet the demands of large scale MANETs, we need a more effective architecture to handle node failures, link breakage, node joins and departures, while still maintaining the same or increased timely message delivery.

2.2 Existing MANET Tree-based P/S Architecture

To the best of our knowledge, for MANETs, most tree-based P/S interconnection approaches either employ a single P/S tree structure in the whole network, or build one P/S tree for each publisher. The latter model can accommodate only a limited number of designated publishers, and thus suffers from a lack of scalability.

The single-tree strategy is typically used; and due to the resemblance between multicast and P/S in MANETs, in [35] the P/S tree is derived from multicast tree construction and maintenance mechanisms of MAODV (Multicast Ad hoc On-Demand Distance Vector Routing Protocol) [45, 46].

In large scale MANETs, as the average distance increases between the root and a node in the P/S tree, it becomes much harder to handle failures and topology changes. Moreover, although the single-tree architecture is fully based on the geographical topology, there still exists a possibility that a publisher's message may not reach some of its subscribers via the physically shortest route. That is because the length of the path is hard to upper bound, and is independent of the length of the physically shortest route.

there still exists a possibility that a publisher's message may not take the physically shortest route to reach some of its subscribers, and sometimes the path stretch can get as high as $O(n)$ (in which n is the number of nodes in the network). Generally, low path stretch is hard to achieve in a spanning tree unless with careful design which may be running-time-consuming in decentralized settings. For this issue, interested readers are referred to [24, 40], etc., on construction of low stretch spanning tree.

A hierarchical tree-based interconnection in a MANET self-organizes all the nodes into non-overlapping trees and has specific mechanisms for inter-tree communication. The roots of the trees function as the inter-tree brokers. In Section 3, we will show the advantages of

a hierarchical tree-based interconnection topology over a single P/S tree approach through both qualitative and quantitative analysis.

Motivated by all the above observations, we consider a tree-based approach and explore a hierarchical architecture design for causally related event delivery. The idea of using hierarchical architecture in ad hoc networks is not new. For example, [43] presents a scalable P/S system called SensTrac based on a tree-based hierarchical structure.

SOMER is uniquely novel and different from [43] in several aspects including:

(1) *Tree Construction.* SensTrac uses static clustering via nodes' geographic coordinates using GPS. In contrast, SOMER uses dynamic clustering, allowing nodes to be clustered on-the-fly;

(2) *Inter-tree Communication.* SensTrac uses AODV to find routes among root nodes (leaders), and uses gossip to disseminate information among root nodes, whereas SOMER uses restricted flooding among root nodes, which is more suitable for a mobile environment;

(3) In SensTrac, the query (subscriber) node subscribes to the information of its area of interest (AOI) which is bounded by a given square, whereas our subscription model is more general;

(4) In SensTrac, the query node is the only subscriber, and does not act as a broker, and the publishers are only the sensors in the AOI, whereas we do not have those limitations on subscribers or publishers;

(5) In SensTrac, nodes are stationary and only the query node can move, whereas we allow every unit to be mobile;

(6) SensTrac does not support system reconfiguration for causal dependencies (due to its static node clustering mechanism which has no capability to handle causal dependencies

similar to that of SOMER).

In brief, SOMER is a more general architecture design comparing to SensTrac, as opposed to an application specific design. From now on, we mainly consider tree-based approaches for event delivery on top of MANETs, and we show the structure and the features of SOMER in the following chapters.

Chapter 3

Hierarchical Tree-based Model

Suppose that the system (i.e., a MANET) has already completed self-organization and remains stationary. To build an analytical model from which we get our first motivations, we make the following assumptions:

- All nodes are homogeneous and evenly distributed within a 2-dimensional area; and
- Each node has the same number of immediate neighbors and thus each tree has the same number of immediate neighboring trees. Let the number of neighbors of any node be $N_{Neighbor}$.
- We do not consider the physical topology where nodes are placed to form a string or a circle.

Note that the underlying hierarchical tree-based network model in this chapter is to provide an abstract analysis and rough idea of the architectural motivation of the work. The assumptions made here are not related to the structure and features of SOMER, or for the simulation-based experimental studies in the later chapters.

3.1 Graph Theory Preliminaries

An r-tree is a tree with *root* r. Let $T(r)$ denote such a tree. The *level* of a vertex v in $T(r)$ is the length of the path rTv. Each edge of $T(r)$ joins vertices on consecutive levels, and it is convenient to think of these edges as being oriented from the lower to the higher level, so as to form a *branching*. Each vertex except v on the path rTv, is called an *ancestor* of v, and each vertex of which v is an *ancestor* is a *descendant* of v. Two vertices are *related* in T if one is an ancestor of the other. The immediate ancestor of v is its *predecessor* or *parent*, denoted $p(v)$, and the vertices whose *predecessor* is v are its *successors* or *children*. A *leaf* of a tree is the node which has no successors. We refer to the process of going from a non-root node to the root by way of its parent as "going upward," and going the reverse way as "going downward."

Suppose $N_{Neighbor}$ is subject to a normal distribution with mean of 6 and standard deviation of 4. Obviously, only 4 and 6 are the possible number of neighbors that can produce an evenly distributed physical topology given that we rule out the case where $N_{Neighbor} = 2$ as the third assumption states above; For example, if $N_{Neighbor} = 5$, we would not be able to find a topology to maintain the equality of the distances between any two of neighboring nodes, thus violating the 2-dimensional even distribution assumption.

We provide a sketchy proof for finding valid $N_{Neighbor}$'s here. For $N_{Neighbor} = 3$, it turns out this case is invalid because eventually we would find there are in fact 6 immediate neighbors for each node, which results in contradiction. For $N_{Neighbor} = 5$, as Figure 3.1 shows, suppose that node A has exactly 5 immediate neighbors placed on the corners of a pentagon. Then for node B, we should find at node D, where $|AB| = |BD|, \angle BAC = \angle ABD$, an immediate neighbor of node B due to the isotropy of our model. However, $|CD| < |AB| = |BD|$ makes node D an invalid node in the graph because there should not be any edge that is shorter

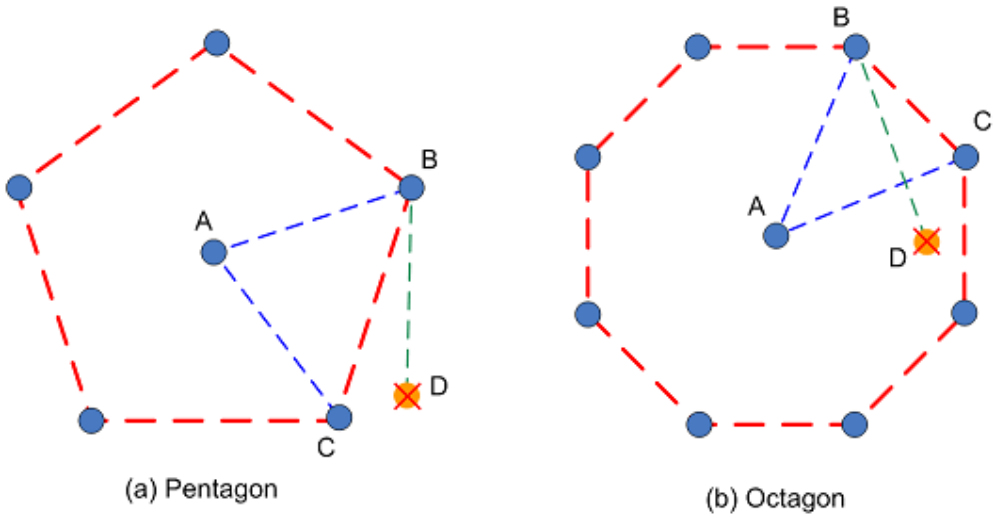


Figure 3.1: Pentagon, Octagon — Invalid Model Example

than the distance between two immediate neighbors. Similar statement applies to the cases where $N_{Neighbor} > 6$ as shown in Figure 3.1. Therefore, we have

$$N_{Neighbor} = 4, 6 \tag{3.1}$$

The network can be represented by a connectivity graph $N(V, E, P)$, where $V = \{v_1, ..., v_n\}$ is the set of nodes, E is the set of links, and $P = \{p_1, ..., p_k\}$ is the set of publishers. Also, let n denote the number of nodes and k denote the number of publishers. Let D_T denote the density of root nodes and $T = \{t_1, ...t_t\}$ denote the set of trees. Let the longest distance from the tree root to a node in its territory be R (i.e., the radius). Let the distance between any pair of neighboring nodes be denoted as $dist_{Min}$. We assume that $0.5dist_{CST} < dist_{Min} < dist_{CST}$, where $dist_{CST}$ is the one-hop wireless transmission range (carrier sensing range), such that a node should only receive the signal from an immediate neighbor.

From Equation 3.1, we can derive that the network connection links form either a grid-like ($N_{Neighbor} = 4$) or a beehive-like ($N_{Neighbor} = 6$) structure. Figure 3.2 illustrates this, where

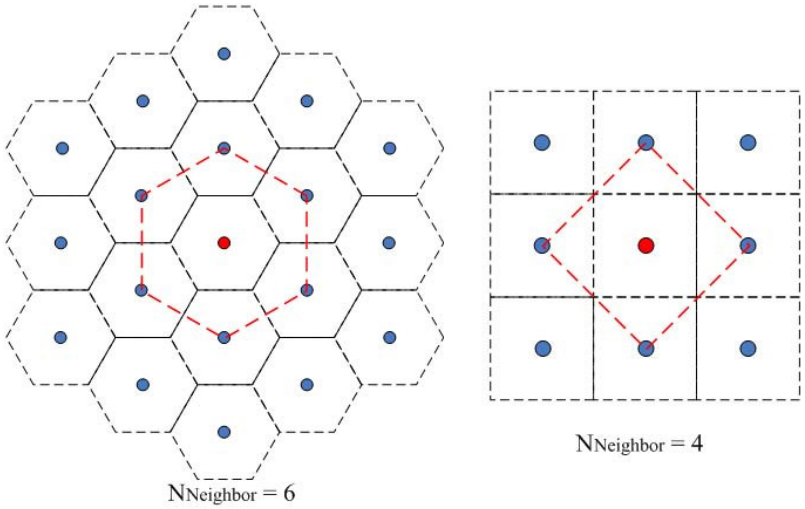


Figure 3.2: Network Model Structures

the solid dots denote nodes in the network, and the red hexagons and squares with nodes at the angles cover the set of immediate neighbors of the center node.

