Computer Science at Kent

A Survey and Performance Evaluation of Scalable Tree-based Application Layer Multicast Protocols

Su-Wei, Tan
Gill Waters
John Crawford

Technical Report No. 9-03
July 2003

A Survey and Performance Evaluation of Scalable Tree-based Application Layer Multicast Protocols

Technical Report 9-03

Computing Laboratory, University of Kent, UK
Su-Wei, Tan   Gill Waters   John Crawford

Abstract— **Application layer multicast (ALM) enables rapid deployment of multicast applications in the Internet. In ALM, application hosts organise themselves into an overlay topology on top of the underlying unicast network. Application data is multicast over the overlay network. In general, ALM protocols can be classified as either tree-first or mesh-first approach. In this paper, we provide a survey and simulation study for the class of tree-first ALM protocols. We investigate the efficiency of HMTP, TBCP, NICE and several variants of transformation-based protocols in terms of tree cost and delay optimisation. To the best of our knowledge, this paper provides the first head-to-head comparison of various tree-first protocols in a single simulation environment. Results show that depth-first searching technique in HMTP can achieve the lowest cost trees. On the other hand, a transformation-based technique that involves only local region nodes is able to construct low latency trees. In addition, we propose an enhancement to TBCP to improve its performance in producing low delay trees.**

Index Terms—**application layer multicast, distributed tree building protocol, performance evaluation.**

1 INTRODUCTION

Due to the need for efficient techniques for group communications, multicasting has received much research attention since its initial proposal [10]. Initial multicast proposals have focused on network layer based solutions in which multicast packet replication and forwarding mechanisms are all implemented in network routers. Indeed, network layer solutions make the most efficient use of network resources such as bandwidth. Unfortunately, global deployment of IP multicasting has been hindered for a number of practical reasons [11]. Consequently, various alternative proposals, such as simple multicast [28], source specific multicast [16] and application layer multicast (ALM) [4, 9, 13, 15, 18, 20, 21, 23, 25-27, 29-31, 33, 34, 36, 37] have been suggested. Out of these proposals, ALM has been in the limelight in recent years. This is because it utilises the existing Internet unicast protocols, and facilitates instant deployment without modifying the existing network infrastructure.

In the ALM architecture, all multicast related tasks such as group membership management, packet replication and forwarding are handled by the end hosts. ALM members form an overlay network, which is a set of unicast connections among themselves, for multicasting. The main objective of an ALM solution is to construct and maintain the overlay for efficient data transmission.

During the past few years, many protocols have been proposed to build efficient application layer overlay. Basically, we can categorise these protocols into centralised and distributed approaches. The centralised approach assumes that the pair-wise metrics such as delay between any two members are available at a well-known server. The server then computes an optimum delivery tree based on the desired optimisation objective, e.g. minimising delay or cost. The computed result is then distributed to all the members to construct the delivery tree. This solution, albeit simple, does not scale well and is subject to the single point of failure problem.

In the distributed approach, every multicast member executes an instance of a tree building protocol and, cooperatively, the members construct the ALM overlay. The overlay serves two purposes: as a control topology to maintain membership information and as a data topology to multicast data packets. Normally, the control topology is a mesh structure with multiple paths between members. On the other hand, the data topology is usually a tree to ensure loop-free routing. In a rough sense, we can classify the existing distributed proposals into two groups: “mesh-first” and “tree-first” based on the sequence that the mesh and tree are constructed.

Building a tree at the application layer is fundamentally different from building one at the network layer. In the network layer, the tree computation is done at routers which are equipped with network topology information. In addition, as the packet replication and forwarding are done at the router level, a packet can be delivered to every destination without unnecessary duplication in the network. On the other hand, an application layer host has either no or limited access to the routing information. This results in unnecessary packet duplication and prolonged latency. Consequently, the goodness of an ALM solution is often measured by the relative performance compared with the network layer multicast solution. Three performance metrics normally used are *link stress*, *stretch* [9] and *tree cost* [36]. The link stress metric is defined per-link as the number of identical packets flowing over a single link. The stretch metric or relative delay penalty (RDP) is defined per-member as the ratio of the distance between two members along the overlay and the corresponding direct unicast distance. The tree cost is defined per-tree as the sum of the cost of the overlay tree links. Typically, the cost of an overlay tree link is measured in terms of delay. These metrics measure the quality of the data delivery tree of an ALM solution. In addition, to achieve scalability for large member population, an ALM solution should have a low protocol overhead. The protocol overhead includes the control messages used during the joining process of new members, and tree maintenance and improvement processes.

This work investigates the efficiency of several existing tree building approaches for tree cost and latency optimisation using the above mentioned performance metrics. We focus on protocols based on the tree-first approach as it is the most direct way to construct distribution trees. In addition, it provides a basis to understand various optimisation techniques that can be applied to mesh-first approach. We classifying the existing tree-first protocols into three main groups:

i.) transformation-based,
ii.) local region rearrangement-based, and
iii.) cluster-based.

Figure 1 depicts a taxonomy of some existing tree building protocols adapted from [12] based on the above grouping. In this work, we evaluate the performance of HMTP (transformation-based), TBCP (local region rearrangement-based), NICE (cluster-based) and several variants of transformation-based protocols using an identical simulation environment.

In the next section, we discuss several design issues in the class of a tree-first approach. Section 3 reviews some existing tree-first based protocols. In Section 4, we describe our simulation settings. In section 5, we present our simulation results and discussions. In Section 6, we propose an enhancement to TBCP. We briefly discuss several related works in Section 7 and finally, Section 8 concludes the paper.

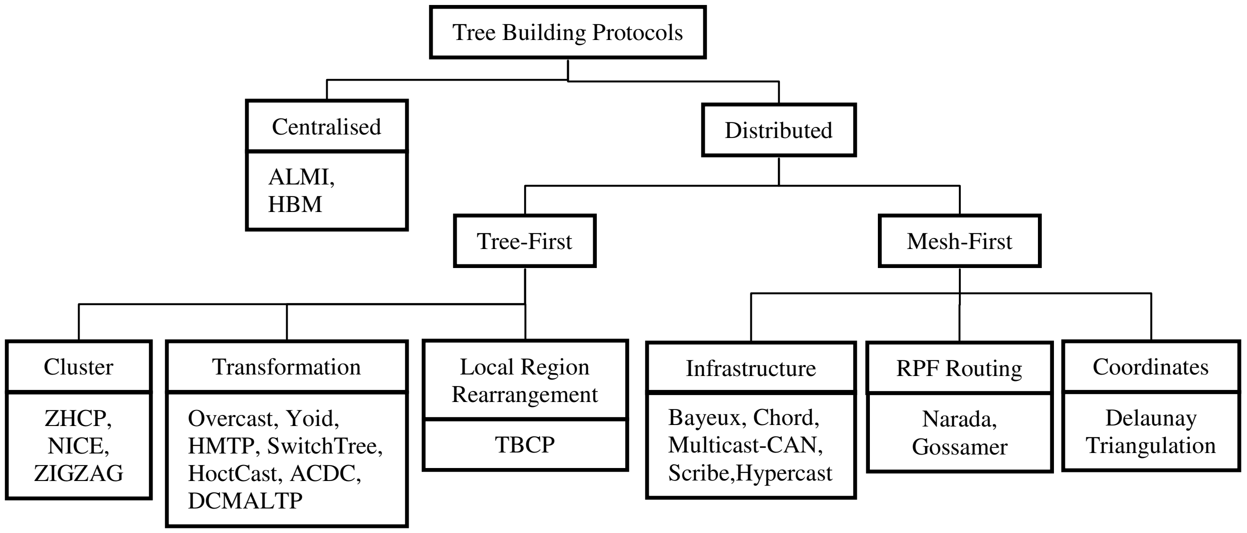


Figure 1: Taxonomy of Existing Tree Building Protocols (adapted from [12])

2 DESIGN ISSUES IN TREE-FIRST PROTOCOLS

2.1 System Architecture

The study of ALM often centred around two system architectures: peer-to-peer architectures and proxy-based architectures. In a peer-to-peer architecture, all functionality is pushed to the end hosts participating in the multicast session. In the later, an organisation that provides value added services deploys proxies at strategic locations on the Internet. End hosts attach themselves to proxies that are topologically close to them and receive data using a unicast service. Since our main focus is to study the efficiency of the tree building and optimisation techniques, we refer to an end system simply as the entity that actually takes part in the tree building process, which can be either one of the above.

All ALM protocols require a bootstrapping mechanism for a new member to discover existing members for a multicast session. Normally, a well-known Rendezvous Point (RP) is assumed to be available to maintain the membership information. Typically, the RP is not a member of the session. It acts as a query server that returns a list of existing members information to new member that wishes to join the session. Besides this, the RP may also be involved in other functions like partition healing and security. The information about the RP itself may be announced to potential group members through some out-of-band mechanisms such as web publishing or electronic mail.

Tree-first based protocols build the distribution tree directly. The constructed tree is rooted at a single node. For single-sender applications like content distributions from a well-known source, the tree root acts as the data source. Examples of protocols that are designed for this type of applications are Overcast [18], HostCast [21], ACDC [20] and DCMALTP [6]. On the other hand, multi-sender applications like video conferencing use a group-shared tree to distribute data. In this case, all members can send data over the shared tree. The root of the shared tree can be any of the group members. In practice, it can be the member who initiates the multicast session. It is possible for a protocol to change the root during the course of the session. Yoid [13] and HMTP [36] are examples of such protocols. An exceptional case in the class of tree-first approach shown in Figure 1 is the NICE protocol. The protocol builds a hierarchy of clusters as its control topology and derives source specific trees from the control mesh. As the control structure and delivery tree are built simultaneously, and the hierarchy built is in the form of a tree, it is considered as tree-first protocol. NICE is designed for large-scale multi-sender applications.

In tree-based overlay solutions, the distribution tree is defined by a set of parent-child relationships among the members. Each member, except the root, joins the session by becoming a child of an on-tree member. As end hosts normally have processing capability and limited interface bandwidth, the number of outbound traffic streams (fanout) that can be transmitted at a time is restricted. This in turn limits the number of children[1] that an overlay node can maintain. In all protocols to be discussed, a newcomer initially contacts the RP to obtain a list of existing members information. One or some of these members will be selected as potential parents where the newcomer will begin its joining process. How the initial potential parent is selected and how it evolves during the course of joining differ from one protocol to another. Once grafted to the tree, members communicate via the control topology to keep the membership information up-to-date. As network condition may change over time, some protocols provide refinement mechanisms to maintain or improve the quality of the distribution tree.

2.2 Issues with Tree Structure

A tree, by definition, is loop-less. Therefore, routing in a tree can be done by a simple flooding mechanism: when a node receives a packet from one of its tree links, it replicates and sends the packets out via the other tree links maintained by it. When an on-tree node leaves or fails, its existing children have to graft to the tree again by selecting a new parent. It is important that these nodes do not select their descendents as new parents. Therefore, loop detection; avoidance and termination become important design issues.

One simple loop avoidance and detection technique is for each on-tree node to maintain a root path, i.e. a list of

[1] We define the fanout of an on-tree node as the number of outbound replication for a traffic stream it can manage at one time. Therefore, it is the number of an on-tree node's children (plus one for the parent node if it acts as a data source in a group-shared tree).

nodes between the root of the tree and the node itself. With the root path, loops can be avoided if a node rejects all joining requests from nodes contained in the root path it maintains. In the case of loop detection, each node periodically forwards its root path to its children. When a node receives a root path that contains its own address, it knows that a loop has been formed and triggers the loop termination process. While these will prevent most loops from forming, loops may still be formed in some rare cases. This issue is discussed in great detail in [13]. Some other protocols like NICE, rely on the rules that construct the overlay to avoid loops (see Section 3.3.1).

Another issue with the tree structure is the loss of the root node. When the root of a tree leaves or fails, the tree is partitioned. To detect a tree partition, the tree root periodically delivers heartbeat message down the delivery tree. When an on-tree node does not receive a heartbeat message from the root for some time period, the node assumes that the root is dead and regards itself as a new root. As more than one node may regard themselves as the root at one time, the well-known RP can be used to resolve this. In this case, periodically, all nodes that regard themselves as the root send an "I am root" message to the RP. When the RP detects multiple roots, it selects one of them to be the root.

2.3 Optimisation Issues

ALM is implemented at end host or a dedicated server at a site. Unlike network routers, these systems have either no or limited knowledge of the underlying network topology. However, an efficient data delivery tree is one that is able to reflect the underlying topology well. To infer the underlying network metrics (e.g. delay and bandwidth), end-to-end measurement techniques have to be used. Typically, these techniques require probe packets to be sent between two end hosts for some time interval. It is impractical for each host to perform measurement to all other participants. In other words, a practical solution has to work with partial topology information in a distributed manner. Furthermore, end systems have limited processing capability and interface bandwidth, which restricts the fanout of the delivery trees.

In this section, we discuss several issues involved with optimising the distribution tree in terms of tree cost and latency.

2.3.1 Minimising Tree Cost or Resource Usage

In network layer multicast routing, tree cost is determined largely by the summation of individual link costs. The cost of a physical link could be an administratively configured value or the bandwidth cost. In terms of application layer routing, a tree link is a unicast tunnel between two nodes, which may actually traverse multiple physical links. While it is possible for an end host to measure the latency or path bandwidth to another host, it may not be possible to obtain the underlying path cost between the members directly. Therefore, we quantify the tree cost as the resource usage of the underlying network. A low cost tree is suitable for delay insensitive applications such as bulk file transfer.

If we let E_T be the set of overlay tree links and $d_o(e)$ be the delay of overlay link e in the tree. The tree cost is defined as,

$$tree\,cost = \sum_{\forall e \in E_T} d_o(e) \qquad \text{Equation 1}$$

In [9], network resource usage of an overlay solution is defined as the sum of the product of physical links' stress and their propagation latency. Since a physical link may be traversed by multiple overlay links (which results in stress), it may be counted more than once in the above equation. Therefore, equation 1 also roughly reflects the resource usage in the network.

In order to achieve a low cost tree, it is desirable for each host to attach to the nearest on-tree member. This optimisation problem is related to the well-studied degree constrained minimum spanning tree problem, which has been proven to be *NP*-hard even under a centralized computation model [19]. A practical solution in the context of ALM is further complicated by the need to work with limited topology knowledge. Figure 2 shows an overlay tree with maximum number of children of two. In the figure, the newcomer, *N* is trying to join the tree rooted at node *r*. From the figure, it is clear that the best parent for *N* is node *x*. However, initially *N* may only have the information about the tree root, *r*. This is a question of topology discovery, i.e. how *N* will eventually find its best parent, i.e. *x*?

