A Simulator for Peer-to-Peer Overlay Algorithms

Vladimir Blagojevic

A thesis submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Master of Science

Graduate Programme in Computer Science
York University
Toronto, Ontario
June 2004

ABSTRACT

The Peer-to-Peer (P2P) approach is not the first attempt to achieve large scale Internet multimedia streaming. P2P multimedia streaming is born out of the frustration that IP multicasting has not materialized even a decade after its initial specification. In P2P streaming overlays, peers are connected into a logical overlay structure emulating a multicast network. Since data is replicated at the application layer through a set of unicast connections formed at each peer, such systems are undoubtedly less efficient than IP multicast. Although recently there have been a multitude of P2P overlay streaming proposals, it is yet unclear which ones present the best solutions. The major obstacle in reaching such a conclusion involves the difficulties of evaluating large-scale overlay algorithms. Having a continuing trend where each research team develops its own ad-hoc overlay simulator would undoubtedly jeopardize the progress of this interesting field of research. There is a pressing need for a unified and unbiased simulation framework that enables a strong push forward for P2P overlay media streaming. It is our opinion that the simulator we have developed is an important step in the right direction. As far as we know we are the first to provide a unified and sophisticated P2P media streaming overlay simulator. We have demonstrated our simulator's extensibility and adaptability by implementing four leading media streaming overlay algorithms. Our simulator comes with a rich set of join, duration, and bandwidth distributions as well as overlay metrics. Furthermore, the size of the synthetic network, as well as the number of peers in the overlay in our simulations, is two orders of magnitude larger than ever considered in simulations of P2P tree-based overlay algorithms.

ACKNOWLEDGEMENTS

First of all, I owe a debt of gratitude to my supervisor Franck van Breugel. Without his insightful guidance and support, I could not have completed this thesis. I am grateful for each and every hour he has contributed to the development and refinement of the research and writing.

Eshrat Arjomandi, my co-supervisor, has been extremely supportive of me over the years, providing guidance, counsel, and generous funding.

I am grateful to the remaining members of my thesis examining committee, Jonathan Ostroff, Stephen Watson, and Suprakash Datta for their time and helpful feedback.

Duc A. Tran from the University of Dayton has been very forthcoming with answers to all of my questions regarding the ZIGZAG P2P streaming system. Anukool Lakhina from Boston University has also been extremely helpful with inquiries related to the BRITE topology network generator.

Mojoj majci

TABLE OF CONTENTS

Chapter 1

Introduction

Beginning with the explosion of Napster in 1999, the advent of peer-to-peer (P2P) computing has dramatically shifted attention from centralized systems to more scalable, decentralized and distributed systems. P2P systems differ from more conventional centralized network systems in the sense that the computing load is more equally shared between all participating computer nodes, which are called peers. Peers participate in P2P systems by sharing their resources toward a common goal. In the case of Napster this goal is file sharing.

Although the Napster file sharing system has been presented as a poster child of P2P computing, it cannot be regarded as a pure P2P system due to its centralized directory index. This central directory index stores information about files available from all currently participating peers. Another often cited example of a P2P system is the Gnutella network [2]. Gnutella is a completely decentralized P2P file sharing system that does not have a centralized directory index like Napster. However, such a high degree of decentralization comes with an increased searching cost. Gnutella uses an expanding search mechanism. A search query is first performed at the peer nodes directly connected to the node that originated the search. A search is then further expanded to the immediate neighbours of those nodes queried in the first round and so forth.

There has been a significant renewal of interest in decentralized systems within the research community as well. Most of the research has focused on solving limited scalability issues of the search mechanism employed in systems like Gnutella. Several research groups have independently presented P2P systems that are essentially based on distributed hash tables [36, 37, 39]. In these P2P systems, objects are associated with keys and each peer node is responsible for storing a

certain range of keys. Given the key of the object, the operation lookup returns the identity of the node storing that object. The lookup message is efficiently forwarded among the nodes. The expected number of forwarding steps from the node originating the search to the node hosting the desired object is $O(\log n)$ [37], where n is the number of peer nodes in the system. In order to provide better data availability and fault tolerance, the hashtable contents are usually replicated.

1.1 Multimedia streaming

Another viable application area of P2P systems is multimedia streaming. One of the biggest shortcomings of the classical client/server model of multimedia streaming is the excessive bandwidth requirement usually associated with streaming. In current working solutions, employing the classical client/server model, the number of receiving clients scales linearly with costly resources (hardware and bandwidth). For example, an Internet radio station streaming content at 128 kbps can hardly scale to thousands of clients due to both hardware and bandwidth constraints and their costs.

Multimedia streaming is also possible through content delivery networks (CDN). A CDN is a dedicated network of servers that enables content distribution from geographically and Internet strategic locations. CDN companies like Akamai own a network of powerful servers and offer services of content distribution to third parties. A CDN transparently directs content consumers to the best available network server. CDNs offer good quality of service, however, they are extremely expensive.

One of the reasons that led to the recent emergence of CDNs was the unsuccessful development and deployment of IP multicasting technologies [18]. The development and deployment of IP multicasting has been plagued with issues such as group management, congestion and flow control, security and network management [20]. Under the current IP multicast model there is no mechanism that prevents a host from creating a multicast group, from becoming a member of a group or from sending data to a group. Thus, multicast networks would be much more susceptible to malicious flooding attacks than unicast networks. Multicast systems would be even more desirable as a flooding target due to the

opportunity of an attack of a larger scale than unicast networks. In their seminal paper [17] arguing for a move of multicasting implementations from the IP layer to the application layer, Chu, Rao and Zhang claim that IP multicasting has been doomed to failure since it attempts to squeeze application layer functionality into the stateless IP router layer. Current routers are stateless while multicasting would require stateful routers to maintain per group information. Internet Service Providers (ISPs) are also reluctant to deploy multicast routers due to the lack of reliable network management and administration tools. All of these shortcomings combined have ultimately led to a slow adoption rate of IP multicasting by ISPs and Internet users.

The P2P approach to multimedia streaming is not the first attempt to mass-scale multimedia streaming but is rather born out of the frustration that IP multicasting (in theory the ideal solution) has not caught up even a decade after its initial specification [21]. P2P based systems, almost exclusively implemented in the application layer, could potentially present an effective solution to mass-scaling of multimedia streaming. Indeed, most recent research studies in the context of mass-scaling multimedia streaming involve building multicast P2P structures in the application layer space. Such application layer structures are also referred to as *overlay* networks since they are built on top of the conventional IP networks. In P2P media streaming systems, rather than replicating packets at the router level, packets are distributed through the application layer of the participating peers.

1.2 Overlay algorithms and their simulation

Peers are connected into a logical overlay structure emulating a multicast network. Since data is replicated at the application layer through a set of unicast connections formed at each peer, such systems are undoubtedly less efficient than IP multicast in terms of network usage. For example, in application level multicast it is highly possible that sometimes multiple copies of the same packet might travel through the same underlying physical network link. IP multicast is thus usually used as a benchmark to measure the efficiency of P2P overlay structures by providing a lower bound for performance measurements. An overlay tree is efficient if the performance penalty induced by migrating multicast functionality into the application level space does not significantly increase the network cost

compared to IP multicast. A good overlay tree will have a small deviation from the IP multicast tree in regards to the defined network metrics. Essentially, we are interested in the overhead incurred by using overlay structures instead of IP multicast.

Although recently there have been a multitude of P2P overlay streaming proposals, it is yet unclear which ones present a good solution. The major obstacle in reaching such a conclusion involves difficulties of evaluating large-scale overlay algorithms. One would have to develop and test a system consisting of thousands of physical computer nodes. This is rather impractical if not almost impossible. In the absence of resources to test an overlay consisting of thousands of nodes, one usually resorts to P2P overlay simulators. Simulators help in evaluating overlay algorithms without having to deploy the overlay over thousands of physical network nodes. However, there seems to be a strong tendency among P2P overlay researchers to develop their own simulator. There are several drawbacks to such an approach as well. The most important one is the use of ad hoc simulators for research purposes. Banerjee et al. [4] have published the details of their P2P overlay streaming simulator, while other research projects that we have reviewed [19, 34, 41, 45] briefly mention the high-level details of their own simulators. p2psim [7] is another example of a P2P simulator. However, it focuses on distributed hash table based overlays rather than media streaming overlays. Previously developed overlay streaming simulators used a synthetic network of at most ten thousand nodes while the largest overlay size was two thousand nodes, whereas our network has half-a-million nodes and ten thousand overlay nodes. Utilizing appropriate *join* and *duration* node distributions is important as well. A join distribution describes the rate of peer node arrival in the overlay while a duration distribution determines the length of stay in the overlay for each peer. Researchers have mostly used simple join and duration distributions that can hardly be a representative model of the real Internet. In contrast, we have carefully chosen join and duration distributions that reflect the unpredictable nature of the Internet. In addition, all simulators we have reviewed used a peer bandwidth distribution where all nodes have the same bandwidth capability or the peer bandwidth distribution has been ignored altogether. Such a bandwidth model is in clear contrast with the most recent research. In our simulations we acknowledge the suggestion of Saroiu et al. [38] that there exists a high degree of bandwidth heterogeneity among overlay peers.

Having a continuing trend where each research team develops its own over-

lay simulator would undoubtedly jeopardize the progress of this very interesting field of research. We strongly believe that there is a pressing need for a unified and unbiased simulation framework that enables a strong push forward for P2P overlay media streaming. It is our opinion that the simulator we have developed is an important step in the right direction.

1.3 Overview

In Chapter 2 we give a summary and definitions of the most commonly used performance metrics in P2P simulations. Besides frequently used metrics such as *stress* and *stretch*, we focus on control overhead as well. We give special attention to join and leave control overhead since these two metrics can pinpoint troubling tendencies for overlay scaling. We base our join and leave control overhead measurements on the number of peer contacts. Join control overhead records the number of peer contacts required to add the arriving node to the overlay. Leave control overhead tabulates the number of peer contacts that are required to repair an overlay after a node leaves or crashes. Finally we take a closer look at the *robustness* metric. The robustness metric focuses on the proneness of the overlay trees to fragility. In order to be robust, an overlay should seamlessly and efficiently handle a node leave or crash as well as overlay restructuring. We introduce two specific robustness metrics, namely *glitch ratio* and *shed ratio*, which we believe are new.

Chapter 3 provides an overview of the three different approaches to building overlay trees, namely the tree-first, the mesh-first and implicit tree building overlay algorithms. We discuss the differences between them as well as their applicability to media streaming. We focus specifically on tree-first overlays since there is a general understanding in the research community that the tree-first overlays are better suited for large-scale media streaming. We provide a review of four recent tree-first overlay algorithms while using a consistent terminology in presenting the overlays.

In Chapter 4 we explore the current state-of-the-art research in synthetic Internet topology generation. In order to have a representative P2P overlay simulation environment the most suitable topology generator should be used. We investigate the history of the Internet topology generators and give arguments for

choosing the BRITE Internet topology generator for our unified simulation framework. Special attention is given to often-neglected simulation parameters, namely peer bandwidth distribution as well as peer join and duration distributions. We use a peer bandwidth distribution based on recent measurements studies [38] of the Gnutella P2P network. We believe that this bandwidth distribution is a good representative of the general Internet population that is interested in media streaming. Our simulation framework can accept any other distribution as well. Join and duration distributions are other simulation input parameters that have been usually overlooked in previous simulations. We also provide reasons for choosing specific join and duration distributions in our simulations. Again our simulation framework can accept any join and duration distribution, which we demonstrate in Chapter 5. As far as we know, our simulator is the first that takes the peer bandwidth distribution and the peer join and duration distributions seriously.

Chapter 5 gives an overview of previously conducted P2P overlay simulations along with their accompanying results. P2P overlay simulators were mostly developed as part of the overall overlay algorithm research projects. Naturally, the focus of those research projects was on the overlay algorithms themselves. We give a detailed summary of the recent P2P simulation environments as well as the simulation results. We then proceed to recreate these simulation environments using our simulator and to verify the published simulation results. By repeating the results of the previous simulations using our unified simulation framework we achieve two things. Firstly, we demonstrate the ability of our simulator to effortlessly plug in the desired overlay algorithm as well as the simulation parameters from the previous simulations. Secondly, we confirm that our implementations of the simulated overlay algorithms are correct. Having asserted the correct implementation of the four overlay algorithms, we can proceed to simulating the overlay algorithms with simulation parameters that we believe are reasonable. We provide a concluding overview for each of the four overlay algorithms that we implemented and studied. Simulation result trends are discussed for each of the defined metrics and various trade-offs are explained.

In Chapter 6 we give a summary of our research along with future directions. As far as we know we are the first to provide a unified and sophisticated P2P media streaming overlay simulator. We have demonstrated our simulator's extensibility and adaptability by implementing four leading media streaming overlay algorithms [19, 34, 41, 45] along with a rich set of metrics. Furthermore, the size

of the synthetic network, as well as the number of peers in the overlay in our simulations, is two orders of magnitude bigger than ever considered in simulations of P2P tree-based overlay algorithms.

Chapter 2

Performance metrics

While various P2P multimedia streaming research groups have considered different overlay performance metrics, Banerjee and Bhattacharjee [10] have summarized the most common performance metrics that various research groups have measured: stress, stretch, control message overhead and robustness.

2.1 Stress

Stress is a metric intrinsically tied to the underlying physical network links. The stress that an overlay induces on the underlying physical network has been a focus of previous overlay research. Chu et al. [17] and Zhuang et al. [46] define stress of a physical link as the number of identical copies of a packet carried over that physical link. Banerjee and Bhattacharjee [10] expand this definition from physical links to overlay nodes. They define stress as a metric defined per link or router node of the topology that counts the number of identical packets sent by the protocol over that link or router node.

In IP multicast there are no redundant packets sent over any physical network link. Therefore, the stress on each physical link in IP multicast is 1. Stress higher than 1 is an indicator of a deviation from an ideal solution. Overlay nodes in certain instances send more than one identical packet through the same underlying physical link.

By taking time-varied snapshots of the overlay structure and measuring the

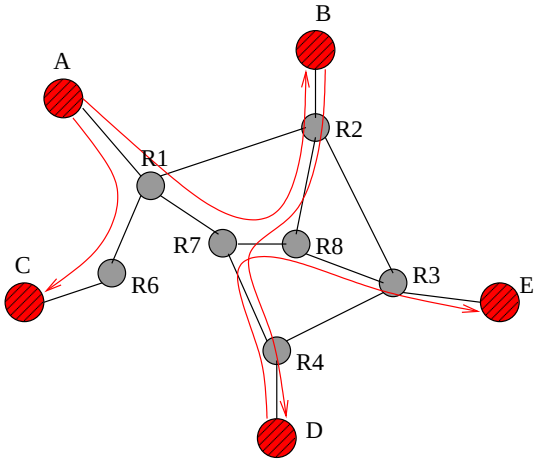

Figure 2.1: Stress on physical link R7-R8 is the highest

stress of the underlying physical links at those given instances we will be able to deduce stress deviation of the overlay from IP multicast.

