ABSTRACT
Event broker networks are used to interconnect publishers and subscribers in an event-based distributed computing system. Event broker networks are overlay networks formed over the underlying physical network (underlay). In modern distributed applications, high availability in the presence of runtime failures is an important concern. This paper presents an asynchronous distributed algorithm for constructing and maintaining an underlay aware overlay which ensures 3-degree of availability in the presence of node and link failures in the underlying physical network. We prove theoretically that our algorithm is correct. The time complexity of our algorithm is estimated to be $O(diameter \times degree)^2$ of the network and the message complexity is $O(diameter \times degree)$. We have investigated the scalability of the algorithm on a simulation testbed. The preliminary scalability test results demonstrate the scalability of our method.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems; D.2.8 [Software Engineering]: Metrics

General Terms
Availability, Event Based Systems

Keywords
Overlay Networks, Triconnectivity, Underlay Awareness, Availability

1. INTRODUCTION
Event broker networks are used for event dissemination in large scale event-based applications. Event brokers[1] form an overlay network over the physical network consisting of broker nodes as well as other (non-broker) nodes. Overlay networks facilitate applications to run smoothly without being aware of, and disturbed by, the intricacies and variations of the underlying network. In order to provide this support to applications, the overlay network should ensure the availability of routing paths in the event of node and link failures in the underlying physical network. The overlay has to be formed and maintained dynamically, while the event broker software and applications are running. As the network of computers is an asynchronous distributed system, asynchronous algorithms are required for the formation and maintenance of overlay networks. This paper outlines an asynchronous distributed algorithm for the construction and maintenance of an overlay network which guarantees the existence of three physically node disjoint paths between every pair of overlay nodes. Such an overlay network can be used for routing events without any changes in the presence of two node failures or two link failures, i.e., having an availability of degree three [2]. We chose to focus on an availability of degree three as higher degrees of availability would place a constraint of higher physical degrees (network connections) on the broker nodes in the network. The concept of our algorithm can be extended to provide availability of higher degrees.

The paper is organized as follows. Section 2 describes event broker networks, availability in event broker networks and the concept of underlay awareness. Related work on available overlays is presented in Section 3. Section 4 illustrates the graph theoretic concept of triconnected graphs, which is closely related to overlays of availability three. The Trimarg algorithm design and analysis form Section 5. Section 6 presents the proof of correctness for the Trimarg algorithm. Sections 7 and 8 describe the experimental framework for carrying out the simulation studies and present our experimental results. Future work in this area is discussed in Section 9, which also concludes the paper.

2. BACKGROUND

2.1 Event Broker Networks
In a distributed event processing system [1], event publishers and event subscribers are clients which are connected through a set of event brokers. In a multi-broker publish-subscribe system these event brokers are connected in a peer to peer fashion[1] to form an overlay network over the underlying physical network (underlay)[3].

Event brokers are smart machines where the event based middleware software is installed. The network of event brokers basically forms a sub network of the underlay network. An overlay network is a logical abstraction of the underlying physical network and can be represented as a graph with vertices corresponding to the over-
lay nodes and edges corresponding to the overlay paths between
the pairs of overlay nodes. Multiple overlays can be constructed
based on a single physical network. An overlay network can be
represented as a graph $N = \langle B, L \rangle$, where the vertex set $B$ is
the collection of overlay nodes, and the edge set $L$ is the collection
of paths between overlay node pairs that are determined by the ap-
lication (in this case the event broker network). Each broker can
directly communicate only to its neighbouring brokers, to which it
has direct overlay links. This overlay network of brokers can be
structured in many ways. Standard graph topologies like ring, tree,
star, chordal ring[4] etc. are seen in literature. However, the per-
formance of such an overlay network is tightly linked to that of the
underlying physical network (also termed underlay).