Assume that a straight path can be established between any pair of neighboring trees' roots. It can be seen that for any root node to reach another root node, if their trees are not neighbors, it needs a route by way of other root nodes, and this route takes at most one turn along the route. Let β denote the angle of the turn if there is one. Now,

$$\beta = \begin{cases} 120^o, \text{ for } N_{Neighbor} = 6 \\ 145^o, \text{ for } N_{Neighbor} = 4 \end{cases} \tag{3.2}$$

Let $s(p_i, v_j)$ be one of $p_i \in P$'s subscriber which is associated to the tree rooted at $v_j \in V$. We estimate the average distance from a node to its tree root to be $\frac{R}{2}$. Let $length(u, v)$ denote the length of the path between two nodes u and v. Now, the length of the path

between p_i and $s(p_i, v_j)$ is:

$$\begin{aligned} &length(p_i, s(p_i, v_j)) = length(p_i, r(p_i)) + \\ &\qquad length(r(p_i), v_j) + length(v_j, s(p_i, v_j)) \\ &= \tfrac{R}{2} + length(r(p_i), v_j) + \tfrac{R}{2} \\ &= R + length(r(p_i), v_j) \end{aligned} \tag{3.3}$$

Let $dist(u, v)$ denote the straight-line distance between two nodes u and v. We can bound the length of the path between p_i and $s(p_i, v_j)$ as:

$$\begin{aligned} &length(r(p_i), v_j) \leq \tfrac{dist(r(p_i), v_j)}{|\sin\beta|} \Rightarrow length(p_i, s(p_i, v_j)) \\ &\leq \begin{cases} R + \frac{2}{\sqrt{3}} dist(r(p_i), v_j), \text{ for } N_{Neighbor} = 6 \\ R + \frac{2}{\sqrt{2}} dist(r(p_i), v_j), \text{ for } N_{Neighbor} = 4 \end{cases} \end{aligned} \tag{3.4}$$

For a large-scale MANET, $dist(p_i, s(p_i, v_j))$ is approximately $dist(r(p_i), v_j)$. Therefore,

$$\begin{aligned} &length(p_i, s(p_i, v_j)) \\ &\leq \begin{cases} R + \frac{2}{\sqrt{3}} dist(p_i, s(p_i, v_j)), \text{ for } N_{Neighbor} = 6 \\ R + \frac{2}{\sqrt{2}} dist(p_i, s(p_i, v_j)), \text{ for } N_{Neighbor} = 4 \end{cases} \\ &= R + \frac{2\sqrt{2}}{\sqrt{N_{Neighbor}}} dist(p_i, s(p_i, v_j)) \end{aligned} \tag{3.5}$$

3.2 Path Overhead Ratio

We define the *path overhead ratio* (POR) based on the variation of the concept of *path stretch* (i.e., the ratio of the length of the route between two nodes to the length of the shortest path (both in Euclidean distance)) as the extra number of hops required by the path over that of the straight line distance. The number of hops is proportional to the path length,

and hence:

$$\begin{aligned} POR_{SOMER} &\leq \frac{length(p_i,s(p_i,v_j))-dist(p_i,s(p_i,v_j))}{dist(p_i,s(p_i,v_j))} \\ &= \frac{2\sqrt{2}}{\sqrt{N_{Neighbor}}} + \frac{R}{dist(p_i,s(p_i,v_j))} - 1 \end{aligned} \tag{3.6}$$

By applying this POR in a large-scale network where $R \ll dist(p_i, s(p_i, v_j))$, we obtain:

$$POR_{SOMER} \leq \frac{2\sqrt{2}}{\sqrt{N_{Neighbor}}} - 1 \tag{3.7}$$

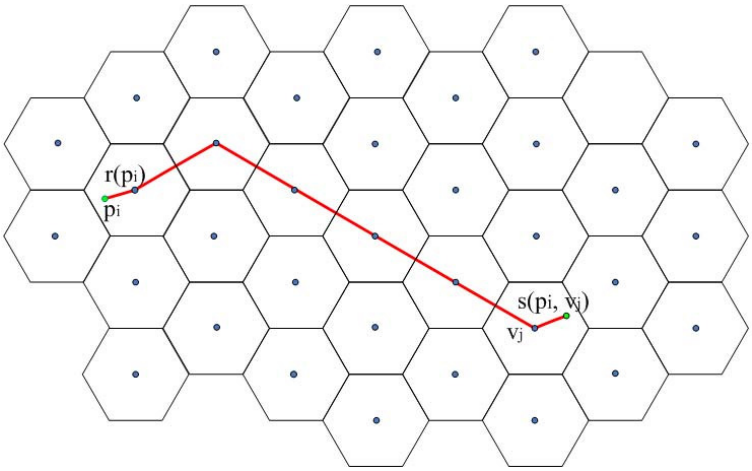


Figure 3.3: Advantage over Single-tree Model

Figure 3.3 gives an example showing the path from p_i to $s(p_i, v_j)$ with red lines. In the figure, the blue solid nodes are root nodes and the solid hexagons denote the abstract territories of the trees rooted at the center nodes.

Now we can observe that for the hierarchical tree-based architecture, POR can be bounded small-order parameters (and in large networks it can be as small as $\sqrt{2}-1$ or $\frac{2}{\sqrt{3}}-1$); whereas for single-tree-based and structure-less approaches, the length of the path could sometimes be up to the order of the total number of nodes. Thus, the benefits of the hierarchical tree-based architecture include the following:

(1) event delivery can be completed in a more timely manner;

(2) shorter and bounded path length implies lower probability for a delivery to fail, despite frequent link breakages and node failures; and

(3) the multi-tree structure gives enough leeway to devise *multi-path* event delivery schemes through the use of multiple neighboring trees and multiple border routers (leaf nodes) among the trees, and may also balance the network traffic and alleviate congestion.

Chapter 4

Basic Architecture Design

4.1 Overview

SOMER's design employs the hierarchical tree-based interconnection as shown in Figure 4.1.

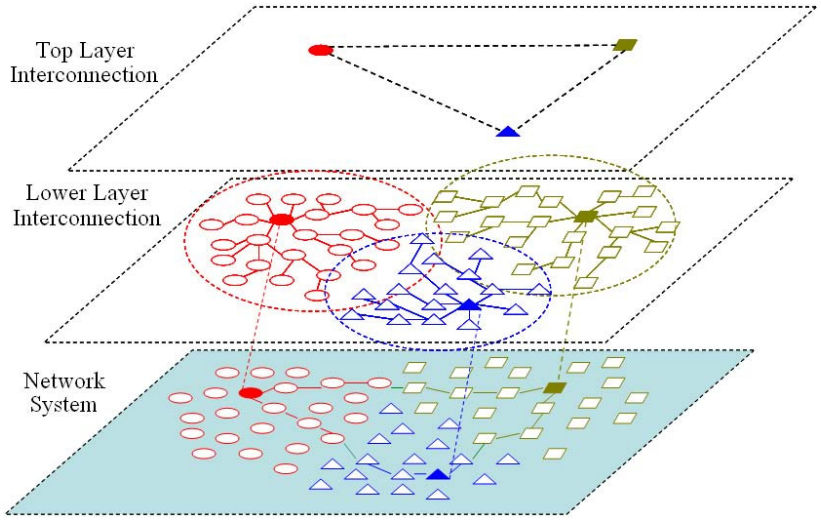


Figure 4.1: Architecture Overview

At the lower layer (*inner-tree level*), nodes are self-organized into multiple P/S trees rooted at several *root nodes* across the network. At the top layer (*inter-tree level*), root nodes act

as super publisher/subscriber nodes and constitute an overlay network.

4.2 Inner-tree Level Interconnection

The *publish/subscribe tree* (PST) is constructed distributively in a request/reply fashion. Any node v_0 that has not joined a PST broadcasts the *join request* message and waits for its immediate neighbors that are currently in a P/S tree to reply with a *join reply* message. After a timeout for the neighbors' reply expires, if the node gets any replies, it greedily selects the best candidate neighbor from the replies as its parent according to the *parent evaluation metric* (PEM); otherwise, it continues to broadcast new *join request* messages. Each node in the system can only join one PST.

4.2.1 Inner-tree Subscription and Publication

Each node v_0 has its own subscription interest set $s(v_0)$, termed *inherent subscription*. On joining the tree, node v_0 sends a *subscription* message containing $s(v_0)$ upward toward the *grafting node* for each of its ancestors to match, and subsequently publish information to v_0, which is now a leaf in that tree. We define a non-leaf node u_0's *effective subscription* $S(u_0)$ as the "combined" subscription formed by merging u_0's inherent subscriptions with all of its descendants' subscriptions. Thus, the grafting node of node v_0 is the closest node (in terms of number of hops) in the upward path, whose effective subscription overlaps with that of v_0. In this way, each non-leaf node maintains data structures for its successors' subscription interests, and in u_0's parent $p(u_0)$'s view, u_0 is a child that has the subscription interest of $S(u_0)$.

