# Reliable Routing of Event Notifications over P2P Overlay Routing Substrate in Event Based Middleware

Shruti P. Mahambre and Umesh Bellur
Indian Institute of Technology Bombay,
Mumbai, India
{shruti,umesh}@it.iitb.ac.in

## Abstract

*Event Broker Networks (EBN) 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. However quality of service based routing schemes are rare and our work addresses this gap. Specifically we look into the prospect of routing events based on reliability requirements of subscribers for an event type being delivered via the EBN. In this paper, we formally define reliability and propose a multiplicative model which calculates reliability of the P2P overlay routing substrate 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 routing substrate, 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. Our algorithm also determines a path with higher reliability than the path established by Hermes [5]. We present initial simulation results, using the Hermes middleware simulator.*

## 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 routing substrate over the publish-subscribe(pub-sub) [10] brokers. Overlay routing substrate, provides decentralized, application-level routing, that helps disseminate events from publishers to subscribers. 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. 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 [10], makes this model highly suitable for large scale distributed applications. In an event based middleware, information provided by the producers is distributed in a timely manner to the consumers.

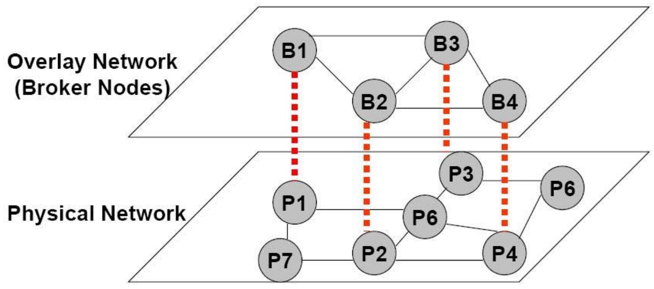

**Figure 1. Layered Broker Network**

Pub-Sub systems are deployed on overlay networks. An overlay network [6] 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, i.e., a single hop in the overlay network, could result in multiple hops in the physical network. Overlay nodes run applications in a distributed manner providing middleware functionalities. Most event based middleware are implemented using the two layers, as shown in Figure 1. The bottom layer represents the physical network comprising of routers and links. The overlay network layer represents the P2P overlay routing substrate which provides an abstraction of the *distributed hash table*[1] [7]. A node in this P2P overlay routing substrate, may be mapped to an event broker, which forms a part of the event dissemination tree, used for routing events.

The main contribution of this paper, is a multiplicative model that is proposed to compute the reliability of a path in the overlay network, and to route event notifications to client along a path meeting client specified reliability requirements. The paper is organized as follows. In Section 2, we provide a formal definition of reliability over a P2P routing substrate in a publish-subscribe domain. We compare our work with existing work in this field in Section 3. Section 4 provides a formal definition of the problem. In order to determine reliable paths for routing event notifications, we propose a multiplicative model for reliability measurement in Section 5. In Section 6 we propose a Pruning algorithm, which relies on the reliability model and finds a reliable path on the overlay routing substrate. We compare our algorithm, with a flooding algorithm in Section 6.2 and present the complexity analysis of our algorithm in Section 7. We present initial simulation results, in Section 8 that verify our claims and conclude with a discussion regarding future work in Section 9.

## 2. Reliability in Publish Subscribe

Quality of Service [3] or QoS refers to the ability of a system to provide services, such that the users expectations for all non-functional criteria, are satisfied. 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 this paper our focus is on computing reliability of paths, and setting up paths in the overlay network, that meet the reliability requirement of the subscriber. We define the delivery *reliability* of an event, in the context of a pub-sub system, as the ratio of the number of notifications delivered to the subscriber of a particular event *type*, against the number of publications of the event *type*. In a type based scheme, event subscriptions are filtered based on their types. The notion of an event *kind*, is matched to an event *type* [2]. A type based model facilitates type safety at compile time and leads to closer integration of the language and the middleware [10]. We run our simulations on the Hermes middleware, which provides type-based [5] and type-and-attribute-based [5] routing of events. We now define *reliability*, in the context of a publisher-subscriber system.

