Autonomic Peer-to-Peer Connectivity Using Overlay Networks

K. Ravindran*
Department of Computer Science
City University of New York (City College)
Convent Avenue at 138th Street, New York, NY 10031 (USA)
E-mail address: *ravi@cs.ccny.cuny.edu*

The paper deals with connectivity reconfigurations to handle the changes in the characteristics of the distribution path between client and server sites on an infrastructure network. For a given session-level configuration, dynamic network changes (e.g., increase in delays over a path segment or a node failure) may trigger the reconfiguration of a distribution path. With reconfiguration activities occurring asynchronously at the connectivity management level, a coordination among these activities is required to serialize the changes to an interconnection structure such that the reconfigured distribution path continues to maintain the tree-structure. In a way, our model provides a *redundancy* in the distribution paths along the temporal dimension (on top of a rich connectivity in the physical infrastructure — as in overlay networks and wireless ad hoc networks). The paper offers the tree-structured connectivity to support a variety of applications: such as QOS management in CDNs and server replica coordination. Policy functions that prescribe constraints on the tree setups (e.g., QOS support) can augment our algorithms to suit specific application domains.

* **K. Ravindran** is with the Department of Computer Science, City University of New York (City College), New York, NY 10031, USA — E-mail: *ravi@cs.ccny.cuny.edu*

1 Introduction

An overlay infrastructure network consists of nodes that can provide session-level connectivity to application end-points over back-to-back connected transport links (e.g., TCP and UDP links). Besides data forwarding, these nodes can also carry out some application-specific functions such as layered video multicast, traffic monitoring, and OAM activities. Content distribution (CDN) is an example of applications that benefits from infrastructure networks, where each node can function as a data caching point for clients to reduce the access latency [1]. As another example, [2] employs a overlay network of proxy nodes to implement distributed firewalls for protecting a server against 'denial of service' attacks. The infrastructure nodes may be deemed as 'middle boxes' that exist within the infrastructure network but can be exercised by the application through some signaling mechanism[1]. Regardless of the application setting, we look at a fundamental problem in this paper, namely, providing *logical connectivity* among application end-points residing in nodes $\mathcal{A}$ through an underlying infrastructure network.

The connectivity among nodes $\mathcal{A}$ can be viewed as an *acyclic graph* $\mathcal{G}(\widehat{\mathcal{M}}, \widehat{\mathcal{E}})$ that has vertices in nodes $\widehat{\mathcal{M}}$ and edges $\widehat{\mathcal{E}}$ over the transport links between $\widehat{\mathcal{M}}$ such that $\mathcal{A} \subseteq \widehat{\mathcal{M}} \subseteq \mathcal{M}$ and $\widehat{\mathcal{E}} \subseteq \{(x,y)\}_{\forall x,y \in \widehat{\mathcal{M}}}$, where $\mathcal{M}$ is the set of nodes in the entire network and (x,y) refers to the direct transport link between nodes x and y. $\mathcal{G}(\widehat{\mathcal{M}}, \widehat{\mathcal{E}})$ represents a *canonical* structure that provides connectivity among the nodes $\widehat{\mathcal{M}}$, with exactly one path between any pair of nodes, i.e., a tree that spans the nodes $\widehat{\mathcal{M}}$.

Architecturally, an *infrastructure connectivity layer* (ICL) manages the interconnection of nodes and links. The logical topology layer (LCL) executes an algo-

[1] The architectures studied elsewhere such as Scattercast [3], End-system Multicast [4] and Application-level Multicast [5] may be viewed as depicting infrastructure networks in some way. These architectures evolved for supporting multicast delivery of video and images in heterogeneous settings.

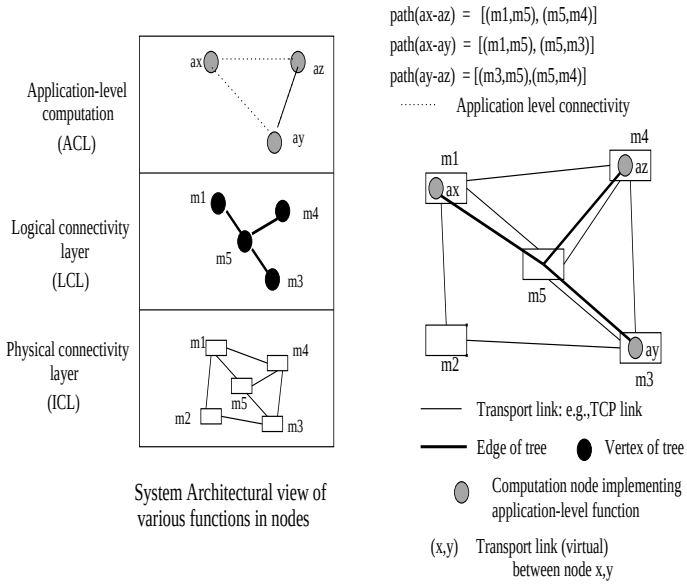

Figure 1: cycle-free connectivity in networks

rithm to reconfigure the connectivity between the application-level end-points (ACL) in the presence of node crashes and link failures. See Figure 1. Here, $\mathcal{M} = \{m_1, m_2, m_3, m_4, m_5\}$, $\mathcal{A} = \{m_1, m_3, m_4\}$, and $\widehat{\mathcal{M}} = \{m_1, m_3, m_4, m_5\}$. Suppose the ACL implements a CDN with, say, a_x being a content source and a_y being a client. The LCL sets up a content access path by rooting $\mathcal{G}$ at a_x and identifying the set of adjacent edges that connect to a_y from the root, namely, $\{(m_1, m_5), (m_5, m_3)\}$. In[2] general, the ICL provides the support mechanisms to facilitate tree reconfigurations by the LCL. If, in Figure 1, m_5 fails for instance, the ICL notifies the LCL about the failure, whereupon the LCL replaces m_5 with m_2 in $\widehat{\mathcal{M}}$ and $\widehat{\mathcal{E}}$ with a new set of edges $\{(m_1, m_2), (m_2, m_3), (m_3, m_4)\}$. The encouching of reconfiguration algorithm into the LCL and parameterizing its execution with $(\mathcal{M}, \{(x, y)\}_{\forall x,y \in \mathcal{M}})$ from the ICL and $\mathcal{A}$ from the ACL depicts a sound engineering practice in network designs.