Overlay tree building algorithms should strive to achieve as low as possible stress deviation from the IP multicast. Let L be the physical link that connects source A to the rest of the network. If A has n children in the overlay tree, then stress of the link L will be at least n. For example, in Figure 2.1 node A has 2 children, B and C, hence the stress on the physical link that connects node A to network router R1 has a stress of at least 2. Overlay network building algorithms can hardly influence stress of some backbone links since most of the Internet traffic goes through them. For example, given the same physical network in Figure 2.1 and the overlay tree where computer A is the source of the stream, we can conclude that the highest stress of 3 for this instance of overlay is on physical link R7-R8.

Link stress is trivially calculated by simply keeping track of physical links used when overlay links are established and removed. We assume that the overlay link between two nodes A and B is created on a shortest path between A and B within the physical network. Figure 2.2 gives an outline of the algorithm. The running time of the algorithm depends on the implementation of Dijkstra's shortest path algorithm. Assuming the priority queue is implemented using an array, finding a shortest path takes at most $O(V^2)$ where V is the number of physical nodes in

```
overlay_add(node n)
  let p be parent of n in overlay
  find shortest path sp between node n and p in physical network
  for each physical link pl on that path sp
    stress(pl) = stress(pl) + 1

overlay_remove(node n)
  let p be parent of n in overlay
  find shortest path sp between node n and p in physical network
  for each physical link pl on that path sp
    stress(pl) = stress(pl) - 1
```

Figure 2.2: Stress algorithm pseudocode

the network graph. Given that we have n parent/child pairs in the overlay, the running time of the stress algorithm is at most $O(nV^2)$.

2.2 Stretch

Stretch is a metric that evaluates the overall quality of the overlay by taking measures for each node in the overlay. Chu, Rao and Zhang [17] defined stretch of a node as the ratio of the path latency from the source to the peer node along the overlay to the path latency of the direct unicast link. A unicast path between two nodes has a stretch of 1. Ideally we would like to have each member's stretch as close as possible to 1. Clearly, it would be simple to construct a multicast overlay where the source unicasts data to each of the n node members thus resulting in each node having a stretch of 1. However such an overlay is constructed on the expense of $O(n)$ stress on the link L as discussed in previous section.

Returning to our example overlay from Figure 2.1 and assuming that each edge has the same latency we can conclude that the shortest unicast path from node E to the source is along the path E-R3-R2-R1-A, thus yielding path length of 4 units. However, the path along the overlay between node E and the source node A is 15 unit lengths. Therefore the stretch of the node E is 3.75.

```
stretch(n):
up = length of shortest path between n and source in physical network
p = parent of n in overlay
top = 0  //overlay path
until n = source
  top += length of shortest path between n and p in physical network
  n = p
  p = parent of p in overlay
return top / up
```

Figure 2.3: Stretch algorithm pseudocode

The algorithm that computes the stretch of the particular node is straightforward. Although the algorithm outline is given in Figure 2.2, potential implementations vary depending on the actual data structures used in representing the particular overlay. The running time of the stretch algorithm also depends on the implementation of Dijkstra's shortest path algorithm. Taking the same assumption as in the stress algorithm, finding the n shortest paths corresponding to the overlay paths from each node to the source takes at most $O(nV^2)$ where n is the number of nodes in the overlay and V is the number of physical nodes in the network graph. Shortest paths are calculated on demand, using the same algorithm as used in our stress calculations.

2.3 Control message overhead

Control message overhead is very important for overlay scalability. Overlay algorithms maintain the overlay by exchanging control messages between peer nodes. Ideally we would like to minimize peer control message communication overhead and localize it to the neighbouring peers. There have been several proposals to appropriately measure the control message overhead. One of these proposals comes from Liben-Nowell, Balakrishnan and Karger [29]. They suggest that the bandwidth percentage used on control message overhead in each node is the most appropriate measurement for efficiency of overlay algorithm control overhead. If the bandwidth consumed by peer nodes on the control overhead grows even lin-

early with the respect to the total number of peers then scalability of the P2P network would be seriously limited.

Tran et al. [41], the authors of ZIGZAG overlay system, have a different proposal. They have not measured the control overhead by the amount of bandwidth taken but rather by the average number of nodes contacted during the execution of the overlay mutating operations (i.e., join, leave, restructure). They indicate that a successful control protocol should have the same overhead for overlay maintenance regardless of the overlay size. Indeed, special attention has been devoted to ensure that the number of nodes in the overlay does not affect ZIGZAG's control overhead. As discussed in Section 5.2.3, ZIGZAG's control overhead assumptions have been verified in simulations.

In our simulations we measure control overhead according to the latter approach. There are two main reasons for this choice. First, implementing a representative simulation of the amount of bandwidth taken by the control protocol is difficult and error prone. More importantly, the second reason is that the control overhead proposal by Tran et al. quickly identifies problematic overlay tendencies during various overlay operations. For example, such tendencies would be that the control overhead scales linearly with the number of nodes in the overlay. Overlay algorithms should have sublinear control overhead in order to scale to thousands of nodes.

In the simulations we conducted special attention was given to join and leave control overhead. The join overhead at peer node P is measured by the number of the nodes P contacts during the join procedure. The leave or crash control overhead measurement is not as straightforward. The details of the leave or crash control overhead are very specific for different overlay algorithms. We have abstracted the leave or crash control overhead measurements to the total number of node contacts that resulted due to leave or crash of a certain node.

2.4 Robustness

Robustness concerns fragility of overlay trees. Overlay trees should have efficient failover capabilities when nodes leave or crash. Robustness can be measured by the percentage of nodes that have not received all the data during various leave

or crash scenarios. The amount of missing data is usually important as well - depending on the context of the multicast application. In voice multicast streaming even small amounts of lost data may cause the listener to notice annoying glitches while in video streaming small amounts of missing data might not be so perceptible.

We have identified two classes of overlay algorithms related to node leave or crash events. The first type has an individual rejoin algorithm where nodes are allowed to search for a new parent individually after their current parent leaves or crashes. The second type has a coordinated rejoin algorithm where nodes are coordinated in their search for a new parent. ZIGZAG, OMNI and some of the SpreadIt rejoin algorithms are classified as coordinated rejoin algorithms. The HMTP rejoin algorithm is an individual rejoin algorithm.

A simulator for the SpreadIt overlay streaming system [19] has especially focused on modelling packet loss experienced during various join/leave scenarios. Specific attention was given to this metric for a rather different reason. SpreadIt data channels are based on the connectionless and lossy UDP protocol and thus SpreadIt is susceptible to packet loss.

In our simulations we measure robustness in two ways resulting in two specific robustness metrics - the overlay glitch ratio and the overlay shed ratio. The first one is the percentage of nodes that were receiving the stream but are unable to find a new stream source after restructuring in the overlay. These nodes would be temporarily disconnected and in a real-life scenario experiencing annoying streaming glitches. The ratio of the total number of nodes experiencing stream interruptions and the total number of nodes in the overlay is the overlay glitch ratio.

Note that we consider all the possible causes for overlay restructuring depending on the specifics of a certain overlay algorithm. Such causes include among others join, leave, crash and regular overlay restructuring/maintenance as dictated by a certain overlay algorithm.

The second, more fine-grained measurement is based on tracking the amount of time ticks that the stream interruption is experienced in a certain node. If a stream is not received in a certain time window the node is disconnected. In this way we simulate the disgruntled users. The ratio of total number of nodes

disconnected in such a way and the total number of nodes in the overlay is the overlay shed ratio.

We have also introduced two thresholds related to robustness. The first one, the drop threshold, is the number of simulation ticks after which a node that is temporarily disconnected from the overlay is dropped from the overlay. An orphaned peer node that is unable to rejoin the overlay is not immediately dropped from the overlay. Such node is temporarily disconnected and unable to receive streamed data. We have set the drop threshold to 3 ticks. If a temporarily disconnected node is unable to reconnect for 3 ticks it is dropped from the overlay. The second robustness threshold, the delay threshold, is related to individual rejoin algorithms. If a node running an individual rejoin algorithm is unable to find a new parent in a given amount of time, it is dropped from the overlay as well. In the case of the delay threshold we need a finer time metric than a number of ticks. Thus in our simulations we track the delay accumulated while the rejoining node is searching for a new parent. We have set this delay to be a hundred times the average parent/child delay. Our reasoning is that since the average parent/child delay on the Internet is in the hundreds of milliseconds scale the rejoin delay threshold, in the real Internet, would be in the tens of seconds scale. We assume that a peer stream receiver will be most certainly annoyed after not receiving any media stream for more than ten seconds.

Chapter 3

Overlay algorithms

Although there exist various application layer multicast approaches, they all have a common ingredient that comprises of data and control tree building algorithms. Data tree algorithms construct a tree for data delivery while control algorithms connect peers into a control data exchanging overlay. Depending on the order of data and control overlay construction, P2P media streaming systems can be classified into mesh-first, tree-first and implicit systems [10].

In a mesh-first based overlay [16, 17] nodes are connected in a mesh topology of unicast links between peer nodes. The mesh is usually directly used for the exchange of control messages between peers. Single source multicast trees are then constructed on top of meshes by using well known IP multicast algorithms [21]. In the mesh-first approach, overlays constructed are suitable for multisource broadcasting.

Tree-first algorithms [19, 23, 27, 34, 45] organize peer nodes into a single source data multicast tree directly without any prior mesh construction. After a peer node joins the multicast tree it discovers a few other member nodes and establishes control data exchange links with them. Tree-first overlays usually consist of one broadcast node (source) and many receiver nodes (all other nodes).

In implicit tree building algorithms [11], the mesh and the multicast tree are simultaneously constructed by the protocol and no additional member interactions are needed to generate one from the other or vice versa.

Various P2P multicast streaming solutions focus on different overlay structure

properties in order to achieve good overlay trees. Protocols for node join and leave are defined by taking into account those properties of the overlay. However, given the fact that the number of peers participating in the system, like in other P2P applications, changes rather rapidly, it is essential that P2P systems implement a tree management algorithm that continually repairs the overlay. Tree management protocols strive to achieve an ideal overlay structure given such a rapidly changing system, but will most probably never achieve an ideal overlay structure.

One of the objectives of our research is creating a P2P streaming network simulation environment where we will simulate various proposed application level P2P streaming solutions. In order to be scalable to tens of thousands of nodes, the simulator is based on a flow network level granularity rather than packet level such as the ns2 [5], ssfnet [8] and javasim [3] simulators. By comparing various proposed P2P streaming solutions we will verify their performance in terms of the defined metrics.

In the rest of this chapter, we will overview the tree-first overlay algorithms and in turn summarize each of the four tree-first overlay algorithms that we have implemented.

3.1 Tree-first overlay algorithms

As previously mentioned, tree-first algorithms construct a shared tree for delivering streaming data in the overlay. In order to produce scalable overlays, each peer in the overlay is aware of only a constant number of other peers. All tree related algorithms are distributed in nature and invoked within peers themselves. A join algorithm guides each new arriving member to a certain parent within the tree, thus effectively leading to an explicit construction of the data delivery tree. Subsequently, after a node has been added to the overlay, each node keeps track of a set of "neighbour" nodes. The size of this "neighbour" set is bounded. In some overlays, the nodes keep track of the "neighbour" set by either creating additional links to the "neighbour" nodes or simply by keeping and refreshing the state of the "neighbour"nodes. Such additional links, sometimes referred to as mesh links [11], complement the data delivery tree in the sense that they increase its robustness and help in node crash recovery.

A leave algorithm, initiated from the leaving node, removes the leaving peer node from the overlay. Crashes are usually handled differently. When a node crash is detected, peers affected by the crash have to be reorganized in the overlay. If the leaving/crashing node did not have any children, the overlay data delivery tree is not affected and no further actions have to be taken. Otherwise, a certain type of tree restructuring has to be performed in order to keep the remaining peer nodes connected. The mesh links and information about "neighbouring" nodes is also updated to reflect the changes.

Besides join and leave algorithms some overlays define additional tree improvement algorithms. Tree improvement algorithms are invoked periodically from certain peer nodes. Their goal is to improve the overlay tree with respect to metric(s) of interest.

All tree-first algorithms have an explicit control over the maximum number of children a node can have. Given a limited peer uplink bandwidth capacity B and a data delivery stream rate R, each node can have in between 0 and B/R children. Each peer node chooses its parent during execution of the join algorithm itself. It may also choose a parent during tree improvement algorithm execution. Furthermore after a node has been orphaned, it has to find a new parent.

In the following sections we present a summary of the SpreadIt, HMTP, OMNI, and ZIGZAG tree-first overlay algorithms.

3.2 SpreadIt

The SpreadIt overlay algorithm [19] is an example of a tree-first overlay algorithm. Peer node P, wishing to join a SpreadIt overlay, starts its join process by sending a join request to the source node. SpreadIt node R, receiving a join request, accepts the request if itself is unsaturated. Otherwise, R redirects the joining node P to one of its immediate children nodes depending on the selected join policy. The join policies are:

Random - P is redirected to a random child of R.

Round-Robin - P is redirected to one of R's children in a round-robin fashion.

Smart-Placement (SP) - P is redirected to R's closest, in terms of network distance, child.

Knock-Down - if P is closer to R than any of R's children then R redirects its farthest child and accepts P. Otherwise, same as SP.

Smart-Bandwidth - if P's bandwidth capacity is bigger than R's, P takes the place of R. R and all its children become children of P. Otherwise, any of the above mentioned policies can be used.

The join algorithm proceeds recursively until a node R willing to parent P is found. If, after a predefined number of recursive steps, no such node R is found, the join fails and the algorithm exits.

The SpreadIt overlay treats the intentional leave and accidental crash separately. A leaving node P is first unsubscribed from its parent. If P happens to be a leaf, no further action is necessary. Otherwise, if P is a non-leaf node, P's descendants have two options for recovery. Either immediate children of P rejoin and the rest of the descendants rely on them, or each descendant rejoins separately. Since a SpreadIt node does not keep any additional "neighbour" peer state besides information about its parent and immediate children, the options for rejoining are limited. The new parent for the orphaned descendants is selected according to one of the following policies:

Grandfather-All (GFA) - P chooses its parent R as the new parent for all descendants. P sends a message to all its descendants to rejoin at R.

Root-All (RTA) - P chooses the source as the new parent for all its descendants. P sends a message to all its descendants to rejoin at the source.

Grandfather (GF) - P chooses its parent as the new parent for its immediate children. The rest of the descendants rely on their parents to restore the stream.