[1] DHT is a building block for p2p applications, that enables a group of distributed hosts to manage the mapping from keys to data values representing host nodes in the p2p overlay

**Definition of Reliability in Publish-Subscribe**

We define reliability in publish-subscribe domain as follows.

**Notation -** $n(e_\tau)$ is the number of notifications sent for an event type $\tau$. $p(e_\tau)$ is the number of publications of an event of type $\tau$. $\tau_{max}$ is the total number of event types in the system. $s_{max}$ is the maximum number of subscribers in the system. $e^s_\tau$ represents a subscriber *s* which has subscribed for an event $e$ of type $\tau$. $\mathbf{R}$ is the Reliability value ranging from [0..1] The reliability $\mathbf{R}$ attained at subscriber *s* is given by 1.

$$\mathbf{R}[e^s_\tau] = \frac{n(e_\tau)}{p(e_\tau)} \qquad (1)$$

Using equation 1, reliability is measured at different levels in an event based system as follows:

### 2.1. Per Subscriber Per Event Type

The delivery reliability for the notification of an event type, is measured for a single subscriber subscribing for a single event type. A subscriber may subscribe for multiple event types. We can measure the reliability that the subscriber attains for each event type that it has subscribed for, as shown in equation 1.

### 2.2. Per Subscriber All Event Types

A subscriber can measure the total reliability it has attained, by calculating the reliability it attains across all the event types for which it has subscribed. The reliability attained by a subscriber *s* for all the event types, it has subscribed for, is given by equation 2

$$\mathbf{R}[e^s] = \frac{\sum_{\tau=1}^{\tau_{max}} \mathbf{R}[e^s_\tau]}{\tau_{max}} \qquad (2)$$

### 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 $\tau$, across all the subscribers is given by 3

$$\mathbf{R}[e_\tau] = \frac{\sum_{s=1}^{s_{max}} \mathbf{R}[e_\tau^s]}{s_{max}} \tag{3}$$

### 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 equations 2 and 3 the reliability of the entire system is as given by equation 4

$$\mathbf{R}[e] = \frac{\sum_{s=1}^{s_{max}} [(\sum_{\tau=1}^{\tau_{max}} \mathbf{R}[e_\tau^s])/(\tau_{max})]}{s_{max}} \tag{4}$$

## 3. Related Work

TAROM [9] 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 secondary paths to (i.e. paths between pairs of two non-interacting sources and destinations) two independent primary paths may interfere. If both the primary paths fail simultaneously, the secondary 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 [1] nodes also use active probing to infer quality of virtual links, and passive observation of the traffic, and then 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 reliability model, that considers both, the node and the link reliabilities when establishing reliable paths in an overlay network. Unlike TAROM, we establish secondary paths in order to achieve higher reliability in the network, by increasing redundancy, and not merely for establishing backup paths. If the primary path (i.e. the path established by the middleware) also meets the reliability requirements of the subscriber, then we send notification along the primary path to the subscriber.

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 uses Pastry [8] which is a routing substrate for wide area P2P applications. Pastry performs object location, and application level routing in an overlay network. Each node in Pastry has a unique numeric node identifier. Assuming a network of $N$ nodes, Pastry can route a message in $logN$ steps to any destination. Hermes routing algorithms are built over Pastry and they 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.

