QoS in Event Based Middleware

Annual Progress Seminar 2 - Report

Shruti P. Mahambre
(Roll no. 04429801)

under the guidance of
Prof. Umesh Bellur

Kanwal Rekhi School of Information Technology
Indian Institute of Technology Bombay
2006
Abstract

Event Broker Networks are a scalable incarnation of the publish subscribe paradigm for building asynchronous systems. These take the form of overlays of broker nodes and several routing schemes exist that deliver events from publishers to subscribers efficiently on different overlay structures. The decoupled and asynchronous nature of event based architectures has made it a popular choice for large scale software systems today. While a number of principles on which such architectures ride, have been well established, attention has now turned to exploring non-functional attributes of such systems. In this effort, we first present a taxonomy of event based middleware and an accompanying survey of existing event based middleware efforts centered around the provision for qualities of service guarantees. In the process of putting together this survey we have also identified open research problems in the area. Specifically we look at the prospect of routing events based on reliability requirements of subscribers based on the event type, via the broker network. We propose a reliability model, which measures reliability of the overlay network, and an algorithm based on this model, to deliver event notifications to the client. We employ a technique called ‘pruning’, by which we restrict flooding the entire overlay network, when finding a reliable path. The complexity analysis of our algorithm shows that it finds a reliable path with a lower message complexity, as compared to the flooding approach. We also verify our claims, with simulation results, using the Hermes[Pie04] middleware simulator.
# Contents

Abstract i

List of figures iii

List of tables iv

1 A Taxonomy and Classification of Event Based Middleware 2
   1.1 Introduction ................................................. 2
   1.2 Exploiting the Taxonomy ................................... 3
   1.3 Interpreting the Taxonomy .................................. 3
   1.4 Related Work ............................................... 4
   1.5 Middleware Architecture ................................... 5
   1.6 Taxonomy of Event Based Middleware ....................... 6
       1.6.1 Core .................................................. 6
       1.6.2 Discussion ........................................... 11
       1.6.3 Services .............................................. 12
       1.6.4 Discussion ........................................... 15
   1.7 Classification of Event Based Middleware .................. 15
   1.8 Research Issues ............................................ 18
   1.9 Conclusion ................................................ 18

2 Routing for Reliability in Event Based Middleware 19
   2.1 Introduction ............................................... 19
   2.2 Reliability in Publish Subscribe .......................... 20
       2.2.1 Per subscriber per event type ......................... 20
       2.2.2 Per subscriber all event types ....................... 20
       2.2.3 All subscribers for an event type ................... 21
       2.2.4 Entire System .......................................... 21
   2.3 Related Work .............................................. 21
   2.4 Reliability Model .......................................... 22
       2.4.1 Assumptions .......................................... 23
   2.5 Our Approach ............................................... 25
List of Figures

1.1 Taxonomy Key ......................................................... 3
1.2 QoS and Adaptation in Middleware .......................... 5
1.3 Core of the Middleware .......................................... 6
1.4 The Event Model ................................................... 7
1.5 Overlay Topology of the Middleware ......................... 8
1.6 Subscription Scheme of the Middleware ..................... 10
1.7 Middleware Services ............................................... 12
1.8 QoS in Middleware ................................................ 13

2.1 Layers in the Broker Network Architecture ................... 19
2.2 Reliability Model of a Node ...................................... 23
2.3 Path Establishment in the Pruning Algorithm ................. 26
2.4 Actual v/s Expected Reliability for a Subscriber ........... 31
2.5 Percentage of Messages generated by Pruning, in comparison to Flooding 31
2.6 Comparison of message complexity for Pruning and Flooding 32
List of Tables

1.1 Classification of event based middleware ......................... 17
1.2 Classification of Middleware based on Support for QoS ........... 17
Chapter 1

A Taxonomy and Classification of Event Based Middleware

1.1 Introduction

Event based middleware is a powerful and scalable paradigm, capable of building large scale distributed systems. It extends the notion of publish-subscribe, and also facilitates the construction of an overlay network over the publish-subscribe brokers. Middleware alleviates the heterogeneity of large scale distributed applications, by enabling interoperability amongst its components.

In an event based middleware, information provided by the producers is distributed in a timely manner to the consumers. A common service interface provided by event based systems is the publish-subscribe (pub-sub)[PT03]. In a pub-sub paradigm, the producers publish information, consumers subscribe to this information, and receive notifications for the same. This information is encapsulated in a structure termed as the event. This model relies on an event notification service, provides storage and management for subscriptions and efficient delivery of events. There is anonymity amongst both, the senders and the receivers, and the communication between them is asynchronous. The fully decoupled nature in terms of time, space and synchronization[PT03], makes this model highly suitable for large scale distributed applications.

Pub-Sub systems are deployed on overlay networks. An overlay network[PB03] constructs a user level graph on top of an existing networking infrastructure, such as the Internet, using a subset of the available network links and nodes. An overlay link is a virtual link in this graph and may consist of many actual links in the underlying network. Overlay nodes run applications in a distributed manner providing middleware functionalities. Event based services are used in a large number of applications, in various domains. These domains, may require the provision of guarantees for the services they demand from the middleware. However, few of the existing middleware [Pie04], [AR02],[GC01] provide limited support for qualities of service. Moreover, in the process of ensuring qualities of service, the middleware may require to be adaptive in nature. Taking these factors into
1.2 Exploiting the Taxonomy

In the next section, we discuss how the taxonomy may be exploited by users, when classifying event based systems. Section 1.3 tells users how the taxonomy should be interpreted. In Section 2.3, we differentiate our work with existing approaches to middleware classification. Section 1.5 presents an architecture of an adaptive event based middleware which provides support for quality of service guarantees. We present a detailed taxonomy of the core features of the middleware and also the services it provides in Section 1.6. At the end of the taxonomy, we present a classification of existing event based middleware, illustrating the usage of the taxonomy in Section 1.7. Finally, we identify some research issues, which are an outcome of this classification and conclude in Section 1.9.

1.2 Exploiting the Taxonomy

The taxonomy presented in this report, is represented in the form of a hierarchy of properties which define a middleware. These properties, (i.e., core characteristics of the middleware), serve as a basis for identifying features, required by an event based system, in order to ensure the non-functional services(i.e. qualities of service). For eg, the classification shown in Figure 1.3, identifies a set of features, which are directly or indirectly affected, when providing service guarantees in a middleware. In this regard, the taxonomy also identifies, characteristics of a middleware, which reflect its adaptive nature. The taxonomy may lead to identification of groups of systems, providing quality of service support in existing middleware, by using the classification shown in Figure 1.8. Moreover, this taxonomy is beneficial in determining combinations of the above mentioned properties, which assist in architecting an adaptive middleware, with support for quality of service guarantees.

1.3 Interpreting the Taxonomy

The taxonomy is represented using figures and text. Each figure is depicted as a hierarchy, which is a tree shaped structure, with the root occurring at the top, and the leaves at the
bottom. The connections between the nodes of the tree, are shown using arrows. The *the filled arrow* represents an *or* relationship between the connecting nodes, i.e., either of the paths should be chosen. The *straight lined arrow* represents an *and* relationship between the connecting nodes, i.e., all paths should be traversed. The *line ending with a filled circled* represents both, the *and* and *or* relationship, i.e., the user will have to traverse at least one of the paths and at most all the paths. At the highest level, we have classified, event based middleware as having a *core* and a set of *services*. The user then traverses the hierarchy, by selecting the nodes along the path, which suit the characteristics of the middleware. This process is continued, till the user reaches a leaf node. Leaf node, indicates a termination of the path in the tree.