Root (RT) - P chooses the source as the new parent for all its immediate children. The rest of the descendants rely on their parents to restore the stream.

In the case when non-leaf node P crashes, P's descendants do not know the identity of P's parent. Thus the descendants can only use the RTA and RT policies for the crash recovery.

The SpreadIt overlay does not employ any additional tree-improvement algorithm nor do nodes keep track of any "neighbour" nodes except for the parent and the immediate children.

3.3 HMTP

The Host Multicast Tree Protocol (HMTP) [45] is another example of a tree-first overlay construction algorithm. Unlike SpreadIt, HMTP uses a rendezvous node to discover the root of the tree. The node joining the HMTP tree "remembers" parent nodes visited during join which allows the joining node to backtrack and branch out in a search for unsaturated parent nodes. Also unlike SpreadIt, HMTP uses periodic tree improvements to ensure that the overlay is as congruent as possible with the underlying physical network. Finally, HMTP uses an individual node rejoin algorithm which is in contrast with SpreadIt's collective join algorithms.

Peer node P wishing to join an HMTP overlay starts its join process by querying the overlay's Host Multicast Rendezvous Point (HMRP) node. An HMRP is a rendezvous node allowing newly arriving peers to discover overlay membership. Since the arriving peer node P starts its join process from the root, it first contacts HMRP which knows the identity of the overlay root. Starting from the root, P tries to locate an overlay member closest (by network latency) to itself that is willing to be its parent. Such a node, having available bandwidth and willing to parent node P is called a valid node. Peer P keeps a local stack of valid potential parent nodes.

The HMTP join algorithm considers all overlay peer nodes as potential parents. Each joining node has a stack S to keep track of the discovered potential parents. The join algorithm executed by joining node P starts by pushing the root node on the stack S. The join algorithm proceeds until a suitable parent is found or the stack S of potential parents is empty. In the first step of the join algorithm, the top of the stack S is declared the current potential parent T.

Round-trip network delays from a joining node P to T and each of its children are measured. In each iteration of the join algorithm there are three possible options. First, if there is no node among T and its children that has enough bandwidth to serve P, then stack S is popped and the iterative step is repeated. Second, if there exists a child C of T having available bandwidth and C is closer to P than T itself then T is pushed on the stack S along with rest of its children. C is put on the top of the stack S and the iterative step is repeated. Otherwise, in the last remaining option, node T is picked as the parent of P and the algorithm exits.

An HMTP overlay peer keeps information about other peers on its path to the root node. This path is called the root path. Peer node P wishing to leave the overlay first notifies its parent and children nodes. If node P is a leaf no further actions are necessary. If node P is not a leaf, each child C is responsible for finding a new parent independently. Node C first removes itself and its parent P from the root path and pushes its root path on to the stack such that its grandparent is the first peer to be popped from the stack. After a random delay, the join algorithm is run using the root path as a stack of potential parents. The random delay is introduced in order to lower the probability of all children contacting the grandparent at the same moment.

Since the network environment is constantly evolving, the HMTP overlay tries to adapt by periodically running an improvement algorithm. Each peer node runs the improvement algorithm separately by re-running the join algorithm. In order to prevent overwhelming the root node with join requests, a random node from the re-joining node's root path is selected as a starting point for the improvement algorithm.

In order to branch out in a search for closer nodes, peer nodes other than the closest one can be selected as a potential parent. However, the parent switch is made if and only if the new parent is closer than the current one by a certain threshold. Introduction of a threshold prevents a peer node oscillation between a pair of parents. The improvement algorithm invocation frequency is a function of proximity to its current parent.

HMTP nodes do not explicitly create additional mesh links. However, besides keeping root path information, each node caches information about a constant number of other nodes visited during tree walks. The root path is used in cycle detection and failure recovery. Each time peer node P switches its parent, the root

path of a new parent is verified not to contain P. Thus tree cycles are avoided. The root path is also used by rejoining orphaned nodes as a list of potential parents for rejoin. In rare cases of failure recovery when all nodes in the root path are unavailable, cached nodes are used instead.

3.4 OMNI

OMNI [12] is yet another example of a tree-first overlay algorithm. The OMNI overlay defines two types of peer nodes. Multicast service nodes (MSN) are high bandwidth, dedicated peer nodes, providing efficient data distribution to a set of peer nodes. MSNs provide the data stream to other MSNs as well as to the client peer nodes. Clients are non-dedicated nodes that can also serve other non-dedicated nodes but cannot serve MSNs. Thus MSNs are positioned in levels closer to the root of the tree while other non-MSNs tend to move towards the leaves. Unlike SpreadIt and HMTP, OMNI has both local and global tree transformation. OMNI and HMTP have a similar join algorithm. However, OMNI's leave algorithm is a collective leave algorithm similar to SpreadIt.

Each MSN keeps the following state information:

- Path to the root
- Number of descendants

  $$s_i = c_i + \sum_{j \in Children(i)} s_j$$

  where c_i is the number of clients being directly served by i, $Children(i)$ is the set of MSNs being directly served by i and finally s_i represents number of clients being directly served by all MSNs in the subtree rooted at i
- Aggregate subtree latency

$$\Lambda_i = \begin{cases} 0 & \text{if } i \text{ is a leaf MSN} \\ \sum_{j \in Children(i)} s_j I_{i,j} + \Lambda_j & \text{otherwise} \end{cases}$$

where $I_{i,j}$ is the unicast latency between MSNs i and j

- Latency to its parent and children

The OMNI overlay defines tree improvement algorithms on two levels of the overlay hierarchy. We will describe both local and global tree improvement transformations in turn. Both types of tree improvement algorithms are executed by MSNs only.

Local tree improvement algorithms require interactions between MSNs that are at most two levels apart. Each MSN attempts to perform a local transformation periodically. A local transformation is performed if the average subtree latency would be reduced by that local transformation. There are no thresholds introduced to prevent possible oscillations.

Child-promote

If a grandparent G has available bandwidth, one of its grandchildren is promoted to be its child, if such an operation reduces the aggregate latency at G.

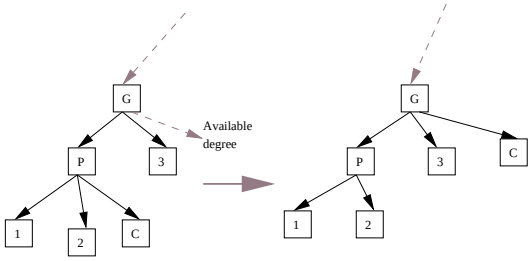

Figure 3.1: Child-promote

Parent-child swap

In this local transformation, the parent P and a child C swap their places if such a transformation reduces the aggregate latency at the grandparent G. If child C has no available degree to be a parent to P, P itself becomes the parent to one of C's children. The child selected is the child that reduces the aggregate latency the most. The degree needed at C to parent P is thus created.

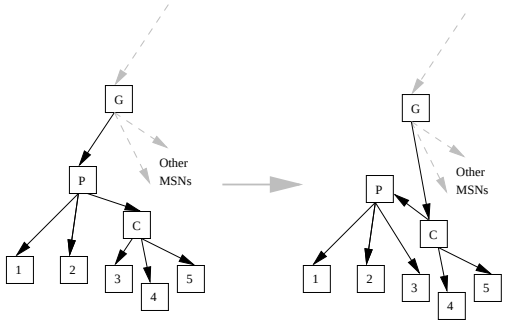

Figure 3.2: Parent-child swap

Iso-level-2 swap

In this transformation MSNs X and Y that have a common grandparent are swapped. Parent of X, P, becomes the new parent for Y and old parent of Y, Q, becomes the new parent for X. This transformation is performed if it reduces the aggregate latency at the grandparent node.

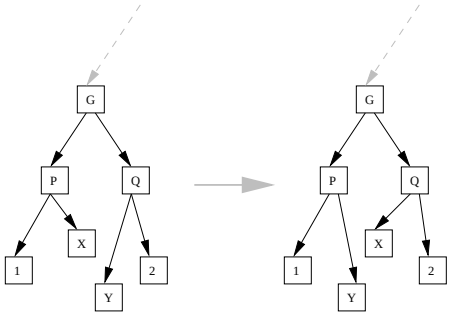

Figure 3.3: Iso-level-2 swap

Iso-level-2 transfer

The iso-level-2 transfer local transformation transfers node X from its current parent to a new parent which is on the same level as its old parent. The new parent has to have bandwidth available. Both the old and the new parent of X have the same grandparent G. The iso-level-2 transfer is performed if it reduces the aggregate latency at G.

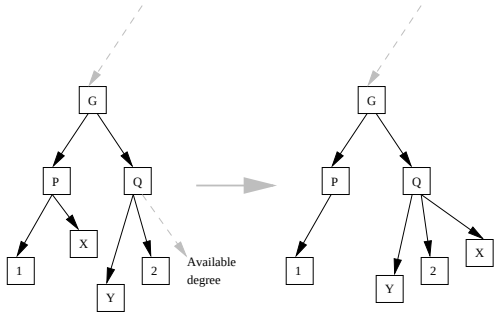

Figure 3.4: Iso-level-2 transfer

Aniso-level-1-2 swap

This transformation involves two MSNs, X having a parent P, and Y having P as grandparent. There is a restriction that X cannot be the parent of Y. Nodes X and Y swap their positions. The operation is performed only if it reduces the aggregate latency at P.

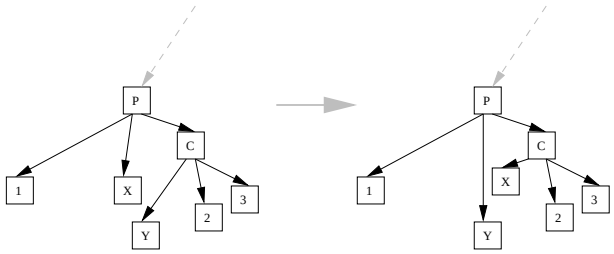

Figure 3.5: Aniso-level-1-2 swap

Local transformations lead to a local subtree aggregate latency minimum. If multiple local transformations are possible the one that reduces the aggregate latency the most is executed. However, as shown in [12] local transformations cannot guarantee a global minimum. In each local transformation period, an MSN chooses, with a low probability, to perform a global transformation as well. In a global transformation MSN P switches its tree position with another MSN Q. MSN P tries to find another MSN Q by performing a random tree-walk. P and Q cannot be each other's descendant or ancestor. The swap transformation is performed with probability 1 if the aggregate latency of the least common ancestor of nodes P and Q decreases and with probability of $e^{-\Delta/T}$ otherwise. T

is the temperature parameter of the simulated annealing technique [28] while Δ represents the increase in the aggregate latency of the least common ancestor of nodes P and Q.

The OMNI overlay specifies join and leave algorithms for MSNs while the details of join and leave algorithms for client nodes are not given. A joining MSN starts its join algorithm by sending a join request to the root MSN. At each level of the tree, a joining MSN N has three options:

- If the currently queried MSN P has bandwidth available, N joins as a child of P.
- N chooses child C of P and joins as parent of C and child of P. The cost of this option can be calculated through interactions between nodes N, P, C and the children of C.
- N retries the join process from some child of P, say R. The cost of this option is approximated to a hypothetical cost if N would join as a child of R.

The first option is always given precedence over the other two. If the first option cannot be executed the lowest cost option among the remaining two is chosen.

If a leaving MSN is a leaf then no further overlay restructuring is needed. Otherwise, one of the departing MSN's children is promoted to the position of its departing parent. The specific child is chosen such that the aggregate latency is minimized the most. The other children of the departing MSN simply execute the join algorithm starting at the newly promoted child.

Banerjee et al. assume that MSNs fail rarely. Therefore, no crash recovery algorithm has been specified for MSNs.

3.5 ZIGZAG

Compared to the previously reviewed algorithms, ZIGZAG [41] has some very unique characteristics. The ZIGZAG overlay organizes peers into bounded-size

clusters and builds a data delivery tree on top of those clusters. ZIGZAG does not have a rendezvous node and its join and leave algorithms are rather different from those of SpreadIt, HMTP, and OMNI. ZIGZAG has strict enforcement rules that guarantee the height of the tree to be $O(\log_k N)$ and the node degree to be $O(k^2)$, where N is number of peers and k is a constant parameter.

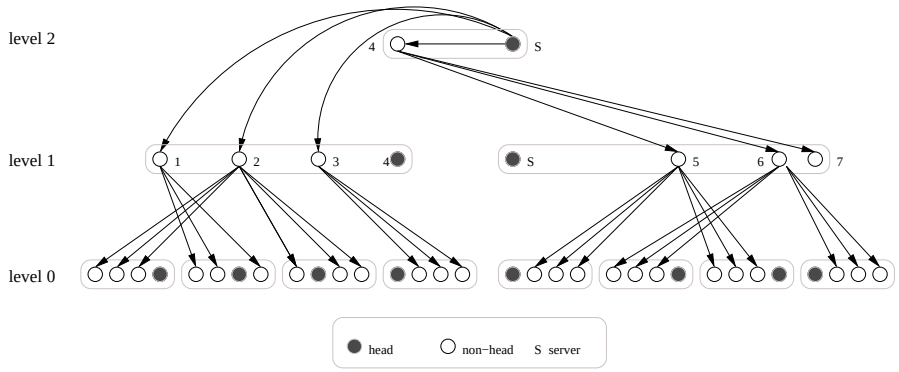

Figure 3.6: ZIGZAG overlay H=3 and k=4

In order to understand the join and leave algorithms we first have to look at the logical organization of ZIGZAG overlay peers. Peers are logically organized in a multi-layer hierarchy of clusters. ZIGZAG's overlay tree construction is governed by this logical organization. If H is the number of layers and $k > 3$ is a constant, then the following holds:

- Layer 0 (bottommost) contains all peers. The top layer contains the server. All peers of layer i are also part of layer $i-1$. If two peers belong to the same cluster in layer i then they belong to different clusters in layer $i-1$.
- Top layer (H-1) has only one cluster of size $[2, 3k]$ while all other layers have clusters of size $[k, 3k]$.
- One particular peer in every cluster is selected to be the head of the cluster. The server is the head of the top layer cluster. If a peer is the head of a cluster, it is a member of the cluster in the layer above. If a peer is a head of cluster at layer i, then it is also the head of its clusters at layers $0, 1, ..., i-1$.

The data delivery tree is constructed on top of the logical peer organization. The rules for the data delivery tree are the following:

- The server is the root of the data delivery tree.
- A peer can have data streaming links to other peers only when it is in its highest layer. For example, peer 4 in Figure 3.6 has no links at layer 1. It has links only at its highest layer, namely layer 2.
- Peer P, in its highest layer i, can only serve data to peers located at layer $i-1$. In addition, peer P serves data to all cluster members of a cluster at layer $i-1$ except the head of that cluster. However, peer P cannot serve data to peers that belong to the same cluster as peer P in a cluster at layer $i-1$. Thus, for example, peer 3 in Figure 3.6, whose highest layer is 1, serves data to cluster mates of node 4 at layer 0. The exception is the server, which serves streaming data to its immediate top layer cluster members.