Tree reconfigurations are already prevalent in some application settings (such as multicast data routing [6]). However, the emergence of newer application domains that demand connectivity reconfigurations in a scalable and autonomic manner (such as CDNs) necessitates a *generic algorithmic framework* to realize tree reconfigurations. The framework can offer self-configuring building blocks that can be instantiated to suit specific applications.

[2]Since the infrastructure nodes need not be just routers (they may be, say, TCP end-points), protocols such as DVMRP and PIM developed in the IP world need to be modified for use in infrastructure networks.

2 Why tree-structured connectivity ?

In this section, we make a case for using a communication-level abstraction of trees to maintain peer-to-peer connectivity.

Though a tree appears to be more vulnerable to partitioning if a failure occurs closer to the root, we get around this problem in two ways. First, we use an "unrooted tree" to provide cycle-free connectivity among application-level computation nodes (realized on top of a 'connected graph' topology of overlay nodes). Second, the tree is in a perpetual state of reconfiguration to "heal" around failed network nodes and links in the underlying connected graph.

The unrooted topological structure of our tree reduces (but does not eliminate) the likelihood of a node/link failure causing more disruptions in connectivity than a failure occurring in a different part of the tree. The perpetual state of 'healing' a tree from breakages offers guaranteed connectivity among peer application nodes over certain (slow) time-scales. Thus, the spatial redundancy implied by the mesh architecture of SRIRAM [7] to provide a sustained connecctivity is equivalently realizable (in a loose sense) by resorting to continuous recovery activities, along the temporal dimension, on our "unrooted trees". A good thing about trees is that the reconfiguration activities are algorithmically tractable, due to the regularity and recursivenss of tree-based topological structures. Also, tree management protocols are readily available (from existing works on ALM and overlay multicast).

Overall, our framework of autonomic tree constructions over infrastructure networks can anchor the development of applications for QOS management and for fault-tolerance in a generic manner.

3 Tree reconfiguration problem

Our framework rests on the premise that $\|\mathcal{A}| \leq |\widehat{\mathcal{M}}| \ll |\mathcal{M}|$. Reconfigurations occur when a node joins $\mathcal{A}$ or leaves $\mathcal{A}$ in the midst of application-level computation, causing $\widehat{\mathcal{M}}$ to expand or shrink, as the case may be. These induced reconfigurations are in addition to those occurring due to network-induced

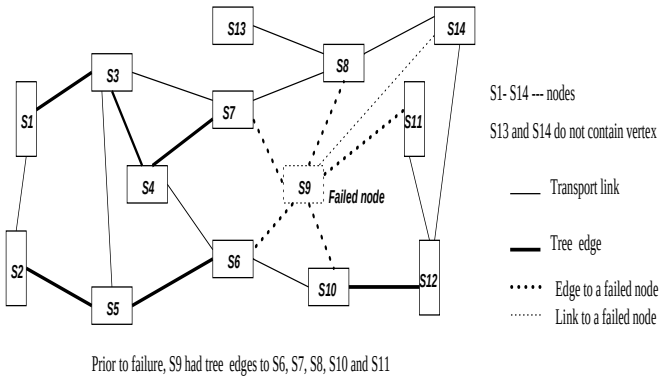

Figure 2: A fragment interconnection scenario

changes in the infrastructure topology[3].

3.1 Tree fragmentation

We model the reconfiguration problem as one of connecting together two or more subgraphs of $\mathcal{G}$, known as *fragments*, which hitherto are not connected with one another [8]. A reconfiguration algorithm executes on various nodes in the infrastructure network to interconnect the fragments. A key requirement is that the concerned fragments coordinate with one another so that the new tree obtained from the fragments does not contain cycles.

See Figure 2 for example. Prior to failure, the node S_9 had edges to its neighbors S_6, S_7, S_8, S_{10} and S_{11}. After the failure, the sets of nodes $\{S_1, S_3, S_4, S_7\}$, $\{S_2, S_5, S_6\}$, $\{S_{10}, S_{12}\}$, $\{S_8\}$ and $\{S_{11}\}$ are the fragments (nodes S_{13} and S_{14} are not in any fragment since they do not contain any vertex). A required interconnection may be the creation of edges over $\{(S_{12}, S_{14}), (S_{14}, S_8)\}$, $\{(S_{12}, S_{11})\}$, $\{(S_6, S_{10})\}$ and $\{(S_4, S_6)\}$. In this case, avoidance of cycles requires coordination among the fragment interconnection activities to ensure that no edges are created over $\{(S_1, S_2)\}$ or $\{(S_3, S_5)\}$. Such a synchronization may be achieved by serializing the fragment interconnection activities in a consistent order at various nodes.

[3]Since application-level configuration changes usually occur over a slower time-scale relative to changes in the network topology, the number of reconfiguration invocations in our approach will be far less compared to that in existing approaches that reconfigure upon network-triggered events across the entire topology. For example, link congestion due to traffic overload may occur over a 5-*sec* time-scale in the Internet, whereas joins and leaves of computation nodes in the video conferencing session may occur over, say, a 5-*minute* time-scale.

The fragment interconnection activities may also be subject to problem-specific requirements. In an example of multicast routing for video distribution, the reconfiguration algorithm may need to ascertain the availability of enough bandwidth to transport data over the new tree segments. Referring to Figure 2, a minimal-hop routing will create a tree segment over (S_6, S_{10}), and then over either (S_1, S_2) or (S_4, S_6) based on which has enough bandwidth to transport video. Such requirements may be embodied into policy functions realized on various nodes that are invoked by reconfiguration control messages.

3.2 Concurrent reconfigurations

In existing approaches, addition and removal of vertices to a spanning tree are made either in a sequential manner or by collecting the entire topology information at a centralized site [9, 10]. These approaches are suitable for smaller sized systems in which the topological changes are infrequent, and when they do occur, global synchronization to serialize the addition and/or removal of vertices is not expensive. In the case of infrastructure networks where the number of fragments can be large and/or the fragment formation can be highly dynamic (as in CDNs), potential concurrency among various additions and/or deletions should be exploited to achieve reasonable levels of performance.