In order to maintain a specific quality of service and to be ro-
bust, the event based middleware forming the overlay must adapt
to changing conditions both in the application requirements and
changes in the computation and communication infrastructure pro-
vided by the underlying physical network of computers. amongst
changes that can happen in the environment, physical node failure
and link failure are crucial as they may block the communication
to clients, if there are no alternate paths readily available or effi-
ciently computable in the network [4]. The overlay needs to be
underlay aware so that it adapts itself to the faults or failures that
occur in the underlay. The main focus of this paper is on improving
availability in event based systems by providing an overlay network
that is adaptable to node and link failure triggers[4]. The proposed
overlay is a general fault tolerant overlay which can be used for
event dissemination in event based middleware. We propose a dis-
bursed algorithm for formation and maintenance of an underlay
aware, available overlay formation for the event broker network.
The next subsection briefly describes the model of availability in
Event Broker Networks used in this paper.

2.2 Availability in Event Broker Networks
Availability of a system is defined in standard literature as the frac-
tion of time for which a system is up (usable). In [2] the availability
of a general network is discussed and the degree of availability of
a network is defined and related to the node connectivity of a net-
work. According to [2] a network with an availability of degree $k$
has probability of non-availability of a path between any two pair
of nodes as $(1- (1-p)^n)^{\frac{k}{n}}$, where $n$ is the average path length, and
p is the probability of failure of a link. Such a network should have
a topology that guarantees the existence of $k$ independent paths be-
tween any pair of nodes. In this paper, we extend the definition of
availability and model it for overlay networks. In the context of
event broker networks, the availability of the overlay network is
the primary concern and the availability of the physical network
is of consequence only to the extent of the influence it has on the
availability of the overlay network.

2.3 Underlay Awareness
Clearly, knowledge of all aspects of the physical network cannot be
known by the overlay network as:

1. A centralized repository of information is not generally main-
tained. The information is distributed, as is the network.
2. The underlying network changes dynamically and hence main-
taining current information at all nodes is difficult and resource-
consuming.
3. The magnitude of information is large and storing this at every
broker node is not practical.

[5] Provides a taxonomy of overlay networks and classifies under-
lay aware networks as proximity aware and quality aware. Proxi-
mary aware overlays are aware only of neighborhood distance infor-
mation. Quality aware networks are aware of other aspects of the
links from the node, such as disjointness and performance param-
ters.

In this paper we describe algorithms for an underlay quality aware
overlay. The overlay nodes gather and store information about the
underlying path for the overlay links originating from them includ-
ing information about the nodes in the path and information about
the node overlap in the overlay links from the node. However, the
magnitude of this information is reduced by the fact that this in-
formation is required only for the nodes that “matter” in the path.
We present a brief classification of the nodes in the physical graph
relevant in this context.

2.3.1 Node Classification
All the nodes in the graph may be classified as follows - overlay
nodes, expander nodes, connector nodes and trivial nodes. Figure
1 illustrates this concept.

Overlay nodes Overlay nodes are the nodes selected to be brokers.
They should satisfy the criterion that their physical degrees are
greater than or equal to $k$, where $k$ is the degree of con-
nectivity required.

Expander nodes Expander nodes are nodes (non brokers) with
degree more than two. They have the potential to exist in
more than one physical path. Hence any overlay link that
contains such nodes can overlap with other overlay links,
which contain the same expander node. Hence overlay nodes
should remember such nodes that exist in their overlay links.

Connector nodes These nodes have degree equal to two. They
cannot be a part of more than one path between two higher
degree nodes and are hence “collapsible” as far as disjointed-
ness studies are considered, unless their degrees get changed.

Trivial or client nodes These correspond to “pendant” nodes in
graph theory terminology and have a degree of just one. They
cannot be guaranteed a connectivity more than one, and can
serve only as client nodes, if at all, in an overlay of connect-
vity $k$ higher than one. Hence these nodes can be ignored
for overlay construction.

2.3.2 Link numbering
The links associated with every node in the physical network are
ordered and specified by their link number for the node. For ex-
ample, a node with degree five has a fixed ordering 1, 2, 3, 4, and 5
for its outgoing links. The ordering is fixed based on a function of link
type and bandwidth, and uniquely determinable for every node.

2.3.3 Underlay information
The overlay node stores a sequence of three tuples for each of its
overlay links. The three elements of the tuple are $(node-id, link1,
link2)$, where $node-id$ identifies the physical nodes in the sequence,
and $link1$ and $link2$ are the link numbers of the links of the node.
The end nodes in this sequence correspond to broker nodes and the
intermediate nodes are expander nodes. Connector nodes are not
included as they cannot possibly overlap. Pendant nodes cannot be a part of such a path. This information is used by the broker nodes for overlay construction. In the next section we present two different perspectives of available overlays.