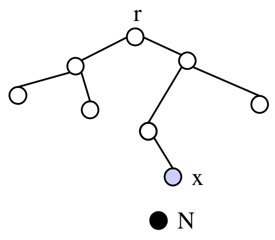


Figure 2: Problem in Distributed Overlay Tree Cost Optimisation

Another problem that often arises in tree cost optimisation is the triangle problem as illustrates in Figure 3 (a). Let *d*(*x*,*y*) depicts the unicast distance between node *x* and *y*. In the figure, the distances between the nodes follows the relationship: *d*(*A*,*B*) > *d*(*A*,*C*) > *d*(*B*,*C*). In this case, it is obvious that the arrangement in Figure 3 (b) provides a cost effective solution. However, if *B* joins the group after *A* and before *C*, a greedy decision that always seeks to graft a new node to the nearest on-tree node will end up in the arrangement in panel (a).

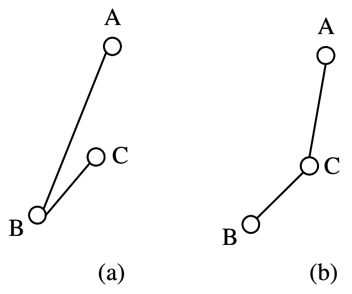


Figure 3: Triangle Problem

2.3.2 Minimising Latency

Two cases arise in latency optimisation. For single sender applications, the objective is to minimise the maximum distance from the sender to any other members. For multi-sender applications that use a shared tree, the objective is to minimise the maximum distance between any two on-tree nodes. In the rest of this paper, we use *root-diameter* and *tree-diameter* to differentiate the objectives of these two service models. Root-diameter optimised trees are useful for applications like live webcast and streaming media from a well-known server. On the other hand, tree-diameter optimised trees are important for multi-party applications like video conferencing.

Obviously, a source-rooted shortest path tree provides the best delay performance. In terms of application layer routing, this corresponds to a unicast star overlay rooted at the source. However, this solution results in high link stress and tree cost as many redundant physical links will be used. Besides, the number of simultaneous receivers that can be supported is constrained by the processing capability and interface bandwidth of the sender. The problems of finding the minimum tree- and root-diameter trees with a fanout constraint have been proven to be NP-hard in [32] and [24] respectively. In both works, the authors focused on the centralized computation model where the membership information and distances among the members are readily available. Independently, they proposed heuristics similar in nature for the problems (see Section 4.3 for a description of the algorithms).

Considering the problem of building a low root diameter tree, the objective of a newcomer is to choose a parent that provides the smallest latency from the root without violating the fanout constraint. Figure 4 illustrates a decision problem faced by a distributed protocol. In the figure, *r* is the root node while *N* is the newcomer. If *r* has a maximum fanout that is more than two, it is clear that *N* can be accepted as the child of *r*. However, if the maximum fanout of *r* is just two, there are a number of possible configurations even in this simple example (see Figure 4 (b), (c) and (d)). This problem is related to the triangle problem discussed above if the newcomer is forced to join to one of the children of *r*, i.e. configuration (b) or (c). While the solution shown in (d) is the best possible configuration, it will increase the sub-tree delay to *y*.

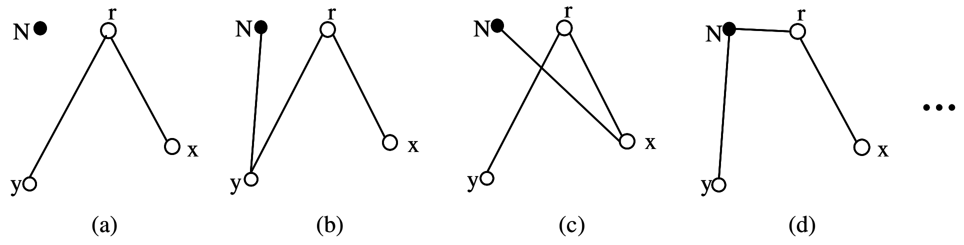


Figure 4: Problem in Distributed Latency Optimisation

2.4 Protocol Overhead

Protocol overhead can be divided into join overhead and maintenance overhead. Join overhead represents the communication cost involved in the joining process of a node. The communication cost may involve protocol messages and measurement messages. Protocol messages are join, query and reply packets used to establish a peering relationship between two hosts. Measurement messages are probe packets that are used to infer the underlying network metrics like delay and bandwidth. A classic example is the Ping program that measures the round trip delay between two end hosts. While different joining strategies impose varying degree of overhead on the network, a protocol can allow a new member to temporarily attach to an on-tree member to reduce join latency.

The maintenance overhead involves failure detection and recovery, loop detection and tree improvement. Failure detection requires peering nodes to exchange periodic refresh messages in the control structure. Some tree-based protocols use only the parent-child relationships in the data tree for this purpose, e.g. HMTP. Other protocols like Yoid and HostCast create additional links to improve the protocol's fault tolerance. As the refresh messages are normally small fixed-size packets, they impose little traffic on the network. If a protocol host does not receive a refresh message from its control path peer, it will assume that the corresponding peer has failed. A failure of a non-leaf node in a tree implies a partition. Typically, this results in a re-join process by the children of the failed node.

Maintaining the loop-free feature in the distribution tree in tree-based protocols is much easier than mesh-based protocols. As discussed in previous section, a root path can be used for loop avoidance and detection. In a mesh, multiple paths may exist between any two nodes. In order to avoid packet duplication, a loop-free routing strategy like reverse path forwarding needs to be used. This implies that a routing protocol is needed to disseminate the topology information among the members. As routing protocols often result in high communication overhead, this has restricted the use of the mesh-based protocols such as Narada for large-scale applications [9].

In tree-based protocols, a node gets involved in tree refinement by switching parent or swapping position with other nodes. In order to select a switch or swap target, a potential list of targets needs to be identified. This process normally involves end-to-end measurements from the node to the targets. Therefore, the overhead depends on the size of the target set. There is a tradeoff between the message overhead with the possibility of finding a better target. To reduce the measurement overhead, caches can be used.

3 TREE-FIRST PROTOCOLS

In the following section, we provide a survey of some existing tree-based protocols. This section is not meant to be an exhaustive survey of all tree building protocols. Rather, it concentrates on the tree building techniques for latency and tree cost optimisations. We classifying the existing protocols into three main groups: i.) transformation-based, ii.) local region rearrangement-based, and iii.) cluster-based according to the way that they construct and optimise the overlay trees. We begin the discussion with the transformation-based protocols, followed by local region rearrangement-based protocols and finally, cluster-based protocols.

3.1 Transformation-based Protocols

This is a class of protocols that use transformations like switching or swapping to optimise the overlay tree. Protocols in this category differ in the type of transformation, selection of targets and scope of the target set. The protocols classified under this category are Yoid [13], Switch-Trees [15], HMTP[36], ACDC [20], HostCast [21] and DCMALTP [6].

3.1.1 Yoid

Yoid [13] is one of the earliest ALM protocols. It consists of a set of protocols forming a new architecture for general content distribution. In Yoid, the first member to join the group is designated by the RP as the root of the tree. Each subsequent joining member contacts the RP and obtains a list of current members. One of these members will become the parent of the newcomer. The parent selection depends on the target application. For example, a member might attach to a topologically closest on-tree node or a node that provides the smallest delay from the data source. Besides the delivery tree, Yoid maintains a mesh structure for fast failure recovery. Yoid uses refinement to improve the delivery tree. The refinement is not targeted to produce an optimised tree, rather, it is to avoid routing pathologies. In the refinement process, a node is allowed to switch to a new parent that provides a better latency or packet loss performance.

A Yoid host uses two techniques to discover potential parents: i.) *random walk* and ii.) *depth-first searching*. In the random walk method, a node sends a discovery message with a time-to-live (TTL) field to its parent on the tree. The message is randomly forwarded from neighbour to neighbour with the TTL value being decremented at each hop. The node at which the TTL reaches zero will reply to the message source. In the depth-first searching technique, the searching begins at a non-descendent node and moves down the tree from the node in a depth-first manner.

3.1.2 Switch-Trees

In [15], a family of switch-parent algorithms was proposed: switch-sibling, switch-one-hop, switch-two-hop and switch-any-hop. We use SwitchTree to refer to this class of protocols. In SwitchTree, a newcomer is initially joined to the root node and depends on periodical refinement to relocate its on-tree position. In the refinement process, each node periodically estimates the distance of a set of nodes which may become its switching target. A switching decision is based on the optimisation metric; if the objective is to minimise the tree cost, a node will try to switch to a node that is closer than its existing parent; if the objective is to minimise the root diameter, the node will attempt to switch to a parent that provides a lower root delay. Figure 5 (a) shows a simple three nodes network with *p* as the root node. Panel (b) illustrates the configuration for a low latency problem while panel (c) shows the solution for a low cost tree.

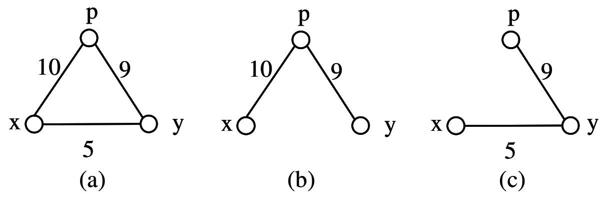
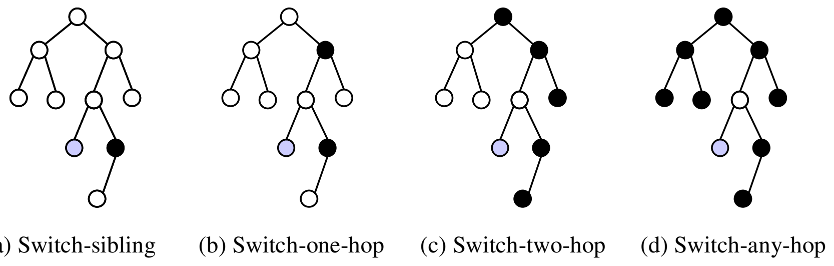


Figure 5: Delay and Tree Cost Solutions

The SwitchTree target set is confined within a predefined local region. Figure 6 illustrates the four algorithms proposed in [15]. In the figure, the grey node is the node wishing to perform a switch while the black nodes are the set of potential parents. In this class of protocols, no solution is provided for the triangle problem. The Banana Protocol [14] is a specific example of a switch-one-hop protocol.



(a) Switch-sibling (b) Switch-one-hop (c) Switch-two-hop (d) Switch-any-hop

Figure 6: Family of Switch-Trees Protocol

3.1.3 Host Multicast Tree Protocol (HMTP)

The HMTP [36] is another example of switch parent protocol. It is designed to build low cost trees for multi-sender applications. HMTP differs from SwitchTree in the initial joining method and the selection of target set. Since HMTP tries to build low cost trees, it attempts to place members at a close proximity together (see Section 2.3.1). To achieve this, a newcomer uses a recursive greedy depth-first searching technique to find the potentially closest parent node. In this method, a newcomer measures the distance from itself to the potential parent (initially is the root node) and its children. The newcomer decides whether to attach to its current potential parent or to search further down the tree based on the measured results. If the nearest node found is one of the potential parent's children, the newcomer continues to search down the tree by using the nearest node as its new potential parent. A stack is used to store the potential parents found during the searching process. The process continues until the current potential parent is the nearest node. When this happens, the newcomer issues a join query to the potential parent. If the request is rejected, the newcomer retrieves the most recently visited potential parent from the stack and repeats the above processes. Once attached to the tree, each node periodically performs a re-join operation using the same procedures from a node that it randomly picks from the root path to improve the tree quality. During the re-join operation, the protocol allows a non-nearest member to be chosen as potential parent so as to explore other branches of the tree.

In HMTP, a heuristic is developed to handle the triangle problem (see Section 2.3.1). Referring to Figure 3 (a), assume that node C is the newcomer and it has found B as the nearest node during the searching operation. The protocol performs a triangle optimisation by letting C knows the distance between B and B's parent, A. Based on this information, C will try to attach to A if $d(A,C)$ is smaller than $d(A,B)$. Otherwise, it will attach to B. After C attaches to A, B may discover C in its next refinement process. This process is limited by the maximum number of children allowed by A.

3.1.4 Adaptive low-Cost, Delay Constrained (ACDC)

Another example of a switch parent protocol is ACDC [20]. ACDC is designed to build low cost and bounded delay trees for single source applications. The cost metric considered in [20] is independent of the delay metric. ACDC's system of constructing a probe set or switching targets distinguishes it from SwitchTree and HMTP. In order to maintain a targeted delay bound, each ACDC host keeps the information about the delay from the root and the largest delay to its descendents.

In the protocol, a newcomer first contacts the well-known RP to obtain a list of existing members. An on-tree member is randomly picked as its initial parent. Once attached to the tree, each member participates in two protocol phases: collection and distribution. The collection phase is used to gather membership information of the entire delivery tree while the distribution phase delivers this information to all the on-tree nodes. This allows a member to discover the other participants in the multicast group and uses them as switching targets.

In the collection phase, starting from the leaf nodes, each node informs its parent of the subset of its descendents along with other information. This information continues to propagate up to the tree root. The size of the message propagating from one node to another is kept constant. Once the root has received the information from all of its children, it begins the distribution phase by distributing a probe set to each of its children. This process ends at the leaf nodes. The collection and distribution phases constitute an epoch that is triggered when the previous epoch ends. When a node receives a probe set, it performs measurements to the target nodes and switches to the target if it will improve the optimisation objective, i.e. minimise cost while preserving delay bound. The probe sets are carefully formed such that the ordering of the tree is preserved, i.e. the transformation will not result in loop. Besides, the protocol ensures that over time, each node will probe all other nodes in the tree. The probe set construction process utilises the tree property that a tree is loop-less and rooted at a single node. Therefore, we named this technique as *tree-based discovery*.

As the switching process is based on a greedy decision, there are times that sub-optimum solutions can happen. Referring to Figure 4, now assume that node *N* can only achieve its delay bound by attaching to *r* but *r* cannot accommodate a new child. If one of *r*'s children, say *x* has *r* as its best possible parent, but can still maintain the delay bound by switching to another node, say *y*, ACDC allows *x* to switch to *y*. To assist with this solution, each ACDC host keeps a best alternative parent that it found during the measurement process and provides this information to its current parent. Based on the gathered information from all of its children, a parent node chooses a child as a *wean* target. A wean target will, in its next measurement process, try to switch to an alternate parent even if the new parent is not better than the existing one. This happens only if the delay bound is not violated.

3.1.5 HostCast

HostCast [21] is another variant of the switch parent protocols. It is designed for single-sender delay-sensitive applications such as media streaming. Similar to HMTP, all newcomers in HostCast treat the tree root as their initial potential parent. Unlike HMTP, there may be multiple potential parents at a time for a newcomer and no measurement is done during the joining process.