The ZIGZAG join algorithm adheres to the rules specified above. Since there is no dedicated rendezvous node, joining peers contact the server directly. If the overlay logical organization has only one cluster, the joining peer node P is simply added to the cluster and connected to the server. Otherwise, P is bumped further down the multicast tree until finding the appropriate leaf peer X that will parent P. Suppose that while walking down the multicast tree, node P sends a join request to node X. There are two options at each recursive step:

- if X is *addable*, select a child of X, Y, that is *addable* and the overlay delay from P to the source through Y is minimal.
- otherwise select a child Y such that Y is *reachable* and the overlay delay from P to the source through Y is minimal.

A peer node X is *addable* if there exists a path in the overlay tree from X to a layer 0 peer whose cluster size is smaller than $3k$. This specific cluster size is introduced to allow adding of a new node in that particular cluster without violating the ZIGZAG cluster size requirement. A peer node X is *reachable* if there exists a path in the overlay tree from X to a layer 0 peer. For example, node 5 in Figure 3.6 is both *addable* and *reachable* while node 7 is neither. The overlay

delay from a peer to the source is not measured using the physical network delay. Instead, the path in the multicast tree from the peer to the source is considered. For each pair of adjacent peers on this path, the delay between them is measured, and these delays are added together.

The ZIGZAG overlay treats the peer voluntary leaves and crashes the same way. Due to the specifics of the ZIGZAG control protocol, its parent and children as well as the cluster mates become aware of P's departure. If P's highest layer is layer 0 then no further work is necessary. Otherwise, both the logical and the physical organization have to be changed.

Suppose P's highest layer was level j in cluster A. For each child in a cluster at level $j-1$ that P parented, the head of cluster A, X, is responsible for finding a new parent. X simply picks a cluster mate with a minimal degree.

Logical reorganization is equally simple. Since P used to be the head of j many clusters at levels $0, 1, ..., j-1$, the new head for those clusters has to be elected. A randomly chosen layer 0 cluster mate of P, Y, replaces P as a head for all clusters at levels $0, 1, ..., j-1$. In addition, Y selects its new parent to be the same peer that was parenting node P. If the cluster size becomes smaller than k or greater than $3k$ ZIGZAG specifies additional logical and physical reorganizations. See [41] for details.

ZIGZAG does not explicitly create mesh links. However, in order to keep track of its logical and physical organization, each ZIGZAG node P exchanges state information with a constant number of other nodes. P communicates with its parent, its children, and with its cluster mates in each cluster to which it belongs. From the above rules, we can derive that each node has at most $(3k-1)^2$ children. Furthermore, each node belongs to at most $\log_k N$ clusters and each cluster has at most $3k-1$ peers. Therefore, the worst-case control overhead is $O(\log_k N)$ - recall that k is a constant.

Chapter 4

Simulator

In recent years, intensified interest in peer-to-peer computing as well as in Internet media streaming have resulted in the research and development of a variety of P2P media streaming overlay protocols. While we have reviewed only four tree-overlay protocols, in the coming years we will undoubtedly witness the creation of many others.

Some overlay researchers, including Deshpande et al. [19], have implemented real P2P overlay streaming applications to test their overlays. While verifying P2P overlays in simulations can hardly compare to the ideal of real life application testing, the vast majority of researchers do not have the resources for such attempts and have thus resorted to simulations of P2P overlays.

Using simulators, researchers attempt to verify various quantifiable overlay assumptions such as overlay tree depth and width, as well as commonly measured metrics such as link stress and stretch. However, having each P2P overlay research team developing their own custom-designed simulators to test overlays is disadvantageous in various ways. Firstly, developing an overlay simulator is not a trivial task. It takes the resources away from the actual focus of the research - the overlay design. Secondly, if each research team designs its own simulator, they can hardly compare or verify each others results due to the bias introduced in such an approach. For example, one research team would usually pay special attention to all the intrinsic details of their own algorithms while possibly overlooking the same level of details in other algorithms. Researchers also tend to arrange the simulation setup and the input parameters which produce advantageous results for their own algorithms compared to other algorithms.

Many of these issues can be addressed with a common overlay simulator. The network research community has been successfully utilizing the ns simulator [14] since 1996. One of the main benefits of the ns simulator is its ability to simulate fine-grained details of network phenomena. For example, researchers have used ns to investigate low-level TCP details (selective and forward acknowledgment, congestion notification), router queuing policies (random early detection, class based queuing), multicast transport (scalable reliable multicast), multimedia (layer video, audio and video quality of service), wireless networking and various other problems [14].

Although ns has been used for scalable reliable multicast (SRM) and multicasting in general [14], the simulations performed were very detailed. Such simulations cannot be large-scaled. Indeed, while the ns design requirement allows for the abstraction of low-level simulation details [14], there has been little, if any, evidence that ns is able to scale simulations to the requirements of today's P2P overlays. Our doubts concerning the scaling of the ns simulator up to the requirements of P2P overlays were confirmed by the ns manual chapter - "Tips and Statistical Data for Running Large Simulations in NS" [26] that boasted examples of network topologies containing one thousand network nodes. [1]

In overlay simulations, OMNI researchers used a network topology of 10K nodes and up to 512 peer nodes, while the ZIGZAG overlay was tested on a 3240 nodes network topology with up to 2000 peer nodes. HMTP simulations used a network topology of 1K nodes and up to 500 peer nodes.

4.1 Internet modeling

To model the Internet effectively, a detailed understanding of underlying issues is crucial. The Internet is constantly evolving, its large-scale topological structure is hard to understand, and there has been little agreement about what determines a good topology generator in general. Although there have been many attempts to model the Internet, there is a consensus among Internet topology researchers that the question of accurate Internet topology generation remains open [40].

[1] It must be noted that there have been some successful attempts to scale ns simulations to 50K nodes using several custom extensions and modifications [35].

Researchers, however, agree that the Internet should be modelled at two distinct detail levels. The first level is the autonomous system (AS) level. In the Internet, an autonomous system is the unit of router policy. It is either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of a single administrative entity (such as a university, a business enterprize, or a business division). Nodes and edges in the AS level topology graph thus represent peering relationships between different ASes. The second representation is the router level, where nodes represent routers and edges are one hop IP-level links. There are various ongoing projects that are attempting to map the Internet on the router level using traceroute-like utilities [25]. Obviously, the router graph represents the Internet at a much finer level of granularity than the AS graph.

Internet topology generators have various modelling goals, focusing on a certain aspect of the Internet. Based on those modelling goals, Internet topology generators are usually classified into three categories: random, hierarchical and degree-based generators.

The first Internet topology generators were random graph generators. The main representative of the random graph generators is the Waxman generator [43]. This generator randomly assigns nodes to points in a plane and creates links between the nodes based on a probability function of the Euclidian distance between the nodes. The closer the node, the higher the probability that an edge is created between them.

The hierarchical topology generators appeared next. Hierarchical generators started with an assumption that the Internet is not a random graph but that it has a rather distinctive hierarchical structure. Hierarchical generators, as their name implies, focused on replicating the Internet's hierarchical structure. GT-ITM [15] is a representative of hierarchical topology generators and one of the first Internet topology generators to gain wide attention in general. GT-ITM, and its Transit-Stub generation model received wide acceptance due to its ability to reasonably replicate the Internet's hierarchical structure. GT-ITM equates stub domains to the interconnected local area networks and the transit domains to wide- or metropolitan-area networks. GT-ITM's Transit-Stub generation model first creates a connected random graph where each node represents an entire transit domain. Each transit domain node is then transformed into a connected random graph, representing a transit backbone, where each node could be a

gateway to a stub domain or a router to other transit domains. Finally, stub domains are generated and attached to the each node in the transit backbone.

GT-ITM has been considered the state-of-the-art Internet topology generator until a seminal paper by Faloutsos et al. [22] revealed that the Internet's degree distribution is a power-law. Degree distribution refers to the network node connectivity distribution on both the intra and the inter AS level. A power-law relationship between two scalar quantities x and y is of the form $y = x^k$, where k is a constant. Using real Internet traces, Faloutsos et al. observed four power-laws:

- P1 (rank exponent) relationship between the outdegrees of nodes sorted in decreasing order and the ranks of the nodes in such order. The outdegree d_v of the node v is proportional to the rank of the node, r_v, to the power of some constant.
- P2 (outdegree exponent) the frequency of an outdegree, f_d, is proportional to the outdegree d to the power of some constant.
- P3 (hop-plot exponent) the total number of pairs of nodes, $P(h)$, within h network hops is proportional to h to the power of some constant.
- P4 (eigen exponent) the eigenvalues, λ_i, of the adjacency matrix of a topology, sorted in decreasing order are proportional to i raised to some constant power.

Since the hierarchical generators do not produce power-law adhering topologies, some members of the research community have concluded [40] that the hierarchical generators are unsuitable for Internet modelling after all. The seminal paper of Faloutsos et al. has been a motivator for a new generation of Internet modelling generators that aim to replicate the Internet power-law degree distributions - degree based generators. Several degree-based Internet topology generators emerged recently [9, 30].

After the emergence of the degree-based generators, Zegura, the principal author of the GT-ITM topology generator, agreed [44] that the GT-ITM does indeed not adhere to power-law distributions and that it is probably not suitable for AS network representation. However, Zegura claims that their topology generator is still a suitable candidate for the generation of moderate size router

level topologies. Zegura has argued that their method builds topologies whose high-level structure fairly well reflects the high-level structure of the Internet.

It is a widespread belief [40] that it is more important for topology generators to accurately model the large-scale structure of the Internet, for example its hierarchical structure, rather than modelling the local phenomena like the degree distribution. Since we needed large-scale network topologies for our simulations we wondered if degree-based generators would be the best fit for our simulations after all? Which class of generators do indeed most accurately resemble the Internet when looking at the large-scale properties?

Tangmunarunkit et al. [40] compared hierarchical and degree-based generators using various topology metrics in an attempt to find out which class of generators more accurately models the large-scale structure of the Internet. Following the belief that topology generators that are focused on the Internet's large-scale structure properties model the Internet more accurately, Tangmunarunkit et al. started with an assumption that hierarchical generators would thus generate more accurate large-scale Internet topologies. Much to their own surprise their findings pointed in the opposite direction. Tangmunarunkit et al. found that degree-based generators are significantly better at representing large-scale Internet topologies at both the AS and the router level. This paradoxical finding would lead to a conclusion that degree-based generators must create hierarchical topologies as well since it is well known that the Internet has hierarchical structure. And indeed Tangmunarunkit et al. found that although degree-based generators do not aim at representing hierarchical properties of the Internet, the power-law nature of degree distributions results in a substantial level of hierarchy.

4.2 Network topology

Selecting a particular topology generator depends on the several factors, including the nature of the research, the size of required topology, as well as the importance of the various Internet characteristics for a certain research project. As discussed above, the most appropriate solution for the large-scale P2P simulations would be a generator from the degree-based class of generators. The BRITE [30] Internet topology generator has recently emerged as a universal topology generator. We believe that BRITE is the best choice from the degree-based generators, and

Internet topology generators in general, for several reasons:

- Representativeness - BRITE produces accurate large-scale synthetic Internet topologies. Accuracy is reflected in both hierarchical and degree-distribution properties of the Internet.
- Flexibility - BRITE generation models can easily be enhanced by additional node and link metadata like, for example, firewalls, bandwidth and delay.
- Extensibility - BRITE's object-oriented architecture provides an extendible model framework for adding entirely new generation models.
- Efficiency - BRITE generates large-scale topologies of more than 500,000 nodes using reasonable CPU and memory resources.
- Interpretability - BRITE is able to import topologies from various topology generators and combine them with other topologies. BRITE can also import real Internet traces and use them in a topology generation model. Besides standard BRITE export format, several other export formats are built-in and new formats can be easily added.
- Portability and user friendliness - BRITE is implemented in both C++ and Java and includes a GUI.

From the vast array of available Internet topology generation models provided by BRITE, we have the chosen a hierarchical generation model. BRITE's hierarchical generation model allows a separate generation model for the AS and the router level representations. In the light of recent power-law research by Faloutsos, Tangmunarunkit's comparison of hierarchical and degree-based generators, we could have selected a one level power-law adhering model. The main reason for choosing the hierarchical model is the necessity for the Internet wide area network representation in P2P overlay simulations. The real life deployment scenario for most of the P2P overlay media streaming systems is most likely the Internet itself. The combination of inter AS policy routing and intra AS shortest path routing determine a routing structure on any wide scale network of peers. In order to capture these important routing phenomena in our simulation we need a two level hierarchy.

The actual BRITE generation model we used is the top-down hierarchical topology generation model. BRITE allows for the use of a different model for

each hierarchy level. Since Tangmunarunkit et al. have found power-law graphs on both the AS and the router level we have used the power-law adhering Barabasi-Albert [13] generation model for both levels of the hierarchical generation model. Barabasi and Albert suggested that there are two reasons for the emergence of a power law [22] in the frequency of outdegrees observed in network topologies: incremental growth and preferential connectivity. Incremental growth refers to continually enlarging networks formed by the addition of new nodes. Preferential connectivity is described as a tendency of newly added nodes to connect to existing nodes that are highly connected or popular. Medina et al. [31] consider two additional factors that might be a cause for emergence of power-laws: geographical distribution of nodes and locality of edge connections. Medina et al. argue that geographical distributions of nodes that are skewed (e.g. heavy-tailed) appear to be a realistic cause for power-laws. Locality of edge connection mimics node distribution in a sense that those few heavily populated areas have highly connected nodes, while the rest of the nodes are loosely connected. All of the four factors are implemented in the BRITE topology generator.

The top-down generation model approach first generates the AS level topology. In a second step, each AS level node is expanded into a router level topology using the same generation model.

Router level topologies from different AS nodes are interconnected according to the connectivity of the AS level topology. If (i, j) is a link in the AS level topology, then a node u from the router level topology associated with AS node i, and a node v from the router level topology associated with the AS node j, can be connected in four different ways:

- Random: u and v are picked randomly.
- Smallest degree: u and v are nodes with the smallest degrees in the respective router topologies.
- Smallest degree non-leaf: u and v are nodes of smallest degree in respective topologies but not leaves (i.e. having a degree of at least 2).
- Smallest k-degree: u and v are nodes from the respective topologies having a smallest degree which is at least k where k is a user specified constant.