## 4. Problem Statement

- $p_i$ is a publisher *i*, s.t. $p_i \in \mathbf{P}$, where $\mathbf{P}$ is the set of publishers and $s_i$ is a subscriber *i*, s.t. $s_i \in \mathbf{S}$, where $\mathbf{S}$ is the set of subscribers.
- $\lambda$ is the reliability threshold requested by the subscriber for an event it has subscribed for.
- $\tau$ is an event type
- $p_i(\tau)$ is a publisher *i*, s.t. $p_i \in \mathbf{P}$, which has published an event of type $\tau$ and $s_i(\tau)$ is a subscriber *i*, s.t. $s_i \in \mathbf{S}$, which has subscribed for an event of type $\tau$
- $n_i$ is a node *i*, s.t. $n_i \in \mathbf{N}, \forall i = 1..\mathbf{N}$, where $\mathbf{N}$ is the set of brokers in the overlay network and $l_{i,j}$ is a link between nodes $n_i$ and $n_j$, s.t. $l_{i,j} \in \mathbf{L}$, where $\mathbf{L}$ is the set of links in the overlay network.
- $\Theta$ represents the path between nodes *i* and *j*. This path comprises of intermediate nodes and links, i.e., $\Theta = \{n_i\ l_{i,x}\ n_x\ l_{x,y}\ ......\ n_w\ l_{w,j}\ n_j\}$ i.e., $\Theta$ is a set of nodes and links on a reliable path established between a publisher and subscriber for an event type.
- $\mathbf{R}$ $[\Theta]$ is the reliability of the path

  Given a single subscriber $s_i(\tau)$, single publisher $p_i(\tau)$ and $\lambda$, we determine a path $\Theta$ s.t.

$$\mathbf{R}[\Theta] \geq \lambda \tag{5}$$

## 5. Multiplicative Model for Reliability

In order to compute reliability of a path, we consider, the path of an event notification to be a *series system* [11], 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. 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 $l_{i,j}$, 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**[$\Theta$] of the entire path is given by the following equation

$$\mathbf{R}[\Theta] = \Pi_{i=1}^{\mathbf{N}}[1 - P(\omega_i)]\Pi_{i=1}^{\mathbf{N-1}}[1 - P(\phi_i)] \qquad (6)$$

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 the next section we discuss the proposed algorithm in detail.

## 6. Our Approach

In this section, we propose an algorithm which leverages the Pastry [8] routing algorithm, based on the reliability measurement model proposed in Section 5. Given a threshold value for reliability, by the subscriber, our algorithm tries to determine the *most reliable path*[2], for delivery of an event notification, with minimum possible message complexity. We can determine a highly reliable path in the overlay network, by flooding the entire network. This results in a high message complexity. The Pastry overlay substrate has the property that it is able to route a message to a given destination in maximum $logN$ hops, where N is the number of nodes in the overlay network. We use this property of the Pastry overlay routing substrate, in order to control the message complexity introduced by the flooding approach. We modify the Pastry routing algorithm to determine a set of paths, along with their reliabilities and send an event notification only along paths which meet the reliability requirement of the subscriber. The Pruning Algorithm described in Section 6.1, is discussed at the level of a single event broker, and is applicable to every broker in the reliable paths being established.

[2]Most reliable path is one which has a reliability value greater than or equal to the threshold specified by the subscriber

### 6.1 The Pruning Algorithm

**Notation** - $R_p$ represents the reliability of the partial path, or the path upto which the reliability has been calculated so far. *List*[*PartialPath*$_{nodes}$] represents a list of nodes occurring in the partial path established so far. We can *add*() and *remove*() nodes from this list. The *PartialPath*() operation, returns the list containing the *PartialPath*$_{nodes}$. The *PastryRouting*() operation refers to the routing algorithm of the Pastry overlay substrate.

**Algorithm 1** PRUNING

```
Require: $R_p, currentNode, DestId,$
Ensure: The value of R$_p$
 1: $Nbors \leftarrow GetNeighboorhoodSet(currentNode)$
 2: for all $i$ such that $1 \le i \le Nbors$ do
 3:   if $currentNode_i \notin List[PartialPath_{nodes}]$ then
 4:     $P(\omega_i) \leftarrow GetNodeBlocking()$
 5:     $P(\phi_i) \leftarrow GetLinkLoss()$
 6:     $R_p \leftarrow R_p * [1 - P(\omega_i)][1 - P(\phi_i)]$
 7:     $List[PartialPath_{nodes}].add(currentNode_i)$
 8:     $CurrentLevel_{flooded} \leftarrow Level_{flooded} + 1$
 9:   end if
10:   if $Nbors_i \notin List[PartialPath_{nodes}]$ 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:       $List[PartialPath_{nodes}] \leftarrow PartialPath()$
17:     end if
18:   end if
19: end for
20: $PastryRouting(List(R_p), SourceId)$
21: $SendNotification(List(R_p))$
```