1.4 Related Work

The taxonomy presented in Meir et al. [MC05] deals with classification of existing event based systems as a programming paradigm. It identifies fundamental properties of event based middleware and classifies them broadly based on *event model* and *event service* criteria, with a further classification on the basis of its *functional* and *non functional* features and its *organization* and *interaction model*. Using this taxonomy as a basis for a programming model of an event based middleware, we have come up with our own taxonomy, which focuses on run-time issues related to middleware. Our taxonomy centers around quality of service guarantees and adaption in middleware. It is based on the event based middleware having a *core*, encompassing all functional features, and *services* which are optional. We further the definition of an event model as a *communication model* in [MC05] by taking into account, event characteristics such as, attributes, hierarchical relationships between events and compositional relationships between events.

Eugster et al [PT03], present a detailed survey of event based systems, which is based on a common denominator of the various modes of decoupling in asynchronous systems. The full decoupling of the communication entities, is discussed in three dimensions, in terms of time, space and synchronization. The main focus of the paper is on implementation issues underlying the publish/subscribe schemes, and how these issues have been addressed in existing middleware. This paper also identifies some common qualities of service relevant in pub-sub paradigm. We use this paper as a basis for our work, in identifying features that will require to form an inherent part of any event based system, that intends to support quality of service guarantees and classify them.

Martin et al [BEMBR91], present a taxonomy, as a hierarchy of questions and answers, about the features of distributed computing systems (DCS). This taxonomy, though not specific to the pub-sub paradigm, can be used to compare existing general purpose distributed computing systems and provides a means to classify systems and also serves as
1.5. Middleware Architecture

As shown in Figure 1.2, we represent an event based middleware in the form of a layered architecture. We distinguish the core of the middleware, from the services it provides, as follows - The core comprises of functional features, which form an integral part of any middleware. For eg: The core defines the event model, the subscription scheme used to route events in the model and the overlay topology which facilitates this routing. The services supported by the middleware may be functional or non-functional in nature. In essence, functional services also form a part of the functional features of a middleware. However, they do not comprise the core, since they are not mandatory to the functioning of any middleware. Security and mobility, are functional services, i.e., their functioning is directly related to the core of the middleware, where as fault tolerance, load balancing and quality of service (QoS) guarantees are non-functional in nature. The middleware may support a subset of the above mentioned services. In the taxonomy presented in our paper, we focus on QoS, which is a non-functional service, affecting all the layers of a middleware. At the services layer, is the QoS Programming Model which comprises of the QoS specification provided by the user. This specification is then mapped to the lower layers (i.e. the core) and translated into the required resource reservation metrics. Hence we depict the QoS parameters, namely reliability, message ordering and delivery semantics, to be cutting across all the layers of a middleware. If the service guarantee demanded by a user is not renegotiable, then the middleware may have to adapt itself in order to meet the user requirements. Adaptation leads to changes in the core of the middleware as well, thus affecting its functional characteristics. Hence as shown in Figure 1.2, Adaptation also cuts across all the layers of the middleware.
1.6 Taxonomy of Event Based Middleware

As shown in Figure 1.3, the event based middleware forms the root of our taxonomy. Every event based middleware has a core and a set of services that it supports. We differentiate between the services and the core, in order to capture the fact that the core defines properties which are mandatory for the functioning of a middleware, and the services are additional features which a middleware may support - which do not form a part of the core functionality of a middleware.

1.6.1 Core

The core of a middleware as can be seen from Figure 1.4 represents features related to the basic functionality characterizing any middleware. The primary task of an event based middleware, is to pass messages, which are encapsulated as events, in the event broker network. The essence of an event based middleware will be captured by its event model. Every event based middleware will have an event model which defines properties of the events it supports, also known as the programming model of the middleware. An overlay topology, which determines the event dissemination routes and a subscription scheme, representing the semantics of expression used by the clients, when subscribing or publishing events.

1.6.1.1 Event Model

- **Event Hierarchy**
  Events in a system may be represented in the form of a *hierarchy*. Hierarchical relationships amongst events lead to a polymorphic dispatch of events in the system. For eg: An event maybe a supertype of other events. A subscriber subscribing to a supertype receives notifications from all subtypes of the event, as well as the supertype. DAC[PTE00] supports a topic based event hierarchy, i.e., events in DAC are represented in the form of topics. Subscribers express their interest in these topics, and notifications are sent for the subscribed topic along with notifications.
1.6. Taxonomy of Event Based Middleware

for all its subtypes. In Hermes, every rendezvous node is aware of their descendent types, and hence, subscriptions to the parent rendezvous node, can be supported by inheritance routing.

- **Event Composition**

  Composite Events [Pie04] provide a higher level of abstraction for event subscribers. A composite event service enables event subscribers to subscribe to complex event patterns, as opposed to subscribing to all primitive events which make up the pattern and then form the detection themselves. The Hermes [Pie04] middleware, provides support for a composite event service. Composite events may be temporal or spatial in nature. Temporal composite events lead to the formation of a composite based on temporal dependencies between primitive events. Spatial composite events are unrelated in time, and are published based on a pattern of event subscriptions observed by the composite event detection engine.

1.6.1.2 **Overlay Topology**

As shown in Figure 1.5 Event Based Middleware may also be categorized on the basis of its overlay topology model. Overlay topology, is a layer above the physical network, and represents logical links and broker nodes, which determine event dissemination routes in the system. An overlay network can be represented as a graph with vertices corresponding to the overlay nodes and edges corresponding to the overlay paths between the pairs of overlay nodes. The nodes for the overlay network and the edges are decided by the application. Hence a middleware architecture may support one or multiple overlay topologies. In this section, we classify overlay networks for event based middleware. The actual physical network is described as the underlay of the overlay network. Overlay networks used in network applications can be classified based on the method adopted for the overlay construction. Overlays may be constructed with or without underlay awareness. Effec-
1.6. Taxonomy of Event Based Middleware

![Overlay Topology Diagram]

Figure 1.5: Overlay Topology of the Middleware

Figure 1.5: Overlay Topology of the Middleware

tively, overlay topologies influence the design of routing algorithms in the middleware, which further affect the selection of optimal routes when ensuring service guarantees.

- **Underlay Aware**
  The underlay aware overlay construction strategy maintains correspondence between the physical and the overlay networks. We can further classify underlay aware overlay networks, as those which make use of the structure of the physical network topology during construction, and those which along with the structure-awareness, also continuously monitor the changing network characteristics like bandwidth and traffic on links, the computational load at each physical node, buffer sizes and node and link failures. These overlays are termed as *adaptive overlays*. Such overlays reflect the changes in the physical network and maintains the correspondence with the physical network throughout the lifetime of the application. The number of network characteristics to be monitored can vary with the application for which the overlay is constructed. Tools are available for measuring the underlay properties and reporting to the overlay network. Hence depending upon the need of the application it can monitor and measure a predefined set of network properties. At present there are no adaptive overlays exist for the event broker network application. As against this, *structure aware overlays* maintain the correspondence with the physical topology during the initial phase of the overlay formation. Thereafter, the changes in the physical network, if any, are not reflected in the overlay network. In other words the overlay networks assume a static underlay. Most of the overlay networks used in event based middleware, like Rebeca, Gryphon, Hermes [Pie04], and Jedi [GC01] fall into this category. Once the static overlay network, has been constructed, it may follow different overlay topologies as shown below.
- **Hierarchical Topology**
  In a hierarchical topology, the nodes are connected in a hierarchy of parent-child relationships. Each server in the topology has a number of clients. These clients may be publishers, subscribers or other servers (brokers) in the network. Each server will have a special connection to a parent server. The parent server is able to receive notifications, advertisements and subscriptions from all its clients, but will only send back notifications. Jedi uses a hierarchical overlay topology for dispatch of events.