2.3.4 Underlay Aware Available Overlays
A given physical network can form the basis for many overlays networks. The availability guaranteed by an overlay can be manifested in two ways. It may be explicit, which means that different paths that are node disjoint in the overlay also map to physically node disjoint paths. On the other hand, an overlay’s availability may be implicit, or hidden, which means that node disjoint overlay paths are not necessarily node disjoint at the physical level, but the existence of different node disjoint paths at the underlay level, for each pair of overlay nodes, is guaranteed by the overlay. On this basis we classify available overlays as

1. Manifest
2. Latent

![Figure 1: Types of nodes in the underlying network](image1)

![Figure 2: Availability Manifest Overlay](image2)

![Figure 3: Availability Latent Overlay](image3)

Availability Manifest Overlay: An Availability Manifest Overlay can be defined as follows. Let $G = (V, E)$ represent a physical network, and $N = (B, L)$ be an overlay network based on $G$, where $B \subseteq V$ and $L$ consists of overlay links which are paths in $G$ between two vertices of $G$ that also belong to $B$. $N$ represents an availability manifest overlay network, if for any two nodes $b_1$ and $b_2$ belonging to $B$, if $(b_1, x_1, x_2, \ldots, x_k, b_2)$ and $(b_1, y_1, y_2, \ldots, y_k, b_2)$, where $x_1, \ldots, x_k$, and $y_1, \ldots, y_k$, all are broker nodes, be two overlay paths between $b_1$ and $b_2$ such that $x_1, \ldots, x_k$, and $y_1, \ldots, y_k$, are disjoint sets, then the two paths are also disjoint at the underlay level. In other words, the set $(p_1, x_1, p_2, x_2, \ldots, p_k, x_k)$ and the set $(q_1, y_1, q_2, y_2, \ldots, q_k, y_k)$, where $p$’s and $q$’s represent the physical nodes along the path, are also disjoint sets. This is illustrated in Figure 2. Given a physical network and an overlay graph, constructing an availability manifest overlay on the physical graph can be directly mapped to the fixed, node disjoint subgraph homeomorphism problem[6], which is NP-complete for general subgraphs (overlays). However for simple subgraphs, such as triangles and two node disjoint paths between two nodes, the fixed, node disjoint subgraph homeomorphism problem has linear time solutions [6],[7]. Manifest overlay creation requires a complete knowledge of the underlay, which is a practically difficult proposition in the context of event broker systems. Moreover, manifest overlays are not absolutely necessary if availability is the only requirement. Hence, we focus on latent overlays.

Availability Latent Overlay: An availability latent overlay of degree of availability $k$ on a graph $< V, E >$ is defined as $A = (B, L)$, where $B$ is a subset of $V$, and $L$ is a set of links which represent paths in the physical network, and for any $(b_1, b_2)$ belonging to $B$, there exist $k$ node disjoint paths in the physical network between the nodes corresponding to $b_1$ and $b_2$. This is illustrated in Figure 3. Hence in an availability latent overlay, two distinct overlay links can have node overlaps, but between any two broker nodes, $k$ node disjoint paths are guaranteed.

3. RELATED WORK
Event based Middleware is a well studied area and a number of research projects have been carried out in this field. Our survey on the EBM research projects and study of the type of overlay networks used and their support for availability is summarized in Table 1.
4. TRICONNECTIVITY IN GRAPHS

A k-connected (or k node connected) undirected graph $G = (V, E)$ is such that there are $k$ pairwise node disjoint paths between every pair of nodes in $V$.

Whitney's theorem [17], a fundamental theorem in graph theory states that the node connectivity of a graph cannot be more than the minimum degree among all the nodes. Hence a graph cannot have a connectivity and thereby a degree of availability more than the degree of its lowest degree node. If the graph represents a physical network of computers, nodes of degree greater than three are rare, as it would entail more than three network connections per machine. Hence, we focus on 3-connected graphs, which are also called triconnected graphs [7].

A separation pair [17] is a vertex pair $(a, b)$ of a connected graph such that the subgraph $G'$ of $G$ induced by $V - \{a, b\}$ is disconnected.