The Pruning algorithm is divided into three stages. Initially the Hermes middleware determines a path for sending an event notification. This path does not taken into consideration the reliability requirement of the subscriber. Once this path is established, the next stage of the Pruning algorithm, involves partial flooding, followed by Pastry routing. We will now describe algorithm in 1 in detail.

- **stage 1:-** 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 [5]. However instead of sending a notification to the subscriber, we 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.

- **stage 2:-** With this we ensure that the source is aware of the destination. We now proceed to the second stage of the pruning algorithm, in which we establish multiple paths between the source and destination. In step 1 of algorithm 1, we get the set of *neighborhood nodes* [8] for the current node, which is being flooded and perform *partial flooding* (i.e., flooding upto a particular level only and not the entire network). *Flooding enables the establishment of multiple paths, and increases the chances of finding a path which has a reliability value meeting the threshold reliability requirements of the subscriber.* 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, and in step 5, we obtain the link loss rate. We calculate *partial path reliability*[3] as given in the formula in step 6. This node now becomes a part of the partial path established so far(step 7). In step 8 we increment the level of flooding. We now continue the flooding process, i.e., we find nodes in the neighborhood node set of the current node, flood each neighborhood node, if it is not already a part of the partial path established so far (steps 10, 11). *If a neighborhood node is not in the partial path established so far, it implies that the neighborhood node has not yet been flooded.* We continue flooding, till the level of flooding that has been specified, and recursively call the Pruning algorithm (step 14).
  With this, we complete the partial flooding, which is necessary to introduce redundancy in the system in order to meet the reliability requirements of the subscriber.

- **stage 3:-** Once the flooding is complete, the message is then routed to the destination using Pastry Routing(step 15). The *PartialPath* 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 20. 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 21 does the following

[3]Partial Path Reliability, is the reliability of the path being established, from the source node till the current node in the route

**stage 3a:-** Find a set of paths, having reliability values, which are greater than or equal to the *threshold* reliability requirement specified by the subscriber. Send the notification along the path having maximum reliability value, from this set.

**stage 3b:-** 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. Two cases arise out of this scenario.

**Case 1: Disjoint Paths** - In this case all the paths being combined are disjoint, i.e., they do not have any common nodes and links. If *R* is the reliability of path $\Theta_h$, and there are *H* 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 7.

$$\mathbf{R}_c = 1 - [\Pi_{h=1}^{H}(1 - \mathbf{R}[\Theta_h])] \tag{7}$$

**Case 2: Intersecting Paths** - Here the paths being combined are non-disjoint, i.e., subsets of the paths, intersect with each other. We take a union of all the intersecting paths in order calculate the joint reliability. Given *R* is the reliability of path $\Theta_i$, and there are *H* paths which need to be combined in order to attain a reliability value greater than threshold, we obtain a union of all the paths, and calculate the reliability as shown in equation 8.

$$\begin{aligned}\mathbf{R}_c &= \sum_i \mathbf{R}[\Theta_i] - \sum_i \sum_{j>i} \mathbf{R}[\Theta_i \cap \Theta_j] \\ &+ \sum_i \sum_{j>i} \sum_{k>j} \mathbf{R}[\Theta_i \cap \Theta_j \cap \Theta_k] - .... \\ &... + (-1)^{H+1}\mathbf{R}[\Theta_1 \cap \Theta_2 \cap \Theta_3 .... \cap \Theta_H]\end{aligned} \tag{8}$$

We use an incremental approach when calculating multi-path reliability, i.e., we keep on adding one path at a time to the existing set of paths, till we attain the reliability specified by the subscriber. At each step, we store reliabilities of the paths which have been combined so far. Therefore, when we add a new path, we only need to calculate the joint reliability of the new path with the existing ones as shown in equation 8. 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.