With each reconfiguration activity causing updates to various regions of a tree, a major issue is the serialization of changes to the tree. If the changes do not cause any cycles to be formed in the connectivity structure, they can occur in parallel; otherwise, they should be serialized. Consider, for instance, the fragments $\{S_1, S_3, S_4, S_7\}$, $\{S_2, S_5, S_6\}$ and $\{S_{10}, S_{12}\}$ in Figure 2. The interconnection of, say, S_1, S_2 and that of S_6, S_{10} occur in different regions of the tree, and hence can proceed in parallel. But an interconnection of S_1, S_2 and that of S_4, S_6 occur in the same region, and hence only one of these interconnections should take place to avoid a cycle. Furthermore, certain activities need to be causally ordered, such as the leave of a node from a tree occurring after a join of this node to the tree has taken effect. Referring to Figure 2, suppose S_{13} joins the tree, and leaves the tree quickly thereafter. If, say, S_{13}, S_8 sees these events in the correct order but not S_7, an 'orphan' edge may get created over (S_7, S_8). Thus, the formation of cycles and/or having redundant edges are symptomatic of inconsistent ordering of reconfiguration activities.

Overall, though multiple reconfiguration activities can be in progress concurrently, the changes to a tree structure should appear as if the various interconnec-

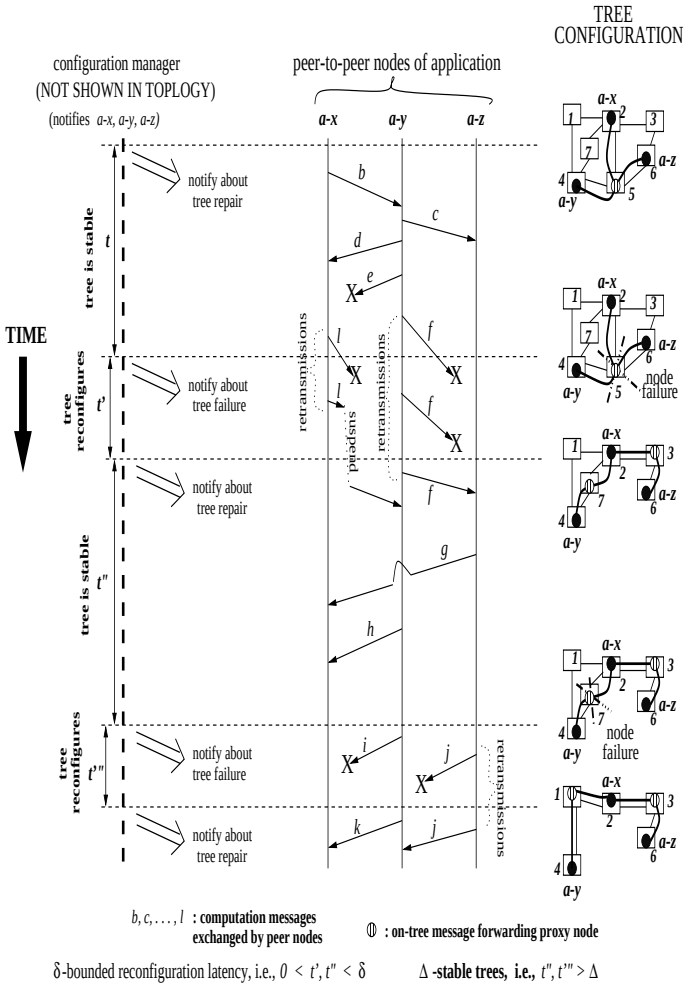

Figure 3: Illustration of ACL-LCL interactions

tions were done by one reconfiguration activity at a time. A distributed realization of this serialization may employ a total ordering of the interconnection activities as seen by the participating nodes.

3.3 LCL-ACL interface (API)

The meta-information exchanged across the ACL-LCL interface, among other things, are the upper bound on tree reconfiguration latency (δ) and the lower bound on how long a tree remains stable (Δ). We assume that the failure of nodes in a communication path are detectable by application-oriented testing mechanisms. Assuming a rich connectivity between various nodes in the PCL, an algorithm can reconfigure the tree-structured connection within a bounded interval of time δ. The algorithm may determine δ based on the failure detection mechanism and the geographic placement of ACL nodes in the infrastructure topology[4]. Whereas, Δ is determined by the type of failures and the functional characteristics of system elements.

Figure 3 illustrates the ACL and LCL interactions for a sample scenario of peer-to-peer message flows. The peer nodes a_x, a_y and a_z are interconnected by a tree-strucured communication path. The data paths from a_x to a_y, a_y to a_z, a_z to a_y, and a_z to a_x exhibit non-lossy FIFO behaviors; whereas, the data path from a_y to a_x exhibits a lossy FIFO behavior. The non-lossy data paths may be realized by session-level protocols that employ, say, TCP-based infrastructure node-level connectivity. The protocols for these paths resort to timeout-based message retransmissions to recover the messages lost during path reconfigurations (such as messages f and j). Furthermore, the protocol along the path a_x-a_y also employs deferred message transmission, i.e., suspends the message transmissions from the time of tree failure to the time of tree repair[5] — such as message l. The lossy data path a_y-a_x may be realized, say, by employing UDP-based infrastructure node-level connectivity.

The δ and Δ parameters cast on the semantic properties of the session-level data paths between peer nodes — such as QOS and communication synchrony.

We now describe how our model of temporal redundancy of communication paths between peer nodes can be employed for application support.

4 Tree-based application-level support

From the standpoint of using trees for peer-to-peer application support, our model may be viewed as providing redundancy in connectivity between peer nodes along the temporal dimension. This is unlike the currently prevalent models of providing spatial redundancy in the comunication paths between peer nodes as part of application support. Our model of temporal redundancy of communication paths is realized in the LCL by providing for a rich node-level connectivity in the ICL[6].

[4]The control messages required by a reconfiguration algorithm are assumed to flow 'in-band' along the data paths over which the peer nodes exchange computation messages.

[5]The failure and repair notifications are posted from the configuration manager executing in the LCL.

[6]In other words, our algorithmic procedures for maintaining tree-structured connectivity transform the spatial redundancy of infrastructure-level nodes onto a temporal redundancy of communication paths between peer nodes in the application.