4.2.2 Parent Evaluation Metrics (PEM)

PEM is the key mechanism for constructing and maintaining the PST. There are various PEMs, including:

- Shortest-path Metric (SM);
- Publication-overhead-aware Metric (PM); and
- Combination Metric (CM) of SM and PM.

SM is a most straightforward metric, in which a node chooses the neighbor at the lowest level as its parent. PM uses the subscription information of the other nodes in the tree to decrease the total number of messages needed to complete the delivery of one publication. In [28], the authors propose a publication-overhead-aware metric that requires a' priori knowledge of the exact rates of invocation (publication) for each category of subscription; otherwise the lack of that knowledge will cause high inaccuracy in a distributed system where the rates of invocation vary over time or even are unpredictable as is the case in our motivating scenarios. Thus, their metric not only suffers from the requirement to have that global information to function properly, but also implies that the PEM they use is limited to topic-based publish/subscribe systems. Note that in a content-based system, subscriptions are so finely grained that events are not classified according to any predefined categories but rather according to the properties of the events themselves. The PM in [6] requires *periodic* messages to go upwards (towards the root) in the tree through a probabilistically chosen candidate parent until it reaches a new grafting node. However, in a large-scale network where subscription interests vary greatly, especially for a content-based system, finding a grafting node may require traversing a significantly long distance and incurring much higher overhead than SM. In [6], the authors claim an advantage of less "overhead" over

the MAODV-based PST approach. However, by "overhead," they only count the messages used for publication without considering the considerable overhead incurred for a better path detection initiated periodically by each node other than the root (level-0) and level-1 nodes.

In [28], the authors also give a combination metric (CM) which calculates the product of the level of a candidate node and the predicted publication overhead to evaluate each of the candidate parent nodes. However, as long as the aforementioned problems with PM exist, the combination metric still suffers from some impractical requirements for it to work properly.

Based on these observations, we choose Shortest-path Metric (SM), similar to the approach in the MAODV-based PST and that in [43]. Obviously, SM inherently improves the timeliness of event delivery.

4.3 Inter-tree Level Interconnection

A natural question that arises for this distributed architecture is which nodes should be the roots, given no centralized administration. Another question is how the inter-tree communication mechanism is designed to facilitate event routing.

4.3.1 Root Node Selection and Tree Merging

A node's role (either root or non-root) in the system is designated for the first time when the P/S middleware is initiated.

Root Selection

We use a random-number-based strategy for distributed root selection. Let D_T denote the density of roots. In fact, D_T has a significant influence on the performance of the architecture, as shown in Chapter 6. Given the size range of a MANET, which often is known at least approximately based on the MANETs application, we can select the best D_T based on experimental studies, which is reasonable. And in fact, as shown in Chapter 6 with the following tree merging and new root selection mechanisms, the system will self-adapt to the initial topology, self-adjust and converge to a stable state, without preselecting a value for D_T. However, a better initial D_T value shortens the system self-adaptation time.

A random number *Rand* within the range of 0 to $Rand_{Max}$ is produced by the P/S middleware on each node. For a node v, that is:

$$\begin{cases} v \text{ is a root node,if } 0 < Rand(v) \leq D_T \cdot Rand_{Max} \\ v \text{ is a non-root node, otherwise} \end{cases}$$

Tree Merging

To deal with the degradation of performance caused by roots lying geographically too close (e.g., one or two hops away from each other), a *tree merging* mechanism is employed.

A commonly used tree maintenance technique is the periodic *refresh* message broadcast with a sequence number, from the root throughout the tree. Every node rebroadcasting this message replaces the *level value* field with its own level value. Whenever a node v_0 receives a foreign tree node u_0's *refresh* message, it calculates the root-to-root distance between $r(v_0)$ and $r(u_0)$ from the level numbers of v_0 and u_0. If the root-to-root distance falls below a threshold $Merging\_Thres$, either v_0 or u_0 sends a *merging request* message to the root, and

```
   input: local node i's level number level(i), level number level(v) of foreign node v,
          r(v)'s address
 1  r2rdistance ← level(i) + level(v) + 1;
 2  if r2rdistance > Merging_Thres then
 3      if i ≠ r(i) then
 4          Send merging message with r(v)'s address to r(i) to initiate tree merging
            process;
 5      else
 6          Initiate tree_merging_process(r(v)'s address);
    /* Function tree_merging_process() */
 7  tree_merging_process(r(v)'s address):
 8  if My address is higher than r(v)'s address then
 9      I will remain as a root;
10      initiate the partition merging process;
11  else
12      I will wait for the other root to initiate the partition merging and I will resign
```

Figure 4.2: Tree Merging Algorithm

the merging process is initiated. The tree merging process is similar to the partition merging in MAODV [46], in which the root with a higher ID retains its role and takes over the other tree. Figure 4.2 describes the tree merging algorithm.

New Root Node Selection

New root node selection is required based on the following considerations. To better accommodate MANET dynamics, it is important to observe that the topology changes due to node mobility and failures can cause the distances among the roots to change. Such dynamics together with *tree merging* can cause the number of trees (i.e., the number of root nodes) to decrease. Thus, schemes must be designed for handling root node failures and mobility, and for controlling the tree sizes to be below a certain upper threshold. The basic idea is to select a new root and start a new tree to overcome those adverse conditions.

```
   input: local node i's level number level(i)
   /* For random number based strategy */
1  if level(i) > New_Root_Thres or Out_Period-timer expires then
2  |  Rand(i) ← new random value;
3  |  if 0 < Rand(i) < D_T · Rand_Max ·α then /* α > 1 */
4  |  |  I become a root;
5  |  else if Out_Period-timer expires then
6  |  |  start Out_Period-timer again;
```

Figure 4.3: New Root Node Selection Algorithm

When a node v's level is above a certain predefined threshold New_Root_Thres, or it has not been a member of any tree for a certain time frame Out_Period, the node checks the possibility of advancing itself to a root. A new random number will be produced raising the probability of the node being accepted as a root. After the new root's "birth", it broadcasts invitations to "crop" tree members from other trees, until the level value of its leaf nodes is no smaller than that of at least one of the leaf nodes' neighboring foreign nodes. This algorithm is illustrated in Figure 4.3.

Note that in a reliable wireless network with little topology change, our tree merging and new root selection strategies for system maintenance would introduce very little additional traffic overhead.

4.3.2 Inter-tree communication

Inter-tree Route Establishment

Similar to the mechanism for tree merging, whenever a node v_0 receives a foreign tree node u_0's *refresh* message, it calculates the root-to-root distance between $r(v_0)$ and $r(u_0)$ from the level numbers of v_0 and u_0. If the new root-to-root distance is smaller than the current value,

node v_0 records u_0 as the next hop destination for inter-tree routing, and send a *route report* message upwards. On receiving the route report message, v_0's parent $p(v_0)$ checks its local root-to-root distance value. If the new value reported is better, $p(v_0)$ will also update its record and send a route report message upwards. In this way, the root will always keep the freshest feasible route for inter-tree communication. Further, a node only registers the next hop destination and the corresponding root-to-root distance for inter-tree routing, requiring only small memory usage. When an inter-tree message (e.g., a publication or notification message) arrives at a foreign node u_0, u_0 will simply forward this message upward to its root. Now, we claim that the route established between two roots of the neighboring trees is the shortest.

Theorem 4.1. *The path p, which is established for inter-tree communication between the roots a and b of two neighboring trees, is the shortest one.*

Proof. Assume that the inter-tree path p connecting $T(a)$ and $T(b)$ established in SOMER is not the shortest one. Then there exists a distinct path p' such that

$$length(p') < length(p).$$

Let the last node of $T(a)$ that lies on p be v_a, and the last node of $T(b)$ that lies on p be v_b; in the same way we have v'_a and v'_b for p'. Then we obtain:

$$\begin{aligned} &length(p') = length(a, v'_a) + length(v'_b, b) + 1 \\ &< length(p) = length(a, v_a) + length(v_b, b) + 1. \end{aligned}$$

Also, we have:

$$\begin{aligned} &length(a, v_a) = level(v_a), length(v_b, b) = level(v_b), \\ &length(a, v'_a) = level(v'_a), length(v'_b, b) = level(v'_b). \end{aligned}$$

According to our inter-tree route establishment process, if there is such a path that $level(v'_a)+level(v'_b) < level(v_a)+level(v_b)$, then we should have already found that path based on the level numbers, resulting in a contradiction. Thus, the assumption does not hold and the theorem is true. □

Inter-tree Overlay Event Routing

Advertisement messages are widely used in P/S systems. On joining a P/S tree, a node sends both its subscription and publication interests to the root nodes. From the top layer view, root nodes can be treated as super publishers/subscribers, with the *effective subscription* and the *effective publication* capabilities. Event routing at the inter-tree level is based on periodic advertisement messages flooding across root nodes such that the root nodes' effective P/S interests are propagated across the top layer. Though the flooding among root nodes incurs message overhead, it is worth it when node mobility is high. (We show in Chapter 6 that the overhead is reasonable.) When a root node r_0 receives an advertisement message forwarded by a neighbor root r_1 originally from r_2, r_0 updates r_2's P/S interests and the inter-tree route entry for r_2, and updates the corresponding next-hop root-address with r_1's address. In this way, the inter-tree routing uses the shortest reverse path to effectively propagate event notifications.

Chapter 5

Self-reconfiguring Architectural Facilitation for Event Causal Dependencies

5.1 Causal Graph Construction

The architectural P/S system management is done at an entity level, with the abstraction from a transaction level. That means in this architecture, nodes are being manipulated to maintain the system structure, while causally-related events flows act as the momentum behind. That reflects that a conceptual transition from event causal dependency to node causal dependency is necessary. As the example in Chapter 1 shows, a node causal graph is a graph comprised of causally-related nodes of the P/S system, created by mapping the event message graph onto the physical system of nodes.