- **Acyclic Peer To Peer Topology**
  In an acyclic peer to peer topology, servers communicate with each other as peers. The server-server links are non-directed arcs and the configuration of connections among servers produces an acyclic graph. This leads to a special protocol that allows bi-directional flow of subscriptions, advertisements and notifications.

- **Generic Peer To Peer Topology**
  Generic peer-to-peer topology, removes the constraint of an acyclic graph from acyclic peer-to-peer topology. This topology allows bi-directional communication amongst two servers. However the network of connections among servers is a generic graph, which may possibly lead to multiple paths between servers.

- **Hybrid Topology**
  A hybrid topology supports a combination of topologies, i.e., it exhibits different topologies at different levels of granularity. Siena [Car98] poses a good example of a hybrid topology. For eg: Some clusters of subnets in Siena have very intense traffic of local events, and only a small fraction of these events is visible outside the cluster. Here, for efficiency reasons, a generic graph topology might be preferable inside the cluster while the high-level topology could be set up as an acyclic graph.

- **Transit Stub Topology**
  Transit Stub topology [EZB96] model is used to generate large scale topologies that resemble the structure of the internet. Basically, a transit-stub topology comprises of multiple domains. Each of these domains represents the Internet autonomous systems. A domain is a stub domain if it never carries any transit traffic, otherwise it is called a transit domain. The purpose of transit domains is to interconnect stub domains efficiently, thus forming an Internet backbone for other autonomous systems. Hermes uses transit-stub topology model for routing of events.

- **Underlay Unaware**
  Here the construction of the overlay is done using distributed hash tables. The logical identifiers of the overlay nodes and the identifiers of the neighbors of a node are
1.6. Taxonomy of Event Based Middleware

generated based on a random hash function. The overlay assumes full network connectivity and does not measure any of the physical network properties in the overlay construction process. Chord [SMK+01] constructs underlay unaware overlays.

1.6.1.3 Subscription Scheme

Subscription scheme as shown in Figure 1.6, is the manner in which subscribers specify their interest in an event, also known as subscription scheme. A topic based scheme, will expect the subscriber to subscribe to a topic/subject. In content based scheme, the subscriber will have to provide information pertaining to the content of the event it has subscribed for. And the type based scheme expects the subscriber to subscribe to an event based on its type.

- **Topic/Subject Based Scheme**
  
  In this scheme, participants publish events and subscribe to individual topics. Each topic is identified by a keyword. This is similar to group communication, where similar topics are grouped together. DAC uses topic based routing scheme. Yancees has its own topic based dispatcher. Topics typically overlap, and hence a hierarchy of topics may be introduced, in order to facilitate classification of topics. This hierarchy is similar to event hierarchy discussed in Section 1.6.1.1.

- **Content Based Scheme**
  
  The subscriptions in content based scheme are based on the actual content of the event, i.e., the events are not classified according to a topic, but according to the properties of the events themselves. This enables a more fine grained search, as compared to the topic based scheme, and reduces the set of matching events for a subscription. The properties of the events maybe internal attributes of the data structures carrying the events. Content based routing scheme is supported by Jedi[GC01], Siena and Elvin.

- **Type Based Scheme**
  
  In type based routing scheme, the events are classified according to their type[PT01]. This is a variation of the topic based routing scheme, where events are regrouped according to their commonalities in structure and content. Type based scheme,
facilitates a closer integration of the language and the middleware and also ensures type safety at compile time. IndiQos supports type based routing scheme. Hermes also supports type based routing, with an additional attribute level filtering.

- Hybrid Scheme

This routing scheme is a combination of the routing schemes discussed in the sections above.

1.6.2 Discussion

In this section, we have categorized an event based system, according to its functional characteristics, which play a significant role in the functioning of any middleware. However, each of these also affect the decisions taken when providing service guarantees. The event model, determines properties of events and relationships between them. The middleware will have to take into account the hierarchical and compositional relationships between events, when sending notifications.

The overlay topology is instrumental in determining the event dissemination route. Again, this route, will have to adhere to the service guarantees requested by the clients, when routing events. In the hierarchical topology, the server is a critical point of failure, i.e., when server fails, it disconnects all its client subnets, and also all parent subnets reachable from its parent server. Acyclic peer-to-peer topology suffers similar problems since there is a lack of redundancy in the connection graph. In comparison, the generic peer-to-peer topology is more robust, since there is redundancy in connection between different points in the network. However, owing to this redundancy, there is a need to design specific algorithms to choose the best path and to avoid cycles. Hybrid topology overcomes these issues, by enabling the network to exhibit different topologies at different levels of granularity. And the transit-stub topology model is well suited to emulate the internet domain.

The subscription scheme (routing scheme), determines the level of granularity attained when filtering subscriptions. A limitation of the topic based scheme is that, it is static in nature and also offers limited expressiveness. The content based routing scheme improves on this by providing a subscription based on the actual content or properties of the event. However, even though the content based scheme, enables a fine grained search, this scheme may influence the response time for notification of an event, since it would increase overlay latency, for a notification, owing to the elaborate search as compared to topic based scheme. Type based scheme is similar to topic based scheme, with the additional feature, that it ensure type safety at compile time. However there is a trade-off between the overall latency as against the finer grained search, owing to the additional search criteria, which may lead to violation of service guarantees. In the next section, we discuss these service guarantees in more detail.
1.6. Taxonomy of Event Based Middleware

1.6.3 Services

As seen in Figure 1.7, an event based middleware may support services, which are functional or non-functional in nature. Services are categorized as functional, if they form an inherent part of the core functioning of the middleware. For eg: Security is a functional service, since it directly forms a part of the event model (secure events, authentication, authorization, access control of events). However, Quality of Service guarantees are an additional service supported by the middleware, which is not related to the core functionality of the middleware. Hence QoS is classified as a non-functional service. We elaborate on QoS as a non-functional service guarantee provided by the middleware. QoS or Quality of Service[CI01] refers to the ability of a system to provide network and computational services, such that the users expectations for all non-functional criteria, such as timeliness and performance quality are met. As seen in Figure 1.7, constraints in event based systems are applied at different levels of abstraction[BZ05]. We identify a specific class of service guarantee called as Middleware QoS. This class, comprises of a set of quality of service parameters, which are applicable to an event based middleware. Middleware QoS defines a set of service guarantees, which are specific to an event based system. These service guarantees, follow a publisher-offered, subscriber-requested pattern. It is the task of the event-broker network to ensure that the quality constraints are met with, when delivering events. As shown in Figure 1.8, we have identified five classes of service guarantees - Reliability, Delivery Semantics, Message Ordering, Latency and Bandwidth.

- **Latency**
  The Latency requirement, also known as time to delivery, is specified as time between a publisher publishing an event and a subscriber for the same event, receiving a notification for the same. It is the task of the overlay network to effectively reduce the overall latency of event notifications, in an event broker network.