Initially, a newcomer sends a join request message to its potential parent (the root node). The potential parent replies with its current children and whether the request is accepted or rejected. If the request failed, the potential parent is stored in a list called non-potential parent list (NPL). Nodes in the NPL are sorted based on the time at which their replies are received. If the requesting newcomer has no more outstanding requests when a rejection was received, it retrieves the first node from the NPL. The children of the selected node will become its new potential parents. The requesting process continues until the first positive reply is received.

Once a HostCast node has found its parent (called a primary parent) in the tree, it establishes a set of peering relationships with its grandparent and parent's siblings to form a control mesh. These links are called secondary links and the corresponding peers are called secondary parents. Figure 7 (a) and (b) shows an example of a data delivery tree and its corresponding control mesh respectively. If the maximum number of children a node can handle is *k*, it can be shown that the maximum number of peering relationships (primary and secondary), i.e. control overhead, a node has to maintain is O(k^2).

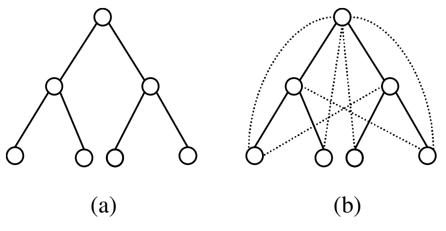


Figure 7: Data and Control Topologies in HostCast

HostCast uses its control mesh for tree improvement and fast failure recovery. Periodically, the tree root transmits small fixed size probe packets to its child nodes in the mesh. If a node receives a probe packet from its primary parent, it forwards the packet to all its children in the mesh. Otherwise, the packet will not be forwarded. Each probe packet carries a timestamp to estimate the delay and bandwidth of the root path. This is done using a moving averaging equation similar to TCP's round trip time estimation. With the measurements from the primary and secondary parents, a node can change its primary parent to improve the root delay. The above one-way measurement technique requires all HostCast hosts to have a synchronized group clock.

In order to solve the triangle problem, HostCast uses a substitution procedure that involves swapping position between a child and its parent. Referring to Figure 3 (a) again, node *C* can initiate a substitution process to achieve the configuration in 3(b) if its grandparent provides a smaller root delay (which should be true) and itself has free degree to accept another child. In addition, the protocol requires that the process can be done only if the number of descendents of the node is larger than that of its current primary parent. The substitution process involves messages to confirm the possibility of substitution and adjustments of the control mesh after the swapping.

3.1.6 Degree-Constrained Minimum Average-Latency Tree Protocol (DCMALTP)

In [6], the authors studied the problem of finding a degree-constrained minimum average-latency tree for single-sender real-time applications. They considered a network model where a number of Multicast Service Nodes (MSNs) are deployed to act as ALM forwarding entities for a set of clients. The objective is related to the minimum root diameter problem with additional consideration on the client size that each MSN has to serve.

Their solution consists of two phases. The first phase is the initialisation process where the root node uses a simple centralized computation to arrange the set of MSN nodes. The centralized algorithm uses only the degree bound of each MSN and the distances between the root node with all the participating MSNs. In the algorithm, the root node first sorts the list of MSNs in an increasing order of distance from itself. It then fills up the available degrees of MSNs in this increasing sequence.

After the initialisation, the MSNs try to perform a set of local transformations to improve the objective function. Figure 8 illustrates the set of transformations used. Panel (a) and (b) show the switch parent and parent-child swap transformations. Panel (c) shows a swapping process between two nodes having a same grandparent, this is called Iso-level-2 swap. An iso-level-2 transfer process can be performed if a node, say *x* found that there is free degree at the same level. In panel (d), a swapping operation happens between two nodes that are not on the same level of the tree. This is called aniso-level-2 swap. Each of the local transformation is performed only if the resultant arrangement reduces the optimisation objective.

As these local transformations involve only close neighbours on the trees, it is possible to drive the solution to a local minimum. In order to try to achieve global optimisation, the protocol considers a transformation that involves a node, say *x*, with a randomly chosen non-descendent node. The swap target is selected using a random walk technique as described in Yoid (see Section 3.1.1). When the target is found, *x* performs a swap operation if the operation improves the objective function. Otherwise, *x* uses a simulated annealing technique to probabilistically decide if the operation should be done.

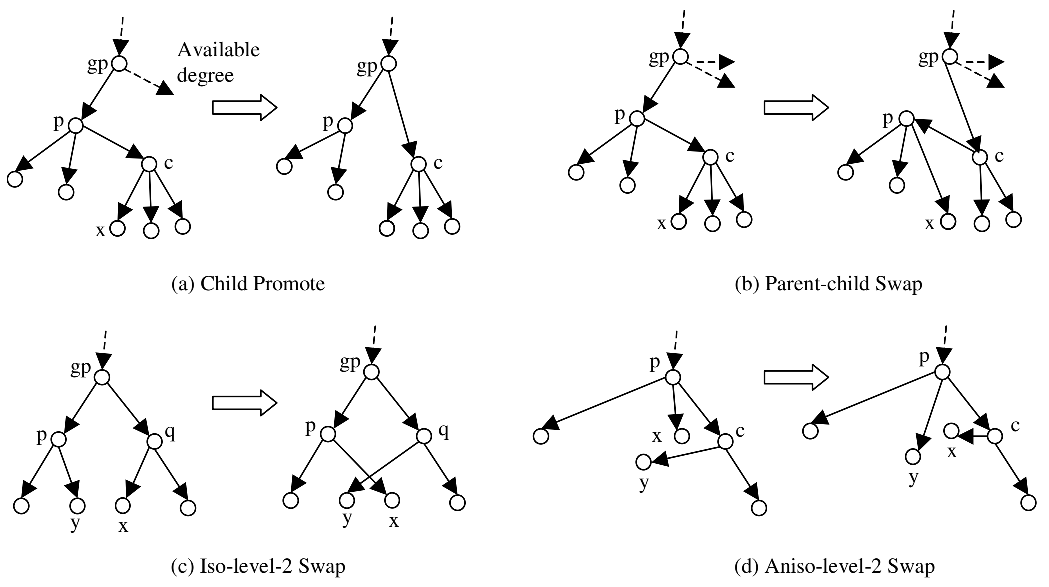


Figure 8: Local Transformations in DCMALTP

3.2 Local Region Rearrangement-based Protocols

This is a class of protocols that allows nodes within a local region in a tree to rearrange their configurations for better performance. We can view this as a variant of the transformation-based protocols as the rearrangement process also involves switching and swapping operations. However, unlike transformation-based protocols where the operations are computed and performed independently by each member, a representative node is used to compute and distribute the new configuration for the local region. Currently, only TBCP [25] is classified under this class of protocols.

3.2.1 Tree Building Control Protocol (TBCP)

TBCP is a generic tree building protocol. Like HMTP, it is designed to reduce the convergence time by building a "good" tree as early as possible. Unlike HMTP and other previously mentioned protocols, it uses the local region rearrangement technique to build the delivery tree.

In the TBCP joining process, a newcomer estimates the distance from itself to the potential parent (initially the root node) and its children. The distance information is then sent to the potential parent. On receiving such information, the potential parent consults a set of potential local configurations and calculates a score value for each configuration. The configuration with the best score value is chosen to rearrange the nodes involved. Figure 9 depicts the set of local configurations tested. In the figure, node *N* is the newcomer, *P* is *N*'s current potential parent and node 1, 2 and 3 are *P*'s children. Based on the selected configuration, if *N* can attach to *P*, it will be sent an acceptance message. If *N* or one of the *P*'s existing children (we called it a victim node) has to move down the tree, it will be informed of a new potential parent. The victim node will perform the joining process from the new potential parent. We give an example based on the configuration in the left panel of Figure 9 (c). In the configuration, *P* will accept *N* as a new child while forcing node 3 down the tree. To achieve this configuration, *P* issues an acceptance message to *N* and a redirection message with new potential parent as node 1 to node 3.

TBCP uses a configurable score function to evaluate the goodness of a configuration. Therefore, the final shape of the resultant tree depends on the function and the metrics involved. In [25], the following score function is chosen.

$$\textit{score function} = \max_{m\in(C\cup\{N\})} D(p,m) \qquad \text{Equation 2}$$

where *D*(*i*,*j*) is defined as the distance between node *i* and *j* along the tree, *p* is the current potential parent, *C* is the set of the *p*'s children and *N* the newcomer.

Based on the score function, the configuration with the smallest score value is chosen. It is an attempt to minimise the maximum overlay distance in the set of local configurations, and thus, hopefully will achieve global optimisation. If the score function is suitably chosen, TBCP can overcome the triangle problem in the local region. No tree improvement mechanism is proposed in [25].

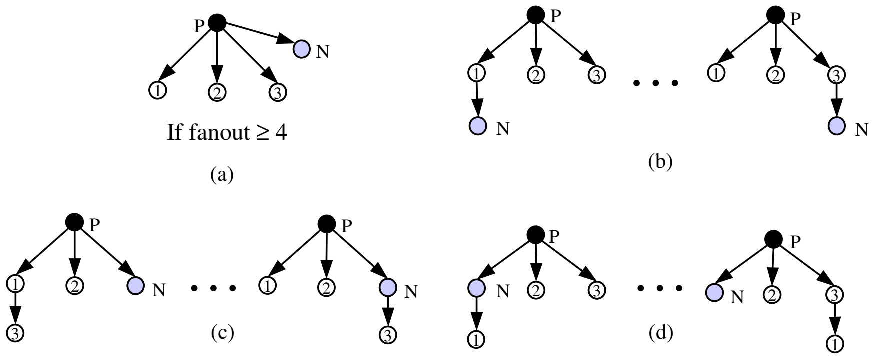


Figure 9: TBCP Local Configurations

3.3 Hierarchical Cluster-based Protocols

Hierarchy has long been viewed as an efficient way to achieve scalability in network routing protocol design. NICE [4], ZIGZAG [34] and ZHCP [26] are three distinctive protocols which try to achieve low control overhead by building an overlay of hierarchical clusters. All these protocols build the hierarchy of clusters in a recursive manner. The hierarchical clusters serve as the control structure for membership management, fault recovery and performance refinement. The data delivery tree is derived from the cluster relationship among the members.

3.3.1 NICE

NICE [4] is designed for low bandwidth multi-sender applications with a very large member populations. In the protocol, group members are arranged into a set of hierarchical layers of clusters. The layers are numbered from 0 to the highest layer, *n*. A single parameter, *k* is used to bound the size of the cluster to between *k* and $3k - 1$. If the number of members is given as *N*, the height of NICE hierarchy is at most $O(\log_K N)$ where *K* represents the maximum number of cluster members (i.e. $3k - 1$).

In the protocol, all group members join the lowest layer (layer 0) clusters. Within each cluster, a representative node called the cluster head is elected. Collectively, the cluster heads at layer 0 form a set of layer 1 clusters. A set of cluster heads is elected from layer 1's clusters, which form the next higher layer clusters. This process is continued until the top-most layer where only a single cluster exists. Based on the arrangement described, it is clear that if a node is present in some cluster in layer *i*, it must also present in one cluster in each of the layer 0 to layer *i*-1. In other words, if a node is the cluster head at layer *i*, it must also be a cluster head at each of the layers below it. An exceptional case is the top-most layer where the cluster head is only present at the highest layer, and in no other layer. Figure 10 (a) shows an example of a NICE hierarchy.

In NICE, a newcomer begins its joining process from the top-level cluster head which is also known as the bootstrapping entity. If the current hierarchy has only one layer, the newcomer will join the existing layer. Otherwise, the cluster head returns a list of next lower layer cluster heads to the newcomer. The newcomer estimates its distance to these cluster heads and tries to join the cluster of the nearest one. If the selected cluster head is not at the lowest layer, the joining process continues further down the hierarchy. This process is similar to HMTP's depth-first search technique, except that a NICE host will always join the lowest layer. Once a newcomer has successfully attached to a cluster, it sends a periodical heart beat message to each of its cluster siblings to establish peering relationships.

Consequently, the control structure in a cluster is a fully connected mesh among the members (see Figure 10 (b)). Periodically, a node is also involved in a refinement process to try to find a nearer cluster to which it should attach.

A cluster head periodically checks the size of its cluster. If the cluster is too large (> 3*k*-1), it activates a cluster split operation. To split an existing cluster, the cluster head partitions the cluster into two equal-sized sub-clusters, such that the maximum radius (in the graph theoretical sense) of the new clusters are minimised. Then, the centre of each cluster is elected as new head. On the other hand, if the cluster is undersize (< *k*), the cluster head initiates a cluster merge operation with its nearest cluster peer at the highest layer it presents. By using the graph theoretic centre as the cluster head, NICE hopes to strike a balance between the tree cost and latency among the members.

The NICE protocol's hierarchical structures implicitly define the data delivery paths, which are a set of source-specific trees. In NICE data forwarding, when a node *h* receives a packet from a node *p*, it will replicate and forward the packet to all clusters of which it is a member at each layer, except the clusters where *p* is also a member. Figure 10 (c), (d) and (e) show three examples of data forwarding tree in NICE. In the figures, the black coloured nodes are the data sources. Due to this forwarding algorithm, a NICE host can have as many as $O(K\log_K N)$ peers in its data path. This also represents its worst-case control path overhead.