### 6.2 The Flooding Algorithm

In the flooding algorithm, we send messages to all nodes occurring in the *neighborhood set* [8] of the current node. This process is continued till we reach the destination. 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* [8] of a node. The procedure for calculating reliability of the path, and sending notification, is similar to that discussed in the Pruning algorithm in Section 6.1. The flooding algorithm has been discussed in detail in [4].

## 7 Complexity Analysis of Pruning Algorithm

In this section, we determine the message complexity for the Pruning algorithm, and discuss its implications.

- The Hermes middleware determines a path for notification of the event, using the Pastry routing algorithm, which we call as the *Primary Path*. The messages generated are twice $(logN)$,(assuming that the network consists of *N* nodes) because, once the primary path is established, the destination node collects the reliability value of the path and routes it back to the source, using Pastry routing.

- 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. And for each of the paths established when flooding, we use Pastry routing to route message to destination. Therefore, the total number of messages introduced in the system currently are $(logN)n^k$. An additional $(logN)$ messages are introduced, to route the reliability values back to the source using Pastry Routing algorithm.

- Once the paths are established, the notification is sent along the paths, which meet the reliability requirement of the subscriber. The maximum number of paths along which the event notification can be sent are $(logN)n^k$ (in case of multipath routing).

The *total number of messages* **M** generated in the system, when establishing routes using the pruning algorithm, is given by the following expression.

$$\mathbf{M} = 2(logN) + \sum_{i=1}^{k} n^i + (logN)n^k + (logN)$$

We can infer that, the message complexity of the Pruning approach is of $\mathbf{O}((logN)n^k)$.

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 entire network. We verify our claims with simulation results presented in the next section.

## 8 Simulation Results

In this section, we present initial simulation results, obtained with an implementation of the Pruning routing algorithm. The parameters for our simulations are as shown in the table. All experiments were performed using an underlying transit-stub topology [5] with 5 autonomous systems, each having 20 nodes. This represents the physical network as shown in Figure 1. We assumed a static overlay network, and generated random values for reliabilities of nodes and links. The results of our simulations have been averaged across the values obtained for algorithm runs for 10 different publishers distributed across the overlay routing substrate for an event type. The event notification was sent to a single subscriber, subscribing for an event type. In our experiments, the main focus has been on determining the reliability of the path attained by the Pruning algorithm, and the number of messages generated when achieving this reliability as compared to flooding the entire network. The

| **Simulation Parameters** | |
|---|---|
| Transit Stub Topology | 100 nodes |
| Number of Event Brokers | 25 |
| Neighborhood Set Size | 4 |
| Leaf Set Size | 3 |
| Level of Flooding | 2 |
| Number of Publishers | 10 |
| Number of Subscribers | 1 |
| Number of Event Types | 1 |
| Reliability | Value between 0 and 1 |

reliability attained by the subscriber, was evaluated against the reliability the subscriber originally requested in Section 8.1. We then compare the message complexity of Pruning and Flooding to determine a reliable path in Section 8.2.In Section 8.3 we take a look at how the Pruning algorithm tends to Flooding, with increase in the levels of flooding. Finally, in Section 8.4, we study the influence of the connectivity of the overlay routing substrate, on the message complexity of the Pruning algorithm.