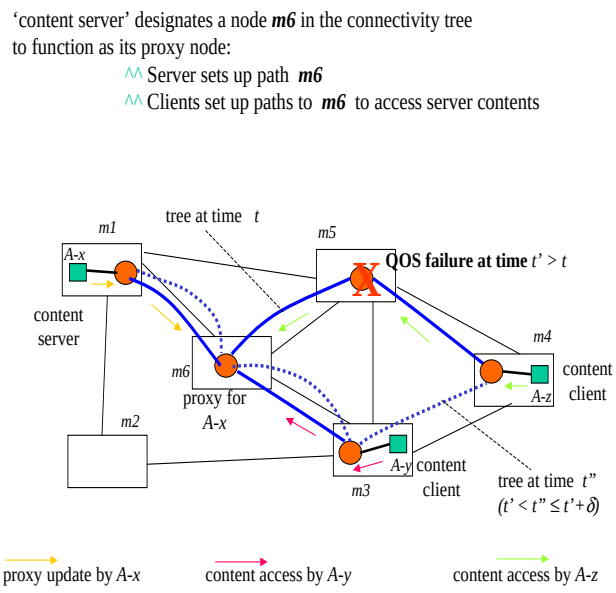

Figure 4: Proxy-based structure of a CDN

We employ tree-structured node-level (logical) interconnections over infrastructure networks as the building block for peer-to-peer connectivity required in many distributed applications. Two applications are examined: i) QOS management in 'content distribution networks', and ii) fault-tolerance by server replication.

4.1 'content distribution networks'

CDNs may employ an overlay of caching proxies placed at multiple nodes of an infrastructure network, to control the quality of service (QOS) of content access at a server site [1] (e.g., AKAMAI and Digital Island). With client access patterns changing over time and the information contents being dynamic, connections that can be automatically reconfigured to realize efficient access to remote contents are necessary. An approach is to incorporate policy-based control of the operations of a CDN based on various communication-level performance factors. These factors include the location of clients in the network, their access patterns and the frequency of content changes.

The interactions between the server and proxy nodes and the clients and proxy nodes are realizable on top of our tree-structured connectivity. See Figure 4. When a proxy cache p (such as node m_5 in the Figure) determines that it cannot meet the latency requirements of a client C, the parent of p can declare a QOS failure and invoke a tree reconfiguration along the path connecting p to C. The invocation (across the ACL-LCL interface) can carry a parameter γ depicting the acceptable latency between C and p. The latency parameter may then be used in the tree reconfiguration algorithm to determine the set of candidate path segments to connect C to the parent of p, and then choosing one of these paths. Specifically, the γ parameter may bound the reconfiguration latency δ to a value such that:

$$\gamma > (\delta + d),$$

where d latency between p and C along a candidate path. Assuming that d for various candidate paths is known from a QOS management protocol, the reconfiguration algorithm may use this information in deciding on a path between p and C.

Using the tree stability parameter Δ, the QOS management function in the ACL may determine many client-level performance parameters — such as how many client requests can be serviced under the current delay and bandwidth parameters of the distribution paths. Furthermore, the knowledge of δ allows the QOS manager to determine the client requests that are likely to not meet the access latency requirements during a tree reconfiguration. Referring to Figure 4, the access requests generated by client A_z during the reconfiguration time interval $[t', t'+\delta]$ might suffer extra delays, in addition to the delays incurred along the path segments m_1-m_6-m_3-m_4. To enable such a performance control, the QOS manager may be a part of the configuration manager that maintains the tree-structured distribution path.

As can be seen, our tree reconfiguration approach can be employed to control the QOS of content delivery in highly dynamic settings.

4.2 Server replica voting

An object service may be replicated on different nodes of an infrastructure network for fault-tolerance purposes. And, a client may access the object at any of the server replica sites. The replication may be supported by multicast communication paths between the replicas and the clients, realized over a tree-structured interconnection among them.

Since a server site can be compromised by malicious attackers, a client may resort to some form of decentralized voting among the replicas to determine the correct result from a server. Typically, it is required that a majority of the replicas are non-faulty — which allows employing a majority voting on the

various server outputs. Decentralized voting has an algorithmically tractable solution in the presence of malicious process-only failures, with an appropriate multicast communication support between the replicas and the client. But, consensus is difficult (or, even impossible) to achieve when the communication paths between replicas are also vulnerable to attacks — say, one or more nodes in a communication path getting compromised by intruders (as in wireless ad hoc network settings).

The prescription of an upper bound δ on the reconfiguration latency preserves the synchrony properties of the communication paths. To elaborate, if the paths already guaranteee delay bounds in the absence of infrastructure node failures, these delay bounds are only increased by δ when path reconfiguration is incorporated, i.e., the paths continue to maintain the synchronous nature of message flows. On the other hand, if the paths do not guarantee delay bounds even in the no-failure cases (i.e., are asynchronous), the prescription of δ does not change this absence of delay guarantee. Yet another setting is possible where delay bounds of the communication paths do exist in the no-failure cases but are not known to the ACL[7]. Such paths continue to remain as 'partially synchronous' [14] — even with the passing of δ parameter from LCL to ACL.

With the prescription of a lower bound Δ on how long the connectivity between server replicas will remain stable, we reduce the voting problem as one of determining a majority result in the presence of only process attacks during a limited time interval Δ. See Figure 5. This allows using the voting algorithms developed to deal with process-only failures under various types of communication synchrony. For instance, the Byzantine agreement protocol proposed elsewhere for Internet-style networks [12] (which are 'partially synchronous' communication paths) can be employed after the afore-mentioned reduction of the voting problem in the presence of path reconfigurations.

As can be seen, our approach deals with failures by resorting to spatial redundancy for the server processes and temporal redundancy for the interprocess communication channels[8].

[7]This may be either because the LCL does not know the delay bounds or the API does not allow the meta-information on delay bounds to be passed on from LCL to ACL.

[8]In contrast, the existing approaches deal with failures by using spatial redundancy for both processes and channels (say, providing primary and backup channels for interprocess communications and switching from the primary to a backup channel upon a failure). See [13] for a generalized discussion on spatial versus temporal redundancy as a means to achieve Information Assurance.