We must be careful when extending the concept of causality from events to the nodes in

a network where events are generated and consumed. The two necessary conditions for an event b to be a "causal child" of event a are:

- b subscribes to a's publication; and
- b's receipt of a's publication event message is the sufficient condition for b to generate a new publication to its subscribers.

In turn, a is b's "causal parent". The corresponding causal relationship is denoted by $a \Rightarrow b$. For instance, given a causal dependency specification as "$(a \Rightarrow b) \wedge (a \Rightarrow c) \wedge (a \Rightarrow d) \wedge (b \Rightarrow e) \wedge (c \Rightarrow e)$", the corresponding causal graph is shown in Figure 5.1.

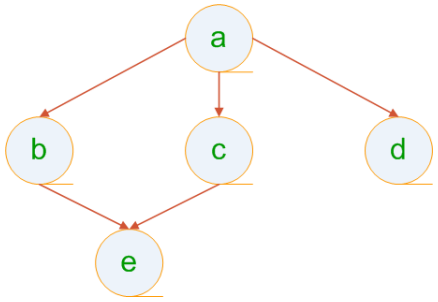


Figure 5.1: Event Causal Graph

Event causal dependencies are deemed strictly anti-symmetric (which may be different from some logical causal relationships). However, the anti-symmetry does not hold among the networked causally related nodes. In Figure 1.1 we can find the retro-causal relationships between nodes.

To simplify the problem, we make careful assumptions on the the composition of the causal graph.

(1) We assume that a node can only be either a "causal descendant" or a "causal ancestor" of another node but not both. This preserves the acyclicity of the node causal graph. In

fact, if that happens we can always break the retro-causal ties by splitting all of them from the original causal graph and creating new disjoint causal graphs.

(2) The causal graph can be reduced to a causal tree if we assume that there is at most one "causal parent" for each causally-related node. However, that abstraction is too strong. A more general and simplifying assumption is that all of one's "causal parents" (if there is any) are at the same level (we set the original event source node to be at Level 0; then one level up as one descendant down on the causal graph).

(3) We do not place any assumption on the number of causal graphs that the system may have (i.e., the system can have any number of original event source nodes and any number of disjoint causal graphs).

(4) In this way, the whole causal graph is reduced to a set of disjoint acyclic directed causal graphs. On this quasi-tree-like logical structure, we borrow the self-explaining concepts of "root" and "leaf" that are used for the tree structures.

Now, advertising the subscription/publication interests throughout the system enables the root nodes of the physical trees to build the causal graph for the whole system.

5.2 Self-reconfiguration with Awareness of Event Causal Dependencies

The original physical trees stop their natural growth when each node has joined in a tree. The tree merging and new root selection strategies confine the tree sizes with $Merging\_Thres$ and New_Root_Thres. However, this may not be the "ultimate" solution for a system with event causal dependencies. We explore further performance improvements based on the following

observations and discussion.

(1) A physical tree of a comparatively larger size typically has more neighboring trees, implying increasing possibilities to establish direct inter-tree connection with more other trees. Consequently, the probability increases for a causally-involved node to get to its causal children through fewer intermediate forwarding nodes.

(2) One physical tree's expansion means other neighboring trees' contraction, as if trees are contending for nodes which can only belong to one tree at any given time. The policy for resolving trees' contention for nodes must be carefully designed. The goal is to shorten the event notification time along the P/S causal chains that originate at the "causal root" until they reach the "causal leaves" in the causal quasi-tree-like graph.

(3) For a given physical tree, if it contains more "causal nodes" at lower levels in the causal graph, or a larger number of causal nodes, it should be given greater preference in the resource contention resolution policy for physical expansion. That is because lower level causal nodes in the causal graph usually have greater influences on the subsequent event notification deliveries that directly or indirectly depend on those nodes in the corresponding causal chains. For instance, in Figure 1.1, any delay that occurs for the event notification delivery between the commander and the agents will have an impact on every subsequent causal chain, whereas the soldier-relay link will only have an influence on one causal chain.

(4) Furthermore, for the causal nodes covered by a physical P/S tree, if a large number of their causal descendants are already covered by the same tree, there is a possibility that the trees expansion will not contribute to the timely and reliable event delivery across the causal graph as much as before when only a few of the causal descendents are covered in the tree. This is because, the profit margin of the tree expansion decreases

and the expansion may cause undesirable consequences in terms of timeliness of event deliveries. For example, when tree t_1 crops nodes from tree t_2 by expansion, if those nodes have their causal parents in tree t_2 and those causal parent nodes are still in t_2 after t_1s expansion, then the event deliveries from those causal parent nodes to the nodes now seized by t_1 will have to make a detour by passing the root of t_1. While this could be avoided by sharing nodes P/S information and nodes geographical information on the inter-tree level, it would add much to the cost of inter-tree communication. Further, in some application settings, it could be quite difficult (or costly) or impossible to get nodes geographical information.

Here, we define a tree's *expansion potential* (EP) as the metric for evaluating the potential benefit of a tree's expansion for event delivery across the causal graph. For a physical tree T, let $N_{Causal}(T)$ be the number of causal nodes that T covers, $Avg\_lev_{Causal}(T)$ be the average level value in the quasi-tree-like causal graph containing the causal nodes covered by T, and γ be the percentage of causal descendants that is covered by T for all the causal nodes in T. Now, $EP(T)$ is given by:

$$EP(T) = \frac{(N_{Causal}(T(v)) + \eta)}{(Avg\_lev_{Causal}(T) + \eta) \cdot (\gamma + \eta)}. \tag{5.1}$$

Here, η is a parameter with a constant positive value to offset the variables that could be zero.

It is the leaf nodes which actually carry out the expansion (i.e., contending) through periodically requesting neighbor foreign nodes for a comparison of each other's *local expansion potential* (LEP). Because we need to also constrain the tree's branches from abnormal growth,

a node v's LEP is:

$$\begin{aligned} LEP(v) &= \frac{EP(T(v))}{(level(v)+\eta)} \\ &= \frac{(N_{Causal}(T(v))+\eta)}{(Avg\_lev_{Causal}(T(v))+\eta)\cdot(\gamma+\eta)\cdot(level(v)+\eta)} \end{aligned} \tag{5.2}$$

For two nodes u and v involved in such contention, the one with the higher LEP will win, and the loser will join the winner tree as a child of the winner. It is possible that a node may constantly change its affiliation. To avoid that situation, we stop two nodes from contending when:

$$\frac{1}{1+\delta} \leq \frac{LEP(v)}{LEP(u)} \leq 1+\delta \tag{5.3}$$

Here, δ is a parameter to tune the contention severity.

The result of the contention may change the size of a tree. The contention is allowed to take place only when the merging threshold or new root selection threshold is not violated.

The causality awareness module can be integrated as an add-on to the existing event routing hierarchical architecture, as done in SOMER. Yet we should note that only reconfigurable architecture can support such dynamic self-architectural-facilitation to causal dependencies.

Chapter 6

Experimental Evaluation

We conducted sets of simulation experiments using NS-2 with the Random Trip Mobility Model package [4] to generate sets of random MANET topologies.

6.1 Simulation Environment Setup

Our simulation environment is built with each node having its own subscription interest. A randomly selected set of nodes acts as publishers. We used the model of Number Intervals [28] for P/S pattern generation. We used a large number interval as the interest pool, and a node's subscription interest is represented by a random subset of the interest pool. We call it a match when the number associated with a published event falls into the range of a node's interest. We do not restrict publishers to have constant event injection rates. The publishers either choose a constant rate or a randomly varying rate to publish. We carried out 250 runs for each set of experiments and, when applicable, we provide the 95% confidence intervals in Appendix A for the results reported below. Table 6.1 details the simulation settings.

Table 6.1: Simulation Settings

Parameter	Range/Value
Simulation Area	$1000m \times 1000m$ – $3300m \times 3300m$
Network Size	50 – 250 nodes
Simulation Period	1000s
Node Wandering Velocity	0 – 10 m/s
Pause Time	1s – 3s
MAC Protocol	IEEE 802.11 with 2Mbps Bandwidth
Root Node Density (D_T)	1% – 18%
Wireless Transmission Range	200m
Proportion of Publishers	30% – 40%
Proportion of Causal Nodes	5% – 15%
Proportion of Causal Nodes	10%
Tree Refresh Period	10s
Advertisement Period	10s
Transmission Jitter	5% – 10%
Advertisement Period	10s
$Merging\_Thres$	1 – 3
New_Root_Thres	5 – 10
Out_Period	15s
Tree Contention Tuner (δ)	0.2 – 0.4

6.2 Architecture Evaluation and Analysis

6.2.1 Influence of D_T

As mentioned in Chapter 4, D_T is a parameter that can affect the system's convergence time.

To evaluate the values of D_T, we deactivated the tree merging and new root selection mechanisms to ensure that the number of roots does not change during the experiments. Using delivery time as the performance metric, we measured the average time cost in 250-node networks for both single-publish-hop and complete causal graph event delivery. We normalized the delivery time with respect to the minimum value in each case, so that the trend can be

clearly observed.