- **Bandwidth**
  Bandwidth denotes the percentage of bandwidth, available across the path during event transfer. If a subscriber does not specify a requirement, then default values are
assumed, which provide the maximum possible bandwidth available along a path. IndiQos provides support for Latency and Bandwidth parameters. IndiQos uses Netpipe, which is an Infopipe\textsuperscript{1} responsible for transferring events between different hosts. The length of the Netpipe represents the latency and the width of the Netpipe represents its maximum bandwidth.

- **Reliability**

  Delivery Reliability in an event based system is defined as the ratio of number of notifications sent for an event type, to the number of events published of that type. Assume that \( n_i \) represents the number of notifications sent for an event type \( \tau_i \) and \( p_i \) is number of publications of event of type \( \tau_i \), then the reliability \( r \) for and event of type \( \tau_i \), is given by equation 2.1

\[
  r[e(\tau_i)] = \frac{n(\tau_i)}{p(\tau_i)} \tag{1.1}
\]

Delivery reliability, maybe be measured at different levels in the system -

- Per subscription per event type
- Per subscriber all event types
- All subscribers for an event type
- Entire System

Details of reliability parameter are discussed in Section 2.2

\textsuperscript{1}Infopipes are software components that can be connected to each other to form an overlay network
• **Delivery Semantics**

Delivery Semantics refers to the level of guarantee of message delivery which the event broker network provides a subscriber. The subscriber specifies delivery semantics for each event type, in terms of notifications which are mandatory, and whether the duplication of notifications is required. Delivery semantics come into play at the last hop, i.e., just before the notification reaches the subscriber. They are dependent on two boolean values - Reliability for an event type, i.e. Reliability Factor $r_f$, calculated as shown in 2.1 and support for Duplicate messages, i.e., $\theta_a$. The least level of a delivery guarantee is best effort.

\[
\neg[r_f] \land [\theta_a]
\]  

(1.2)

This is a default guarantee provided by the broker network. As can be seen from 1.2, it does not require the network to be reliable. The notifications are sent without any guarantee of delivery and duplication. The at most once delivery semantic as shown in equation 1.3,

\[
\neg[r_f] \land \neg[\theta_a]
\]

(1.3)

ensures that a subscriber receives maximum one notification of an instance of an event type. However, it may receive multiple notifications from different event types, which it has subscribed for. The at least once delivery semantic as described in 1.4

\[
[r_f] \land [\theta_a]
\]

(1.4)

ensures that the subscriber receives at least one notification of an instance of an event type. So the subscriber can receive multiple notifications of the same event type, albeit from different instances of that event type. The exactly once delivery semantic, described in 1.5

\[
[r_f] \land \neg[\theta_a]
\]

(1.5)

ensures that the subscriber receives a notification from exactly one instance of an event type.

• **Message Ordering**

The message ordering property identifies the sequence in which the notifications are delivered to the subscriber(s) w.r.t other notifications in the event based system. The event broker network, may adopt a default temporal ordering in the form of first in first out, (FIFO) or last in first out, (LIFO).

If $\Theta$ is a time stamp associated with events $e$ of type $\tau_i$, $\tau_j$ in the system, and $n$ is notification message then FIFO ordering is expressed as shown in 1.6 and LIFO message ordering maybe be expressed as shown in 1.7.

\[
\Theta[e(\tau_i)] < \Theta[e(\tau_j)] \Rightarrow n[e(\tau_i)] \rightarrow n[e(\tau_j)]
\]

(1.6)
1.7. Classification of Event Based Middleware

\[
\Theta[e(\tau_i)] < \Theta[e(\tau_j)] \Rightarrow -n[e(\tau_i)] \rightarrow n(e(\tau_j)) \tag{1.7}
\]

If the notifications are not ordered, then we assume that the default ordering is either random or unordered. Another form of ordering is Total Ordering as shown in 1.8. In this case, if an event notification of an event type is to be sent to multiple subscribers, then it is desirable that all notifications of the event type are delivered in the same order to all subscribers. Assume that \(n\) and \(n'\) are the notifications received at subscribers \(s\) and \(s'\) for event types \(\tau_i\) and \(\tau_j\). The total ordering semantics are given in 1.8

\[
n[e(\tau_i)] < n[e(\tau_j)] \Rightarrow -(n'[e(\tau_j)] < n'[e(\tau_i)]) \tag{1.8}
\]

Causal Ordering, is required, if the order between events is defined on the basis of a relative cause-effect relationship. All events in Jedi are causally ordered. Casual ordering can be expressed as shown in 1.9

\[
e(\tau_i) \rightarrow e(\tau_j) \Rightarrow n[e(\tau_i)] \rightarrow n[e(\tau_j)] \tag{1.9}
\]

Prioritization of messages is another form of ordering, in which priority numbers are assigned to events when published, and ordering is done on the basis of these priorities. A higher priority event notification can preempt a lower priority one. If \(p\) denotes the publication of event \(e\) of type \(\tau_i\), \(\tau_j\), then prioritization in an event based system is expressed as shown in 1.10

\[
p[e(\tau_i)] < p[e(\tau_j)] \Rightarrow n[e(\tau_i)] \rightarrow n[e(\tau_j)] \tag{1.10}
\]

1.6.4 Discussion

In this section, we have identified, quality of service parameters, required to be supported by an event based middleware. As can be seen from table 1.1, most of the existing middleware, do not provide support for service guarantees. Infact, latency and bandwidth are assumed to be inherent, when routing events in any middleware. IndiQoS explores latency and bandwidth as QoS parameters in detail. Reliability, Message Ordering and Delivery Semantics have been supported on a small scale so far. Hermes provides reliability as a service guarantee since its routing algorithms use built-in fault-tolerance features which enable event brokers to recover to a consistent system state after failure and Jedi provides, a causal ordering of events. We present the details in table 1.2 in Section 1.7.

1.7 Classification of Event Based Middleware

In this section, we present a classification of the event based middleware, based on their core i.e., their functional characteristics, and the support they provide for quality of service parameters. We have picked Jedi, Siena, Hermes and IndiQoS as a basis for this
1.7. Classification of Event Based Middleware

classification. Jedi, Siena, Hermes and IndiQoS provide partial support for some of the service guarantees we have identified in Section 1.6.3. Siena does not provide support for any QoS parameter, but describes an event model, comprising of single or multiple event servers, with different overlay topologies used in routing events. The overlay topology plays a significant role in the routing of events, when ensuring service guarantees to client. Using these systems as the basis for our work, we further identify features, that would be affected as a result of the provision for service guarantees, and express them in the form of core features of a middleware. The taxonomy of the core identifies a set of features, which are not only required for the basic functioning of any event based middleware, but which would also be useful in architecting a middleware which will support service guarantees. Besides this, table 1.1 also gives us a clear insight into the current level of support provided by existing middleware for service guarantees. Table 1.2 illustrates the same in greater depth.
### Table 1.1: Classification of event based middleware