For our simulations we have selected the smallest k-degree connection method. We have set k to 5 in all generated topologies. This connection method exhibits some form of preferential connectivity between the nodes from different ASes. To the best of our knowledge it is unknown how to best represent AS interconnection. There are many issues involved when two ASes connect and it is not clear if the largest degree routers in two ASes should have a link between them.

4.3 Modeling interdomain routing

Representative simulation of Internet routing has been a challenging task [33]. In order to properly model the complex Internet routing one needs to have a detailed understanding of interdomain routing. Interdomain routing in the Internet is governed by the policy-based border gateway protocol (BGP). BGP allows each AS to administer its own routing policy. A BGP router at AS A can selectively broadcast to other ASes that are reachable from A. These routing policies are constrained by the contractual commercial agreements between ASes.

Since routing in the Internet is determined by BGP, physical node connectivity does not always imply Internet traffic reachability. Relationships between various ASes determine whether or not traffic can flow between physically connected nodes. For example, a customer AS can set its policy so that it does not provide transit traffic between its providers. More specifically, it is not hard to envision a scenario where a customer AS is connected to two provider ASes, A and B. A customer AS has no obligation and probably no interest to set its routing policy to allow traffic flow between its two providers. Thus even though there is a physical path from provider A to provider B (through a customer AS) no traffic will be exchanged between the two providers through that customer AS. In order to simulate Internet routing, AS topologies, generated by topology generators, have to be annotated with the additional metadata representing peering relationships between the corresponding AS nodes.

Naturally, peering relationships have a dynamic nature and are continuously changing. How can we model interdomain routing if the peering relationships between various ASes are constantly evolving? Gao [24] suggested to abstract from details and focus on AS relationships. In particular she presented a heuristic algorithm that classifies the types of AS peering relationships. Gao distinguished

three AS relationship types: customer-provider, peering, and sibling relationships. The real Internet BGP traces available publicly from the Route-Views project were used. After obtaining BGP traces, Gao ran her heuristic AS relationship classifier algorithm to find out that 90.5% of AS pairs are in customer-provider relationships, 8% are in peering relationships and finally only 1.5% are in sibling relationships. Gao verified the inferred results with AT&T's internal routing information and confirmed that 99.1% of the inferred relationships are correctly identified. Also more than half of the inferred sibling-sibling relationships were confirmed using the WHOIS lookup service.

As previously discussed, we use BRITE generated synthetic AS and router topologies for the simulations conducted. Obviously the synthetic AS topology as-is does not have any additional metadata about the AS peering relationships. In order to use AS topologies with annotated AS relationships, as suggested by Gao, one would have to take the following steps. First, the real BGP traces have to be obtained that are available from various public BGP tracers, one of them being the Route-Views. Second, Gao's AS relationship or possibly some new algorithm should be run over the obtained BGP traces. The created AS topology annotated with inferred peer relationships should be imported by BRITE. BRITE's node and edge model has to be updated to accommodate the additional node and edge metadata.

After these steps have been performed, the inter domain routing algorithm can use the additional node and edge metadata to have a more representative inter AS routing. As previously discussed in Section 4.2, we have intentionally made a distinction between inter AS and intra AS routing so that routing improvements can be easily incorporated in the future.

4.4 Bandwidth distribution

Although the P2P overlay simulators that we have reviewed have chosen to abstract from peer Internet bandwidth connection characteristics in their respective simulations, we think that the heterogeneous nature of P2P peers, as observed in a peer measurement study by Saroiu et al. [38], is an important peer characteristic to be captured in P2P overlay simulations.

BRITE has the ability to assign bandwidth to the generated network links and thus the available bandwidth between two network nodes can be determined by the slowest link on the path between them. However, in our simulations we have not calculated the bandwidth by traversing the path that connects two peer nodes but have rather assumed that the slowest link is usually the last hop link that connects the peer to the Internet. A peer's Internet connection bandwidth, determined by its last hop link, is a property that remains constant for the lifetime of a simulation. In fact, in order to simplify the model, we have ignored BRITE's link bandwidth assignment altogether. We have not modelled a peer's downlink bandwidth distribution and have assumed that a peer has an appropriate amount of downlink bandwidth to receive the media stream.

The distinction between uplink and downlink bandwidth is important since the available uplink bandwidth of the computers serving content is of a greater importance. If a media stream being served from a peer P to a peer Q, P's uplink is more likely to be exhausted or congested than Q's downlink thus resulting in a stream interruption for peer Q. Congestion problems lead to a bad quality of service experienced by peer Q. While the Internet user in a corporate or academic environment may not experience significant service degradation when the uplink is used by a P2P application, many users who connect to the Internet through a consumer broadband connection such as cable or DSL experience such problems. In order to serve a data stream to other peers, a peer needs to have available uplink bandwidth. Available uplink bandwidth is a peer's property that may change during the lifetime of the peer's participation in the overlay. Available uplink bandwidth for peer P depends on P's uplink connection bandwidth as well as the number of peers that currently receive their stream from peer P.

Using a specifically designed bandwidth-measuring tool named SProbe, Saroiu et al. have conducted an Internet bandwidth connectivity measurement study of the Gnutella and the Napster P2P peers. Saroiu et al. first used a specialized crawler to collect over a million IP addresses of the Gnutella network peers as well as over half-a-million Napster IP peer addresses. A set of half-a-million randomly selected Gnutella nodes were used for the active bandwidth measurements probing. Downlink bandwidth measurements were successfully conducted on 223,552 Gnutella peers, while uplink bandwidth measurements were successfully conducted on 16,252 Gnutella peers. The Napster measurements were not as successful since there were a number of Napster users complaining about the active crawler probing. However, Saroiu et al. believe that 2,049 successfully

measured Napster peers are a good enough representation of the entire Napster peer population.

The overlay simulator we have developed can accept any specified peer bandwidth distribution as one of the simulation parameters. However, for easy referencing and benchmarking we have used the Gnutella peer distribution as captured by Saroiu et al.

4.5 Peer join and duration distribution

One of the most recognizable characteristics of a fully decentralized distributed system and especially a P2P overlay-streaming system is the highly unpredictable nature of peer node joins and leaves. It is common practice [32] to model these systems in a stochastic, continuous-time setting. We have modelled the join of peer nodes in the overlay by a Poisson distribution and the lifetime of a peer node participation in the overlay by an exponential distribution.

Since we wanted to observe overlay behavior on the various peer population scales we have set the arrival and lifetime parameters accordingly.

Chapter 5

Simulation results

Before delving into the details of our simulation results we will give a brief overview of the simulation experiments conducted by those who developed the SpreadIt, HMTP, OMNI and ZIGZAG overlay algorithms.

5.1 Previous simulation results

All of the reviewed simulation experiments used small ad hoc network topologies and simplified simulation parameters like peer outdegree distributions as well as peer join and duration distributions. After the overview of all four overlay algorithms' simulation results, we describe how those simulations can be replicated within our simulation environment and we compare our results with theirs.

5.1.1 SpreadIt

The SpreadIt simulator aims at creating a representative simulation model of the actual SpreadIt streaming architecture. In [19] no mention was made of the topology generator used. The size of the peer overlay contained a maximum of 1000 nodes and a maximum outdegree of 10 was allowed. Since SpreadIt data channels are based on the lossy UDP protocol, the focus was mostly on modelling the packet loss experienced by the peers in the various join/leave algorithm setups.

A peer node, as modelled by the SpreadIt simulator, can be in three states:

inactive, active and transient. A node is inactive when it is not subscribed to the stream. A node is active when it is subscribed to the stream and is receiving the stream and finally a node is transient when it is subscribed to the stream but is not receiving the stream.

Simulated SpreadIt peers change states in a probabilistic fashion. A peer can make a transition from one state to another at each time tick. For example, an inactive node can move to the transient state with some fixed probability. If a node is in the active state it can move to the inactive state with some fixed probability and so on.

Deshpande et al. simulated the various combinations of SpreadIt join and leave policies and observed the packet loss in such simulation setups. They predicted that the P2P approach would lead to an increase in packet loss since the packets travel more hops from the source to the peers compared to the client/server approach. They found that their SP join policy results in only 2.5 times more packets loss on average per peer node than the classic client/server approach. The increase factor is equal to the average height of a node in the overlay tree. The cause of the increased packet loss is attributed to the cumulative packet-loss effect. Deshpande et al. also concluded that leave policies that intensively compact the overlay tree (GFA, RTA) suffer more packet loss than less intensive tree compacting leave policies (GF, RT). This finding is somewhat counterintuitive since one would expect that less intensive compacting policies would result in higher trees which in turn would lead to a higher packet loss.

Deshpande et al. also investigated the flash crowd effect and concluded that any join policy will perform ten times better than the classical client/server model. They attribute the degradation of performance to the server becoming a bottleneck in handling the setup of streaming data for each client. In the P2P approach a server quickly distributes incoming clients away from the server towards the overlay leaves who then eventually setup the stream for the incoming peers.

5.1.2 HMTP

Zhang et al. [45] performed simulations of their P2P streaming overlay using a random flat topology generated by the Waxman topology generator. The gener-

ated topology had 1000 nodes representing routers and 3300 edges (links). Some additional nodes representing end hosts were generated and attached to router nodes. There was no mention of how many additional nodes were attached to the routers. Peer heterogeneity simulation was simplified as well. No specific peer bandwidth distribution was used and the maximum node degree was set to 8. Three metrics were used to measure the quality of the overlay.

The first one is the tree cost which is defined as the sum of delays on the trees' links. The goal of this metric is to capture the total amount of network resources used by the tree. The ratio of the overlay tree's cost to the IP multicast tree cost is thus the tree's cost ratio. The simulation setup included growing the overlay from 20 to 500 nodes. The tree cost ratio for HMTP ranged from 1.2 for 20 members up to 1.5 for 500 members. To put it in perspective, IP multicast has a tree cost ratio of 1 while the original client/server model, as simulated by HMTP researchers, had a tree cost ratio ranging from 1.7 for 20 members up to 3.5 for 500 members.

The second metric used in the simulation was the tree delay. The tree delay represents the node delay from one node to another node along the overlay's links. The ratio between the tree delay and the unicast shortest path is the delay ratio. This metric is somewhat similar to link stretch except that Zhang et al. have measured tree delays not only between the source and an arbitrary node but between other tree nodes as well. The reason this metric was defined between two arbitrary nodes is that HMTP does not prevent any node to multicast data. They also defined the group diameter as the maximum delay between any pair of nodes. The ratio of the group diameter and the IP multicast tree diameter is called the group diameter inflation. Using the same simulation setup as in the first metric observation Zhang et al. have found that most of the pairs have a delay ratio of 5 while the worst case delay ratio was 12.

The third and final metric is the link load. The link load is equivalent to the link stress. The simulation setup consisted of a 100 member overlay. More than half of the physical links used had a link load of 1. About 90% of the links had a link load of less than 4. There were only a few links having a link load between 4 and 8.

5.1.3 OMNI

The OMNI simulations were conducted on a ten thousand node network topology generated by the GT-ITM network topology generator. MSN peer nodes were randomly attached to the network topology. The number of MSN nodes in various simulation setups varied between 16 and 512 nodes. There was no particular distribution model for simulating peer arrivals and leaves. A predefined set of MSN peers was simulated to join and leave the overlay at particular time instances. The bandwidth model was a uniform distribution with MSN nodes being able to serve between 1 and 5 other MSN nodes.

As the OMNI P2P overlay system has an objective to minimize the average tree latency, most of the simulations were focused on gathering the results for determining the average tree latency. More specifically, the conducted simulations focused on the tree latency results affected by varying the overlay input parameters. The first simulation scenario, the convergence, implied that as the probability of random swap operations increases the search for a global minimum becomes more aggressive. Besides the probability of the random swap parameter, the effect of the temperature parameter was also observed. Higher values of the temperature parameter imply that a random swap that leads to an average tree cost increment is permitted with a higher probability and thus to higher oscillation in aggregate tree latency.

The second simulation scenario, the adaptability, examines the adaptation of the overlay to the various join and leave scenarios. The distractibility measures how quickly the overlay converges to the optimum average tree latency. The setup included an initial overlay tree consisting of 248 MSNs. 64 MSNs would either join or leave every constant number of ticks. Banerjee et al. have found that OMNI reaches to within 6% of the optimal average tree latency very quickly. Such prompt adaptation has been attributed to local transformations.

Banerjee et al. have also verified their assumption that the MSNs that serve more clients would move up the overlay tree towards the source while the MSNs that serve less clients move towards the leaves of the overlay tree.

5.1.4 ZIGZAG

The ZIGZAG overlay simulator used the GT-ITM topology generator to create a 3240 node network topology. The simulation setup included 2000 simulated peers. Simulation input parameters have been customized to create the ZIGZAG clusters containing at most 15 and at least 5 peers. Peers were assumed to have an infinite uplink bandwidth. However, Tran et al. have tracked in their simulations the peer uplink distribution required to create the ZIGZAG overlay. The simulations included three different scenarios: a node failure-free setup, a setup where node failures were allowed and finally a simulation setup comparing ZIGZAG to the NICE [11] P2P streaming system.

In the first simulation scenario 2000 peers were subsequently added to the overlay and the statistics about the control overhead, the node degree, the peer stretch, and the link stress were collected. Tran, Hua and Do have found that the control overhead for joins is not significant. On average a new peer contacts only 2.4% of the whole overlay peer population. The overall control overhead was also low. In order to manage ZIGZAG clusters each peer node exchanged state with on average only 12 other peers. The average outdegree of a peer that forwards its stream to other peers was 10 while the highest outdegree observed was 22. Taking into account that this simulation setup consisted of 2000 peers placed on a network of 3240 nodes the link stress and the stretch were surprisingly good. In this particular simulation setup ZIGZAG had a stretch of 3.45 for most of the peers and link stress of 4.2 for most of the network links used.

In the second simulation setup 2000 peers that were subsequently added to the overlay were subsequently failed in batches of 200 until 1000 were left in the overlay. Most of the failures affected the layer-0 peers and thus no additional restructuring was needed. The control overhead for failures is that on average 0.96 nodes are contacted no matter what the population size is. This finding supports the theoretical analysis that failure overhead doesn't depend on population size. Similar findings were observed in cluster merges.

Tran et al. have provided neither the results of node outdegrees nor those of link stress and stretch in the second simulation scenario. We believe that they have assumed that the results of those metrics would have been similar to the first simulation scenario.