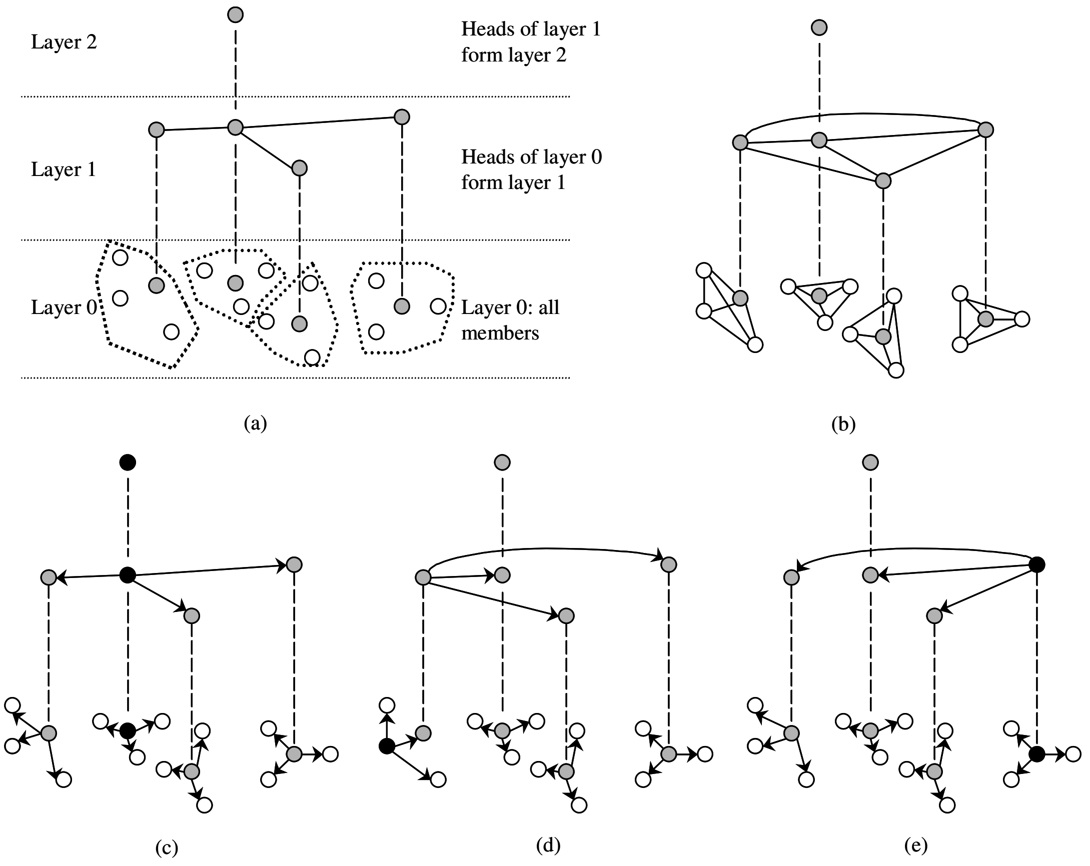


Figure 10: NICE Hierarchical, Control Structure and Data Forwarding Paths

3.3.2 ZIGZAG

ZIGZAG [34] is a recently proposed overlay solution targeted at latency sensitive single-sender media streaming. It uses a similar hierarchical structure as in NICE for control purposes, however, the way that the hierarchy is built is different from NICE. In addition, the distribution tree differs significantly from NICE. Borrowing the definitions for *K* and *N* from the previous section, the height of ZIGZAG's multicast tree is similar to NICE, i.e. $O(\log_K N)$. Unlike NICE, its maximum number of data path peers is given by $O(K^2)$ and it has a lower worst-case control overhead of

$O(log_K N)$.

To explain the data delivery mechanism in the protocol, the following definitions are needed.

- Foreign head – Non-head cluster peers of a node x at layer $j > 0$ is the foreign head of x's layer j-1 members. For example, in Figure 11, non-head cluster member of node 2 at layer 1 ($j > 0$) is node 1. Consequently, its cluster members at layer 0 ($j – 1$) regard node 1 as their foreign head. Similarly, cluster members of node 3 at layer 0 regard node 2 and 3 as their foreign heads.
- Foreign member – Cluster members of a node x at layer j –1 are called foreign members of any x's layer j cluster-mate. For example, in Figure 11, node 1 at layer 1 regards cluster members of node 2 and 3 at layer 0 as its foreign members.

In ZIGZAG, all nodes, when at their highest layer, can serve only their foreign members. An exceptional case is the data sender which is also allowed to serve its own cluster members at its highest layer. In other words, if the highest layer of a node is n, it is only allowed to serve its foreign cluster members at layer $n – 1$ (except the data sender). Figure 11 illustrates a sample of ZIGZAG distribution tree.

Similar to NICE, a newcomer in ZIGZAG needs to join to one of the lowest layer (i.e. layer 0) clusters. The joining process of a newcomer, P begins from the root node. If there is only one single layer at the time of joining, P will become a cluster member of the root and thus, receives data from it. Otherwise, the request is redirected along the multicast tree in a recursive manner similar to NICE. However, instead of finding the nearest node at each stage of redirection, the objective is to find a node that gives the lowest root delay.

Like NICE, ZIGZAG permits a cluster to grow over or under the size bound (i.e. $k \leq$ cluster size $\leq 3k$-1). Unlike NICE, the splitting or merging operation involves only the corresponding cluster, its immediate higher layer cluster from which it receives data and some of its immediate lower layer clusters due to the multicast tree rules. In addition, the procedures used are designed to minimise the number of reconnections due to the split. ZIGZAG provides two refinement techniques to improve the tree quality. The main objective of the technique is to load-balance the service size of nodes within a cluster. The first technique, called degree-based switching, simply considers only the service size in the load-balancing process. In the technique, a node that serves many other nodes will try to send some of them to one of its cluster members with the smallest service size. The second technique, called capacity-based switch, takes into account the bandwidth of a node when making a switching decision.

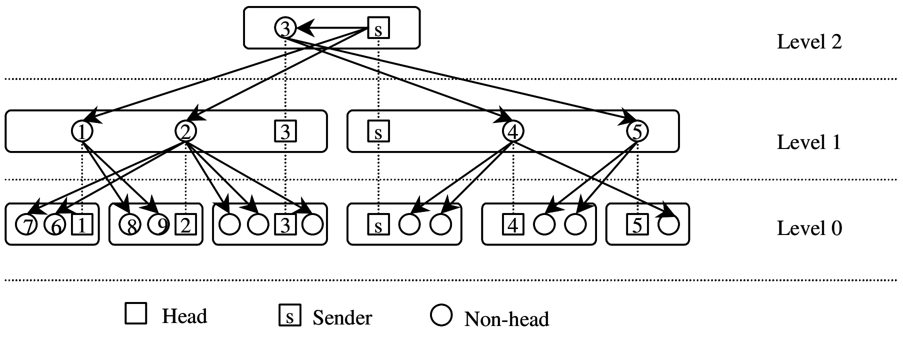


Figure 11: ZIGZAG Data Delivery Tree

3.3.3 Zone-based Hierarchical Clustering Protocol (ZHCP)

In [26], the authors proposed a hierarchical clustering protocol using the innovative notion of zones. Consequently, we refer to the protocol as Zone-based Hierarchical Clustering Protocol (ZHCP). Similar to NICE and ZIGZAG, ZHCP organises multicast members into a hierarchy of clusters. However, the clusters created are unconstrained in size, i.e. the cluster head has no explicit control of the number of members within its cluster. Due to this, the overlay tree constructed by ZHCP is not suitable for data forwarding. Another difference is that the height of the hierarchy created is unbounded.

The principle concept of the protocol is that all members belong to a top-level cluster that is rooted at a well-known node. Members are recursively grouped into smaller sub-clusters, until all clusters obtained are *singleton-clusters* containing only one member. The grouping of members is based on the notion of zones which is defined according to the distance around a cluster head. In [26], an exponential distribution is used to describe the set of zones around a node:

$$zone_0 : 0 < dist \leq 1 \qquad \text{Equation 3a}$$

$$zone_i : (1+\Delta)^{i-1} < dist \leq (1+\Delta)^i , with \Delta \geq 1 \qquad \text{Equation 3b}$$

where Δ is a configurable parameter and *dist* is the distance from the node.

Figure 12 (a) illustrates the zone concept of the protocol. Figure 12 (b) shows a sample of clusters rooted at node *R* and Figure 12 (c) shows the logical tree derived from the cluster relationship in (b).

In the protocol, a newcomer first tries to join to the well-known root node. The root calculates the associated zone for the newcomer based on the distance between itself and the newcomer. If the newcomer is the only node at that zone, it is assigned as the new cluster head of that zone. Otherwise, it is given a list of current cluster heads at the zone and a radius value. The radius value defines the size of the cluster at the zone. The newcomer then compares the radius value with the distance between itself and the nearest cluster head at the zone. If the nearest cluster head is out of the given radius value, the node becomes a new cluster head and informs the root. Otherwise, it continues the joining process from the nearest cluster head found. The above process continues until the newcomer becomes a singleton cluster head.

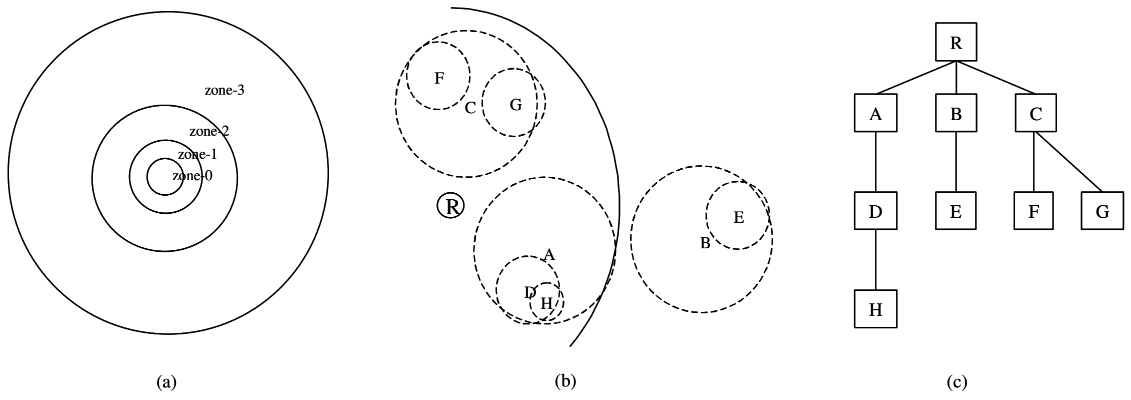


Figure 12: Zone Concept, Hierarchical Clusters and Resultant Logical Tree

3.4 Summary

We summarise the various protocols discussed in previous sections in Table 1.

Protocol	**Type**	**Tree-Type**	**Optimisation Objective**	**Optimisation Technique**	**Membership Discovery**	**Maximum Fan-out**	**Control Overhead**	**Maximum Path Length**	**Applications**
Yoid	Transformation	GST	Avoid routing pathologies	Switch-parents	Random walk + depth-first search	$k + 1$	O(k + mesh edges)	-	Generic content distribution
Switch-Tree	Transformation	SRT/ GST	Generic: Tree cost / root latency	Switch-parents	Local region probing	$k + 1$	Switch-one-hop: O(k) Switch-two-hop: O(k^2)	-	Generic content distribution
HMTP	Transformation	GST	Tree cost	Depth-first search	Depth-first search	$k + 1$	O(k)	-	Bulk data transfer
ACDC	Transformation	SRT	Root latency + tree cost	Switch-parents + wean	Tree-based discovery	k	O(k)	Delay constraint	Delay-sensitive applications: media streaming, interactive data.
HostCast	Transformation	SRT	Root latency	Switch-parents + parent-child swap	Local region probing	k	O(k^2)	-	Delay-sensitive applications
DCMALTP	Transformation	SRT	Minimum average latency	Switch-parents + swapping	Local region probing + random walk	k	O(k)	-	Delay-sensitive applications
TBCP	Local region rearrangement	SRT/ GST	Generic: Tree cost / root latency	Local region rearrangement	Depth-first search	$k + 1$	O(k)	-	Generic content distribution
NICE	Cluster-based	SRTs	Low control overhead for large multicast group	Hierarchical clustering	Recursive clustering	O($\log_K N$)	Max: O($K\log_K N$) Ave: O(K)	O($\log_K N$) overlay hops	Low bandwidth large-scale content distribution
ZIGZAG	Cluster-based	SRT	Root latency	Hierarchical clustering	Recursive clustering	O(K^2)	Max: O($\log_K N$) Ave: O(K)	O($\log_K N$) overlay hops	Delay sensitive applications
ZHCP	Cluster-based	SRT	Low control overhead for large multicast group	Hierarchical clustering	Recursive clustering	Unbounded	Unbounded	-	Control structure

Source rooted tree: SRT
Group shared tree: GST

k: maximum number of on-tree node's children
K: maximum cluster size in NICE and ZIGZAG

Table 1: Summary of Tree-first Protocols

4 Performance Evaluation

4.1 Simulation Set-up

We developed a simple packet-level, discrete event simulator for the evaluations. The simulator assumes a shortest path routing between any two nodes in the network. The network is modelled as a connected graph $G(V, E)$ where V and E are the set of nodes (routers) and links in the network respectively. Each link is associated with a propagation delay. In order to facilitate the evaluation of large networks, the simulator does not model either queuing delay or packet losses.

We used both Waxman and Transit-stub network models [35] available in the Georgia Institute of Technology's random graph generator [1] to generate topologies used in the evaluations. The topologies range from 1000 routers to 2000 routers with average degree of 3 to 4. The multicast members are randomly chosen and attached to the routers. We assume that there is only one member per router and the last hop delay from the member to the router is negligible.

In the simulation, we assume that a well-known RP is available to store the list of members that already joined the multicast session. The first member that joins will become the root of the overlay tree. As our main objective is to study the efficiency of different optimisation techniques, we let each member join the group one at a time. Once attached to the overlay tree, each member independently performs a refinement every 30 seconds (simulator time). Our results were collected from ten independent runs of each simulation scenario.

We study two multicast service models: single-sender and multi-sender. In the single-sender model, we assume that the root of the delivery tree is the traffic source; for the multi-sender model, we assume that all members are data senders.

4.2 Protocols

In the evaluations, we concentrate on two optimisation objectives: latency and tree cost (see Section 2.3). We consider the following protocols: HMTP, a modified version of DCMALTP, some variants of SwtichTrees, TBCP and NICE. These protocols are chosen as representatives of the three classes of protocol discussed in Section 3. For transformation-based protocols, Yoid is excluded as it does not try to optimise for either tree cost or latency. On the other hand, we exclude ACDC as we do not consider the bounded delay problem here. Cluster-based protocols have a different maximum fanout definition (see Table 1) compared with other protocols. In addition, an implementation of NICE is publicly available [3] which enable us to compare and verify our implementation. Therefore, we decided to use NICE in the evaluations.

The following configurations are used for all experiments unless specified otherwise. For NICE, the parameter k is set to 3, this results in a maximum cluster size of 8. For other protocols, the maximum number of children is set to 8.

For TBCP, the score function as in equation 2 attempts to minimise the maximum overlay delay in local regions. It is easy to see that there may be more than one such configuration if one of the overlay distances dominates the others. In order to break a tie, we select the configuration that gives the smallest configuration cost. If we let E_i be the set of overlay links in configuration i, $d(e)$ again be the delay of overlay link e, then the configuration cost, $C(i)$ is given by,

$$C(i) = \sum\nolimits_{\forall e \in E_i} d(e) \qquad \text{Equation 4}$$

Equation 4 is also used as the score function for studying the efficiency of TBCP in optimising the tree cost. We use `TBCP-delay` and `TBCP-cost` to distinguish these two versions.