<table>
<thead>
<tr>
<th>Middleware</th>
<th>Hermes</th>
<th>Siena</th>
<th>Jedi</th>
<th>IndiQoS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core of the Middleware</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event Model</td>
<td>Object Based. Supports event hierarchy and composition</td>
<td>Pattern based model in the form of a triplet (name, type, value)</td>
<td>Object Based (Active Objects and Event Dispatcher)</td>
<td>Object Based with QoS Profiles and QoS conditions</td>
</tr>
<tr>
<td>Subscription Scheme</td>
<td>Hybrid - Type based and Type and Attribute Based</td>
<td>Content Based</td>
<td>Content Based - Pattern matching of events</td>
<td>Type Based</td>
</tr>
<tr>
<td><strong>QoS Support</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overlay Network QoS</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Latency, Bandwidth</td>
</tr>
<tr>
<td>Middleware QoS</td>
<td>Reliable routing in case of failure of node</td>
<td>Not supported</td>
<td>Causal ordering of message</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

### Table 1.2: Classification of Middleware based on Support for QoS

<table>
<thead>
<tr>
<th>Middleware</th>
<th>Overlay QoS</th>
<th>Support for QoS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latency</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>Elvin</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Jedi</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Siena</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Gryphon</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Yancees</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Hermes</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>IndiQoS</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DAC</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
1.8 Research Issues

Our survey has revealed many interesting issues - potentially one for building a comprehensive quality of service framework, based on a theoretical model. It is obvious from this survey, that providing service guarantees in an event based middleware is still in its nascent stages, as most existing middleware, do not fully support any of the quality of service parameters that have been identified. Another potential research issue is to come up with a programming model and a run time model specification, identifying static issues related to QoS-expression, and runtime issues related to QoS implementation in an event based middleware. This leads to further challenges, i.e., identifying the issues brought about by the interplay of QoS parameters, and come up with a conflict resolution matrix of the same. And finally, another potential research issue, this survey highlights is to design an architecture of an adaptive middleware, that will also provide support for service guarantees.

1.9 Conclusion

In this chapter, we have presented a taxonomy for event based middleware, based on its core characteristics and the level of support provided for service guarantees. A shortcoming of most of the existing middleware, is their limited support for expression and enforcement of these guarantees. Primary reason being that, traditional approaches for provision of QoS are based on establishment of direct communication channels, which require to reserve resources when a QoS requirement is specified. In an event based middleware, the participating entities in the communication remain oblivious to each other. This taxonomy also classifies existing event based middleware from the perspective of the support they provide for service guarantees and highlights potential research issues, that are an outcome of the this classification.
Chapter 2
Routing for Reliability in Event Based Middleware

2.1 Introduction

Based on the taxonomy presented in chapter 1, we now focus on the QoS parameters identified in the Section 1.6.3 and elaborate on reliability as a middleware quality of service parameter.

Pub-Sub systems are deployed on overlay networks. An overlay network[PB03] constructs a user level graph on top of an existing networking infrastructure, such as the Internet, using a subset of the available network links and nodes. An overlay link is a virtual link in this graph and may consist of many actual links in the underlying network. Overlay nodes run applications in a distributed manner providing middleware functionalities. As shown in the figure above, Hermes[Pie04] is implemented using three layers. The bottom layer represents the physical network comprising of routers and links. The middle layer represents the peer-to-peer overlay network. This layer provides an abstraction of the distributed hash table. The top layer represents the event brokers, which form the event dissemination tree, used for routing events. Every event broker maps to a node in the overlay network, which in turns maps to a physical network node. A single hop in the overlay network, could result in multiple hops in the physical network.
2.2 Reliability in Publish Subscribe

QoS or Quality of Service [CI01] refers to the ability of a system to provide network and computation services, such that the users expectations for all non-functional criteria, such as timeliness and performance quality are met. Some of the quality of service parameters, which an event based middleware may support are minimum bandwidth requirement, order of event notifications, reliable delivery of event notifications, delivery semantics of an event, latency and availability.

In an event based middleware, we measure the delivery reliability of an event as the ratio of the number of notifications sent against the number of publications of an event type. In a type based pub-sub model, events are filtered according to their types [PT03]. A type based model is analogous to a topic based model, except that the topic-based classification of events, is replaced by a type-based classification. A type-based event model facilitates type safety [PT03] at compile time and leads to a closer integration of the language and the middleware. Hermes middleware provides type-based [Pie04] and type-and-attribute-based [Pie04] routing of events.

Assume that $n$ represents the number of notifications sent for an event type $\tau_i$ and $p$ is number of publications of event of type $\tau_i$, and reliability is expressed as a value varying between 0 and 1, then the reliability $r$ for an event of type $\tau_i$, is given by equation 2.1

$$r[\tau_i] = \frac{n(\tau_i)}{p(\tau_i)} \quad (2.1)$$

Using equation 2.1, reliability is measured at different levels in an event based system as follows:

2.2.1 Per subscriber per event type

The reliability attained by a subscriber $s_i$ for a single event type $\tau_i$, is given by the equation 2.2

$$r[s_i(\tau_i)] = \frac{n(\tau_i)}{p(\tau_i)} \quad (2.2)$$

2.2.2 Per subscriber all event types

A subscriber can measure its total reliability, by calculating the reliability it attains across all the event types for which it has subscribed. The reliability attained by a subscriber $s_i$ for all the event types $\tau_j$, it has subscribed for, is given by equation 2.3

$$r[s_i(\tau_j)] = \frac{\sum_{j=1}^{\tau_{max}} r[s_i(\tau_j)]}{\tau_{max}} \quad (2.3)$$
2.2.3 All subscribers for an event type

The reliability offered by the entire system for a specific event type, can be calculated across all the subscribers, who have subscribed for that event type. The reliability offered by the system for an event type $\tau_i$, across all the subscribers $s_j$ is given by equation 2.4

$$r[s_j(\tau_i)] = \frac{\sum_{j=1}^{s_{max}} r[s_j(\tau_i)]}{s_{max}}$$

(2.4)

2.2.4 Entire System

The reliability of the entire system, is defined as the reliability attained across all the subscribers and for all event types, they have subscribed for. Using 2.2, 2.3 and 2.4 the reliability of the entire system is as given in equation 2.5

$$r[s_i(\tau_j)] = \frac{\sum_{i=1}^{s_{max}} [\sum_{j=1}^{r_{max}} r[s_i(\tau_j)]/r_{max}]}{s_{max}}$$

(2.5)

2.3 Related Work

TAROM [CT05] finds a secondary overlay path that minimizes the joint failure probability for a given primary overlay path between a source and destination. It performs path probing to obtain loss rates in overlay paths and then applies random sampling to estimate the link loss rates. The alternative paths to two independent (i.e. paths between pairs of two non-interacting sources and destinations) primary paths may interfere. If both the primary paths fail simultaneously, the alternative paths are used and hence the common links between the alternative paths may get excess traffic leading to high losses (both in links and at corresponding nodes). Also TAROM is invoked "on demand", i.e., when a node sees the need to establish multiple paths. It is not tied to an event model, where factors such as the arrival rate of event publication, and number of events in the system are relevant. TAROM uses the secondary overlay paths only as backup paths. This approach also does not consider the node failure probabilities which may be crucial as the message queues at each node are of finite length and hence significant losses can occur at overlay nodes with high incoming traffic.

RON [DGA01] nodes also use active probing to infer quality of virtual links, and passive observation of the traffic, and propagate this information to other nodes. Each node uses a variety of performance metrics, and an application-specific path is selected on the basis of these metrics.

Both RON and TAROM do not take into account node failure probabilities. We propose a multiplicative reliability model, which considers both, the node and the link radiabilities when establishing reliable paths in an overlay network. Unlike TAROM, our model routes
events along the most reliable path - i.e., our model will not send notification along the primary path if there exists a secondary path with a higher value of reliability.