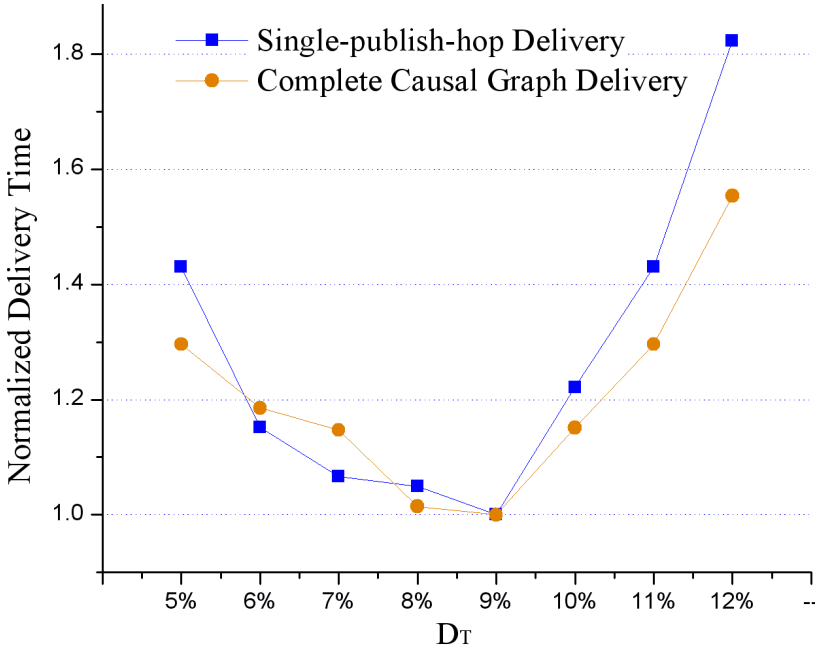


Figure 6.1: Timeliness Performance under Different D_T w/o Tree Merging/New Root Selection

As expected, the curves in Figure 6.1 exhibit the same trend. Further, we observe that at $D_T = 9\%$, both the curves reach the minimum value, indicating that 9% is the best value for D_T under current settings. When D_T decreases from 9%, the average time cost increases because there would be fewer root nodes which act as brokers in the top-layer overlay, degrading the efficiency of the inter-tree route. On the other hand, as the number of trees increases, a publisher and one of its subscribers in the same tree could be separated to be in two trees. Consequently, the event messages from the publisher would have to make a detour through at least two root nodes to reach the subscribers, in contrast with the fact that the messages only need to go through one root node if they were in the same tree. For a given network topology, we can find the best D_T. However, doing so requires some a' priori knowledge of the system. We will show in the following results that we can avoid calculating or preselecting a best D_T.

6.2.2 Tree Merging and New Root Selection

Recall that through the tree merging and new root selection strategies, the system interconnection is self-reorganized, thereby optimizing the tree distribution toward timely event delivery.

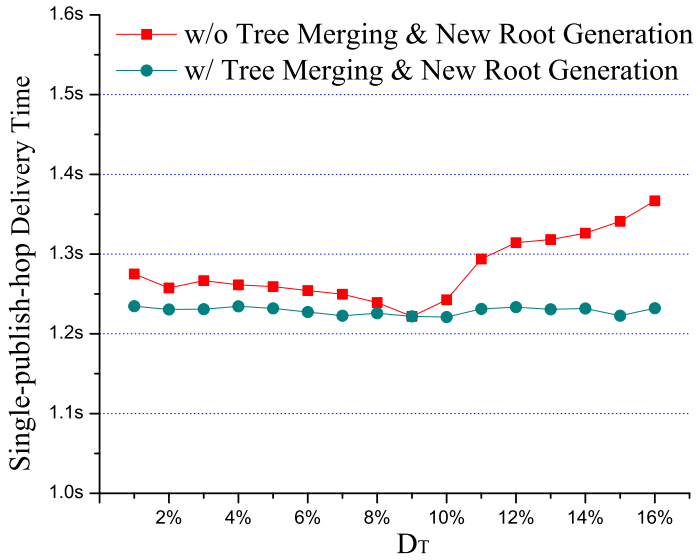


Figure 6.2: Effectiveness of Tree Merging and New Root Selection

From Figure 6.2, we observe that after these strategies take effect, the average time cost for single-publish-hop event delivery is effectively lowered and the system has stable performance as the initial density of root nodes (D_T) varies all the way from 1% to 16%. (We used 200-node networks for these experiments.) This implies that our strategies are stable. The decrease in the average event delivery time is due to the new root selection strategy when the initial D_T is small. The tree merging strategy reduces the average delivery time while the initial D_T is comparatively large.

6.2.3 Event Delivery with Awareness of Causal Dependencies

To evaluate the effectiveness of SOMER's causality awareness design, we performed two sets of experiments via:

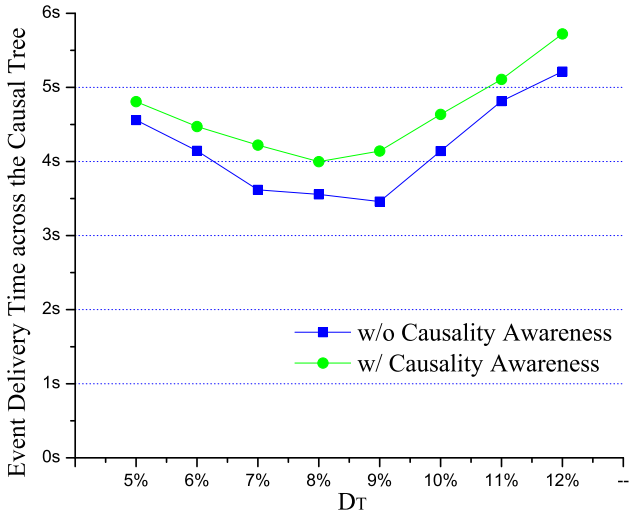
(1) deactivating the tree merging and new root selection mechanisms to observe the "pure" performance gain from system self-reconfiguration for causal event delivery; and

(2) reactivating those two mechanisms, and evaluating the performance gain with every part of SOMER working together.

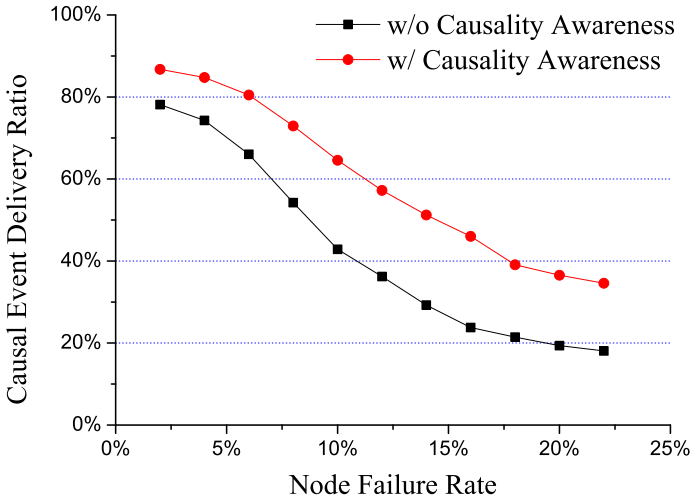
Figure 6.3 is produced by the first set of experiments with 250-node network scenarios. We tested the timeliness and reliability improvement from the architectural support for causal event delivery. The performance metrics we used are event delivery time across the causal graphs and the causal event delivery ratio. The causal event delivery ratio is the success ratio of event delivery through one causal chain. To test reliability, we used node failure rate as a variable, with node wandering speed randomly ranging from 0 to 10m/s.

We observe in Figure 6.3(a) that our design yields as much as over 15% improvement at around the best D_T (9%). In Figure 6.3(b), we observe that SOMER's causality awareness mechanisms also enhance the average success ratio of causal event delivery and thus the system reliability. Because the tree merging and new root selection mechanisms were deactivated during this set of experiments, the performance is still much lower than that of SOMER as a whole.

In the second set of experiments, we evaluate the average timeliness gain with all the algorithms and mechanisms of SOMER working together as a whole. Because trends of average reliability gain would be similar to those of timeliness gain, we only show the results for the timeliness gain here by varying the network size and the tree contention severity tuning

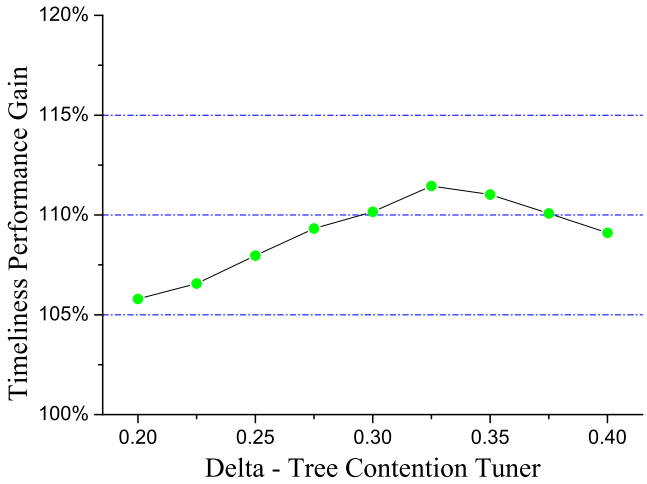


(a) Timeliness Improvement

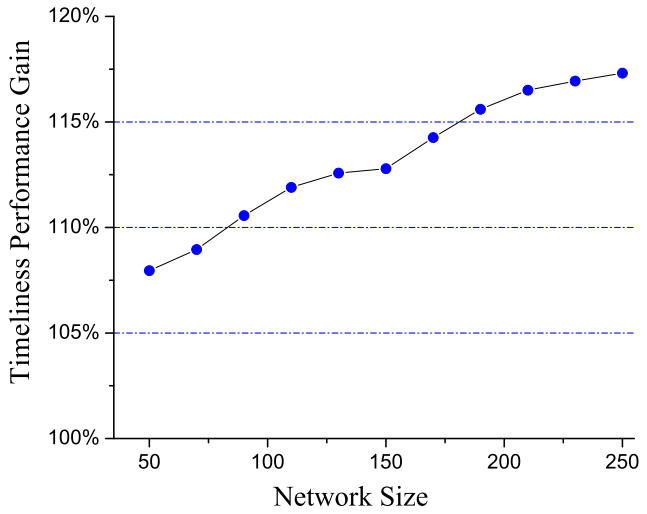


(b) Reliability Improvement

Figure 6.3: Improvement from Causal Event Delivery Architectural Support



(a) Timeliness Gain under Different δ



(b) Timeliness Gain with Different Network Size

Figure 6.4: Timeliness Gain

parameter δ.