We have considered several variants of transformation-based protocols described in Section 3. For switch-trees protocols, we used only the switch-one-hop and switch-two-hop approaches. This is because the switch-sibling technique allows only limited switches and thus, is not efficient. On the other hand, as the switch-any technique imposes no limitation on the scope of the switch targets, it is not a practical solution. As the original switch-trees protocols do not attempt to alleviate the triangle problem, we include a swap operation that allows a child node to swap position with its parent if a triangle problem is detected. Therefore, our version of the switch-two-hop protocol that tries to optimise the root delay acts as an instance of the HostCast protocol. Since switch-two-hop considers more target nodes than HostCast, we expect it to perform better than the original HostCast protocol. We also implemented a protocol that performs the set of local transformations used in DCMALTP (see Section 3.1.6). In this version, the objective is to minimise the latency from the root node. To achieve this, each overlay node maintains its maximum sub-tree delay. A swap or switch operation is performed only if the operation will not increase the maximum sub-tree overlay node measured from the grandparent. We refer to this refinement approach as `localtransform`.

For the class of transformation-based protocols, we consider three different join strategies:

- Root-based – In this variant, all newcomers are initially attached to the tree root. In order to preserve the degree constraint, the root node will inform some of these nodes to switch to other nodes in their first refinement process until it achieves the degree bound. In making the decision, the root tries to keep the nearest nodes as its children. This approach is similar to the simultaneous join approach studied in [15].
- Next-Available – This variant simulates the strategy used in HostCast. Beginning from the root node, a newcomer searches down the tree to attach to the first member that accepts its query.
- Random – In this variant, a newcomer randomly attaches itself to an on-tree member as long as the degree constraint is preserved.

We used the following convention to name the above protocols: <join-method>-<transformation-type>-<objective> where join-method refers to the initial joining methods (`root`, `nextavailable` and `random`) mentioned above; the transformation-type is one of the `onehop`, `twohop`, and `localtransform`; and the objective is either `delay` or `cost`.

For HMTP, we consider the original version that a newcomer begins its join from the root node and a modified version that the newcomer randomly picks an existing member to join. This is to study the efficiency of the depth-first searching technique.

4.3 Benchmarks

In our experiments, we compare the optimisation efficiency of various protocols with some centralised algorithms. For latency optimisation, we use the Compact Tree algorithm (hereafter refer to as CPT) proposed in [32]. CPT is a greedy heuristic for the degree-constrained minimum tree-diameter problem. The algorithm constructs a degree-constrained tree, *T* in an incremental manner starting from the tree root. At each step when adding a new node to the existing component *T*, the algorithm searches for a node that minimises the current objective function value (i.e. tree-diameter). When we consider the single-sender applications, we modified the objective function in the algorithm to minimise the root-diameter. While this algorithm does not provide the optimal solution, it gives a guideline on the efficiency of the various protocols.

For tree cost optimisation, we compute the minimum cost tree using Prim's minimum spanning tree algorithm (hereafter refer to as MSP). The algorithm computes a minimum cost overlay tree using the complete distance information among the members. Therefore, the trees constructed by MSP are the best possible solutions without considering the fanout constraint.

In order to study the cost effectiveness of ALM protocols, we compare ALM approaches with the IP multicast solution and a multi-unicast connections solution. We use source specific shortest path trees computed at the network layer to simulate the DVMRP [10]. The network layer delivery tree is the most efficienct way of multicasting, it enables us to study the penalties (see Section 4.4) incurred by the ALM approaches. In the multi-unicast streams solution, each receiver connects to the sender using an individual unicast stream. While this solution provides the best delay performance, it results in huge bandwidth wastage due to unnecessary packet duplication. In the next sections, we refer to this solution as UnicastStar.

4.4 Performance Metrics

We use the following performance metrics to evaluate the protocols.

- Stress – Stress is defined as the number of copies of an identical packet sent over a single link.
- The delay performance is evaluated in terms of stretch or relative delay penalty (RDP) incurred by ALM solutions over network layer multicast solutions. We used two version of RDP proposed in [7]:
    - $RMP = \frac{\text{maximum overlay delay}}{\text{maximum unicast delay}}$
    - $RAP = \frac{\text{average overlay delay}}{\text{average unicast delay}}$
    - In these metrics, RMP shows the worst-case latency observed by a multicast member while RAP shows the average latency observed by all members for an ALM solution.
- $Tree\,Cost\,Ratio = \frac{\text{overlay tree cost}}{\text{IP multicast shortest path tree cost}}$
    - The overlay tree cost is given as equation 1 at Section 2.3.1. The IP multicast tree cost is the sum of the delays of the physical links used to form the delivery tree.
- Join overhead – Join overhead refers to the number of nodes that need to be contact when a newcomer joins a multicast session.
- Control message overhead – We measure the control message overhead as the average number of bytes of

control packets seen by a network router.
- Convergence of the protocols – We investigate the convergence speed of various protocols.

5 Results And Discussions

In this section, we present and discuss the simulation results. We note that while the absolute performance of various protocols may differ under different network models, the relative performance trends observed are quite consistent across the topologies tested. We present the results obtained with the Transit-stub model and state the similarities and differences in the performance observed.

We divide the results and discussions into six subsections. We first discuss the results for single-sender and multi-sender service models. Then we look at the effect of fanout on the protocols. Next we study the convergence speed of the protocols. Finally, we investigate the protocol overhead during joining and refinement processes.

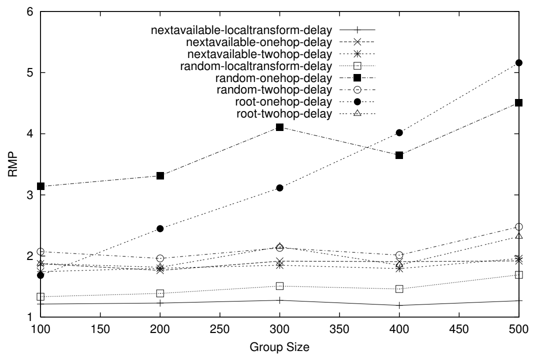
5.1 Single-Sender Model

We begin by presenting the results for RMP and RAP. We first consider the class of transformation-based protocols with delay as their optimisation objective. Figure 13 (a) shows the results. From the figure, we can observe that both local transformation-based solutions (`random-localtransform` and `nextavailable-localtransform`) give the lowest RMP value. This is followed by the switch-two-hop and then, switch-one-hop. This confirms that the performance is improved with the flexibility of the transformations [15]. In terms of joining strategy, the `root` and `nextavailable` versions are generally better than the `random` version. In the root-based strategy, all newcomers are initially siblings (with the root as the common parent) before they switch to other nodes. This provides more choices during the switching process. Therefore, it is more likely to build a reasonably good tree. However, the efficiency comes with the price of having more measurement overhead at the initial stage. In addition, the results of `root-onehop-delay` show that limited switching options can cause worse performance than the `random-onehop-delay` for a large group size. The initial tree built by the root-based strategy is more compact than the random approach. This together with a smaller target set restricts the movement of a node. In the next-available approach, a newcomer is attached to the tree to the first nearest on tree node that has free degree. Consequently, the initial tree layout is partially clustered which leads to a better performance than the random strategy. In the following sections, we use the local-transformation approaches to represent the class of delay-based transformation protocols.

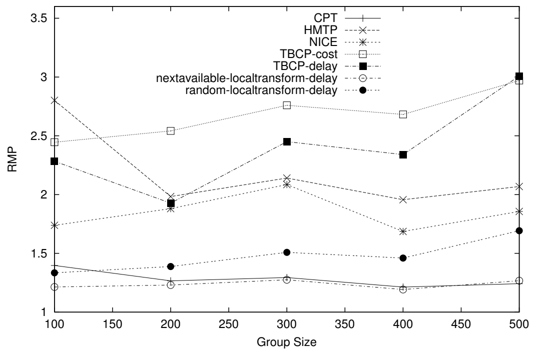
Figure 13 (b) depicts the performance of other protocols in term of RMP together with `random-localtransform` and `nextavailable-localtransform`. From the figure, we can observe that the `nextavailable-localtransform` is comparable to the centralised CPT algorithm. The results also show that the `TBCP-delay` is not able to provide competitive performance with other delay-based solutions. Results from other topologies reveal that this protocol can achieve good RMP for small group sizes, but the performance degrades with increasing member size. This is mainly due to the greedy score function (see Equation 2) that tries to achieve an optimum arrangement in the local area. We provide a detailed analysis for this in the next section. While the results in Figure 13 (b) show that NICE performs better than `TBCP-delay` and HMTP, results from other topologies indicate that there is no clear winner from these three protocols.

It is interesting to note that HMTP gives the best delay performance compared to other cost-based solutions. The performance advantage is more obvious under the Transit-stub model. We believe this is because the depth-first searching technique used by HMTP can better explore the clustering properties of the topology compare with other methods. Other cost-based protocols give a RMP value that ranges from 2 to 6 across the group size. The relative performance among these protocols varies from one topology to another.

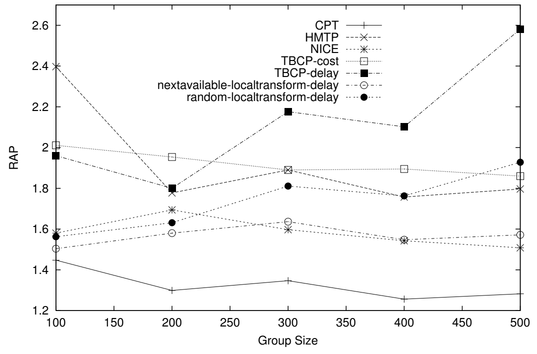
Figure 13 (c) depicts the average latency in term of RAP. The figure shows that CPT is now providing the best result. We see that NICE and `nextavailable-localtransform` solutions show quite similar performance.



(a) RMP vs. Group Size



(b) RMP vs. Group Size



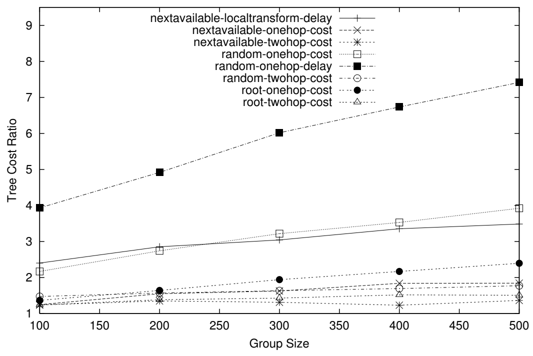
(c) RAP vs. Group Size

Figure 13: Delay Performance

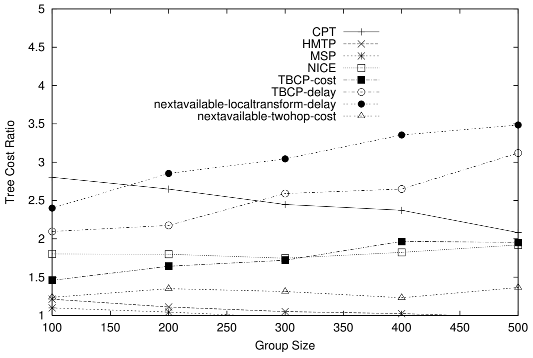
We now look at the performance of various protocols in terms of tree cost or resource usage. In general, the relative performance of the cost-based solutions across the topologies is quite consistent. Figure 14 (a) depicts the results for the class of cost-based transformation protocols. First, in terms of joining strategy, we can see that the performance in degrading order are `nextavailable`, `root` and `random`. In most topologies tested, the performance difference between `nextavailable` and `root` is quite small. The small difference shows by `nextavailable-onehop` and `random-twohop` indicates that the initial tree has great influence for the overall performance. In the following sections, we choose to show the results of `nextavailable-twohop-cost` approach to represent the best cost-based SwitchTree protocols. In the figure, we also include the results from the delay-based `nextavailable-localtransform` and `random-onehop` approaches. The reason is to show that failure in optimising delay (i.e. `random-onehop`) may lead to more resource consumption. The tree cost ratio of other delay-based approaches range from 2 to 4.

Figure 14 (b) plots the tree cost performance for other protocols. We include the results for the minimum spanning tree (MSP) as the optimum solution. The results show that HMTP provides the lowest tree cost in comparison with other distributed protocols. It is followed by `nextavailable-twohop-cost`. `TBCP-cost` and NICE show similar performance across the group size. In general, delay-based protocols give higher tree cost. This confirms with the well-known delay-cost tradeoffs. The centralised CPT is able to exploit the clustering property in the underlying topology when the group size increases. Consequently, a decrease in tree cost is observed. The converse is observed for `nextavailable-localtransform` that tries to optimise the tree latency. In addition, the tree cost performance given by `TBCP-cost` and `TBCP-delay` also increases with the group size. This is because the local region reconfiguration method in the protocol is unable to explore large overlay population.

We note that the UnicastStar overlay gives a tree cost ratio ranges from 6.5 to 8.5 across the group size. It is not included in the graph for a better view of other curves. This confirms that the tree cost or resource usage of most overlay solutions is reasonably low.



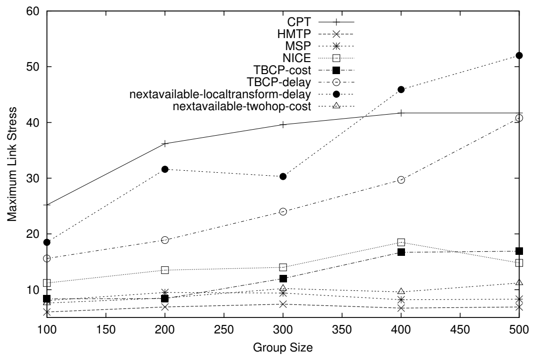
(a) Tree Cost Ratio vs. Group Size



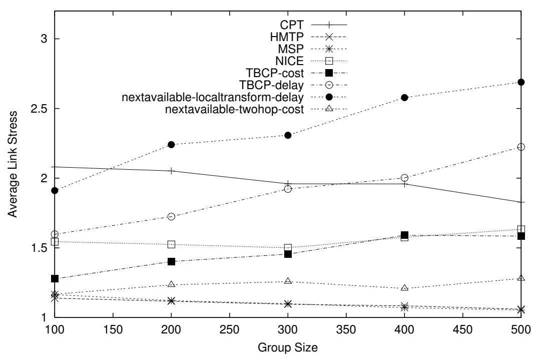
(b) Tree Cost Ratio vs. Group Size

Figure 14: Tree Cost Performance

Figure 15 (a) and (b) depict the link stress given by the protocols. In general, the results show the tradeoffs between link stress and delay. More specifically, protocols that were designed to minimise latency result in high link stress and vice versa. The growth in link stress with the group size is much faster in delay-based solutions compared to cost-based approaches. This is because delay-based solutions tend to build compact trees that use more redundant network links. Considering the results presented thus far, we can observe that the NICE protocol can strike a balance between delay and resource usage. Figure 16 plots the maximum fanout of MSP, NICE, HMTP and `TBCP-cost`. It is clear that the maximum fanout of HMTP, `TBCP-cost` or other constrained protocols is bounded. For MSP, the fanout value depends on the underlying topology. For NICE, the fanout of a node depends on the number of layers it has joined. We recall that the maximum number of data path peers for NICE is given as $O(K\log_K N)$ with N and K as the group size and maximum cluster size respectively. Consequently, neither MSP nor NICE is suitable for bandwidth intensive applications. In [5], an enhanced data path is suggested for NICE. In the enhancement, a cluster head only sends data to its lowest layer cluster members. The transmission to its upper layer is delegated to these lower layer members. This reduces the number of data path peers to K. While this approach can reduce the link stress and resource usage, it results in prolonged delivery latency. Moreover, in order to achieve efficient data delivery, the delegation needs to be carefully selected. Finally, UnicastStar overlay solution gives a very poor stress performance. In particular, the maximum stress value achieved is of the order of the group size with average stress values range from 4.5 to 6 (again, it is not shown in the graph). We can conclude that while this solution provides the best delay performance, it results in the worst resource wastage. In the following sections, we exclude the results from this solution.