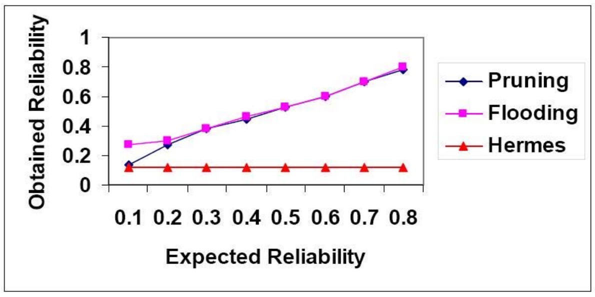

**Figure 2. Actual v/s Expected Reliability for a Subscriber**

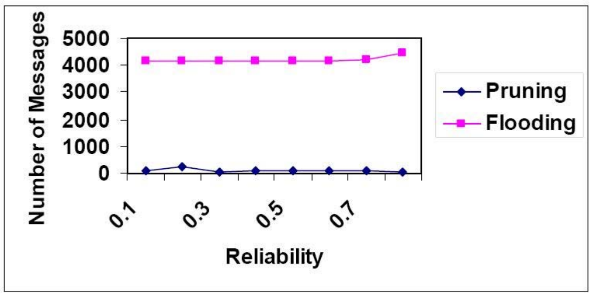

**Figure 3. Comparison of Message Complexity for Pruning and Flooding**

### 8.1 Actual Reliability Attained at Subscriber

In this experiment (Figure 2), we plot the reliability value specified by the subscriber (Expected Reliability) versus the reliability that our algorithm is able to attain (Obtained Reliability). As expected, Hermes, 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 reliability threshold, Pruning chooses a path that has maximum reliability amongst those paths which have a reliability value greater than or equal to the threshold. It is observed that, as we increase the reliability threshold, both Pruning and Flooding combine paths in order to meet the reliability requirement of the subscriber. Combining paths, sometimes results in a higher reliability than the specified threshold, as seen in the graph. Our experiment shows that with an increase in the reliability threshold, Pruning and Flooding always manage to attain a path with higher reliability than the path established by Hermes and that the Pruning algorithm manages to achieve the threshold reliability even with a flooding level of 2. We now compare the number of messages generated by Pruning and Flooding when determining reliable paths.

### 8.2 Message Complexity of Pruning and Flooding

Here we compare Pruning and Flooding in terms of their message complexities for determining reliable paths, given a reliability threshold requirement by the subscriber. We infer from this experiment (Figure 3) that, although both Pruning and Flooding attain the reliability requested by the subscriber, the Pruning algorithm is far more efficient, since it does so with a much lower message complexity as compared to the Flooding algorithm. In the next section, we evaluate the performance of the Pruning algorithm in terms of message complexity with increasing levels of flooding.

### 8.3 Pruning Performance

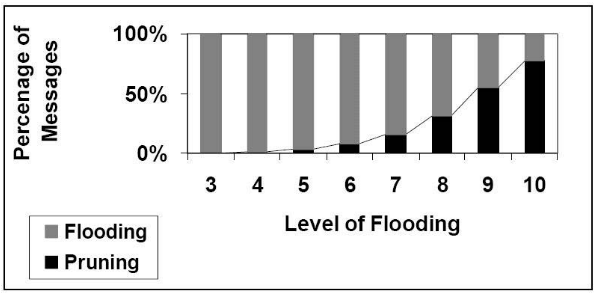

**Figure 4. Percentage of Messages Generated by Pruning, in Comparison to Flooding**

In this experiment, for a given value of reliability for a subscriber, we increase the flooding levels for the Pruning algorithm and determine the number of messages generated at each level. The graph in Figure 4 depicts the percentage increase in the number of messages of pruning in terms of flooding. It is observed that pruning performance degrades with increasing levels of flooding. We conclude from this experiment, that the Pruning algorithm is efficient only if we find a reliable path with an average level of flooding.

### 8.4 Message Complexity of Pruning with respect to Connectivity of the Overlay Nodes

In this experiment (Figure 5), we study how the Pruning algorithm is affected by the connectivity of nodes in the overlay network. We run our experiments, for an overlay network, with nodes having neighborhood set sizes 2, 3 and 4. We assume a fixed value for the reliability requirement specified by the subscriber. It is observed that with a high level of flooding, a densely connected overlay network (n=4), has a much higher message complexity, as compared