The hop-by-hop reliable communication [YA03] approach suggests that the message losses should be recovered on the overlay hop on which they occurred. This is in contrast with the TCP which does error recovery on an end-to-end basis. Since, the overlay link has a lower delay compared to an end-to-end connection that traverses multiple hops, the loss can be detected much faster. The approach involves buffers and processing in the intermediate overlay nodes. These nodes need to deploy a reliable protocol, and keep track of packets, acknowledgements, and congestion control in addition to their regular routing functionality. The main disadvantage of ensuring hop-by-hop reliability is the complexity of the protocol which makes it unsuitable to deploy in large overlay networks. The nodes keep states for each of the transmitted packets, thus shifting the burden of book-keeping from end hosts to each of the intermediate nodes. Secondly, the use of buffers for maintaining states leads to lower availability of the buffers for the application itself. Also, the packets have to be re-ordered at the last hop which causes additional overheads.

Hermes, reliability model has two aspects - the robustness of the middleware against failures and the reliability that is explicitly demanded by clients through a quality of service (QoS) specification. Hermes routing algorithms use built-in fault-tolerance features which enable event brokers to recover to a consistent system state after failure. However Hermes does not provide support for reliability specified by the client as a service guarantee. We focus on the second aspect of the reliability model of Hermes, i.e., providing reliability, as a QoS requirement specified by the client.

In order to determine reliable paths for routing notifications, we propose a reliability measurement model in Section 2.4. Our main focus is on developing a model for measuring the node reliability. We use techniques proposed in [DGA01] and [CT05] in order to determine values for link radiabilities. In Section 2.5 we propose an algorithm, which relies on the reliability model and finds a path in the broker network, with a reliability value greater than or equal to the threshold. We compare our algorithm, with a flooding algorithm and present the complexity analysis of our algorithm in Section 2.6. We present initial simulation results, in Section 2.7.

2.4 Reliability Model

An overlay network comprises of a set of interconnected logical nodes and links, over which events (messages) are delivered. We assume that every broker node, will have a reliability value associated with it. When delivering an event, the brokers will utilize the reliability values provided by the overlay network to determine optimal paths for transmission. The
2.4. Reliability Model

The reliability of an overlay network, is dependent on reliability of a node and reliability of a link. The reliability models discussed in [YA03], [CT05], and [DGA01], in Section 2.3, improve the reliability in terms of link reliability in the overlay network. In our model, we measure the node reliability and assume the reliability values for links, obtained from the link reliability measurement models stated above. In the Figure 2.2 shown above, we assume that there are a total of Bm Broker nodes in the network. We have depicted node B, and some of its neighboring nodes in the figure. We measure the node reliability for node B.

2.4.1 Assumptions

1. Every node represents a broker, in the event broker network, and a message represents an event. Each broker node, is connected to other broker nodes in the network, known as its neighbors. Events are transmitted across the broker network to the clients.

2. The events in our system, occur successively in time, such that the intervals between two successive events are independent. Hence, we say that the arrivals form a Poisson process, with exponential inter-arrival times.

3. Infinite buffer assumption is unrealistic in practise, so we assume that the size of the buffer is N, including the server (i.e. the event broker in this case).

4. Each server can be in state n ranging from 0 to N.

5. $\rho = \frac{\lambda}{\mu}$, s.t. $\lambda$ is the mean inter-arrival time. Event arrivals form a Poisson process $\lambda$. Equivalently the event interarrival times are exponentially distributed with mean
2.4. Reliability Model

1. Service times of events are exponential with mean \( \frac{1}{\lambda} \).

6. Decomposition of Poisson process is again a Poisson process.

7. Superposition of \( n \) independent Poisson processes is a Poisson process. (e.g. If there are two independent Poisson event arrival streams in a buffer, with average arrival rates \( \lambda_x \) and \( \lambda_y \) respectively, then the superposition result implies that \( \lambda_x + \lambda_y \) is also a Poisson process.)

We model the broker node as an M/M/1/N queue. Since the buffer size is finite, the probability that the buffer is full at an arbitrary point in time is given by the following equation

\[
P_n = \frac{(1-\rho)\rho^n}{1-\rho^{N+1}} \tag{2.6}
\]

For an infinite buffer assumption, the probability of being in state \( n \) is given by the following equation

\[
P_n = (1 - \rho)\rho^n \tag{2.7}
\]

\( P_n \) in equation 2.7 represents the probability of a node being in state \( n \) in an M/M/1 queue. The output of the M/M/1 system is again a Poisson process. And the infinite buffer model, is a very good approximation for a finite buffer system (even for moderate buffer sizes). Based on assumption 6 we infer from Figure 2.2, that, the output of the event broker is a Poisson process, which is decomposed into a set of Poisson processes. Similarly the input to a broker, may come from multiple independent Poisson processes. From assumption 7 it is clear, that the input to the event broker, will finally be a Poisson process.

From equation 2.6 we see that, \( P_n \) is the probability that the buffer is full at an arbitrary point in time. Since the arrivals are independent of buffer state, we have \( P_n = P_B \), where \( P_B \) is the blocking probability, i.e., probability that an arriving packet is turned away due to a full buffer. This packet will be dropped.

Hence the probability of reliable delivery offered by a broker node \( P_R \) is the probability that no packet is dropped by the broker node. This is given by following equation

\[
P_R = 1 - P_B \tag{2.8}
\]

Any change in the arrival rate \( \lambda \) will affect the reliability. There are multiple publishers and subscribers connected to a single broker (or server). And various factors such as rate of publication of events by a publisher, number of publishers/subscribers connected to a broker and the rate of arrival of in-transit messages, will affect the arrival rate. As per our assumption, these arrivals are Poisson arrivals with exponential inter-arrival times. The departure rate of one node, will directly affect the arrival rate of another node. Hence,
2.5. Our Approach

with the change in arrival rate $\lambda$, the blocking probability i.e. the reliability of the node will be affected, as given in equation 2.6.

Given a source and a destination, our reliability model, provides a method to determine reliability of a broker node. Any algorithm can rely on this model, to find a reliable path in an overlay network. The broker network is responsible for ensuring the reliability requirement specified by the subscriber. In order to compute reliability, we consider, the path of an event notification to be a **series system** [Tri], in which all components (broker nodes) are so interrelated that the entire system fails if any one of its components fail.

Consider a system with $n$ components, i.e. $n$ broker nodes. All these nodes lie in the path from the subscriber to the publisher for an event type. Let $\omega_i$ be the blocking threshold at node $i$, and $\phi_i$ be the link loss occurring at link $i$. The probability that the node $i$, will drop a packet (i.e. the blocking probability) is given by $P(\omega_i)$. Hence the probability that the node is reliable is $(1 - P(\omega_i))$. The probability that the link $i$, will drop a packet (i.e. the link loss) is given by $P(\phi_i)$. Hence the probability that the link is reliable is $(1 - P(\phi_i))$.

The reliability $R_{path}$ of the entire path is given by the following equation -

$$R_{path} = \prod_{i=1}^{n} [1 - P(\omega_i)] [1 - P(\phi_i)]$$  (2.9)

2.5 Our Approach

In this section, we propose an algorithm which leverages the Pastry [AR01] routing algorithm, based on the reliability measurement model proposed in Section 2.4. Given a threshold value for reliability, by the subscriber, our algorithm tries to determine the most reliable path$^1$, for delivery of an event notification. We modify the Pastry routing algorithm to determine a set of paths, along with their radiabilities and send an event notification only along paths which meet the reliability requirement of the subscriber. The Pruning Algorithm described in 2.5.1, is discussed at the level of a single event broker, and is applicable to every broker in the reliable paths being established.