(a) Maximum Stress vs. Group Size



(b) Average Stress vs. Group Size

Figure 15: Link Stress Performance

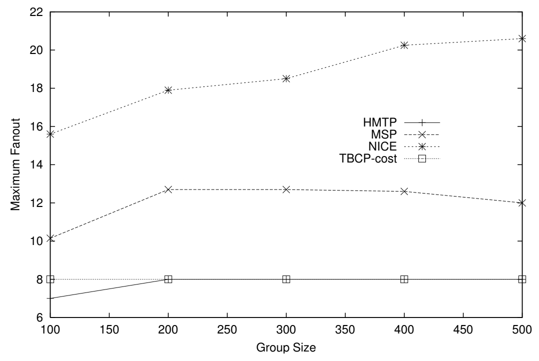


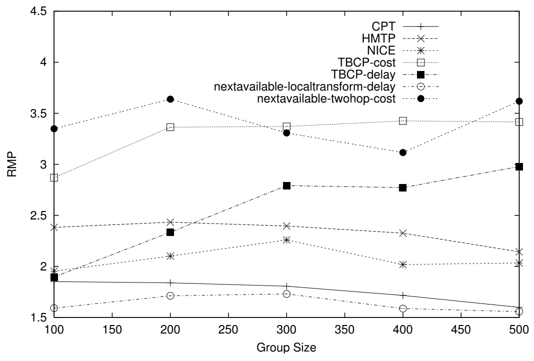
Figure 16: Maximum Number of Fanout vs. Group Size

5.2 Multi-Sender Model

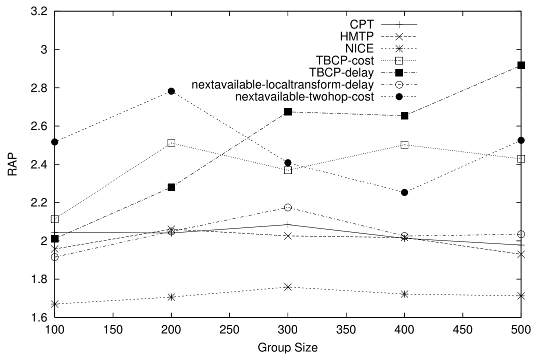
In multi-sender model, all protocols (except NICE) use group-shared trees for data delivery. Figure 17 (a) and (b) depict the RMP and RAP performance respectively. We can observe again that the `nextavailable-localtransform` is able to provide a smaller RMP value compared to the centralized CPT algorithm. It is interesting to note that the version of CPT used in this service model is optimised for tree diameter while the `nextavailable-localtransform` is optimised for root diameter. Again, the results show that `TBCP-delay`'s performance degrades with increasing member population. In terms of average latency, NICE performs the best as source-specific trees are used.

In our experiments, we use the same set of group members for both single- and multi-sender service models. Therefore, all protocols build the same trees in both scenarios. This results in similar relative performance for tree cost and maximum link stress as shown in Figure 14 and Figure 15 (a). While NICE uses a different tree (i.e. source-specific tree) for each sender, the difference in the trees depends on the highest layer in which the sender exists. Since we assume that all members are possible senders, this averages up the effect of members' positions. Consequently, the difference in the performance of tree cost and maximum stress is not great.

In terms of average stress, as the senders may come from any part of the shared tree, reverse physical links are also used in the data delivery. Consequently, we observed a smaller average link stress compared to the single-sender case (see Figure 14 (b) and Figure 18). For NICE, as the source specific tree within a cluster does not use the cost effective centre-based tree, it uses more redundant links. This results in the high average stress performance in NICE.



(a) RMP vs. Group Size



(b) RAP vs. Group Size

Figure 17: Delay Performance

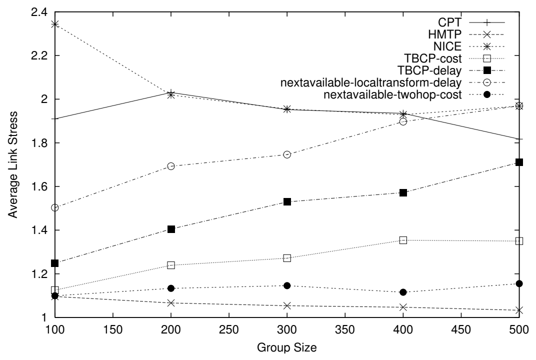


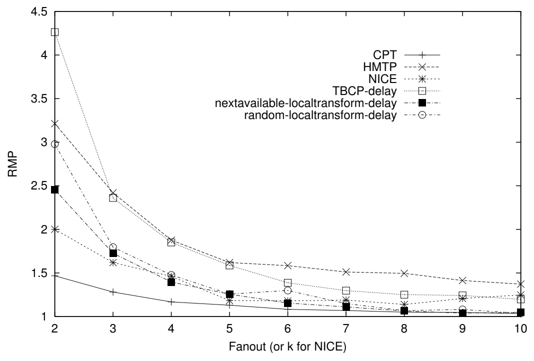
Figure 18: Average Stress Performance

5.3 Effects of Fanout

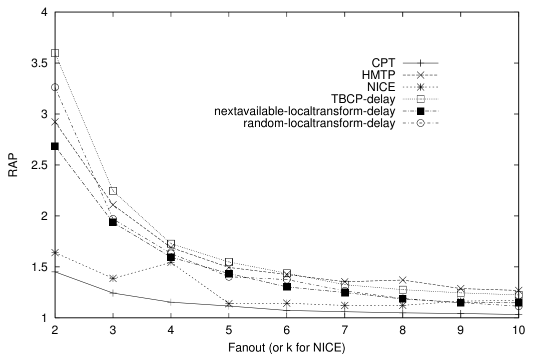
This section investigates the impact of fanout restriction under both homogeneous and heterogeneous environments. In homogeneous environments, we assume that all end hosts (multicast members) have similar interface bandwidth, and thus similar maximum fanout. On the other hand, a heterogeneous environment is characterised by members with diverse interface capacities.

5.3.1 Homogeneous Case

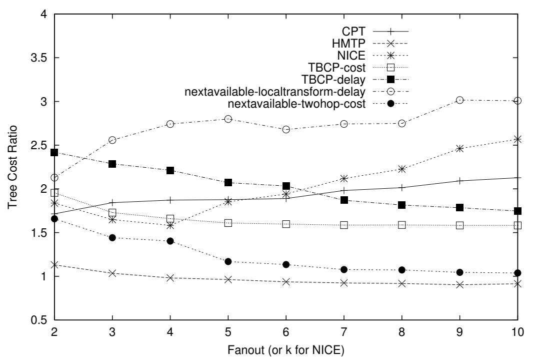
Figure 19 plots the results for the effects of various fanout values for experiments with 500 multicast members. In the figures, for NICE curves, the *x*-axis values represent *k*, the lower bound cluster size instead of fanout. In general, the latency performance is improved with increasing fanout value, as expected. The results also show that the fanout constraint has larger effect on `TBCP-delay` compared to other protocols. In terms of tree cost, larger fanout values result in better resource usage for cost-based solutions, e.g. HMTP and `TBCP-cost`. On the other hand, the performance trends shown by delay-based solutions are less obvious. In particular, we see that `TBCP-delay` produces trees with better resource usage when fanout increases. However, CPT and delay-based local transformation approach show an increasing tree cost with the fanout. NICE shows a distinct trend where the tree cost decreases initially, and increases after a turning point. In terms of link stress, we can observe that the fanout value has little influence for cost-based solutions. Minimising delay requires a tree to be as compact as possible. Consequently, this results in higher link stress when larger fanout values are used for delay-based solutions. For NICE, data is broadcast within a cluster. Consequently, larger cluster size increases the possibility of packet duplication on the physical links. Thus, link stress growth proportionally with *k*.



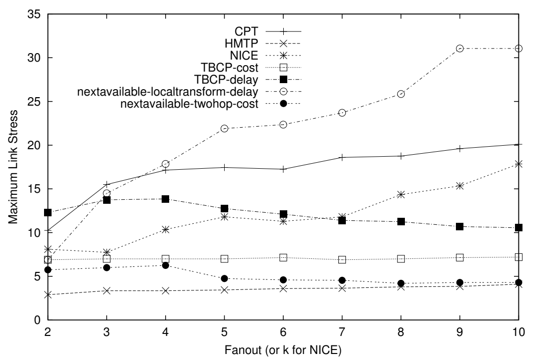
(a) RMP vs. Fanout



(b) RAP vs. Fanout



(c) Tree Cost Ratio vs. Fanout



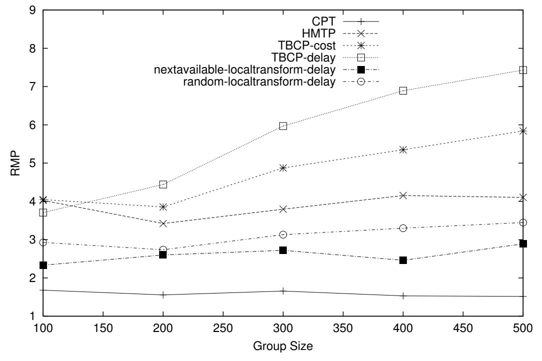
(d) Maximum Stress vs. Fanout

Figure 19: Performance Under Homogeneous Environment

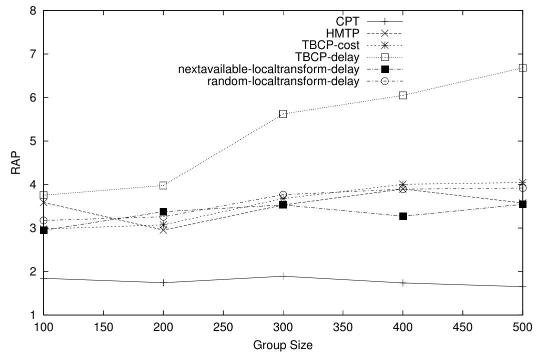
5.3.2 Heterogeneous Case

We simulate a heterogeneous environment by randomly assigning a fanout value ranging from 1 to 8 to each group member. NICE is excluded from the experiments, as we cannot control its fanout value directly. In addition, limited capacity at some nodes imposes a new constraint in the NICE cluster splitting algorithm, which will violate the invariant that all cluster heads must be the centre of the cluster. Our previous results reveal that the choice of cluster head in NICE provides reasonably good average performance for all members. Therefore, we believe that NICE is not suitable for a heterogeneous operating scenario.

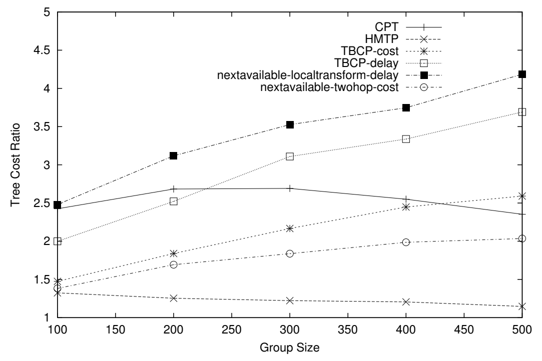
In general, we can observe that the RMP and RAP values for all protocols across the group size (see Figure 20 (a) and (b)) are higher than the homogeneous case as shown in Figure 13 (a)-(c). The results show that the `nextavailable-localtransform` approach is no longer comparable to the centralised CPT algorithm. We believe this is because the transformation made is regardless of the fanout constraint, consequently, if nodes with low fanout value are placed closer to the tree root, the resultant tree will growth faster in height. Focusing on `TBCP-delay`, we can see that the RMP and RAP values increase rapidly with the group size. This again, shows the deficiency of the greedy score function used. In terms of tree cost and stress (see Figure 20 (c) and (d)), there is no significant difference compared to the homogeneous case. HMTP remains as the protocol that provides the best tree cost and link stress performance. Comparing the performance of both cost-based and delay-based solutions, we can observe that optimising cost is relatively less affected than optimising delay in a heterogeneous environment.



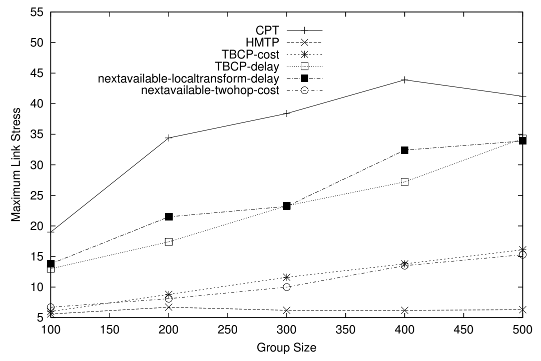
(a) RMP vs. Group Size



(b) RAP vs. Group Size



(c) Tree Cost Ratio vs. Group Size



(d) Maximum Stress vs. Group Size

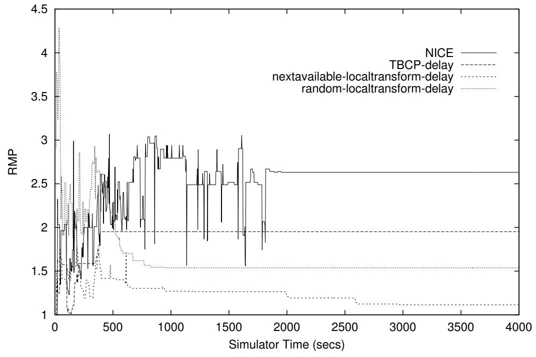
Figure 20: Performance Under Heterogeneous Environment

5.4 Protocol Convergence

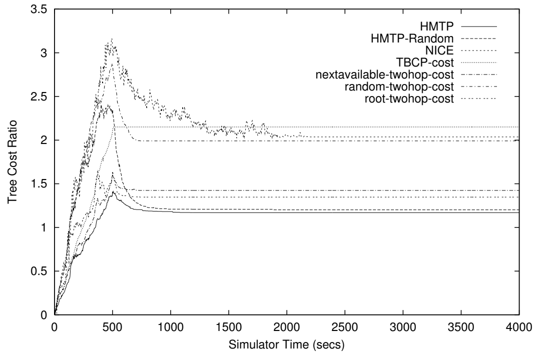
This section presents the convergence properties of the different approaches tested. We show the results obtained for a group size of 500. In the experiments, all members join the multicast session within the first 500 seconds and the refinement period is configured to 30 seconds. Figure 21 (a) depicts the convergence of RMP for delay-based protocols together with NICE. Since TBCP does not perform refinement, no change can be observed after all members join the session. For `random-localtransform`, due to the random join strategy, the initial RMP values may shoot up to a high value (about 4.28). However, we see that RMP value rapidly converges. For the `nextavailable-localtransform`, as the initial tree is partially clustered, the initial RMP values is always well below 2.0. The results from these two local-transformation techniques show that the initial tree is important to achieve a better overall result. The NICE protocol takes much longer times to converge. This is because the protocol tries to maintain the invariant that every cluster head must always be the graph theoretic centre of its respective cluster. A change in a lower layer cluster may result in changes in higher layer clusters. As a result, the protocol takes a much longer time to settle down.