If $(a, b)$ is a separation pair of $G$, then there are at least two nodes $x$ and $y$ in $G$ such that every path between $x$ and $y$ passes through either $a$ or $b$ or both. A triconnected graph has no separation pairs.

A separating cut [17] is a set of vertices $V_c$ of a connected graph such that the subgraph $G'$ of $G$ induced by $V - V_c$ is disconnected. A separating cut of a triconnected graph has at least three vertices.

In general, if a node is added to a triconnected graph in such a manner that it is not disconnected by a separation pair, the resultant graph will be triconnected. This means that it will not be disconnected from the network by the removal (failure) of two nodes or one node. This can be ensured by connecting it by distinct edges to three different nodes of the original triconnected graph.

The Trimarg algorithm, discussed in the next section is based on the same principle. Trimarg, in Sanskrit, means three paths.

The concept of the algorithm can be used to build similar algorithms for higher degrees of availability with different complexities. We develop and present an algorithm for a connectivity of three as higher degrees of connectivity pose a more impractical constraint of greater physical connectivity on the physical nodes in the network.

5. THE TRIMARG ALGORITHM

We start with the design fundamentals of our algorithm.

5.1 Design

The overlay network is initialized with four special broker nodes called the stellar nodes. The stellar nodes network form a wheel [17] network of four nodes. The edges in the wheel network are physically disjoint. This network is statically designed and initialized. The stellar broker network is illustrated in Figure 4. Figure 5 shows the underlying physical network of the stellar broker network.

The initial (stellar) broker has manifest and latent availability of degree three, as can be observed by listing out the paths between broker pairs in Figure 5. The paths contain expander nodes as well as connector nodes. The identities of the expander nodes in the adjacent links are stored at the broker nodes. A node that wishes to join the overlay network of broker nodes does so by connecting it-
The steps in the distributed algorithm work as follows. The new node to join is statically aware of the ids of the stellar broker nodes. It sends a *Join_StellarBroker* message to stellar broker nodes through all its links. It receives messages in return from stellar brokers or other intercepting brokers. It finds three node disjoint paths from the messages it receives and establishes broker links through them. The algorithm is formally stated in Subsection 5.2.

### 5.2 Algorithm

The routines described here are

1. **Make_Broker(B)**: This is executed when a node B wants to join the broker network.
2. **Leave_Broker(B)**: This is executed by a broker B to leave a broker network.
3. **Execute_Broker(B)**: This is a routine executed throughout by a broker which is a part of the network.

---

self to existing broker nodes. The overlay links so formed need not necessarily be node disjoint with existing links. However, the broker network should ensure that the new broker has three pairwise node disjoint paths to every broker node in the existing network.

The node joins the overlay network through links consisting of possibly many expander nodes in the link. Consider the expander node x, which is the first node in this path which is already a part of the existing overlay network's underlay, as illustrated in Figure 6.

At this stage, x is a separating cut of the new broker network, as it disconnects B from the network. The separating cut is of size one. The old network is known to have no separating cuts of size less than three. If now, node B is linked to the existing network through another existing expander node y, the cut separating B from the remaining nodes will be x, y. A third connection to z for B should ensure that the separating cut of B from the existing overlay is three or greater. The two cases that may arise due to relative positions of x and y in the network.

1. **Case 1**: The node x and y lie on common overlay link. This is illustrated in Figure 7. The new link from B to the broker network (to node z) should lie on a different overlay link, other than the one having x and y in it, otherwise the node B has a separating cut of size two. Figure 7 illustrates this. Thus another link needs to be found, such that it meets a different overlay link, like z, to ensure the availability constraints.

2. **Case 2**: The node x, y and z lie on different overlay links. The size of separating cut is at least three. This is illustrated in Figure 8.
The following messages are sent/received by the broker nodes for the construction and maintenance of the overlay.

1. **Join_Stellar_Broker**(*B*) This is sent by a node *B* wishing to join the broker network

2. **New_Contact**(*R, B, path, list*) This is a message sent by a broker *R* which intercepts the **Join_Stellar_Broker**(*B*) message, or by the stellar broker (in case the **Join_Stellar_Broker**(*B*)) was not intercepted by any other broker) to the broker *B* which wishes to join. It contains the knowledge of the path by which the message reached *R* from *B*, and also a list of other brokers and expander nodes, that *B* should try to contact, to get two node disjoint paths to the broker network.