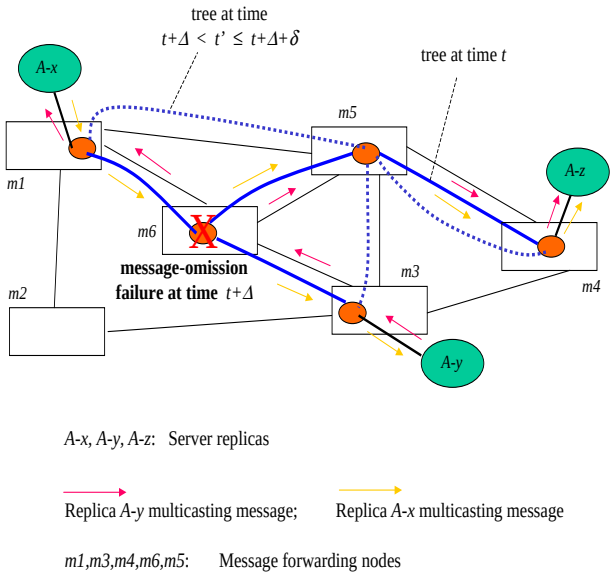

Figure 5: Structure of a server replication system

Our future goal is to further explore the notion of temporal redundancy in peer-to-peer connectivity in the application domains of Information Assurance and QOS Management. To ascertain its feasibility, we now describe the algorithmic elements to reconfigure trees when changes in path characteristics occur.

5 Algorithmic components

We assume that the topology of infrastructure network remains connected, i.e., a node is reachable from every other node through one or more intervening links. Each node knows its immediate neighbors, which implies that a node detects within a finite time the existence of a new neighbor or the loss of connectivity with a neighbor[9].

5.1 Fragment formation

See figure 6 as an illustration. The failure of node w results in the fragments $\{a, b, d, c\}$ and $\{f, g, h\}$. Sim-

[9]For algorithmic purposes, we also assume a FIFO behavior of infrastructure links with no message loss — which is the case with TCP links.

ilarly in figure 6(d), the join of node v results in the fragments $\{v\}$ and $\{a, b, d, c, w, f, g, h\}$. A node belongs to exactly one fragment at any given time. In the presence of multiple reconfigurations, it is necessary that each node belongs to the same sequence of fragments. So a logical clock tick (in an 'event ordering' sense) corresponds to the time interval at which the various nodes belong to a given fragment. Referring to figure 6, suppose the node b also fails, then the nodes a, d, c, f, g, h are members of the fragment caused by this failure after the time at which these nodes (including b) belonged to the fragment caused by the failure of w. Similarly the nodes $(v, a, u, z, d, c, f, g, h)$ are members of fragments caused by the join of v after the time at which these nodes are members of a fragment associated with the failure of b.

Every node eventually acquires knowledge on which fragment it belongs to. A reconfiguration algorithm propagates this knowledge across the nodes of various fragments at chronologically ordered points in time.

5.2 Reconfiguration waves

Each vertex which detects a trigger event, functions as a *coordinator* to initiate reconfiguration activity on behalf of its fragment to connect to other fragments. We refer to the algorithmic projection of this activity as a *wave* (we refer to a wave started by a fragment x as $wv(x)$). If the waves initiated by the coordinators of fragments, say, F_1 and F_2 meet each other at a node, the algorithm needs to resolve the contention among $wv(F_1)$ and $wv(F_2)$ for creating the edges to interconnect F_1 and F_2, so that only one of the waves succeeds in setting up the interconnection. In an algorithmic sense, one of the waves *acquires* the fragments carried by the other waves it meets to form a single larger fragment — say, $wv(F_1)$ acquiring F_2. The fragment acquisition by the winning wave involves taking over the paths generated so far by the losing waves. The wave carrying the combined fragment further propagates to meet other waves that may potentially exist in other parts of the tree.

5.3 Incorporating problem-specific constraints

The selection of links over which a reconfiguration wave propagates (to create candidate edges) may be based on information provided by the ICL about how the paths reachable through various links can satisfy problem-specific constraints. One instance may be QOS-enabled multicast routing. Here, the ICL may maintain a multi-path connectivity table that contains

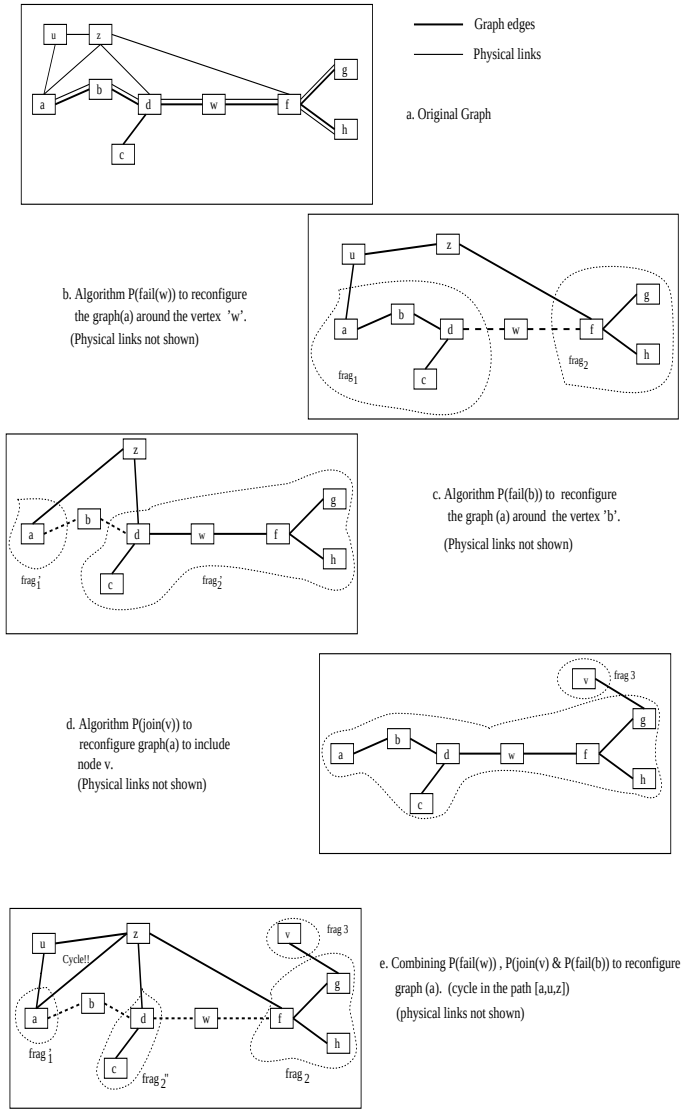

Figure 6: Composing multiple reconfigurations

link weights for selected bandwidth levels, where a link with higher weight may indicate a path segment along that link with the largest capacity at the given bandwidth level. Another instance may be the setting up of secure paths between peer entities. Here, the ICL may maintain information about the security levels of various links, with an insecure link abstrated as a link with inifinte cost. In our model, the proxy agents that realize the ICL abstraction may be implanted with such problem-specific meta-information to guide the reconfiguration of trees.