Figure 21 (b) shows the changes of tree cost ratio with time for NICE and other cost-based solutions. In this experiment, we include a modified version of HMTP to study the efficiency of using its depth-first searching method for cost optimisation. We named the modified version as `HMTP-Random`. This is because in this version, a newcomer begins its join process from a randomly chosen on-tree member. Initially, the cost ratio of `HMTP-Random` is much higher compared to original HMTP, as newcomers are randomly attached. A similar effect is shown by the `random-localtransform-cost` approach. However, the fact that `HMTP-Random` quickly converges to its best value indicates the goodness of the refinement technique. The cost-based `nextavailable-twohop` and `root-twohop` also converge rapidly. Again, NICE takes the longest time to achieve convergence.

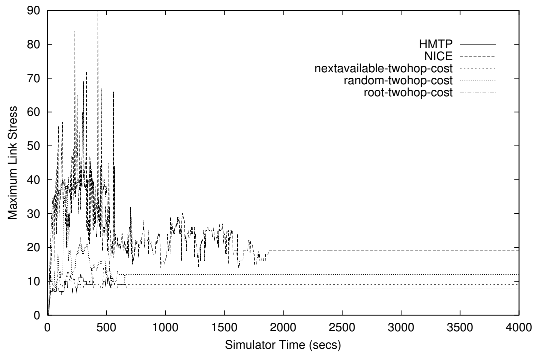
Figure 21 (c) shows the evolution of the stress value over the simulation time. We show the results for NICE, HMTP and three variants of switch-two-hop protocols to compare the stress imposed by different joining strategies. The root-based joining strategy and NICE give the highest stress at the initial stage. In the root-based join strategy, all newcomers are initially attached to the root node. When these nodes perform refinement (switching), they will detach from the root until the root achieves its fanout bound. Consequently, high stress values are observed for the initial stage. Although the NICE protocol restricts each cluster to a size of 3*k*-1, the splitting process is done only periodically. As a result, the maximum number of data path peers for any member may be high. This is as shown in Figure 21 (d). For the `next-available` strategy and HMTP, as the initial tree is more clustered, the stress values are comparatively smaller than other approaches.



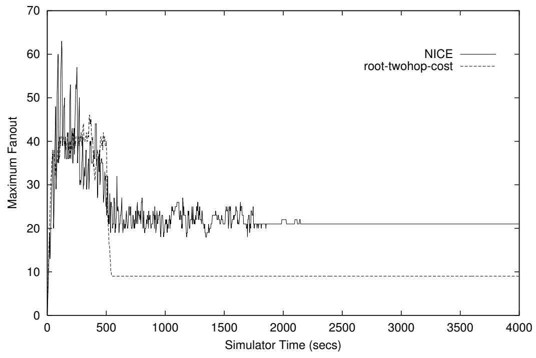
(a) RMP vs. Simulator Time (s)



(b) Tree Cost Ratio vs. Simulator Time (s)



(c) Maximum Stress vs. Simulator Time (s)



(d) Maximum Number of Peers vs. Simulator Time (s)

Figure 21: Convergence Performance

5.5 Protocol Overhead

In this section, we investigate the communication cost for various protocols. As discussed in Section 2.4, the communication cost can be divided into join overhead and maintenance overhead. We first look at these two costs individually, then we present the overall communication cost for two representative protocols.

5.5.1 Join Overhead

This subsection investigates the overhead of various join strategies. As discussed in Section 2.4, the join overhead involves the communication cost of the signalling and measurement messages. We consider only techniques involving end-to-end measurement which incurs much network traffic (i.e. HMTP, NICE, `TBCP-cost` and `TBCP-delay`, and `nextavailable-localtransform-delay`). The join overhead is measured as the number of nodes being contacted during the joining process of a node. We present the results for a multicast session with 500 members. In the experiment, new members are added one at a time.

In Figure 22, we only plot the results for HMTP, HMTP-Random, NICE and `TBCP-cost` to avoid the congestion of the graph. We note that the performance of `nextavailable-localtransform-delay` is close to HMTP. In the figure, we can observe that HMTP has the highest overhead compared to other protocols. In HMTP, newcomers use the depth-first method to search down the tree for an attachment point. The searching is unbounded and very much depends on the underlying topology. The worst-case for this technique happens when the overlay forms a chain, which is unlikely to happen in a realistic network. The worst overhead obtained from the experiments is about 40 for a group size of 500 or 8% of the overlay population. The randomised version of the protocol provides a slight reduction in the join overhead without much performance degradation.

Overall, NICE has the smallest join overhead for the first 300 seconds. After that, TBCP-cost gives a comparable result. The reason is that NICE allows clusters to grow over the size bound before the next cluster check period. Therefore, the initial tree is shorter than it should be. At a later time, more clusters and layers are created which result in higher join overhead. On the other hand, TBCP rearranges local regions to accommodate new members (see Section 3.2.1). Therefore, the join overhead of a new member is mostly determined by its position in the topology rather than the height of the tree.

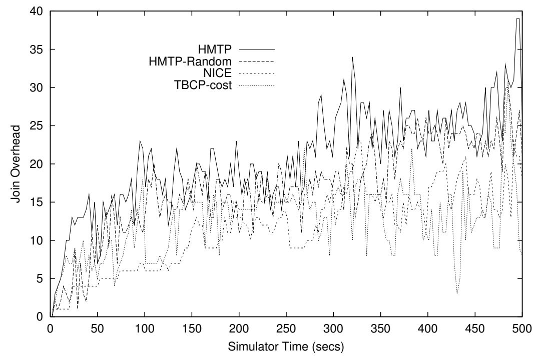


Figure 22: Join Overhead

5.5.2 Tree Improvement Overhead

This subsection compares the overhead of the protocols in terms of tree improvement policy. HMTP and NICE use techniques similar to their respective joining process for tree refinement. Consequently, the overhead is within the worst-case obtained from the previous section. The overhead for transformation-based protocols depends on the scope of their target set. If we assume the maximum fanout of a node as *k*, the probing overhead of each of these protocols is shown in Table 2.

Protocols	Overhead
Switch-sibling	O(*k*)
Switch-one-hop	O(*k*)
Switch-two-hop	O(k^2)
Local transform (DCMALTP)	O(*k*)

Table 2: Tree Improvement Overhead for Transformation-based Protocols

5.5.3 Overall Communication Cost

This subsection presents the overall communication cost for two representative protocols. The communication cost is measured as the average number of control messages seen by each router. We choose to present the performance of HMTP and NICE. HMTP represents the class of protocols that use the parent-child relationship as the control structure. Other protocols grouped in this class are SwitchTree, DCMALTP and TBCP. On the other hand, NICE uses hierarchical clusters as the control structure.

Figure 23 depicts the results. In general, we can observe that the communication cost is reasonably low for both HMTP and NICE, i.e. within 1kbps. In [4], the authors compared NICE with Narada, a mesh-based ALM protocol.

Their results show that the control overhead of Narada is about 200kbps for a group size of 512 while NICE is about 2kbps. The difference in the overhead observed for NICE in Figure 23 and the published results may be due to differences in the underlying topology and the refreshment period used.

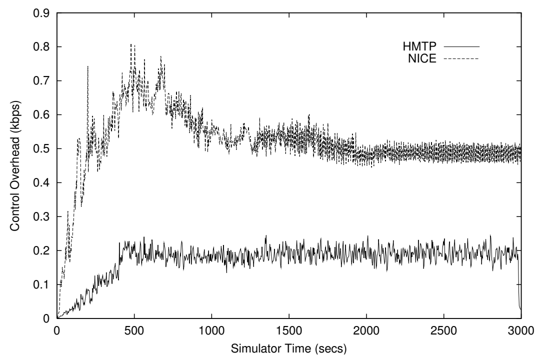


Figure 23: Overall Communication Cost for HMTP and NICE

6 ISSUES WITH TBCP

In the previous section, we presented the results of various protocols in tree cost and latency optimisation. The results reveal that while TBCP can provide reasonably good tree cost performance, the delay performance is not as promising as other solutions. Given the fact that TBCP allows flexible rearrangement within a local region, we believe that the protocol can be extended to improve its latency performance. In this section, we provide an analysis and propose an enhancement to the protocol for delay optimisation.

In [25], the score function used tries to minimise the maximum overlay distance within the local region. In the score function, if the distance of one of the overlay links within the configuration dominates other links, many configurations will give the same score value. Figure 24 illustrates the scenario where the link between node p and c_1 is the dominant link. Panel (a)-(d) in the figure show some examples of configurations that give the same score value. There are several possible ways to break the tie. We have chosen the configuration cost (see equation 4) as the tiebreaker. From our experiments, we have observed that this choice always outperforms its original version where the tie break is arbitrary. Another possible choice is to break tie in a lexicographical order. To explain this, we first assume that the overlay distances of a configuration are sorted in a non-increasing order into a vector $V(d)$. Therefore, the first component of the vector represents the score value of the configuration. For two configurations with equal score value, we compare their subsequent $V(d)$ components and choose the one that gives the first smaller value. While this or other choices may perform better than our selection, we argue below, a greedy score function as in Equation 2 is not suitable for delay optimisation.

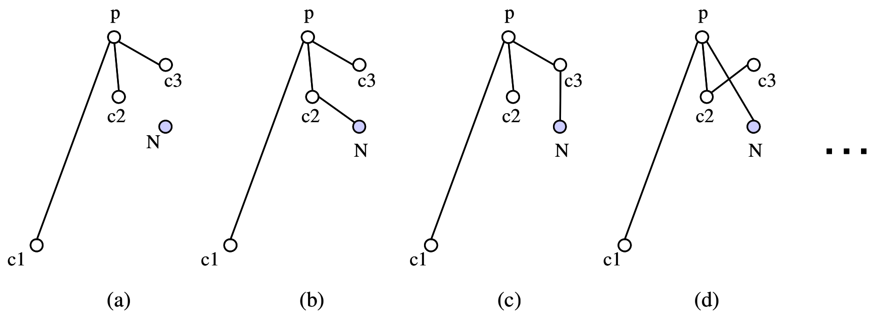


Figure 24: Dominant Link (p,c_1) in Local Configuration

The following discussion is based on an example given in igure 25. igure 25 (a) – (d) illustrates a sequence of tree construction stages using the score function described in section 3.2.1. The example assumes that each node has a maximum fanout of three and *p* is the root of the tree. In panel (a), node *N*1 is the newcomer that wishes to join the tree. We assume that the distance of *d*(*p*,*A*) and *d*(*p*,*B*) are the same and are the maximum in the local configurations. We then further assume that the distance *d*(*p*,*N*1) + *d*(*N*1,*C*) is smaller than *d*(*p*,*C*) + *d*(*C*,*N*1). Therefore, the resultant configuration will be as in panel (b). Now, another newcomer, *N*2 tries to join the tree. As the local configurations only consider the arrangements involving *p*, *N*2 and *p*'s children (i.e. *N*1, *A*, *B*), the best local configuration is as shown in panel (d).

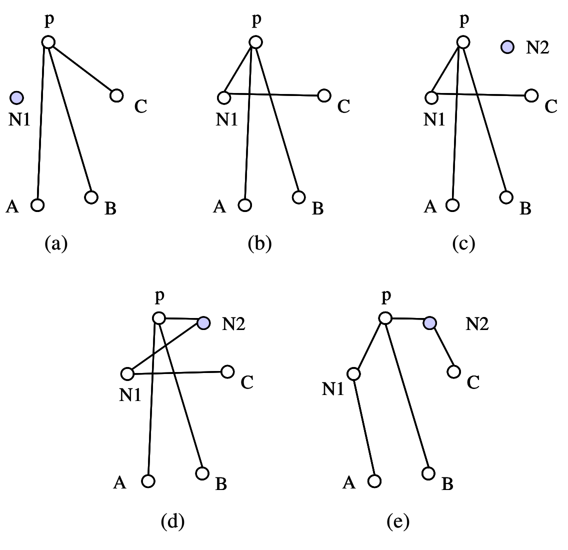
The example above shows a major problem of the greedy score function. In the protocol, the set of configurations used only involves the local region around the newcomer, the current potential parent and its children. A greedy score function such as equation 2 that attempts to provide the best local configuration, will often result in inefficient resource usage. The problem gets worse when the member population increases. Consequently, the resultant tree is not able to achieve the target objective. A better solution for the example would be the configuration shown in panel (e).

We propose a new score function for TBCP. Our score function is aimed to minimise the triangle cost within the local region. Referring to the set of configurations shown in Figure 9, only one change is done in each configuration. Our score function only evaluates the changed segment within a configuration. To explain the score function, we use Figure 26. In the figure, *p* is the potential parent, *x* and *y* are nodes involved in the change. Our score function for a configuration *c* is given as,

$$score(c) = \frac{d(p,x) + d(x,y)}{d(p,y)} \qquad \text{Equation 5}$$

The score function gives the ratio between the distance from *p* to *y* via *x* and the distance between *p* and *y*. We choose the configuration that gives the smallest score value. Effectively, we are trying to minimise the triangle cost created by the change. Figure 27 shows the performance of the new score function compared to TBCP that uses equation 2. In the experiment, we do not consider any improvement technique to the constructed tree. The results show that the new function can provide a good RMP value and at the same time, provide lower stress value.