3. **Establish_Broker_Link**(*B, A, path*) This is a message sent by a broker *B* to broker *A* to establish an overlay link between them with the underlay information in path.

4. **Confirm_Broker_Link**(*A, B, path*) This is a message sent by a broker *A* which receives the establish broker link message from *B*, meant for *A* itself or some other broker *X* to which *B* had sent the **Establish_Broker_Link** message, if *A* was an expander node on the path from *B* to *X*, and subsequently became a broker. Then the broker link is established between *A* and *B*, instead of *X* and *B*.

5. **Check_Path**(*A, B, nodelist*) This is sent by a node *A* to a node *B* to check whether the path by which the **Check_Path** message reaches node *B* has more/less expander nodes than listed in nodelist (nodes in path). This is a periodic refresher message sent between overlay nodes, for the purpose of dynamically monitoring changes in the physical network.

6. **Leave_Stellar_Broker**(*B, nodelist*) This is sent by a node *B* to inform all the overlay neighbours that *B* has ceased to be a broker. Nodelist contains the list of neighbours of *B*.

7. **Bye**(*A, B*) This message is sent by a neighbouring broker *A* or stellar node to allow *B* to stop its routing activities in the overlay.

---

**Algorithm 1 Make_Broker**(*B*)

1. if (degree(*B*)<1) then exit /* does not qualify*/
2. send **Join_Stellar_Broker**(*B*) from all the links of *B* to the stellar brokers
3. while (not timeout)
4. if receive(New_Contact(*A, B, path, list*))
5. then store (path, list) in pathset
6. if pathset is empty
7. then exit /*brokers unreachable*/
8. else select three node disjoint paths from pathset
9. if found
10. then Send(Establish_Broker_Link(*B, A, path*))
11. else if (suggested paths present in the pathset)
12. then send (**Join_Stellar_Broker**(*B*)) through suggested path in list of one **New_Contact** message
13. delete suggested path
14. goto step 3
15. else exit /* fails to find node disjoint paths*/
16. while (not timeout)
17. receive(Confirm_Broker_Link(*A, B, pathlist*))
18. start Execute_Broker(*B*)
19. start routing activity
paths found are (f, k, w) to B, (c, b, a) to S0 and (d, e) to S3 respectively. The new overlay links are shown with thick lines in Figure 11.

The fourth case illustrated is the establishment of three node disjoint paths, to three different brokers, but the paths having nodes in a common overlay link. Consider Figure 12. The aspiring broker node N finds three node disjoint paths, (m, j, k, u) to B1, (v, f, p) to S3 and (t, c, b, a) to S0, and is allowed to join the broker network B with three overlay links to B1, S3 and S0 respectively. The nodes are a part of the overlay link B1-B2, which maps to the physical path (w, k, f, t). But there is no smaller separating cut, hence the new broker node will have three node disjoint paths to the broker network.

A broker that wishes to stop being a broker executes a Leave_Broker(B). It still remains a part of the physical network. This message is also intercepted and replied to by the neighbouring broker which has B as its overlay neighbour with a Bye message. If this node had already ceased to be a broker the Leave_Broker message travels all the way to the stellar node which replies with a Bye message.

We can visualize the process of a new broker node trying to join the broker network as an ancient ship trying to reach land by following a star. The ship stops when it finds land. The new broker node tries to reach the stellar brokers in the sea of IP addresses in the physical network, and in the process stops its search when it finds another broker already a part of the network.

### 5.2.1 Maintenance of the Overlay

Every overlay node executes the Execute_Broker() for overlay maintenance.