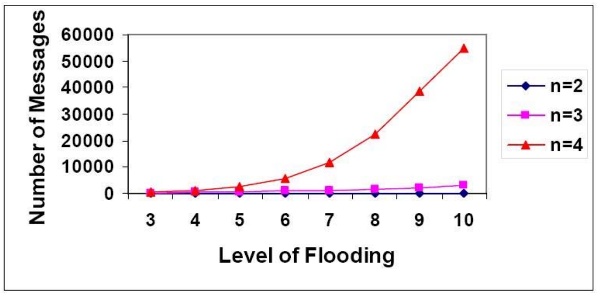

**Figure 5. Message Complexity of Pruning with Varying Sizes of Neighborhood Sets**

to a sparsely connected overlay (n=2), in order to attain the threshold reliability specified by the subscriber. The message complexity increases exponentially for a densely connected overlay, with increase in level of flooding. The Pruning algorithm, shows optimal performance for all levels of flooding, when the network is sparsely connected.

## 9 Conclusion and Future Work

In this paper, we have presented a multiplicative model for determining reliability of paths in a P2P overlay routing substrate, in order to route event notifications. We have also proposed 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. We have tested this algorithm using the Hermes middleware simulator. The algorithm has a higher message complexity in comparison to the Hermes routing algorithm, however unlike Hermes, it ensures a reliable event notification guarantee to subscribers. We have compared our routing algorithm with a Flooding approach. The simulation results show that our algorithm, determines a more reliable path as compared to Hermes, and with a lower message complexity as compared to the Flooding approach. As part of the future work, we plan to test the algorithm for multiple subscriptions and event types and for different topologies of the P2P routing substrate. We also plan to study the effect of hierarchical and compositional relationships between the event types on reliable event notification. As discussed in Section 5, we consider the reliability of both nodes and links, when calculating the reliability of a path. We are currently building an analytical model to measure the reliability of a node on an overlay routing substrate. In our model we will measure the node reliability, and assume the reliability values for links, obtained from the link reliability measurement models in [9] and [1]. We plan to incorporate this model to measure reliability of the nodes in an overlay substrate, and test the algorithm, in a system with dynamically changing values for reliability of nodes and links.

## Acknowledgments

We thank Mahak Patidar for the insightful discussions during the initial phase of this work and Punit Rathod for the very helpful comments on the paper draft.

## References

[1] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. The case for resilient overlay networks. In *Proceedings of Eighth workshop on Hot Topics in Operating Systems (HotOS 2001)*, pages 152–157. IEEE Computer Society, 2001.

[2] P. T. Eugster, R. Guerraoui, and C. H. Damm. On objects and events. In *Proceedings of the 16th ACM SIGPLAN conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001)*, pages 254–269. ACM Press, 2001.

[3] C. Irvine and T. Levin. Quality of security service. In *Proceedings of the 2000 Workshop on New Security Paradigms*, pages 91–99, Ballycotton, County Cork, Ireland, 2001. ACM Press.

[4] S. P. Mahambre and U. Bellur. QoS in event based middleware. *Technical Report - No:28, KReSIT, IIT Bombay*, 2006.

[5] P. Pietzuch. *Hermes - A scalable event-based middleware*. PhD thesis, Queens College, University of Cambridge, UK, 2004.

[6] P. Pietzuch and J. Bacon. Peer-to-Peer overlay broker networks in an event based middleware. In *Proceedings of 2nd International Conference on Distributed Event Based Systems (ICDCSW 2003)*, pages 1–8. IEEE Computer Society, 2003.

[7] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM 2005)*, pages 73–84. ACM Press, Aug. 2005.

[8] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218/2001, 2001.

[9] C. Tang and P. K. McKinley. Improving multipath reliability in topology-aware overlay networks. In *Proceedings of 25th International Conference on Distributed Computing Systems Workshops (ICDCSW 2005)*, pages 82–88. IEEE Computer Society, 2005.

[10] P. Th.Eugster, P. Felber, R. Guerraoui, and A. M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys(CSUR)*, 35(2):114–131, 2003.

[11] K. S. Trivedi. *Probability and statistics with reliability queuing and computer science applications*. Prentice-Hall of India Private Limited, New Delhi, 2004. ISBN:81-203-0508.