5.2 Verification of the previous simulation results

Before conducting any detailed simulation experiments and comparing of the implemented overlay algorithms we wanted to verify the individual simulation results of the HMTP, OMNI, and ZIGZAG overlay algorithms as reported by their respective authors. We have not tried to reproduce the results of the SpreadIt overlay algorithm. Trying to reproduce the SpreadIt simulation results would require a significant change in our overlay simulator. The SpreadIt simulations focus on the packet loss experienced by the peers in the various join and leave algorithm scenarios. Our implementation of the SpreadIt overlay and all other overlays does not involve simulating the probability of packet loss but we rather assume lossless data channels. However, since the SpreadIt overlay construction algorithm is much simpler than the other algorithms that we have implemented we are confident in the correctness of our SpreadIt simulation implementation. High correlation of our simulation results with the published simulation results would indicate the flexibility of our own P2P simulator to reproduce the particular simulation scenarios as well as the correctness of the implemented algorithms.

5.2.1 HMTP

The first step in replication of the HMTP simulation environment was the creation of an approximately similar network topology used in the original HMTP simulations by Zhang et al. However, Zhang et al. do not specify the exact number of the network nodes they used in their simulations except that 1000 nodes were created to represent the routers. An unspecified number of end-host nodes were subsequently added to each router.

We used BRITE's top-down hierarchical topology generation model where both the AS and the router level were created using the Waxman topology generation model. The number of ASes was set to 100 and each AS had 100 routers. Thus the total number of nodes in the network topology was 10000 nodes. In order to replicate HMTP's peer heterogeneity bandwidth model from simulations conducted by Zhang et al. we created a single type of peer node having an uplink outdegree of 8.

The quality of the created HMTP overlay, in the original simulation by Zhang et al. was verified with three overlay metrics: the tree cost, the tree delay and the link load.

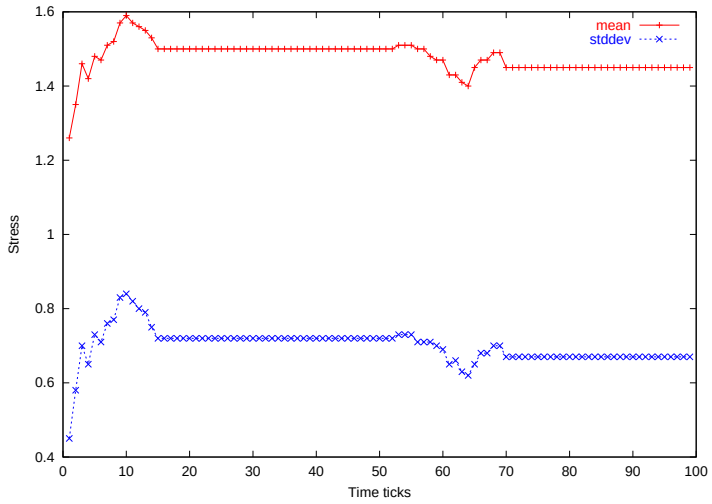

Figure 5.1: HMTP link stress

Since our set of predefined metrics includes the link stress (link load) we have proceeded to verify their link stress findings. The simulation setup involved building a 100 peer node HMTP overlay and taking link stress measurements. Figure 5.1 depicts mean and standard deviation of the link stress during 100 simulation ticks. HMTP nodes are added in batches of ten in the first ten ticks and half of the nodes are failed in between ticks 50 and 70. The link stress simulation results are very similar to the link load results of Zhang et al.

5.2.2 OMNI

In order to replicate the OMNI simulation environment, we used the BRITE generator to create a ten thousand node network topology. We used the top-down hierarchical topology generation model where both the AS and the router level were created using the Waxman topology generation model. The number of ASes was set to 100 and each AS had 100 routers. Since the OMNI simulations used a predefined set of MSN peers leaving and joining the overlay at particular

time ticks, we used our discrete peer arrival and duration distribution model. In order to replicate OMNI's peer heterogeneity bandwidth model from simulations conducted by Banerjee et al. we created five different types of MSNs each having uplink outdegree between 1 and 5.

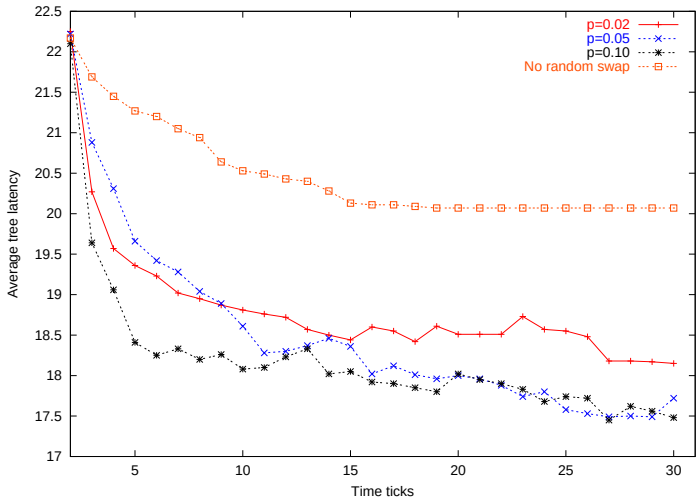

Figure 5.2: Varying the probability for random-swap in 256 MSN convergence setup

The first OMNI simulation we conducted was the 256 MSN convergence simulation. We added 256 MSN nodes to the OMNI overlay and then attached between 1 to 5 clients chosen uniformly at random to each MSN. The final setup included 256 MSN nodes serving a total of 359 clients.

Figure 5.2 shows the results of the 256 MSN convergence simulation. We conducted 4 simulation runs each having a different probabilistic random swap transformation parameter. When the parameter is set to 0, no random transformations occur. The stable value of 20.1 ms average tree latency is reached rather quickly. In this case the fall in average tree latency is accounted to the local transformations only. As we can see in Figure 5.2, the higher the value of the probabilistic random swap transformation parameter the more contentious the search for the global minimum becomes. Using the parameter value of 0.1 allows the OMNI overlay tree to achieve an average tree latency of less than 17.5 ms in about 30 simulation ticks. In this case both the local and the random swap

operations account for the drop in average tree latency.

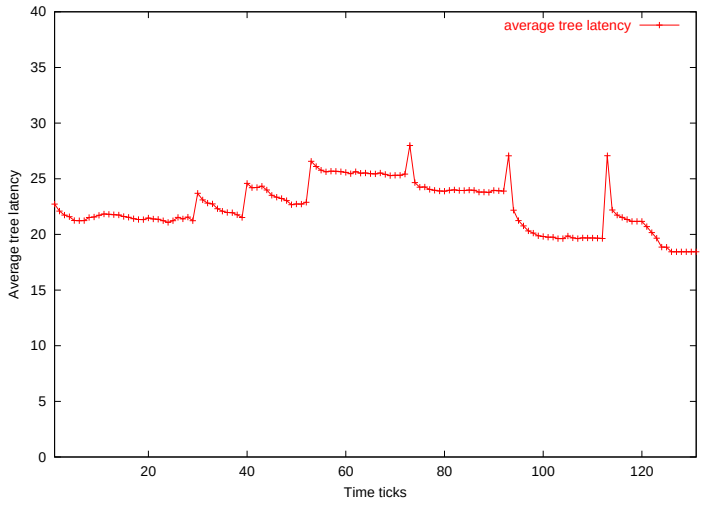

Figure 5.3: Join leave experiments with 248 OMNI nodes

The second OMNI simulation we conducted was the 248 MSN node adaptability simulation. We added 248 MSN nodes to the OMNI overlay in a sequence of 128, 8, 16, 32 and 64 nodes every ten time ticks. Thus at simulation time tick 40 there were 248 MSN nodes. Figure 5.3 plots the average tree latency. Notice how each addition of MSN nodes causes an increase in the average tree latency for each five MSN node addition instances.

At time tick 50 we started crashing 48 MSN nodes in intervals of every twenty time ticks. Notice how similarly to the node addition, node crashes also cause an increase in the average tree latency. However, after both additions and crashes, the average tree latency quickly converges back toward the global average tree latency minimum.

Comparison with the original OMNI simulation results

By verifying the results of OMNI's convergence and adaptability simulations we can assert the correctness of our own OMNI overlay implementation. We were

able to closely reproduce both the results of the convergence and adaptability simulations.

Similar to the results of the corresponding 256 MSN convergence simulation conducted by Banerjee et al. we have also found that as the probability of random swap operation increases the search for a global minimum becomes more aggressive. The average tree latency results of 256 MSN convergence simulation as published by Banerjee et al. are different from our average tree latency results due to the different MSN unicast latency distribution used in these two simulation scenarios. Banerjee et al. used unicast latencies that varied from 1 to 200 ms between various MSNs while in our simulation the average latency between parent/child MSNs was 6.2 ms with a standard deviation of 1.2. The lowest measured unicast delay between parent/child MSNs was 0.9 ms while the maximum was 14 ms. However, the percentage drop of the average tree latency from the start to the end of the simulation is very similar.

The results of the adaptability simulations are very similar to the adaptability simulation results observed by Banerjee et al. as well. Both the bulk node joins and the leaves or crashes cause the spikes in the average tree latency and just as Banerjee et al. observed the average tree latency converges back towards the global average tree latency minimum.

5.2.3 ZIGZAG

The first step in replicating the ZIGZAG simulation environment involves creating a 3200 node network topology. We used BRITE's top-down hierarchical topology generation model where both the AS and the router level were created using the Waxman topology generation model. The number of ASes was set to 32 and each AS had 100 routers resulting in a network topology of 3200 nodes. In the ZIGZAG simulations a predefined set of peers joins and leaves the overlay at each particular time tick. Therefore, we used our discrete distribution model for both peer joins and leaves. Since the ZIGZAG peer heterogeneity bandwidth model has nodes of unlimited uplink bandwidth capacity, we defined a single uplink bandwidth type having an uplink outdegree of 2000 nodes. Since all the ZIGZAG simulation scenarios involve at most 2000 peers we could possibly have a case when the source is parent to all the nodes hence requiring uplink capacity of 2000.

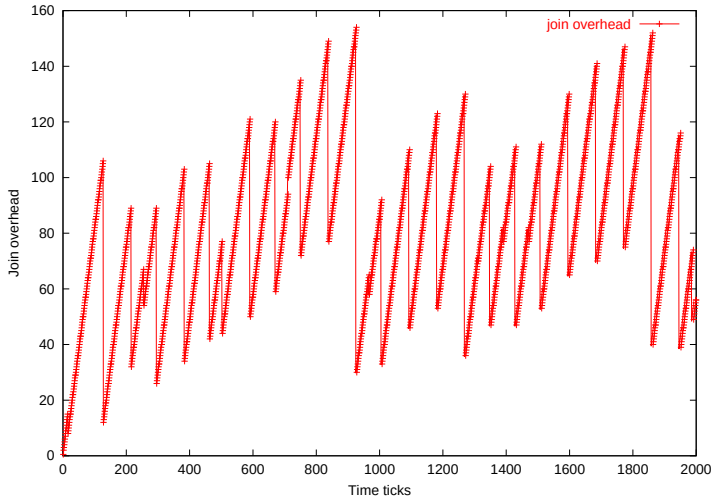

Figure 5.4: Join overhead

The first simulation setup that we replicated was the "no failure" simulation conducted by Tran et al. In that simulation scenario 2000 peers were added to the ZIGZAG overlay and no nodes were allowed to leave or crash. Tran et al. collected fine grained results of their simulations by recording each join, leave, split or merge event. In order to replicate the results on that granularity level we ran our simulator for 2000 simulation ticks and joined one ZIGZAG node in each tick. Since our simulator collects data after each tick we were also able to detect relevant data regarding split, merge and leave events.

Figure 5.4 shows the results of the join overhead metric. Join overhead is measured by the number of peers each newly arriving peer has to contact during the join procedure. As illustrated in Figure 5.4, a joining peer has to contact on average around 80 other peers or about 4% of the total peer population.

The second metric recorded was the split overhead. Split overhead is measured by the number of reconnections required during each cluster split operation. Figure 5.5 shows the number of peer reconnections required during each tick. There is a possibility of more than one cluster split occurring in a particular tick. In total we recorded 280 clusters splits. For each split, there were on average about 9.5 peer reconnections needed. Notice how each high value in split overhead depicted in Figure 5.5 coincides perfectly with a high downfall in the join overhead

depicted in Figure 5.4.

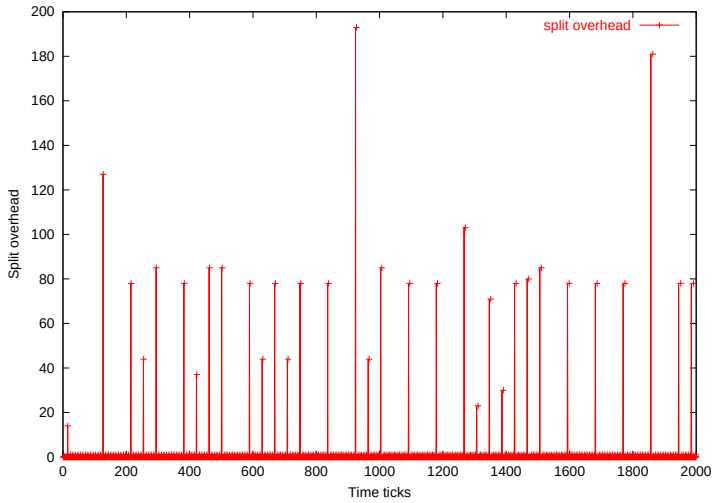

Figure 5.5: Split overhead

The last metric recorded during the "no failure" simulation setup was the average outdegree of the nodes that are serving other peers. Figure 5.6 shows that a peer that serves content to other peers, does so to about 10 other peers. The initial spike in the outdegree is credited to the source serving many peers before the first split.

We have also replicated the second "failure possible" simulation scenario. In this simulation setup, in total 2000 peers were added to the ZIGZAG overlay in the first 1000 simulation ticks. Half of the 2000 peers were then subsequently crashed between ticks 1000 and 2000. Since we used a uniform duration distribution, each tick resulted in approximately one peer leave or crash. At tick 2000 there were 1006 peers left in the overlay.

The first metric we have measured in this simulation scenario was the failure overhead. Failure overhead is captured by the number of required peer reconnections triggered due to a node leave or crash. As depicted in Figure 5.7 most of the failures do not cause any reconnections since these represent failures of the layer 0 nodes. Although a duration distribution function was set to crash one thousand nodes in a thousand ticks there is a possibility that more than one

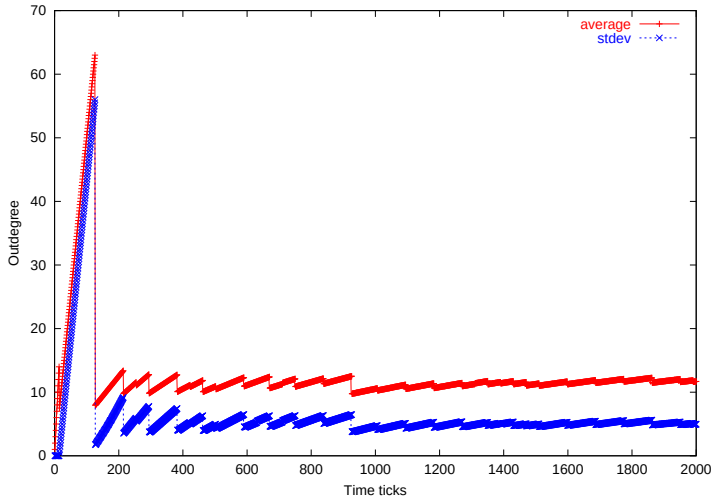

Figure 5.6: Average node outdegree

non-layer 0 node leave or crash occurred in a particular tick. For example, the worst case of failure overhead requiring 24 reconnection was probably caused by more than one non-layer 0 node crash or leave occurring in that particular tick. Non-layer 0 node failures required on average 4.5 peer reconnections.