We now provide a distributed algorithm that embodies the above functional components, and its performance evaluation.

6 A sample algorithm for tree reconfiguration

We limit our goal to providing a canonical distributed algorithm, simply to demonstrate the feasibility of our approach (though other methods of tree constructions — including offline tree computations — are possible).

6.1 Message diffusion

The notion of wave propagation may be procedurally realized by having a coordinator node x sending a control message **Reconf**(cid(x),ev,...) over one or more of its outgoing links to neighbors, where cid(x) is the node id of x and ev indicates the trigger event. Basically, the node x initiating the reconfiguration activity is at the root of a *control message tree* (CMT) that spans over the edges reachable from x. Each node which receives a **Reconf** message forwards it, under certain conditions, through one or more of its outgoing links to neighbors. The selection of links over which the **Reconfig** is forwarded may be based on information provided by the ICL about how the paths reachable through various links can satisfy problem-specific constraints. In QOS-enabled multicast routing for instance, the ICL may maintain a multi-path connectivity table that contains link weights for selected bandwidth levels, where a link with higher weight may indicate a path segment along that link with the largest capacity at the given bandwidth level. Such a forwarding of the **Reconf** message constitutes a wave propagation along specific directions in the tree.

For algorithm purposes, we assume that the routing information maintained by the ICL nodes are mutually consistent — which can avoid message loops[10].

6.2 Coordination between simultaneous waves

Since more than one (on-tree) node could have detected a trigger event, there might be multiple reconfiguration waves for the same event, with each of these waves identifying a set of edges to reconnect the fragments. When such competing waves meet, we should allow one of the waves to continue and acquire the other waves, so that cycle causing edges are not created. The contention resolution in our algorithm uses

[10]In a degenerate form, the message diffusion may amount to sending the **Reconfig** message by resorting to 'flooding' techniques.

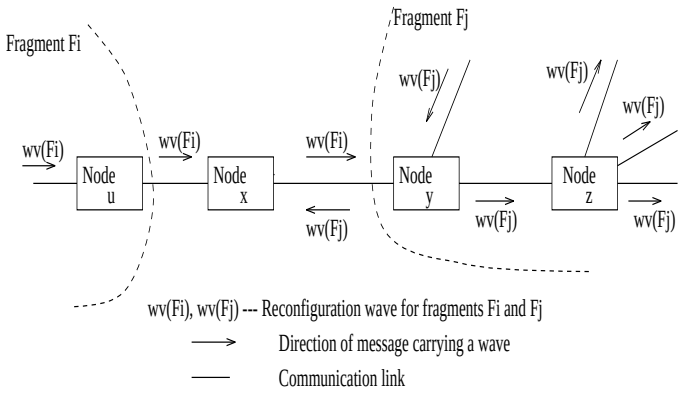

Figure 7: Illustration of a reconfiguration wave

the ranking of coordinator node ids, where the ranking information is made available from the ICL.

A coordinator F_i is higher ranked than a coordinator F_j if cid(F_i) > cid(F_j). Suppose two reconfiguration waves $wv(F_i)$ and $wv(F_j)$ meet at a node z (i.e., compete). See Figure 7. If the coordinator for F_i is higher ranked than that for F_j, a **Stop** message is sent to the sender of **Reconf** on behalf of F_j to inform all the nodes in F_j that no other fragment lower ranked than F_i can contend on the path beyond x.

A node knows its parent in the CMT, i.e., the vertex from which it first receives a **Reconf**. In Figure 7, y is the parent of z in wave $wv(F_j)$ started by fragment F_j. When $wv(F_j)$ reaches a vertex x in fragment F_i, the parent pointers starting from x back to F_j give a path over which F_j and F_i can be connected.

6.3 Fragment Growth

Each node waits for a reply from the links where it sent out a **Reconf** message. A node generates a message **NoCont**, if there are no mor links to forward the **Reconf** message. **NoCont** message mean that a new edge cannot be created along that path. If a node receives **NoCont** or **Stop** messages from all the neighbors whom it is awaiting a reply from, it sends a **NoCont** message to its parent in the CMT. Further, if the node contains a tree vertex with an edge to the leaving node, an **Acc** message is sent to its parent. The **Acc** message indicates the acceptance to connect its fragment to another in the direction pointing to the winning coordinator. With each **Acc** message, an edge is created between the sender and receiver of the message — which causes the fragment to grow.

In the presence of multiple reconfiguration events concurrently taking place, we employ a time-stamp based ordering of reconfiguration activities to serialize

their effects on the tree. With a globally assigned time-stamp sequencing (say, using loosely synchronized clocks), each node totally orders the reconfiguration activities in deciding whether to create an edge over the link to a given neighbor.

Since a node knows only about its immediate neighbors, the state pertinent to a node is whether there is an edge over a link to any of its neighbors meeting a problem-specific constraint. We need to keep this decentralized information consistent, in the presence of multiple reconfigurations concurrently taking place. This is guaranteed by the total ordering of successive tree configurations at various nodes, even though the trigger events for the various reconfigurations might have occurred concurrently.

The various algorithmic elements to effect a tree reconfiguration, namely, reconfiguration waves, fragment growths, fragment coalescing, and time-stamp based serialization are composed in a way to avoid cycles in the overall new tree. The actual algorithmic details are however outside the scope of this paper (see [11]). Note here that the algorithmic framework described above[11] merely purports to establish the feasibility of our model of supporting distributed peer-to-peer applications on top of tree-structured connectivity.

7 Simulation studies

We studied the algorithm by discrete event simulation and collected the performance parameters, namely, the time to reconfigure a tree.

7.1 Simulation model of algorithm

Arbitrary network topologies were generated using 'random graph' techniques, whereupon an auxiliary algorithm was run to create a base tree. The reconfiguration algorithm was then run for this tree under the chosen physical topology. Each node maintains a local view of the physical connectivity information and the tree in the form of a 'node adjacency' list (say, in terms of TCP end-point addresses). Control message transfer delay over a link is treated as a random variable, with prescribed min-max values (in TCP-based infrastructure networks for instance, message delays can be 5-10 $msec$ for a 100-byte control message).