We note that TBCP chooses the best local configuration that optimises the score function at each stage of contact which is largely determined by the relative distances among the set of nodes involved. Consequently, the resultant tree depends on the member joining sequence. Furthermore, the underlying network condition changes over time. Therefore, a tree refinement operation is inevitable to maintain and improve the quality of the delivery tree in a practical environment. An obvious extension of this work is to study the feasibility and efficiency of integrating the tree improvements techniques studied in this paper with the protocol.



igure 25: Sequence of Tree Construction Process

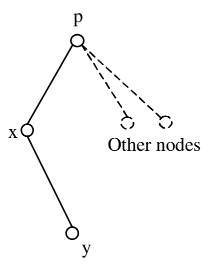
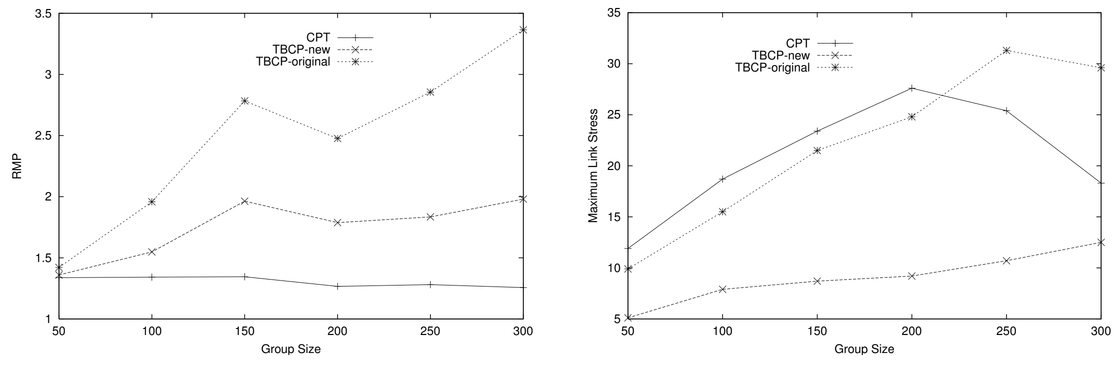


Figure 26: Changed Segment $\{p, x, y\}$ in Local Configuration



(a) RMP vs. Group Size (b) Maximum Stress vs. Group Size

Figure 27: Delay and Link Stress Performance

7 Related Work

This work studied the case for delay and cost optimisation in distributed overlay tree construction. Another important optimisation objective not considered here is bandwidth. Overcast [18] is targeted at bandwidth-intensive non real-time applications. As the bandwidth of a path is determined by the bottleneck link in a path, a bandwidth optimised delivery tree needs to have high bandwidth links nearer to the tree root and have low bandwidth links closer to the leaf nodes. Overcast tries to achieve this configuration by placing new members as far as possible from the tree root without sacrificing the bandwidth from the root. In addition, an Overcast host periodically performs bandwidth estimations between itself and its siblings, parent and grandparent. Based on the measured results, a node may move up or down the tree to maximise the path bandwidth.

Several studies consider the centralised model for overlay tree construction. ALMI [27], one of the earliest ALM proposals is designed for multi-sender applications with relatively small group size (several 10s of members). In ALMI, multicast members perform distance measurements among themselves. A centralised server is used to compute a minimum spanning tree from the measured results. The Host-Based Multicast (HMB) [30] is another centralised solution.

Narada [9] and Gossamer [8] are among the initial mesh-based protocols. In both protocols, the multicast members cooperate to form a connected mesh. Each member then runs an instance of a distance vector like protocol to disseminate the overlay topology information. Source-specific trees are used in the data delivery.

The HyperCast project [2] at the University of Virginia explores the use of *n*-dimensional hypercube and Delaunay triangulations for application layer routing. In [22], multicast members are organised into a logical *n*-dimensional hypercube structure. The hypercube contains 2^n nodes with each node is labelled by a bit string k_n, k_{n-1}, …, k_1 where $k_i \in \{0,1\}$. Nodes in a hypercube are connected by an edge if and only if their bit strings differ in exactly one position. By enforcing a particular ordering on nodes, source specific trees are embedded into the hypercube topology. In [23], Delaunay triangulation is considered. In the protocol, each overly node is assigned an (x, y) coordinate. Using the coordinate system, the overlay nodes infer the Delaunay triangulations to form the overlay topology in a distributed manner. Routing in the overlay is based on a geometry technique called compass routing. The topology created by both protocols is suitable for control purposes.

Another class of mesh-based protocol built on top of the infrastructure created using Distributed Hash Table (DHT) techniques. These include Scribe [31], multicast-CAN [29], Bayeux [37] and Chord [33]. DHT-based protocols provide efficient routing over an abstract namespace. Overlay nodes are mapped onto the namespace and act as keys for message routing. In order to achieve high scalability, these protocols construct overlays without using any measurement techniques. Scribe and Bayeux assume members are able to find close by members on the overlay through out-of-band mechanisms. In [17], the authors evaluate the potential of DHT-based overlays using extensive simulations. They compare multicast-CAN and Chord with NICE and Narada. Their results show that in the primitive form, i.e. without topology information, both multicast-CAN and NICE have a RDP value that is more than NICE and Narada by at least a factor of two. By coupling these two protocols with topology information, they show a comparable performance with NICE and Narada.

8 CONCLUSIONS AND FUTURE WORKS

In this paper, we have looked at various issues in distributed tree construction for ALM. We concentrate on the tree building and optimisation techniques used in some existing tree-first protocols. In particular, we investigate the efficiency of HMTP, TBCP, NICE and several variants of transformation-based strategies by simulations. These protocols represent the three classes of tree-first protocols: i.) transformation-based (HMTP), ii.) local region rearrangement-based (TBCP) and iii.) cluster-based (NICE). The protocols are evaluated based on their ability to build low cost or low delay trees, the penalties imposed by the strategies, convergence speed and protocol overhead in identical simulation environment. We summarise our main findings as follows.

- By using two different topology models (Waxman and Transit-stub), we found that the performance of a protocol varies for different network models. In general, the cost-based solutions are less affected compared to latency-based solutions.

- Tradeoffs of optimising delay with tree cost and link stress. A delay-optimised tree always results in high link stress, and thus high overall tree cost. An interesting note is that a protocol that tries to optimise the delay metric, if not well designed, could result in high bandwidth wastage.
- For cost-based optmisation:
    - The depth-first searching technique in HMTP can effectively construct low cost trees. By using a modified version of the protocol (`HMTP-Random`), we show that the depth-first searching technique provides a fast convergence speed and efficient way to achieve low cost solutions. The disadvantage of this technique is that the worst-case overhead is of the order of the number of participants, i.e. when the overlay forms a chain.
    - The cost-based TBCP gives a comparative solution and a lower join overhead in most topologies tested.
    - Performance of SwitchTree is improved by widening the scope of the local region.
    - The observations indicate that we could combine the advantages of the above three approaches to achieve low cost solutions. More specifically, we can use the TBCP technique to construct the initial overlay trees for its low join overhead. After that, the depth-first searching technique in HMTP can be used to further improve the trees. While the randomise technique during HMTP refinement procedures allows a node to search other tree branches, a more deterministic solution can be achieved by interleaving the depth-first searching with the local region searching method.
- For delay-based optimization:
    - Transformation-based protocols that only involve nodes within a small local region are able to produce low latency trees with low control overhead. The performance is improved with the flexibility of transformation operations.
    - The delay-based TBCP provides reasonably good results for small group sizes, however, the performance degrades when the number of members increase. Our analysis shows that the inefficiency in large group sizes is due to the greedy nature of the score function used. We proposed a new score function that tries to minimise the triangle cost within the local region. The results show that the new cost function is able to provide better delay results with lower resource usage. An obvious future work is to study if the local transform method can further optimise the initial tree produced by TBCP.
    - An interesting result is that the local transformation protocol that attempts to optimise low root diameter trees also achieves low tree diameter trees. In particular, the protocol outperforms NICE which uses source-specific trees in a multi-sender service model. A previous study [4] shows that NICE can provide comparable latency performance in comparison with the mesh-based Narada protocol. While early studies on IP multicasting suggest that source-specific trees have better latency performance than shared-tree, it would be interesting to make a similar comparison at the application layer.
- A clustering technique that uses a graph theoretic center as cluster head, i.e. NICE protocol, can strike a balance between tree cost and latency. However, as the NICE hierarchical arrangement results in high fanout value ($K\log_K N$) at some nodes, it is not suitable for bandwidth intensive applications.
- The effect of fanout and heterogeneity is more obvious for delay-based protocols than cost-based protocols. Poor performance observed in delay-based solutions indicates that heterogeneity is an important design consideration.

This work presents a detailed study on the steady state behaviour of various tree-first tree building and optimisation techniques. The effects of membership dynamics such as node failure have not been studied. This is an important area to investigate.

REFERENCES

1. GT-ITM Topology Generator. Available at: http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz
2. HyperCast Project. http://www.cs.virginia.edu/~mngroup/hypercast
3. Banerjee, S. myns Simulator. http://www.cs.umd.edu/~suman/research/myns/index.html

4. Banerjee, S., Bhattacharjee, B. and Kommareddy, C., Scalable Application Layer Multicast. in *ACM SIGCOMM*, (Pittsburgh, PA, 2002), ACM.
5. Banerjee, S., Bhattacharjee, B. and Kommareddy, C. Scalable Application Layer Multicast, UMIACS TR-2002-53, Department of Computer Science, University of Maryland, College Park, 2002.
6. Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B. and Khuller, S., Construction of an Efficient Overlay Multicast Infrastructure for Real-time applications. in *IEEE INFOCOM*, (San Francisco, USA, 2003), IEEE.
7. Castro, M., Jones, M.B., Kermarre, A.M., Rowstron, A., Theimer, M., Wang, H. and Wolman, A., An Evaluation of Scalable Application-level Multicast Built Using Peer-to-Peer Overlays. in *IEEE INFOCOM*, (San Francisco, USA, 2003), IEEE.
8. Chawathe, Y. An Architecture for Internet Broadcast Distribution as an Infrastructure Service. PhD Thesis *Computer Science*, University of California, Berkeley, 2000.
9. Chu, Y.H., Rao, S.G. and Zhang, H., A Case for End System Multicast. in *ACM SIGMETRICS*, (Santa Clara, CA, 2000), ACM, 1-12.
10. Deering, S. and Cheriton, D.R. Multicast Routing in Datagram Inter-networks and Extended LANs. *ACM Transactions on Computer Systems*, *8* (2). 85-110.
11. Diot, C., Levine, B.N., Lyles, B., Kassem, H. and Balensiefen, D. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, *14*.
12. El-Sayed, A., Roca, V. and Mathy, L. A Survey of Proposals for an Alternative Group Communication Service. *IEEE Network*.
13. Francis, P. Yoid: Extending the Internet Multicast Architecture, ISI, 2000, 38.
14. Helder, D.A. and Jamin, S. Banana Tree Protocol: An End-host Multicast Protocol, University of Michigan, 2000.
15. Helder, D.A. and Jamin, S., End-host Multicast Communication Using Switch-trees Protocols. in *Workshop on Global and Peer to Peer Computing on Large Scale Distributed System (GP2PC)*, (2002).
16. Holbrook, H. and Cain, B. Source-specific Multicast for IP, IEFT, 2000.
17. Jain, S., Mathajan, R. and Watherall, D., A Study of the Performance of Potential of DHT-based Overlay. in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, (2003).
18. Jannotti, J., DGifford, D., Johnson, K., Kaashoek, M. and O'Toole, J., Overcast: Reliable Multicasting with an Overlay Network. in *Symposium on Operating Systems Design and Implementation (OSDI)*, (San Diego, California, USA, 2000).
19. Konemann, J. and Ravi, R. A Matter of Degree: Improved Approximation Algorithms for Degree-bounded Minimum Spanning Trees. *SIAM Journal on Computing*, *31* (6). 1783-1793.
20. Kostic, D., Rodriguez, A. and Vahdat, A. The Best of Both Worlds: Adaptivity in Two-metric Overlay Networks., Duke University, 2002.
21. Li, Z. and Mohapatra, P., HostCast: A New Overlay Multicast Protocol. in *IEEE International Conference on Communications (ICC)*, (Anchorage, Alaska, USA, 2003), IEEE.
22. Liebeherr, J. and Beam, T.K., HyperCast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology. in *Networked Group Communcation (NGC)*, (1999).
23. Liebeherr, J. and Nahas, M., Application-layer Multicast with Delaunay Triangulations. in *IEEE Global Internet Symposium (Globecom)*, (2001).
24. Malouch, N.M., Liu, Z., Rubenstein, D. and Sahu, S., Graph Theorectical Approach to Bounding Delay in Proxy-assisted, End-system Multicast. in *International Workshop on Quality of Service (IWQoS)*, (Miami Beach, 2002).
25. Mathy, L., Canonico, R. and Hutchison, D., An Overlay Tree Building Control Protocol. in *Networked Group Communication*, (London, UK, 2001).
26. Mathy, L., Canonico, R., Simpson, S. and Hutchison, D. Scalable Adaptive Hierarchical Clustering. *IEEE Communications Letters*.
27. Pendarakis, D., Shi, S.Y., Verma, D. and Waldvogel, M., ALMI: An Application Layer Multicast. in *3rd USENIX Symposium on Internet Technologies and Systems*, (2001).
28. Perlman, R., Lee, C., Ballardie, T., Crowcroft, J., Wang, Z., Maufer, T., Diot, C., Thoo, J. and Green, M. Simple Multicast: A Design for Simple, Low-overhead Multicast., IETF, 1999.
29. Ratnasamy, S., Handley, M., Karp, R. and Shenker, S., Application-level Multicast Using Content-addressable Networks. in *Networked Group Communication (NGC)*, (2001).

30. Roca, V. and El-Sayed, A., A Host-based Multicast (HBM) Solution for Group Communication. in *IEEE International Conference on Networking*, (Colmar, France, 2001), IEEE.
31. Rowstron, A. and Kermarre, A.M. Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE JSAC*, *20* (8).
32. Shi, S.Y., Turner, J.S. and Waldvogel, M., Dimensioning Server Access Bandwidth and Multicast Routing in Overlay Networks. in *NOSSDAV*, (Port Jefferson, New York, USA, 2001), ACM.
33. Stoica, I., Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H., Chord: A Scalable Peer-to-Peer Lookup service for Internet Applications. in *ACM SIGCOMM*, (San Diego, California, USA, 2001), ACM.
34. Tran, D.A., Hua, K.A. and Do, T.T., ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming. in *IEEE INFOCOM*, (San Francisco, USA, 2003), IEEE.
35. Zegura, E.W., Calvert, K.L. and Bhattacharjee, S., How to Model an Internetwork. in *IEEE INFOCOM*, (1996), IEEE.
36. Zhang, B., Jamin, S. and Zhang, L., Host Multicast: A Framework for Delivering Multicast to End Users. in *IEEE Infocom*, (New York, USA, 2002), IEEE.
37. Zhuang, S., Zhao, B., Joseph, A., Ratz, R. and Kubiatowicz, J., Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. in *NOSSDAV*, (2001).