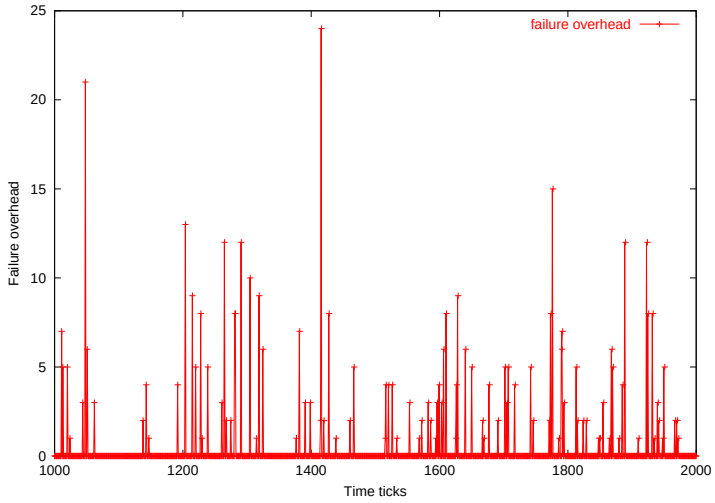

Figure 5.7: Failure overhead

The last metric measured in the second simulation scenario was the merge overhead. Merge overhead is represented by the number of peer reconnections required during cluster merge operations. Figure 5.8 shows the total number of reconnections due to cluster merge calls in each simulation tick. There is a possibility of more than one cluster merge occurring in a particular tick. For example, the worst case of merge overhead was 31 peer reconnections but it was caused by 2 merge calls. We recorded a total of 114 merge calls each requiring on average 5.5 peer reconnections.

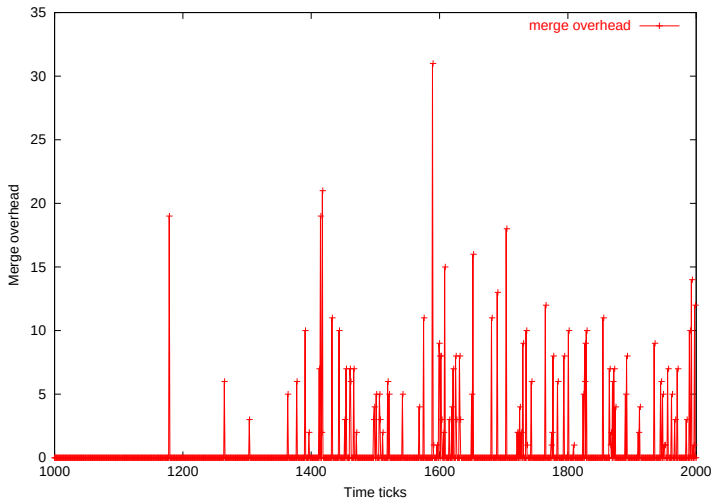

Figure 5.8: Merge overhead

Comparison with the original ZIGZAG simulation results

Arguing the correctness of our ZIGZAG overlay implementation is intrinsically tied to reproducing the original results published by Tran et al. Similarities are most evident from the simulation result data patterns, primarily from the correlation between the split and the join overhead. Other similar data patterns worth mentioning are the "heart-rate" like nature of the join overhead as well as the occasional spikes in the split overhead. Similar spikes are noticeable in the failure and the merge overhead as well.

Both Tran et al. and we got the exact same average outdegree of 10 for the data stream forwarding peer nodes. The join and the split overhead results that

we got were somewhat higher than the average values in join and split overhead from the original simulations. Namely, the average join overhead that Tran et al. got in their simulations was around 50 peer contacts while the average split overhead was 5 peer reconnections. In contrast, we got an average join overhead of 80 peer contacts and the split overhead of 9.5 peer reconnections. Interestingly enough the simulation results of the non-0 layer failure overhead and the merge overhead that we got were somewhat lower than the original results. We have recorded the non-0 layer failure overhead of 4.5 and the merge overhead of 5.5 peer reconnections while Tran et al. observed 10 and 11 peer reconnections respectively.

How do we attribute these minor differences in the simulation results? The most likely causes are the intricate details of the cluster split and merge. We suspect that we did not use the same frequency and threshold of cluster split and merge as Tran et al. did. However, overall there is a great degree of similarity between our results and the original simulation results published by Tran et al.

5.3 Simulation results

After verifying that our implementations of the HMTP, OMNI, and ZIGZAG overlay algorithms are reasonable representations of their original counterparts we can proceed to administer a detailed comparison of these overlay algorithms in our P2P simulator. Conducting a detailed comparison of the overlay algorithms in our P2P simulation environment involves using a predefined set of input parameters that we will review in turn.

Network topology

In Section 4.2 we have reviewed in detail the reasons to use the BRITE network topology generator in our simulations. We have mentioned that we used the two level top-down hierarchical model where both the AS and the router level use the power-law adhering Barabasi-Albert generation model.

The largest network topology used in the P2P media streaming simulations, to the best of our knowledge, was a 10K network topology used in the OMNI

simulations. We thought that it would be beneficial to observe the overlay behavior on larger, more realistic network topologies. We have conducted simulations on two network topology sizes. Both network topologies used the same parameters that adhere to the model we found the most suitable, as defined in Section 4.2. We have conducted overlay simulations on 10K and 100K network topologies. Hereafter, when referring to the specific network topologies used in our simulations we will consistently use the terms 10K and 100K.

Bandwidth distribution

As far as we can tell, most of the previously conducted P2P overlay simulations used various simple peer bandwidth heterogeneity models. In our P2P media simulations we have used a peer bandwidth heterogeneity reference model based on research conducted by Saroiu et al. [38]. We will refer to this bandwidth reference model as the Saroiu bandwidth model. Using the Gnutella peer upstream bandwidth distribution observed by Saroiu et al. we picked one of the following five bandwidths (distributed uniformly):

- 50 kbps
- 400 kbps
- 800 kbps
- 3500 kbps
- 15000 kbps

Each generated peer node that joins an overlay is assigned an upstream bandwidth capability based on a bandwidth reference model used in that particular simulation. For simplicity, we have assumed a stream rate of 128 kbps for all simulations. The highest number of peers to which each peer can possibly stream data in a P2P overlay system is thus bounded by: upstream bandwidth capability / 128 kbps.

Peer join and duration distribution

We have modelled the peer join and duration distributions using Poisson and exponential distributions respectively. We have used the popular scientific library Colt [1] that supports numerous distributions. We have set the parameters of the Poisson and exponential distribution to achieve the desired peer overlay size during various simulations. For simulations involving a network topology of 10K physical nodes we have used the Poisson random distribution with an average rate of 30 peer node joins per simulation tick. For the 100K simulations the average join rate was 60 peer nodes per tick.

Recall that the probability distribution function of an exponential distribution is given by $\exp_\lambda(x) = \lambda e^{-\lambda x}$. We have set the exponential distribution's λ parameter to 0.01 for all simulations. The effect of the λ having the value of 0.01 is best understood by looking at the Figure 5.9. The probability that a peer node will stay in the overlay a certain number of time ticks is equal the area underneath the exponential probability distribution function in that specific time span. Thus, for example, a peer will more likely stay in the overlay in between 0-20 ticks rather than 80-100 ticks.

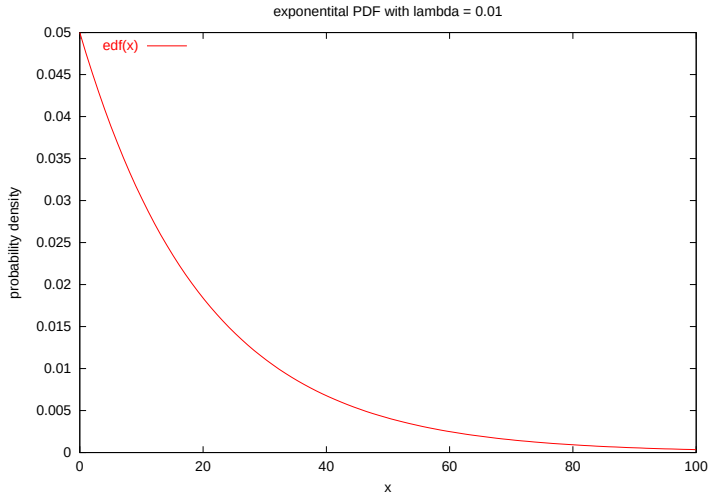

Figure 5.9: Exponential duration distribution

5.3.1 10K simulation results

We will first take a look at the results of the simulations with the following simulation parameters:

- a network topology of 10K physical nodes;
- the Saroiu bandwidth reference model;
- peer join and duration distributions as described in Section 5.3;

Node count

The node count metric is used to show the overlay size growth during the performed simulation. As depicted in Figure 5.10, OMNI and ZIGZAG are able to sustain the desired growth to around 3000 overlay nodes. The HMTP overlay loses nodes during overlay tree restructuring caused by node leave/crash events. Not all orphaned nodes are able to find a suitable parent within a specified delay threshold. The SpreadIt overlay seems to be incapable to grow beyond 800 overlay nodes. We have discovered that the SpreadIt overlay join algorithm is not able to efficiently find the unsaturated peer nodes which can accommodate arriving nodes. In this particular simulation setup, the SpreadIt overlay is increasingly unable to find unsaturated nodes after tick 32.

Stretch

Stretch, as defined in Section 2.2, gives us some insight in the deviation from the ideal unicast paths between the root of the overlay tree and the peer nodes. Recall that a unicast path between two nodes has a stretch of 1. Figure 5.11 shows average peer stretch values for each of the overlays. HMTP's average stretch is below 3 while other overlays' average stretch stays above 3 for most of the simulation. There are two reasons for such an impressive stretch in the HMTP overlay. HMTP has an exhaustive join algorithm that helps a joining node cleverly search for the closest parent node. However, the HMTP join algorithm, as we will later see, results in a somewhat higher control overhead as well. The second and less prevalent reason for the impressive HMTP stretch results is the HMTP periodical tree improvement algorithm. Even if we completely turn of

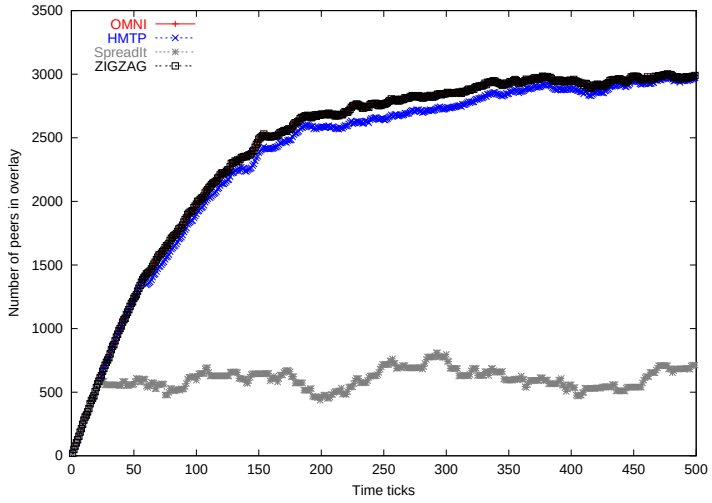

Figure 5.10: 10K node count

the HMTP periodical tree improvement algorithm, HMTP still achieves the best stretch results.

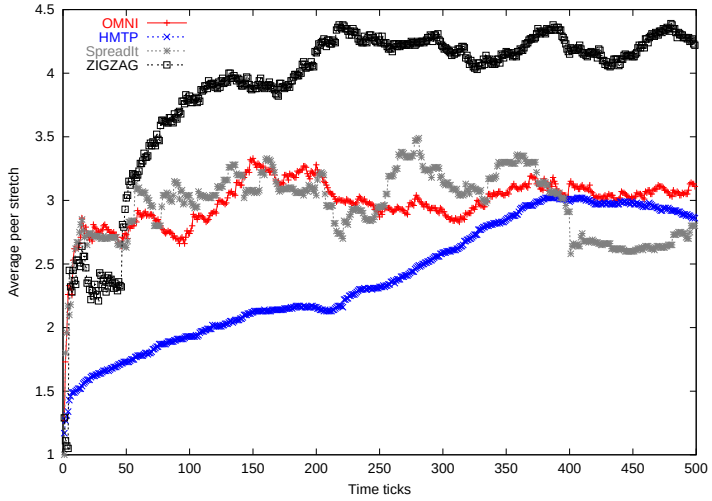

Figure 5.11: 10K stretch

Stress

Stress as defined in Section 2.1 measures the stress that the overlay exhorts on the underlying network. Low stress indicates a small number of duplicate packets being sent across the physical network connections. Figure 5.12 shows average physical network stress for each of the overlays. OMNI and ZIGZAG exhort more or less the same stress on the underlying physical network. Their stress values stabilize between 6.5 and 7. The SpreadIt overlay instigates somewhat lower stress than OMNI and ZIGZAG with stress oscillating between 4 and 5. However, HMTP has an impressively low stress, significantly better than all other overlays, with a steady stress value of 2.5 to 2.8.

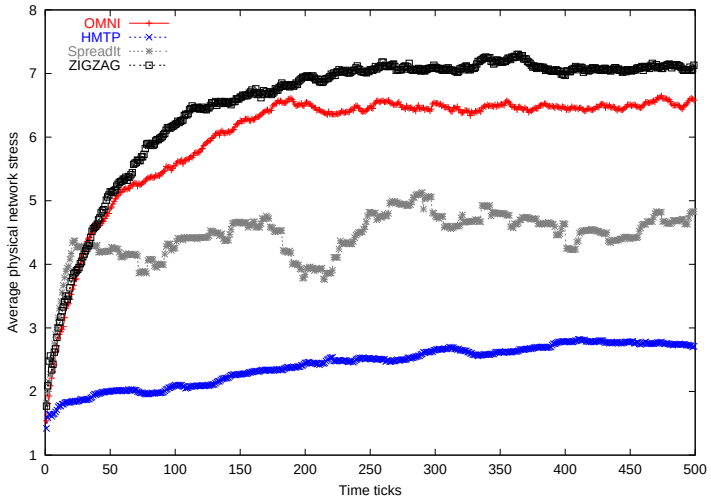

Figure 5.12: 10K stress

Control overhead

As previously mentioned in Section 2.3, we measure control overhead in terms of the number of peer contacts during peer join and leave. Figure 5.13 shows the average join control overhead for the SpreadIt, HMTP and ZIGZAG overlays. We have not measured join and leave control overhead for the OMNI overlay. Although OMNI has defined a join and leave algorithm for MSN nodes, the details of the regular client nodes' join and leave algorithm have been left unspecified.