[11] The algorithmic framework and the underlying constructs have biological analogs where a neurological signaling takes place among different body parts to heal around a failed body tissue.

Simulation was performed for a network of 50 to 55 nodes. We believe this to be representative, since a distribution network usually does not contain lots of middle boxes[12]. The performance data of interest is the time taken to complete a reconfiguration (TTC).

The time-scale chosen is in units of the average transfer delay over a link. We also assume a flooding of **Reconfig** messages, i.e., the forwarding of **Reconfig** over all neighbor links except the parent link in the CMT, for the purpose of simulation. We believe this to be reasonable, given that the number of virtual links in an infrastructure network is not large and the flooding does not require any state information to be maintained in the ICL.

7.2 Time complexity

The TTC observed factors in the parallelism among multiple reconfiguration waves progressing in different parts of the network. As the number of trigger events occurring back-to-back increased, the increase in TTC is less than linear. This is because all these reconfigurations can potentially be handled during a single execution of the algorithm. To elaborate, the reduction in TTC is due to the elimination of duplicate actions that otherwise need to be carried out over multiple (sequential) executions. So, when a wave w meets a lower time-stamped wave w', the responses collected so far by w are acquired by w', as if w' had sent out the **Reconf** messages and explored the CMTs that have responded to w.

The simulated topologies are shown in Figures 8 and 10. The topology-1 has more density than topology-2, where density refers to the (average) degree of physical connectivity in the network. The corresponding TTC results are shown in the graphs of Figures 9 and 11. We observe that as the density of physical topology increases, the TTC increased by about 45% (on an average), which is due to the larger number of cycles in dense physical topologies.

The reconfiguration latency δ, which is relevant to the paper, can be obtained from the TTC results. Though the TTC results are in normalized units and the algorithm we studied is only a representative one, our study establishes the point that a reconfiguration can be carried out in a bounded interval of time, namely, δ.

[12] Though there may be a large number of IP routers in the physical network, the virtual link between two middleboxes in an infrastructure network may abstract out a large number of routers in the path.