2.5.1 The Pruning Algorithm

The Pruning algorithm can be divided into 3 major stages -

- We determine the primary path for event delivery. Primary path is the path along which the event notification travels by default using the Hermes middleware routing algorithm [Pie04]. However instead of sending a notification to the subscriber, we

$^1$Most reliable path is one which has a reliability value greater than or equal to the threshold specified by the subscriber
Algorithm 1 Pruning

Require: $R_p$, currentNode, DestId,
Ensure: The value of $R_p$.

1: Nbors ← GetNeighborhoodSet(currentNode)
2: for all $i$ such that $1 \leq i \leq Nbors$ do
3: if currentNode $\notin$ List(PartialPathnodes) then
4: $P(\omega) \leftarrow$ GetNodeBlocking()
5: $P(\phi) \leftarrow$ GetLinkLoss()
6: $R_p \leftarrow R_p \cdot [1 - P(\omega)][1 - P(\phi)]$
7: List(PartialPathnodes).add(currentNode)
8: CurrentLevelflooded ← Levelflooded + 1
9: end if
10: if Nbors$_i \notin$ List(PartialPathnodes) then
11: if CurrentLevelflooded <= $F_{level}$ then
12: SendMessage($R_p$, Nbors$_i$)
13: Pruning($R_p$, Nbors$_i$, DestId)
14: else
15: PastryRouting($R_p$, DestId)
16: List(PartialPathnodes) ← GetPartialPath()
17: end if
18: end if
19: end for
20: PastryRouting(List($R_p$), SourceId)
21: SendNotification(List($R_p$))

Figure 2.3: Path Establishment in the Pruning Algorithm

return the identifier of the destination node to the event publisher source, along with the reliability value of the path. This is done, since it is possible that the primary path, may not be meeting the reliability threshold specified by the subscriber.
2.5. Our Approach

- With this we ensure that the source is aware of the destination. Using this information, we apply the pruning algorithm to establish reliable paths between the source and destination. In step 1, we get the set of neighborhood nodes [AR01] for the current node, which is being flooded. In step 3 we check to see if the node being flooded already exists in the path that has been established so far. If the current node is in the list, then we do not add it to the list once again. This prevents cycles. In step 4, we obtain the node blocking probability for a node which is occurring on the path for the first time. We calculate partial path reliability\(^2\) as given in the formula in step 5. This node is then added to partial path (step 6). In step 7 we increment the level of flooding. We then flood the neighbor nodes of the current node. In order to prevent cycles, we check if the neighbor node exists in the path established so far as shown in step 9. If not, then if we still haven't flooded to the expected level (step 10), then we recursively call the Pruning algorithm (step 12). If the flooding is complete, then the message is routed to the destination using Pastry Routing (step 14). Once the path is established, the GetPartialPath routine, returns the partial path, upto the point from where we need to resume flooding (step 15). Once all the paths are established, the set of reliable paths, along with their reliability values is sent back to the publisher source node, using pastry routing as shown in step 19.

- Once the reliable paths have been determined, the source node, takes a decision and sends notification along a route, whose reliability is in conformance with the reliability requirement of the subscriber. The SendNotification routine in step 20 does the following -

  - **step 1:** Find a set of paths, having reliability values, which are greater than or equal to the threshold reliability requirement of by subscriber. Send the notification along the path having maximum reliability value, from this set.

  - **step 2:** If no single path exists which meets the reliability requirement of the subscriber, then we combine paths in the system, such that the reliability of the combined paths, is greater than or equal to the threshold. Notification is then sent in parallel, along all the combined paths. We use the greedy approach when selecting paths, i.e., we select paths in descending order of reliability values. If \( R \) is the reliability of path \( i \), and there are \( n \) paths which need to be combined in order to attain a reliability value greater than threshold, then the combined reliability of the parallel paths \( R_c \) is given by equation 2.10.

\[
R_c = 1 - \left[ \prod_{i=1}^{n} (1 - R_i) \right]^n \tag{2.10}
\]

\(^2\)Partial Path Reliability, is the reliability of the path being established, from the source node till the current node in the route.
Since Pruning restricts the level of flooding in the system, we compare our approach with the Flooding approach, which we discuss in the next section.

### 2.5.2 The Flooding Algorithm

**Algorithm 2 Flooding**

**Require:** $R_p$, currentNode, DestId

**Ensure:** The value of $R_p$.

1. $ Nbors \leftarrow \text{GetNeighborhoodSet}(\text{currentNode})$
2. for all $i$ such that $1 \leq i \leq Nbors$ do
   3. if $\text{currentNode} \notin \text{List(PartialPath_nodes)}$ then
      4. $P(\omega) \leftarrow \text{GetNodeBlocking}()$
      5. $P(\phi) \leftarrow \text{GetLinkLoss}()$
      6. $R_p \leftarrow R_p \times [1 - P(\omega)][1 - P(\phi)]$
      7. $\text{List(PartialPath_nodes).add(currentNode)}$
   8. end if
   9. if $Nbors_i \notin \text{List(PartialPath_nodes)}$ then
      10. $\text{SendMessage}(R_p, Nbors_i)$
      11. $\text{Flooding}(R_p, Nbors_i, \text{DestId})$
      12. $\text{List(PartialPath_nodes) \leftarrow GetPartialPath()}$
   13. end if
14. end for
15. if $\text{DestId} \in (\text{LeafSet_currentNode})$ then
   16. $\tau \leftarrow \text{GetNodeBlocking}(\text{DestId})$
   17. $R_p \leftarrow R_p \times (1 - \tau)$
   18. $\text{List(PartialPath_nodes).add(DestId)}$
   19. $\text{SendMessage}(R_p, \text{DestId})$
20. end if
21. $\text{PastryRouting(List}(R_p), \text{SourceId})$
22. $\text{SendNotification(List}(R_p))$

In algorithm 2, we send messages to all nodes occurring in the neighborhood set [AR01] of the current node. This process is continued till we reach the destination. The reliability calculation and event notification is similar to Pruning as discussed in Section 2.5.1. The flooding approach differs from Pruning, since it does not restrict flooding to a particular level. Every node is flooded, till the destination node, occurs in the leaf set [AR01] of a node. The procedure for calculating reliability of the path, and sending notification, is similar to that discussed in the pruning algorithm in 1. Flooding Algorithm described in 2.5.2, is discussed at the level of a single event broker, applicable to every broker in the reliable paths being established.
2.6 Complexity Analysis

In this section, we determine the message and time complexity for the Pruning algorithm, and discuss its implications. The total number of messages $T(M)$, that are generated in the system, when the pruning algorithm is executed are given by the following expression.

$$T(M) = 2(\log_2 N) + \sum_{i=1}^{k} n^i + (\log_2 N)n^k + (\log_2 N)$$(2.11)

- Initially the Primary Path is determined using the Pastry routing algorithm. Assuming that the network consists of $N$ nodes, Pastry can route to a numerically closest node with a given key in less than $\lceil \log_2 N \rceil$ steps ($b$ is a configuration parameter with typical value 4). Hence the number of messages generated is $2(\log_2 N)$ which is similar to the bound offered by Pastry Routing. The messages generated are twice $(\log_2 N)$, because, once the primary path is established, the destination node collects the reliability value of all the paths and routes them back to the source, using Pastry routing.