We first explored the space of δ by obtaining the average performance gain with network sizes increasing from 50 to 250 for a given value of δ. From Figure 6.4(a), we observe a peak performance gain around $\delta = 0.325$. When δ increases this gain reduces due to the degradation of tree contention severity such that the system has less self-reconfigurability for causal event delivery. The gain also reduces as δ decreases because the overhead incurred by severe tree contention adds to the system instability and thus partly negates the performance gain.

Figure 6.4(b) illustrates that SOMER achieves around 10% average performance gain, and higher when the network size scales up. This trend corresponds to Equation 3.6, which states that the upper bound of the path overhead ratio usually decreases for larger size networks. Thus, our experiments illustrate that SOMER's design is scalable and effective toward reducing the total event delivery time across a causal graph.

6.2.4 Timely and Reliable Event Delivery

We compared SOMER's timeliness and reliability against significant past MANET P/S work including:

- SP-COMBO [28], a PST protocol with a combination of shortest path and publication-overhead-aware metrics;
- DSAPST [6], a distributed subscription-aware PST protocol; and
- PS-MAODV [35], an MAODV-based PST protocol.

The network size was varied with a constant D_T in the experiments. We omitted SensTrac from our comparison due to the significant differences between SensTrack and SOMER as

described in Chapter 2. Note that none of the past work addresses event causal dependencies.

Timeliness

We measured and calculated the timeliness performance index by reciprocating the measured time cost for event delivery. Then we normalized each protocol's performance to the worst one's performance. In this way, we can also observe the trend of relative performance as the network size changes.

Figure 6.5 illustrates SOMER's advantage in timely delivery over others. DSAPST performs the worst because the way it constructs the PST is mainly based on a publication-overhead-aware metric as discussed in Chapter 4.

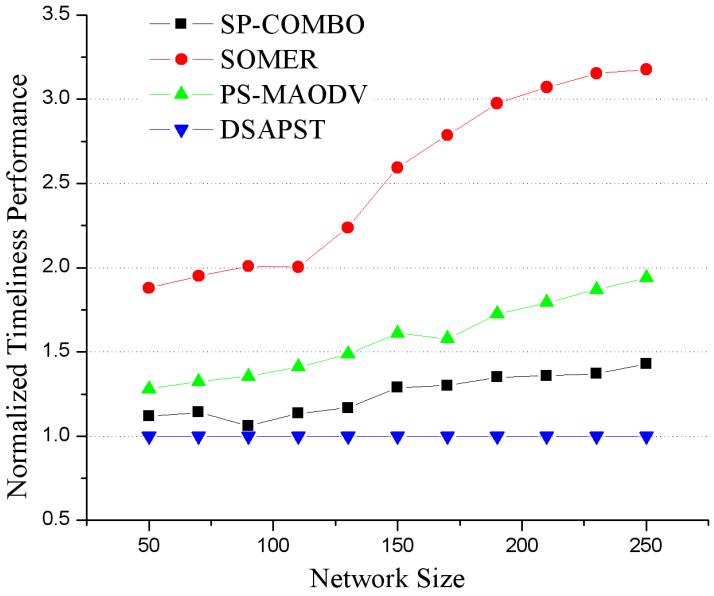


Figure 6.5: Comparison of Timely Delivery

By normalizing with respect to DSAPST's performance, we can clearly observe SOMER's performance gain over the others. SOMER outperforms PS-MAODV, which is the second best, with at least a 30% improvement. As the network size scales up, SOMER's improvement over PS-MAODV increases up to around 100%, illustrating SOMER's superior scalability.

This is due to the increase of the number of trees that can be used as top-layer brokers to make the route closer to the straight line path between two nodes.

Reliability

We measured reliability through *delivery ratio*, which is calculated as the number of copies of event messages successfully received by their subscribers over the number of copies of event messages that should arrive at all of their subscribers, given no failures or no topology changes. We used node failure rate as a variable to assess system reliability, with node roaming speed randomly ranging from 0 to 10m/s. Figure 6.6 shows that SOMER can survive a 12% node failure rate with an over 75% average event delivery ratio. The highest performance improvement of SOMER is an over 10% higher average delivery ratio than that of PS-MAODV.

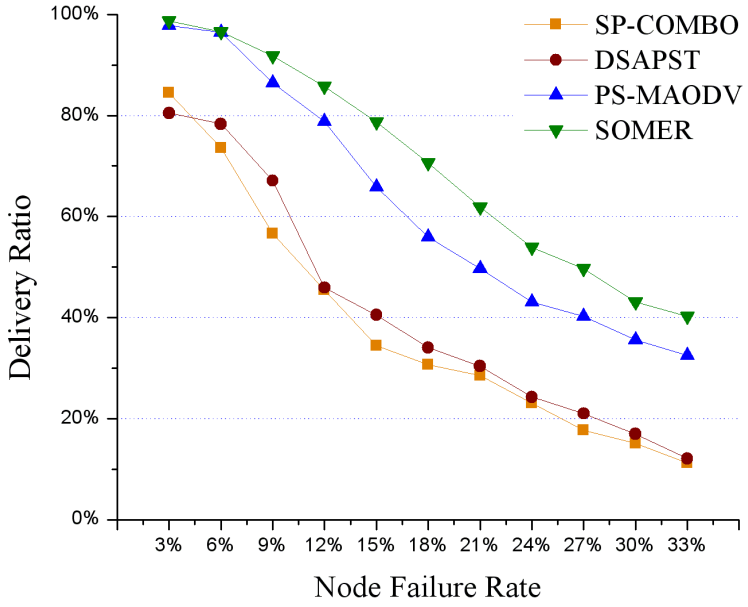


Figure 6.6: Comparison of Delivery Ratio under Node Failure

Note that SOMER also provides a 30% higher average delivery ratio over the other two rivals. SOMER's improvements are due to its inherent distributed multi-tree structure, and

also due to its tree merging and new root selection strategies that effectively counter network unreliability.

Network Traffic Load

For a structured architecture, the traffic load of a P/S system has two sources:

- Structure Maintenance Messages; and
- P/S Messages.

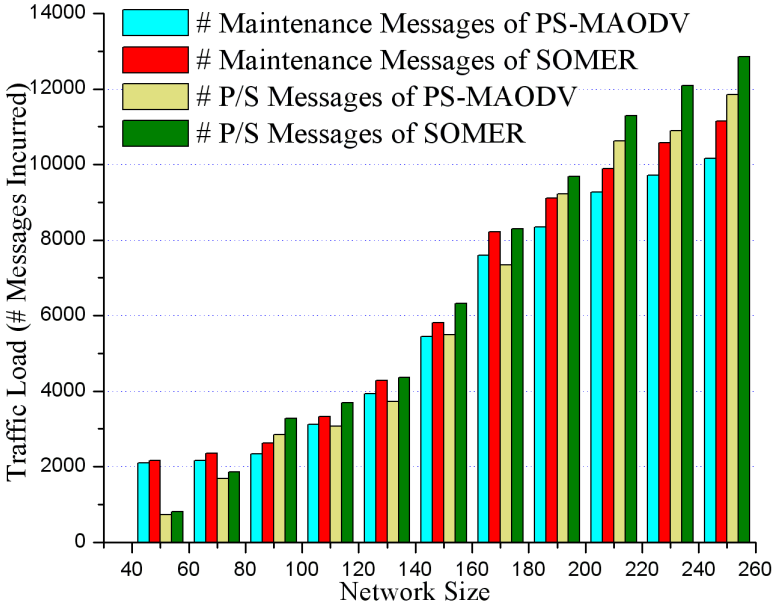


Figure 6.7: Network Traffic Load with Low P/S Load

In tree-based structures, the maintenance messages are those used for maintaining trees. P/S messages include the event notification messages, subscription/unsubscription messages, and advertisement messages if there are any. We measured the number of messages incurred for each source type throughout the entire simulation period. The number of messages increases whenever a message is produced or forwarded. We observed that, among PS-MAODV, DSAPST, and SP-COMBO, only PS-MAODV had relatively acceptable timeliness

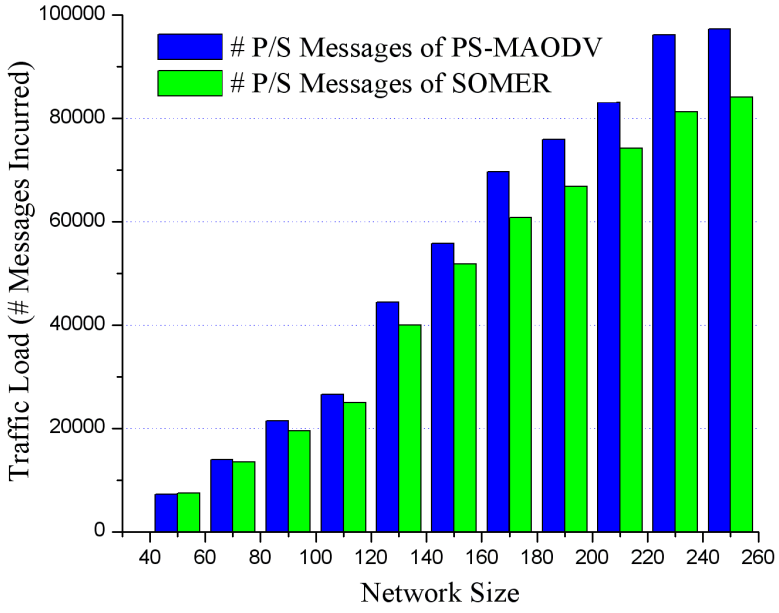


Figure 6.8: Network Traffic Load with High P/S Load

and reliability. Thus, we compared SOMER and PS-MAODV with respect to the network traffic load incurred under the same simulation scenarios and publication patterns.