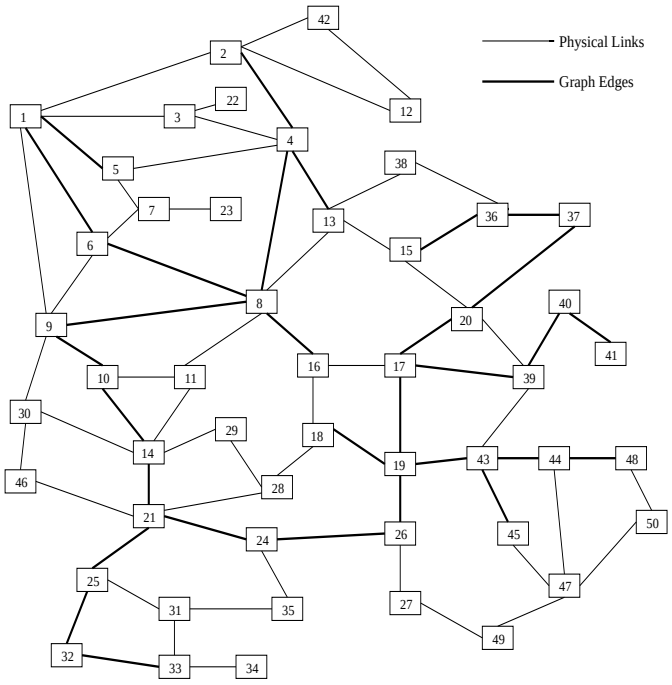

Figure 8: Topology 1

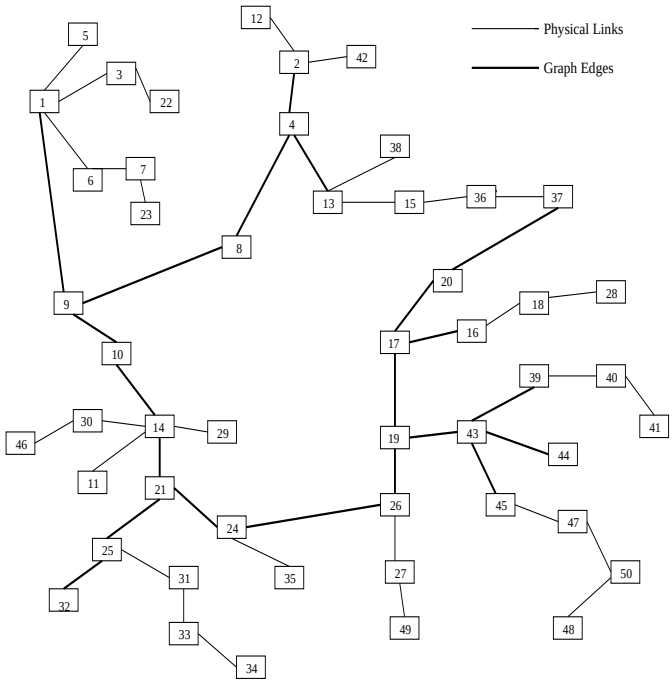

Figure 10: Topology 2

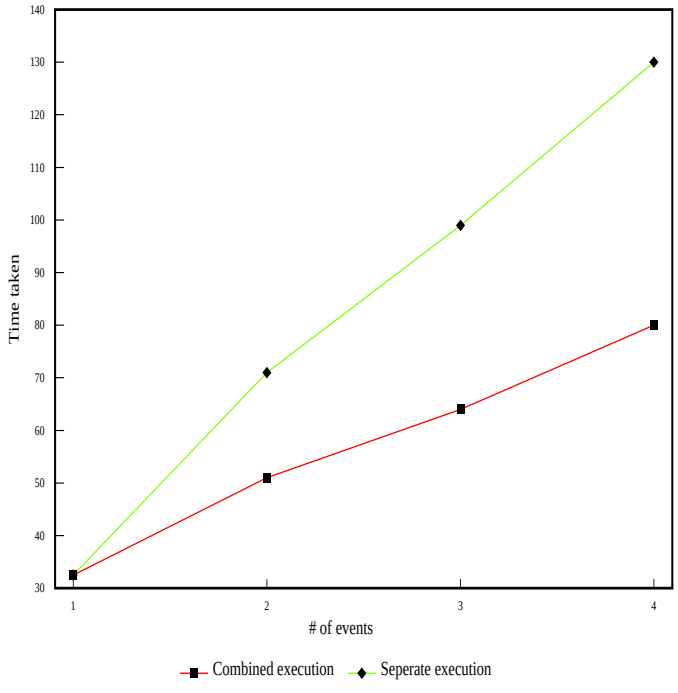

Figure 9: Time complexity (TTC) for Topology 1

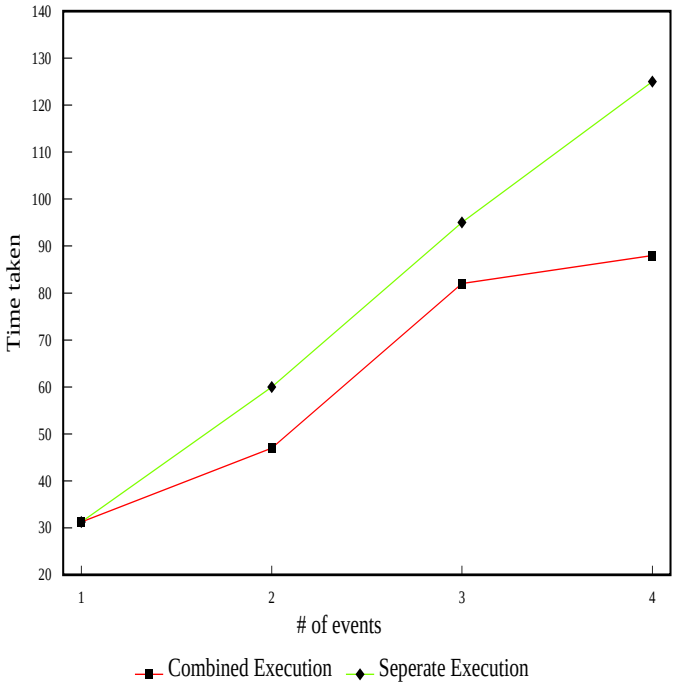

Figure 11: Time complexity (TTC) for Topology 2

8 Summary

Our goal is in developing a generic model for reconfiguration of tree-structured logical connectivity among nodes in an infrastructure network underlying a peer-to-peer system of computation nodes. Example systems where our model can be useful are application-layer routing, CDNs, and communication management for server replication.

In our model, the provisioning of logical connectivity involves constructing a tree that spans a small subset of nodes in the underlying infrastructure network that is just enough to provide the required connectivity. Connectivity reconfigurations are triggered by events such as the joining of a node to or the leaving of a node from the application-level end-point set $\mathcal{A}$. The advantage of our model is that it offers scalability of the connectivity provisioning mechanism in cases where $|\mathcal{A}| \ll |\mathcal{M}|$. Using the currently available tree management protocols (such as the Scattercast, End-system Multicast and Application-level Multicast), one can build infrastructure networks (say, on top of TCP or UDP links) by employing our approach.

The paper provided an algorithmic framework to support our model of connectivity provisioning. In this framework, tree constructions are realized by interconnection of tree fragments that may be formed when nodes join or leave a distributed computation during an application execution. Problem-specific constraints (such as QOS and security) can be prescribed as parameters to the underlying reconfiguration algorithm. A main advantage here is the ability to customize the algorithm execution for specific application-settings, with parameters and policy functions characterizing the applications.

With a proper support from the infrastructure layer — say, using proxy agents, our model of tree reconfigurations can be used as a self-configuring mechanism to support adaptive peer-to-peer network applications.

References

[1] D. C. Verma. **Content Distribution Networks: An Engineering Approach**. In *John-Wiley Publ. Co.*, 2001.

[2] A. D. Keromytis, V. Misra, and D. Rubenstein. **SOS: Secure Overlay Services**. In proc. conf. on *Communication Architectures, Protocols, and Applications*, ACM SIGCOMM'02, Pittsburgh (PA), pp.61-72, Aug.2002.

[3] Yatin Dilip Chawathe. **Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service**. In *Ph.D. Thesis*, University of California (Berkeley), Dec. 2000.

[4] Yang-hua Chu and Sanjay G. Rao and Srinivasan Seshan and Hui Zhang. **Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture**. In proc. *Communication Architectures, Protocols, and Applications*, ACM SIGCOMM01, Aug. 2001.

[5] A. Ghosh, M. Fry, and J. Crowcroft. **An Architecture for Application-layer Routing**. In *Technical Report*, Univ. of Technology (Sydney) and Univ. College (London), May 2000.

[6] S. E. Deering and D. R. Cheriton. **Multicast Routing in Datagram Internetworks and Extended LANs**. *ACM Transactions on Computer Systems*, Vol.8, No.2, pp. 85-110, May 1990.

[7] D, C. Verma, S. Sahu, S. Calo, A. Shaikh, I Chang, and A. Acharya. **SRIRAM: A scalable resilient autonomic mesh**. In *IBM Systems Journal*, vol.42, no.1, pp.19-28, 2003.

[8] R. G. Gallager, P. Humblet and P. Spira. **A distributed algorithm for minimal spanning tree**, *ACM Transactions of Programming languages and Systems*, 30(12), Dec 1983.

[9] B. A. Coan and et al. **Using Distributed Topology Update and Preplanned Configurations to Achieve Trunk Network Survivability**. In *IEEE Trans. on Reliability*, 40(4), Oct. 1991.

[10] J. M. Spinelli and R. G. Gallager. **Event-driven Topology Broadcast without Sequence Numbers**. In *IEEE Trans. on Communications*, COM-37, pp.468-474, May 1989.

[11] K. Ravindran, X. Liu, and M. R. Kumar. **Reconfiguration of 'tree-structured' connectivity in Large Dynamic Networks**. In proc. *GLOBECOM'03*, IEEE Com. Soc., San Francisco (CA), Dec. 2003.

[12] M. Castro and B. Liskov. **Practical Byzantine Fault-tolerance**. In proc. conf. on *Operating Systems Design and Implementation*, IEEE and ACM, 1999.

[13] F. Wang, R. Uppalli, and C. Killian. **Analysis of Techniques for Building Intrusion Tolerant Server Systems**. In proc. conf. on *Military Communications*, MILCOM'03, IEEE, Boston (MA), Oct. 2003.

[14] N. Lynch. **Distributed Algorithms**. In MIT-LCS Lecture notes, 1995.