- In the Pruning approach we flood messages to the neighborhood nodes, upto a specific level, and then follow the next hop routing algorithm of Pastry till the destination. If $n$ is the number of neighborhood nodes for each broker in the network, and $k$ is the level to which we flood, then $\sum_{i=1}^{k} n^i$ indicates the total number of messages introduced in the network, when flooding. Once the flooding is complete, the Pruning approach, uses the Pastry routing technique to route messages to the destination. The maximum number of messages introduced in the system at this point are $(\log_2 N)n^k$. Finally, the destination routes the reliability values back to the source in a single message using Pastry routing. This introduces additionally $\lceil \log_2 N \rceil$ messages in the network. Upto this point, we have the total number of messages introduced in the network for establishing a path.

- Once the paths are established, the notification is sent along the paths, which meet the reliability requirement of the subscriber. In case a single path is not able to meet the reliability threshold, then we sent notification along multiple paths. The maximum number of paths along which the event notification can be sent are $(\log_2 N)n^k$.

From the discussion above, it can be inferred that, the message complexity of the Pruning approach is of $O((\log_2 N)n^k)$. We find that the message complexity is influenced by the number of neighbors for each node, and the level of flooding in the network. So we see that, as the levels of flooding in the Pruning algorithm increase, the Pruning algorithm, will have a message complexity similar to that introduced in the network by Flooding the
A large value of $n$ (number of neighborhood nodes), implies that a smaller value of $k$ (level of flooding) would suffice in finding reliable paths, and this approach can be used in dense networks where each node has a large number of neighbors (i.e., nodes which are closest to the present node according to the proximity metric). So, if we consider a complete graph, then, we require only a single level of flooding, since each node has $N-1$ neighbors.

In order to determine the time complexity of the Pruning approach, we assume an upper bound of $l$ for each task of each process, and an upper bound of $d$, on the time to deliver the oldest message in each channel queue. The time complexity of the algorithm in order to establish reliable paths for an event notification is $O((\log_2 N) (l + d))$. The time complexity for sending an event notification to the subscriber is also bounded by the same expression. This does not change, even in the case of sending an event notification along multiple paths, since, the event notifications are routed in parallel.

### 2.7 Experimental Results

In this section, we present experimental results obtained, with an implementation of the Pruning routing algorithm. The algorithms are implemented in the Hermes middleware simulator. All experiments are performed on an overlay network of 25 event brokers on an underlying transit-stub topology of 100 nodes. We assume a static reliable overlay, with nodes and links having reliability values greater than 0.5. The reliable notifications are sent to a single subscriber subscribing for a particular event type in the Hermes middleware. The reliability attained by the subscriber, is evaluated against the reliability the subscriber originally requested in Section 2.7.1. In Section 2.7.2 we take a look at how the Pruning algorithm tends to Flooding, with increase in the levels of flooding. Finally, the message complexity of the Pruning algorithm is compared with the Flooding algorithm in Section 2.7.3.

#### 2.7.1 Actual Reliability attained at subscriber

This experiment shows the reliability value specified by the subscriber (Expected Values) and the reliability the broker network is able to provide. We vary the reliability from 0.1 to 0.8 for the expected value. The Hermes simulator, does not take into account, the reliability requirement of the subscriber, hence the reliability of the path established by Hermes remains constant. For a given requirement of reliability, Pruning chooses a path that has maximum reliability amongst those paths which have a reliability value that is greater than or equal to the threshold. Sometimes we see that Pruning provides a
2.7. Experimental Results

2.7.2 Pruning Performance

In this experiment, we increase the flooding levels for the Pruning algorithm and determine the number of messages generated at each level. The graph in Figure 2.5 depicts the percentage of messages generated by the Pruning approach in comparison to Flooding. We see that as the flooding level in the Pruning algorithm increases, the message complexity of the Pruning approach tends to the Flooding approach. So the Pruning algorithm is efficient only if it finds a reliable path, given an average flooding level.
2.7.3 Message Complexity of Pruning

![Graph](image_url)

Figure 2.6: Comparison of message complexity for Pruning and Flooding

This experiment shows the number of messages generated for the Pruning and Flooding approach, for expected values of reliability. Since we have seen in Section 2.7.2, that Pruning, tends to Flooding with increase in level of flooding, we take an average value for level of flooding, and then determine the message complexity. It is seen from Figure 2.6, that Pruning manages to find a reliable path in the overlay network each time - however with a much lower message complexity overhead as compared to flooding. The slight increase in the message complexity of the Pruning approach owing to the notification being sent along parallel paths, is also insignificant as compared to the overall message complexity of Flooding.

2.8 Analysis

In this section, we discuss some of our observations based on the experiments conducted in Section 2.7.

- Given a threshold value of reliability by the subscriber, the Pruning algorithm finds a reliable path which has a reliability value greater than or equal to the threshold. As seen in Figure 2.4, flooding is able to establish a path with a reliability value of 0.3. This is the maximum reliability, which can be attained by a single path in the overlay network. Hence it is observed that, as we increase the reliability threshold, flooding approach combines paths in order to meet the reliability requirement of the subscriber. The Pruning algorithm, also follows a similar technique, in which it combines paths in order to achieve the threshold reliability. Combining paths, often results in a higher reliability than the specified threshold, as seen in the figure. We observe that the path established by Hermes, achieves a reliability of 0.3, which remains constant throughout. We see that with increase in reliability
threshold Pruning and Flooding always manage to attain a higher reliability than the path established by Hermes. In the subsequent graphs we compare the message complexity of Pruning with Flooding and the performance of the Pruning algorithm.

- In the Pruning approach, we have seen that, as we increase the levels of flooding the number of messages introduced in the system also increase. We compare the percentage increase in the number of messages in Figure 2.7.2. We see that pruning performance degrades in terms of number of messages with increasing levels of flooding. If we assume that the number of messages generated by flooding to be equal to a 100%, then the graph in Figure 2.7.2 depicts the percentage of messages for the Pruning algorithm in comparison to Flooding, with increase in the flooding level. We conclude from this experiment, that Pruning algorithm will be efficient only if we find a reliable path with an average level of flooding.

- In the next experiment, with an average level of flooding, we compare the message complexity of Pruning and Flooding for increasing values of reliability threshold. We observe that even with an average level of flooding, we are able to determine a reliable path using the Pruning algorithm. However the flooding algorithm also determines reliable paths, albeit with a very high message complexity. Hence we conclude that the Pruning algorithm determines a reliable path as compared to Hermes and with a very low message complexity in comparison to the Flooding approach.

2.9 Current Status and Future Work

This chapter presents a reliability routing model for event notifications and an algorithm which leverages the Pastry routing algorithm, to determine a path having reliability, greater than or equal to the threshold, between a pair of overlay broker nodes. The algorithm has a higher message complexity in comparison to the Hermes routing algorithm, however unlike Hermes, it ensures an event notification guarantee to subscribers. We have implemented this algorithm in the Hermes middleware simulator, and compared it with the routing algorithm of Hermes and the Flooding approach. Our algorithm, determines a more reliable path as compare to Hermes, and one with a low message complexity as compared to the Flooding approach. As part of the future work, we plan to incorporate the reliability model, and test the algorithm, in a system with dynamically changing values for reliability of nodes and links.
Bibliography