First, we conducted simulations with a low P/S traffic load injected. Figure 6.7 shows that SOMER introduces slightly more maintenance messages due to the neighbor-tree route setup and tree merging and new root selection processes. We also observe that SOMER spends slightly more P/S messages (at most about 10%) than PS-MAODV. This is due to the periodic advertisement messages on the top-layer overlay.

Interestingly, we observe that SOMER outperforms PS-MAODV under a higher P/S traffic load, as shown in Figure 6.8. Here, we omit the statistics of maintenance messages because they almost do not change when the traffic load increases. SOMER is able to reduce the average number of P/S messages by dispatching the events along shorter paths from publishers to subscribers. Also, its higher robustness avoids considerable message retransmissions. More injected events means a larger reduction in the average number of P/S messages.

Chapter 7

Conclusion and Discussion

This thesis is based on a more general perspective of P/S systems than is usual. First, it recognizes that P/S information need not be confined to data information but may instead (or indeed, also) be control information. Second, it takes a higher level system view of P/S —normally P/S is considered to take place by one hop (not counting transparent network devices such as switches and routers) from the publisher to the subscribers, but there are cases where receipt of subscribed information causes subscribers to subsequently publish, resulting in causal dependency graphs of publishes. Such graphs raise a variety of interesting issues, including end-to-end timeliness, that to our knowledge have not been addressed before.

In this thesis, we begin to address some of those issues by presenting SOMER, a self-organizing and self-reconfigurable event routing architecture for a class of P/S events that have causal dependencies and time constraints. This architecture is oriented toward ad hoc mobile networks, providing algorithms and mechanisms for P/S tree construction and maintenance with self-organizing and self-configurable capabilities. Through the simulation experiments shown in Chapter 6, we see that SOMER can achieve as much as 30%–100% improvement on timeliness and up to 30% improvement on reliability improvement compared

to previous solutions, such as PS-MAODV, DSAPST and SP-COMBO. Then referring back to our motivating notional example in Chapter 1, the SOMER architecture's reconfigurability will make the multi-hop causal P/S event flow be more timely and reliable throughout, and particularly back to the commander as the causal event sequence initiator in Figure 1.1.

Causal dependencies are common in modern event-based systems. For instance, complex causal dependencies can be found in a customized event-based financial service system with multiple agents servicing multiple investors. Investors trigger the service whenever they have request/requirement updates to push into the service system. An agent servicing a certain number of investors changes their subscription to the financial indexes information and system computing resources accordingly. The customer portfolio module subscribes to the investment decisions that investors have made, and agents also subscribe to changes in the portfolios of their customers. Whenever a new event arrives at an agent from the financial indexes information module, the agent analyzes all the data and makes suggestions with risk and profit probabilities. When at the agent side there is an event matching an investors criteria, a warning signal will be triggered (published) out to the corresponding investor. Such causal event flows can also be found in event-based sensor networks, where sensors are used to detect anomalies and send messages through a gateway to a corresponding control unit; then either actuators are triggered or a false alarm is cleared. Please refer to [31, 49] for details of similar use cases.

Causal dependencies in event-based systems need more attention from both academia and industry given their common use but neglected study. Tasks that lie in a causal chain sometimes are of more significance when the causal chain as a whole influences decision making or situation handling. Any causal link's failure (including timeliness failure) in the middle of a causal chain may cause the failure of the end goal of the causal chain, similar to the occurrence of a failure in a chain of RPCs or remote method invocations. Failures in a

causal graph have more complex potential impacts, and require more sophisticated recovery schemes. Mechanisms are thus needed to (1) recognize causal links; (2) build causal graphs; (3) determine the critical causal links or chains; (4) tune system performance based on the knowledge of causal dependencies.

There exist different approaches to addressing those problems, from a system designer's view, a software developer's view, or a system administrator's view:

(1) here in the design of SOMER, we present a system design approach;

(2) from a software developer's point, software add-ons may be plugged in for P/S middleware;

(3) from a system administrator's point, system configuration strategy profiles can be created and maintained by a system administrator or distributively by application users.

However, to the degree that the problem can be solved during the system design phase, then a crisp software development cycle and simpler system administration schemes, will result in less expensive system and application software development and system administration.

So far, we have only dealt with deterministic time-sensitive causal dependencies; there are also probabilistic causal dependency links, delay-tolerant causal dependency links, etc. By probabilistic, we mean that an event may probabilistically cause a subsequent event to successfully occur. By delay-tolerant, we mean that a causal descendent event may take some time to occur, or may not have a stringent timeliness requirement. Thus, there remains a large design space to explore for further understanding and improving causal P/S systems, their performance, and their reliability. It is likely that application-specific approaches may also be required for certain cases.

SOMER, as it is, may offer insights for subsequent event routing protocol designs. And it

can be easily extended with real-time scheduling strategies and multi-path schemes. The current version of SOMER includes some presumptions:

(1) The design and evaluation of SOMER's framework is implicitly based on the premise that all the nodes in the system are, or can be treated as, homogeneous. When nodes are heterogeneous (i.e., under different configuration, performing different tasks, or using different hardware, etc.), the performance of the system may deviate from the current results due to possible different nodal processing time, network interface buffer length, and network connection type, etc. To make SOMER more adaptable, more system configuration and performance data are needed.

(2) The system causal graphs are assumed to be such that all of one's "causal parents" are at the same level. We can weaken this assumption to adapt SOMER to any general causal graph by splitting those causal links and extracting overlapping or non-overlapping subgraphs from the original causal graph; then the nodes can have logical shadow replications among subgraphs. And similarly for case of the retro-causal arcs, instead of creating new disjoint causal graphs, we may consider using subgraph extracting to "break" those arcs. The graph splitting or subgraph extracting algorithm must be carefully designed such that the system reconfiguration mechanism can be effective at improving the timeliness and reliability of causal event deliveries.

Event, and particularly P/S event, routing in ad hoc networks is still an emerging area. Interesting issues which call for both academic and industrial research include, but are not limited to: (1) the co-existence and mutual facilitation between a typical ad hoc network routing scheme and an event routing scheme; (2) self-organizing protocols for event routing under opportunistic channel communication; (3) geographic routing induced event routing, etc.

Bibliography

[1] R. Baldoni, R. Beraldi, G. Cugola, M. Migliavacca, and L. Querzoni. Structure-less content-based routing in mobile ad hoc networks. In *IEEE International Conference on Pervasive Services*, pages 37–46, 2005.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *the 19th IEEE International Conference on Distributed Computing Systems*, page 262, Washington, DC, USA, 1999. IEEE Computer Society. http://www.research.ibm.com/distributedmessaging/gryphon.html.

[3] A. R. Bharambe, S. Rao, and S. Seshan. Mercury: A scalable publish-subscribe system for internet games. In *the 1st workshop on Network and system support for games*, pages 3–9. ACM Press, 2002.

[4] J.-Y. L. Boudec and M. Vojnovic. Perfect simulation and stationarity of a class of mobility models. In *IEEE INFOCOM*, pages 2743–2754, March 2005.

[5] F. Cao and J. P. Singh. Efficient event routing in content-based publish-subscribe service networks. In *IEEE INFOCOM*, pages 929–940, March 2004.

[6] X. Cao and C.-C. Shen. Subscription-aware publish/subscribe tree construction in mobile ad hoc networks. In *IEEE 13th International Conference on Parallel and Distributed Systems*, pages 1–9, November 2007.

[7] M. Caporuscio, A.Carzaniga, and A.L.Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering*, 29:1059–1071, December 2003.

[8] N. Carvalho, F. Araujo, and L. Rodrigues. Reducing latency in rendezvous-based publish-subscribe systems for wireless ad hoc networks. In *the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, page 28, Washington, DC, USA, 2006. IEEE Computer Society.

[9] A. Carzaniga and C. P. Hall. Content-based communication: A research agenda. In *the 6th International Workshop on Software Engineering and Middleware*, pages 2–8. ACM, November 2006. Invited Paper.

[10] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001. http://www.inf.unisi.ch/carzaniga/siena/.

[11] G.-P. Castellote and P. Bolton. Distributed real-time applications now have a data distribution protocol. *RTC Magazine*, February 2002. http://www.rti.com/docs/RTC_Feb02.pdf.

[12] M. Castro, P.Druschel, A. Kermarrec, and A. Rowston. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, October 2002.

[13] D. Chakraborty and T. Finin. Toward distributed service discovery in pervasive computing environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112, 2006.

[14] R. K. Clark. *Scheduling dependent real-time activities*. PhD thesis, Carnegie Mellon University, 1990.

[15] G. Consolver, D. Ackley, M. Rickard, R. McAfee, and T. Shipchandler. Distributed processor/memory architectures design program. Technical report, TEXAS INSTRUMENTS INC. DALLAS, February 1975. Available at: `http://www.stormingmedia.us/28/2846/A284610.html`.

[16] S. Corson and J. Macker. *Routing Protocol Performance Issues and Evaluation Considerations (RFC 2501)*. Network Working Group, 1999.

[17] P. Costa and D. Frey. Publish-subscribe tree maintenance over a dht. In *DEBS*, pages 414–420. IEEE Computer Society, 2005.

[18] P. Costa, D. Gavidia, B. Koldehofe, H. Miranda, M. Musolesi, and O. Riva. When cars start gossiping. In *the 6th workshop on Middleware for network eccentric and mobile applications*, pages 1–4, New York, NY, USA, 2008. ACM.

[19] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 26(5):748–760, June 2008.

[20] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola. Epidemic algorithms for reliable content-based publish-subscribe: An evaluation. In *the 24th IEEE International Conference on Distributed Computing Systems*, pages 552–561, 2004.