#### Algorithm 2 Leave_Broker(B)

```plaintext
/* this is executed by a broker B to leave a broker network. */
1. leaving = true
2. safe = false
3. over = false
4. while (not safe) wait;
5. stop(Execute_Broker(B))
6. send(Leave_Stellar_Broker(B, brokerlist)) through all the links of B.
7. while (not timeout or not over) 
8. for each node in neighbourlist
9. if not receive(Bye(A, B) message then wait
10. over = true
11. Stop routing in overlay.
```

#### Algorithm 3 Execute_Broker(B)

```plaintext
1. while (true) /* execute until terminated */
2. if receive(Join_Stellar_Broker(A) and leaving==false )
3. then intercept message
4. send New_CONTACT(B, A, path, list)
5. enqueue(B, A, path, list)
6. if receive(Establish_Broker_Link(A, B, path))
7. then send(Confirm_Broker(A, B, path))
8. dequeue(A, B, path, list)
9. add overlay path (A, B, path);
10. if (queue is empty) safe = true
11. if receive(Leave_Stellar_Broker(A, brokerlist))
12. then add new overlay links to the nodes in brokerlist
13. send Bye message(A)
14. for all brokers x in neighbourlist of B
15. send(Check Path(B, x, nodelist1))
16. if receive(Check Path(x, B, nodelist2) and nodelist2 ≠ nodelist1)
17. then update path()
```

Broker addition was discussed in the previous section. The routine for broker addition is also executed by a node already existing in the broker network as an underlay node, on turning into a broker. When the Join_Stellar_Broker message is received by the brokers on the links from the joining broker node, they respond with New_CONTACT and new broker links get established. If in the time interval between sending New_CONTACT and receiving an Establish_Broker, a new broker joins on the path, then the Confirm_Broker is sent by the new broker node, and the overlay link is only to that path. When the broker gets a Leave_Broker message meant for itself, it replies with a Bye message and establishes overlay links to the neighbour of the neighbours, which is present in
the list sent by the leaving broker. Otherwise, it ignores the message and lets it proceed further up the path to the stellar node. In the overlay network every node checks its links information periodically to confirm the underlay knowledge. If new expander nodes are added in the path (say, by adding a new link to a physical node), then that knowledge is updated in the overlay node, so that further node sharing information among links can be correctly assessed. The Check Path(A, B, brokerlist) message does this. The receiving node compares nodelist with the path by which the message actually comes and updates it. While checking the overlay information the old broker links may be modified by the old broker nodes, and new links to the new broker are added.

5.3 Complexity Analysis

Time Complexity: As the algorithm is based on routing to the stellar node in the worst case, the maximum diameter of the network would determine the upper bound on the time for sending and receiving messages. Comparing node disjointedness in the obtained paths is \( O(\text{number of paths} \times \text{path length}) \). As the number of paths is determined by the degree, and path length by the diameter, node disjointedness can be found in \( O(\text{degree} \times \text{diameter})^2 \).

Message complexity: The maximum number of messages are sent for the Make_Broker() routine, as the Join_Stellar_Broker message has to be sent by the joining broker to stellar brokers through all its links. The number of hops of this message is bounded by maximum diameter and the number of links is bounded by the maximum degree. Hence, the message complexity is \( O(\text{degree} \times \text{diameter}) \).

6. PROOF OF CORRECTNESS

We start by giving an inductive proof for the correctness for the algorithm.

Lemma: An overlay graph \( B' \) obtained by the sequential or interleaved executions of the Make_Broker() and Leave_Broker() procedures by different physical nodes and concurrent executions of the Make_Broker() and Leave_Broker() procedures by different physical nodes in a given physical network \( G \), containing an availability latent overlay \( B \) of degree three, is an availability latent overlay of availability degree three, i.e., any broker node in \( B' \) has three node disjoint paths to every other broker node in \( B' \).

Base: The initial network formed by stellar brokers has three node disjoint paths to each other by construction as shown in Figure 5.

Inductive step: If the network of brokers and expander nodes \( B \) has three node disjoint paths between every broker, then a network of broker and expander nodes \( B' \) obtained by the execution of Make_Broker() by a single node, or concurrent execution of Make_Broker() by two different nodes, or by Leave_Broker() executed by a non stellar node by two different nodes, or concurrent Leave_Broker() executions by non stellar nodes, or the concurrent execution of Make_Broker() and Leave_Broker(), also has three node disjoint paths between every two broker nodes which form a part of \( B' \).

Proof: We prove the inductive step by considering the execution of the routines.

Case 1: Execution of a Make_Broker() by a node \( b \).

(i) If \( b \) has degree less than three or it is not connected or too far off (time-out) from \( B \), then no change is made to \( B \), hence \( B'=B \).