[21] P. Costa and G. P. Picco. Semi-probabilistic content-based publish-subscribe. In *the 25th IEEE International Conference on Distributed Computing Systems*, pages 575–585, June 2005.

[22] G. Cugola, E. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *the 20th International Conference on Software Engineering*, pages 261–270, April 1998.

[23] A. Datta, S. Quarteroni, and K. Aberer. Autonomous gossiping: A self-organizing epidemic algorithm for selective information dissemination in wireless mobile ad-hoc networks. In *International Conference on Semantics of a Networked World*, pages 126–143, 2004.

[24] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *the 37th annual ACM symposium on Theory of computing*, pages 494–503, New York, NY, USA, 2005. ACM.

[25] L. Fiege, F. Gartner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middlewares. In *ACM/IFIP/USENIX International Middleware Conference*, pages 103–122, 2003.

[26] L. Fiege, G. Muhl, and P. R. Pietzuch. *Distributed Event-based Systems*. Springer-Verlag B&H, 2006.

[27] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and B. Weihl. Publish-subscribe notification for web services (version 1.0). IBM's White Paper, March 2004. http://www.ibm.com/developerworks/webservices/library/specification/ws-pubsub/.

[28] Y. Huang and H. Garcia-Molina. Publish/subscribe tree construction in wireless ad-hoc networks. In *the 4th IEEE International Conference on Mobile Data Management*, pages 122–140. Springer-Verlag, 2003. ISBN 3-540-00393-2.

[29] E. D. Jensen, C. D. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Real-Time Systems Symposium*, pages 112–122. IEEE, 1985.

[30] M. Junginger and Y. Lee. A self-organizing publish/subscribe middleware for dynamic peer-to-peer networks. *IEEE Network*, 18(1):38–43, 2004.

[31] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan. Service oriented architecture for heterogeneous and dynamic sensor networks. In *the 2nd international conference on Distributed event-based systems*, pages 309–312, New York, NY, USA, 2008. ACM.

[32] E. Lupu, N. Dulay, M. Sloman, J. Sventek, S. Heeps, S. Strowes, S. Strowes, K. Twidle, K. Twidle, S.-L. Keoh, and A. Schaeffer-Filho. Amuse: Autonomic management of ubiquitous e-health systems. *Concurrency and Computation: Practice & Experience*, 20 (3):277–295, 2008.

[33] E. R. B. Marques, G. M. Goncalves, and J. B. Sousa. Seaware: A publish/subscribe based middleware for networked vehicle systems. In *the 7th IFAC Conference on Manoeuvring and Control of Marine Craft*, pages 20–22, September 2006.

[34] E. R. B. Marques, G. M. Goncalves, and J. B. Sousa. The use of real-time publish-subscribe middleware in networked vehicle systems. In *the 1st IFAC Workshop on Multivehicle Systems*, 2006.

[35] L. Mottola, G. Cugola, and G. P. Picco. A self repairing tree topology enabling content-based routing in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, February 2008.

[36] V. Muthusamy, M. Petrovic, and H. Jacobsen. Effects of routing computations in content-based routing networks with mobile data sources. In *MobiCom*, pages 103–116. ACM, 2005.

[37] B. Oki, M. Pfluegel, A. Siegel, and D. Skeen. The information bus —an architecture for extensive distributed systems. In *ACM Symposium on Operating Systems Principles*. ACM, 1993.

[38] OMG. Data distribution service for real-time systems (version 1.2). Object Management Group's Specification, January 2007. http://www.omg.org/technology/documents/formal/data_distribution.htm.

[39] G. Pei, B. Ravindran, and E. D. Jensen. On a self-organizing manet event routing architecture with causal dependency awareness. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 339–348. IEEE Computer Society, 2008.

[40] D. Peleg. Low stretch spanning trees. In *the 27th International Symposium Mathematical Foundations of Computer Science*, pages 68–80, August 2002.

[41] J. L. Pfaltz. Using concept lattices to uncover causal dependencies in software. In *ICFCA*, pages 233–247, 2006.

[42] G. P. Picco, G. Cugola, and A. L. Murphy. Efficient content-based event dispatching in the presence of topological reconfiguration. In *the 23rd IEEE International Conference on Distributed Computing Systems*, pages 234–243. IEEE Computer Society, May 2003.

[43] S. Pleisch and K. Birman. Senstrac: Scalable querying of sensor networks from mobile platforms using tracking-style queries. In *IEEE MASS*, pages 306–315, October 2006.

[44] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[45] E. M. Royer and C. E. Perkins. Multicast operation of the ad-hoc on-demand distance vector routing protocol. In *MobiCom*, pages 207–218. ACM, 1999.

[46] E. M. Royer and C. E. Perkins. *Multicast Ad hoc On-Demand Distance Vector (MAODV) Routing (INTERNET DRAFT)*. Mobile Ad Hoc Network Working Group, 2000.

[47] S. K. Sarkar, T. G. Basavaraju, and C. Puttamadappa. *Ad Hoc Mobile Wireless Networks: Principles, Protocols and Applications*. Auerbach, 2007.

[48] I. Stoica, R. Morris, et al. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11:17–32, Februrary 2003.

[49] W.-C. Tsai and A.-P. Chen. Service oriented architecture for financial customer relationship management. In *the 2nd international conference on Distributed event-based systems*, pages 301–304, New York, NY, USA, 2008. ACM.

[50] R. van Renesse and A. Bozdog. Willow: Dht, aggregation, and publish/subscribe in one protocol. In *IPTPS*, pages 173–183, February 2004.

Appendix A

Confidence Interval Tables

Table A.1: Confidence interval for Figure 6.1.

Delivery type \ D_T	5%	6%	7%	8%
Single-publish-hop	±0.14	±0.12	±0.13	±0.11
Complete causal graph	±0.08	±0.04	±0.06	±0.05
Delivery type \ D_T	**9%**	**10%**	**11%**	**12%**
Single-publish-hop	±0.12	±0.13	±0.12	±0.14
Complete causal graph	±0.05	±0.06	±0.06	±0.06

Table A.2: Confidence interval for Figure 6.2.

PST strategies in use? \ D_T	1%	2%	3%	4%	5%	6%
Yes	±0.13	±0.15	±0.11	±0.12	±0.11	±0.11
No	±0.08	±0.11	±0.09	±0.10	±0.10	±0.09
PST strategies in use? \ D_T	7%	8%	9%	10%	11%	12%
Yes	±0.12	±0.10	±0.11	±0.12	±0.10	±0.09
No	±0.09	±0.08	±0.06	±0.07	±0.07	±0.06
PST strategies in use? \ D_T	13%	14%	15%	16%		
Yes	±0.07	±0.08	±0.07	±0.09		
No	±0.05	±0.06	±0.05	±0.05		

Table A.3: Confidence interval for Figure 6.3(a).

Causality aware? \ D_T	5%	6%	7%	8%
Yes	±0.08	±0.06	±0.07	±0.06
No	±0.08	±0.04	±0.06	±0.05
Causality aware? \ D_T	9%	10%	11%	12%
Yes	±0.06	±0.06	±0.08	±0.06
No	±0.05	±0.06	±0.06	±0.05

Table A.4: Confidence interval for Figure 6.3(b).

Causality aware? \ Node falure rate	2%	4%	6%	8%
Yes	±0.27%	±0.19%	±0.16%	±0.17%
No	±0.18%	±0.15%	±0.15%	±0.14%
Causality aware? \ Node falure rate	10%	12%	14%	16%
Yes	±0.21%	±0.18%	±0.24%	±0.26%
No	±0.16%	±0.17%	±0.25%	±0.28%
Causality aware? \ Node falure rate	18%	20%	22%	24%
Yes	±0.23%	±0.32%	±0.34%	±0.33%
No	±0.23%	±0.34%	±0.32%	±0.32%

Table A.5: Confidence interval for Figure 6.5.

Protocols \ Network Size	50	70	90	110	130	150
SOMER	±0.10	±0.09	±0.13	±0.14	±0.15	±0.14
PS-MAODV	±0.09	±0.08	±0.10	±0.09	±0.09	±0.10
SP-COMBO	±0.08	±0.05	±0.08	±0.08	±0.06	±0.07
DSAPST	±0.03	±0.05	±0.06	±0.06	±0.05	±0.04

Protocols \ Network Size	170	190	210	230	250	
SOMER	±0.11	±0.12	±0.09	±0.13	±0.14	
PS-MAODV	±0.08	±0.12	±0.11	±0.13	±0.09	
SP-COMBO	±0.08	±0.05	±0.08	±0.07	±0.06	
DSAPST	±0.03	±0.05	±0.04	±0.02	±0.04	

Table A.6: Confidence interval for Figure 6.6.

Protocols \ Node Failure Rate	3%	6%	9%	12%
SOMER	±1.47%	±2.59%	±2.56%	±1.77%
PS-MAODV	±1.38%	±1.45%	±2.55%	±2.54%
SP-COMBO	±2.08%	±2.35%	±3.08%	±3.44%
DSAPST	±2.57%	±2.05%	±3.36%	±3.06%

Protocols \ Node Failure Rate	15%	18%	21%	24%
SOMER	±2.31%	±3.18%	±3.35%	±4.04%
PS-MAODV	±3.16%	±2.47%	±3.34%	±2.76%
SP-COMBO	±3.76%	±3.87	±4.25%	±2.98%
DSAPST	±3.15%	±2.97	±4.24%	±3.26%

Protocols \ Node Failure Rate	27%	30%	33%	
SOMER	±2.93%	±2.72%	±1.44%	
PS-MAODV	±2.63%	±3.04%	±2.52%	
SP-COMBO	±3.23%	±1.92%	±2.74%	
DSAPST	±2.23%	±2.24%	±1.32%	