(ii) If \( b \) gets at least three replies that include node disjoint paths to three different broker nodes in \( B \), \( x, y \) and \( z \), then it joins with those links. Suppose the paths it gets are \((b, p_{11}, p_{12}, ..., p_{1n}, x), (b, p_{21}, p_{22}, ..., p_{2m}, y) \) and \((b, p_{31}, p_{32}, ..., p_{3k}, z) \). In each of these sequences there is a first node which is already a part of the existing overlay. Removing any one node, before this first expander node each from any two of these paths, will not disconnect \( b \) as there is a third node-disjoint path from \( b \) to \( B \). As \( B \) already has three node disjoint paths between each of its broker nodes, the removal of any two of these nodes does not disconnect \( B \) either. The removal of any two nodes including those subsequent to these first nodes also does not disconnect \( b \) from the network as \( B \) is known to have no separating pairs, and there exists a third path from \( b \) to \( B \) which does not include these two nodes. The three expander nodes at which \( b \) joins \( B \), form a separating cut for \( B' \). If \( b \) itself was an expander node in \( B \), then it is allowed to join only if it has three disjoint paths, thus ensuring it has no separation cut of size less than three.

(iii) If it does not get two disjoint replies, it uses the information given by one of the replying brokers, which contains information about other expander nodes and broker nodes to contact. Hence if the node is able to link to that expander node, and this path is node disjoint to two other paths already obtained, then also \( B' \) satisfies the availability criteria.

(iv) If it is not possible, the node \( b \) does not join, so \( B'=B \).

Case 2: Concurrent execution of Make_Broker() Suppose nodes \( b_1 \) and \( b_2 \) execute Make_Broker()

(i) if one or both of \( b_1, b_2 \) have lesser degree or get timeout then the proof enumerated in Case 1 holds.

(ii) If nodes \( b_1 \) and \( b_2 \) get three node disjoint paths each and these are to different broker/expander nodes in \( B \), then they can simultaneously join \( B \), and the proof for case 1 holds.

(iii) If two nodes \( b_1 \) and \( b_2 \) select paths that are mutually disjoint but overlap with each other’s paths, it could happen in different ways as we enumerate

1. Both join to same expander nodes/node, but with disjoint paths. Both still get node disjoint paths to each other and to the rest of the network, as they have separating cuts of size at least three. This is illustrated in Figure 13.

2. Both join same nodes, but with overlapping paths. This is illustrated in Figure 14.

3. One of the nodes lies in the path selected by the other as illustrated in Figure 15.

4. Both nodes lie in paths selected by each other. This is illustrated in Figure 16.
Case 3: Leave_Broker() execution.

Leave_Broker() calls, whether executed simultaneously or individually, do not affect the underlying network connectivity, as it is just a change in status of the node(s) from broker to expander.

Case 4: Leave_Broker() and Make_Broker() execute concurrently.

(i) If the brokers contacted by Make_Broker() are different from the node executing Leave_Broker() then they would each execute correctly individually as proved in cases 1 and 3.

(ii) If the broker executing Leave_Broker() is one of the nodes that is contacted by the node executing Make_Broker() then if the contact is made after it starts executing Leave_Broker() then the Join_Stellar_message is simply forwarded, as it would have executed leaving = true. If Leave_Broker is executed by a broker that has already sent New_Contact, it is not allowed to proceed (by wait) until it has confirmed the path. Also, it does not send new New_Contact messages.

Once the leaving broker confirms the path, the other broker would put this as a neighbour and vice-versa, so while continuing with Leave_Broker(), the leaving broker will wait for a Bye message from the new broker node, confirming that it has created other overlay links. Hence the algorithm ensures that B' satisfies the required availability criterion in the presence of concurrent Leave_Broker() and Make_Broker() executions.

7. EXPERIMENTAL FRAMEWORK

The Trimarg algorithm has been tested on a simulated environment using an event based network simulation framework. We have developed this simulation framework by extending the Distributed System SIMulator (DSSIM) used in Hermes [1]. Our simulation framework provides for generation of overlay networks of event brokers. Different event routing algorithms can be incorporated into the simulator as plugins. Thus our simulator acts as a testbed for evaluating the performance of different event routing algorithms on different overlay topologies generated by the simulator based on the overlay formation algorithms integrated into the framework. There is an event driven application (a news dissemination system) simulated on this environment which is used for testing the event routing algorithms on different overlay topologies.

The overlay topology formation discussed in the Trimarg algorithm is tested on a simulated underlying physical network generated using the Boston University Representative Internet Topology gEnerator(BRITE)[18]. The underlay aware overlay topology with three connectivity is generated using the Trimarg algorithm. A preliminary performance evaluation and scalability testing of the algorithm has been carried out. We use a BRITE generated physical network topology of 1000 nodes as the test case for the algorithm. Triconnected overlays of different sizes (10 overlay nodes to 100 overlay nodes) were generated by randomly picking up broker nodes from the physical network. The total time for creating the topology has been measured in incremental steps of 10 overlay nodes. Also the overlay settling time on removal of bro-
kers by execution of Leave_Broker() routines has been observed in steps of removal of 10 randomly selected overlay nodes from the existing overlay network. The memory requirement at each broker node for storing the underlay path information for the incident edges (space complexity of the algorithm) has been monitored for overlay networks of size up to 100. Our initial test results are presented in the next section. The simulations have been repeated 12 times and average values are reported here. Elaborate testing of the algorithm for more performance parameters, on the simulator is in progress.

8. EXPERIMENTAL RESULTS

The scalability of the Trimarg algorithm was tested by increasing the size of the overlay network. The number of overlay nodes was increased from 10 nodes to 100 nodes in steps of 10 by randomly picking up broker nodes from the physical network and the results of the simulation are reported here. The values reported are the average of the values obtained for 12 simulation runs.

We report the initial test results for the experimental framework discussed in Section 7.

8.1 Time for overlay formation

Figure 17 depicts the simulated broker addition time as a function of number of nodes.

Discussion: Figure 17 shows that the time for broker addition is independent of the size of the overlay network, proving the scalability of our algorithm and its adaptability to real networks.

The time for broker deletions is measured on the simulation framework with different node sizes, starting from 100 nodes and decreased in steps of 10 to 10 nodes. The time for deletion of 10 broker nodes is plotted and illustrated in Figure 18.

Discussion: The graph in Figure 18 shows that the time taken for readjusting the overlay on the executions of the Leave_Broker() routine is less than that of the Add_Broker() routine. It is due to the fact that as the underlay information is already available with the neighbouring nodes, on leaving of a broker, more underlay information need not be collected. The average time for readjusting the triconnected overlay is independent of the size of the overlay network emphasizing the scalability of our algorithm.

8.2 Space Requirement for underlay information

Each overlay broker node stores the underlay path information (expander node ids) for every incident edge on it. Here we measure the average space overhead per node by measuring the total memory required for storing the underlay path information. The results of the simulation are plotted in Figure 19.

Discussion: Figure 19 shows that the average memory requirement per broker node is constant, irrespective of the increase in the number of overlay nodes in the broker network. The small variations on the memory requirement is due to the random selection of the broker nodes and the random nature of the physical network used as the input.

We have tested the algorithm with different physical networks and it has been observed that the space overhead remains almost the same.
9. CONCLUSIONS AND FUTURE WORK

High availability in the presence of the runtime failures is an important concern in modern distributed event based applications. This paper is based on an analysis of the availability of broker networks and a study of the theoretical formulation of the problem of determining the availability in a broker network. This paper presents an asynchronous distributed algorithm for constructing and maintaining an underlay aware overlay network which ensures 3-degree of availability in the presence of node and link failures in the underlying physical network. We prove theoretically that our algorithm is correct. The time complexity of our algorithm is estimated to be \(O(\text{diameter} \times \text{degree})^2\) of the network and the message complexity is \(O(\text{diameter} \times \text{degree})\). The scalability of the algorithm has been tested on a simulated environment. The preliminary results of testing and evaluation of the algorithm show that the algorithm is scalable.

It was found that the broker addition time, deletion time and memory requirements for underlay information were independent of the size of the broker network. Detailed tests on the algorithm are in progress. Fault tolerant event routing algorithms, capable of content based event routing are being developed on this underlay aware overlay network of event brokers.

10. REFERENCES