Since Banerjee et al. [12] intentionally left the join and leave algorithm details ambiguous we thought that it would be inappropriate to make any claims about the OMNI overlay control overhead.

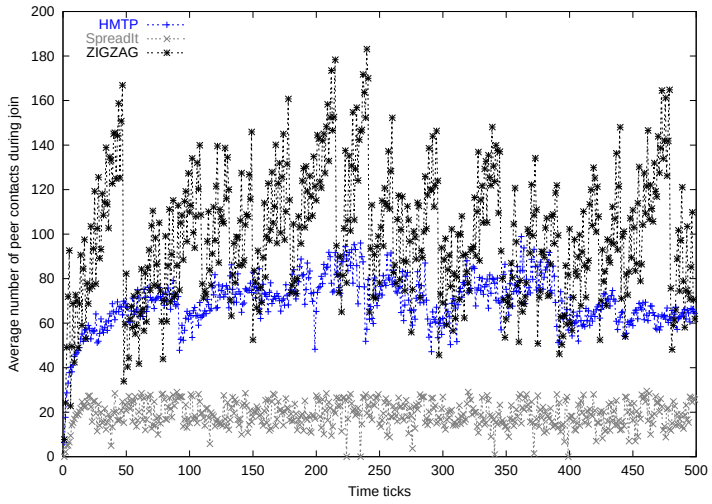

Figure 5.13: 10K join overhead

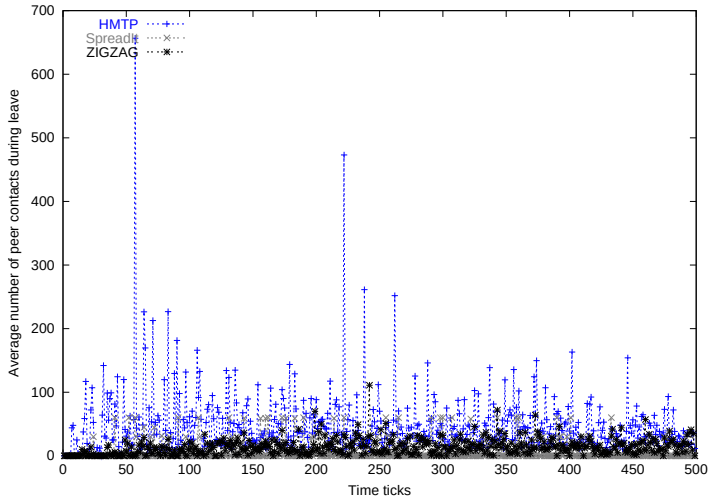

Figure 5.14: 10K leave overhead

The SpreadIt overlay has the lowest join control overhead, rarely surpassing

20 peer node contacts on average. However, given SpreadIt's inability to grow beyond a thousand nodes this fact does not come as a surprise. The ZIGZAG overlay exhibits the familiar, previously discussed "heartbeat-like" join control overhead while HMTP's join control overhead seems to be oscillating around 70 to 80 node contacts.

The HMTP overlay has the highest leave control overhead. The main reason for the high leave control overhead is the rejoin algorithm that allows each orphaned node to individually look for a new parent. In the case when a high upstream bandwidth capable node that serves many peers leaves or crashes, each orphaned node rejoins individually thus resulting in a high number of the rejoin contacts. Figure 5.14 shows the average number of peer contacts during leave events. The HMTP overlay has significant spikes in leave control overhead that reach above 200 peer contacts.

Robustness

Robustness, as defined in Section 2.4, concerns the fragility of overlay trees during tree restructuring. We have excluded ZIGZAG and OMNI from robustness measurements since these two overlays employ a coordinated rejoin algorithm. Coordinated rejoin algorithms, at least in theory, require only a bounded number of contacts to reorganize the overlay tree. For example, in the ZIGZAG rejoin algorithm, the cluster head redirects orphaned nodes to rejoin at a cluster mate of the leaving/crashed node thus requiring only one message for each orphaned node.

As you recall from Section 2.4 we defined and measured two robustness metrics - glitch ratio and shed ratio. The glitch ratio represents the percentage of nodes that were receiving the stream but unable to find a new stream source after restructuring in the overlay. Figure 5.15 shows the SpreadIt overlay having a considerable glitch ratio. The HMTP overlay has a somewhat smaller but still noticeable glitch ratio.

The second robustness metric, the shed ratio, represents the percentage of nodes that have had stream interruption for a certain number of ticks, ultimately resulting in a peer disconnecting from the overlay. We have used shed ratio to simulate the disgruntled users. Figure 5.16 shows the shed ratio for the simulated

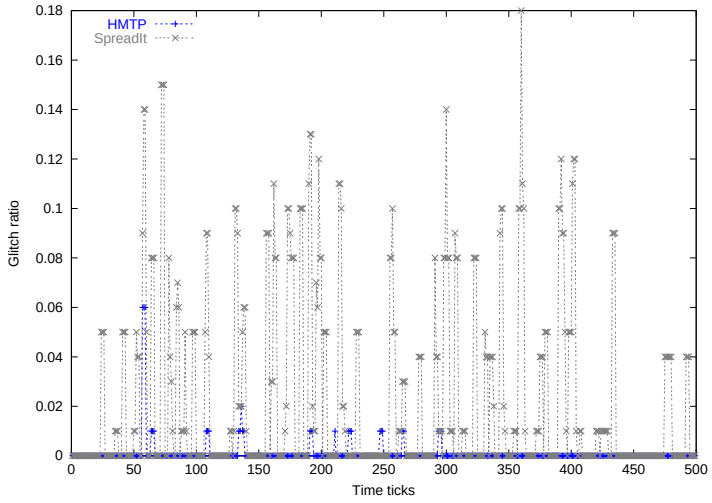

Figure 5.15: 10K glitch ratio

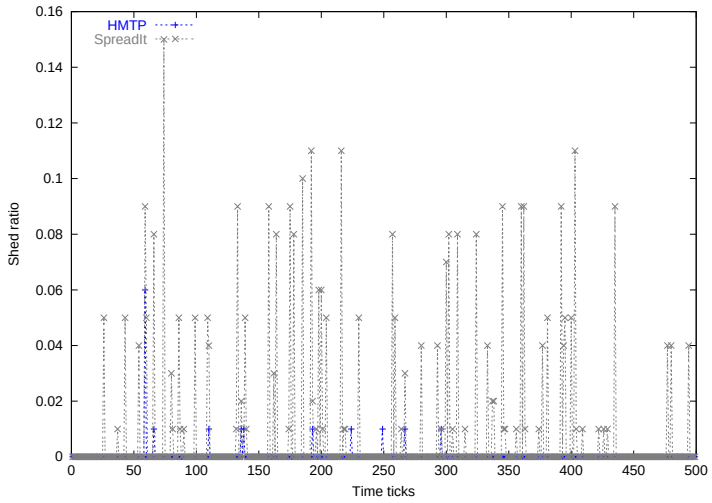

Figure 5.16: 10K shed ratio

overlays. Again, similar to the glitch ratio, the SpreadIt overlay experiences a significantly higher shed ratio compared to the HMTP overlay. Notice how spikes in HMTP's shed ratio coincide with drops in HMTP's node count from Figure 5.10.

5.3.2 100K simulation results

We will first take a look at the results of the simulations with the following simulation parameters:

- a network topology of 100K physical nodes;
- the Saroiu bandwidth reference model;
- peer join and duration distribution as described in Section 5.3.

Node count

As depicted in Figure 5.17, OMNI and ZIGZAG are able to sustain the desired growth to around 6000 overlay nodes. Just as in the 10K simulation, HMTP loses nodes during overlay tree restructuring caused by node leave/crash events. Similarly to the 10K node count simulation results, the SpreadIt overlay does not grow above a thousand overlay nodes.

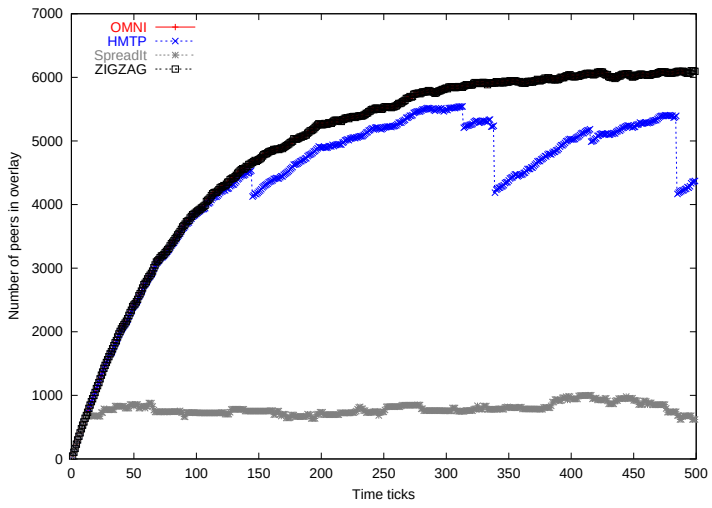

Figure 5.17: 100K node count

Stretch

Figure 5.18 shows average peer stretch values for each of the overlays. Along with the under-performing SpreadIt overlay, the OMNI overlay has the best stretch results. OMNI's average stretch hovers just above 2. ZIGZAG has significantly better stretch results in the 100K than in the 10K simulation. While at certain points of the 10K simulation ZIGZAG's stretch reaches 4.5, the stretch in the 100K simulation is steadily below 2.8. An interesting twist is that HMTP's stretch results are not lower in the 100K simulation as are the stretch results in the other three overlays.

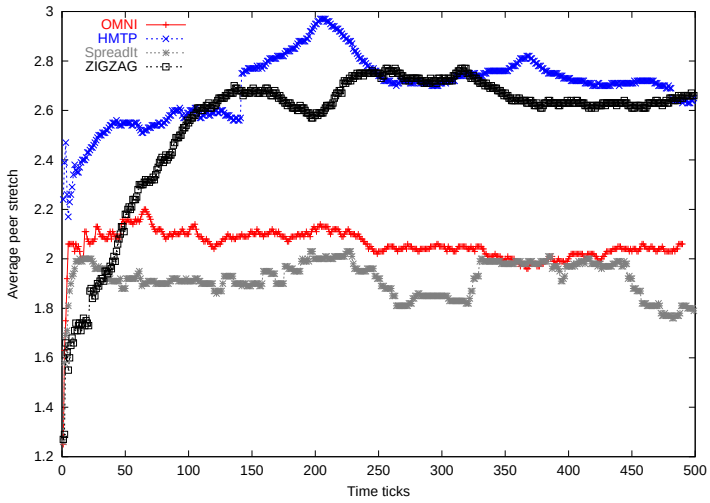

Figure 5.18: 100K stretch

Stress

We have found that, as depicted in Figure 5.19, HMTP has superior stress results in the 100K simulation as well. ZIGZAG's stress is very similar to the stress in the 10K stress results. The OMNI overlay performed slightly better than the ZIGZAG overlay. In fact we can see essentially the same trend in the stretch results for ZIGZAG and OMNI overlays in both the 10K and the 100K simulations. SpreadIt exhorts more or less the same stress on the underlying physical network in both simulation setups as well.

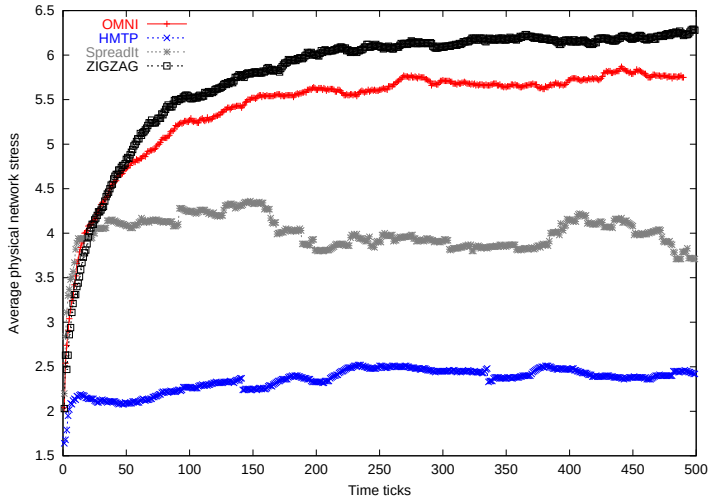

Figure 5.19: 100K stress

Control overhead

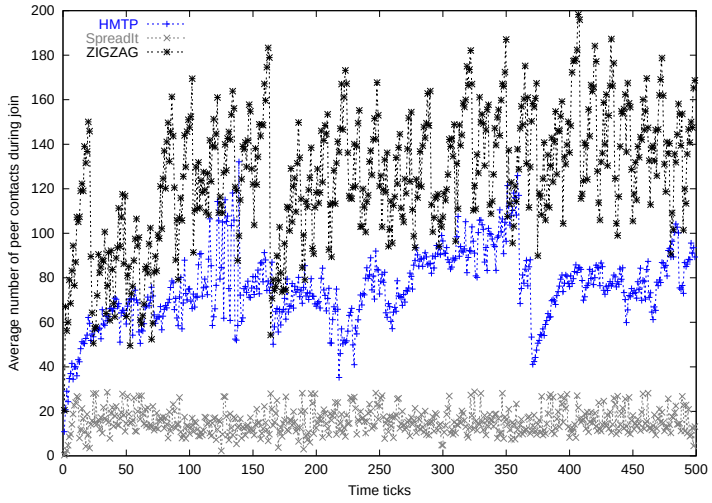

Figure 5.20: 100K join overhead

Figure 5.20 shows the average join control overhead for the SpreadIt, HMTP and ZIGZAG overlays. Just as HMTP's join control overhead was slightly below

60 peer contacts in the 10K simulation the same pattern is repeated in the 100K simulation except that the join control overhead is slightly below 80 peer contacts. The SpreadIt overlay has roughly the same join control overhead as in the 10K simulation, rarely surpassing 20 peer node contacts on average. The ZIGZAG overlay exhibits the familiar, previously discussed "heartbeat-like" join control overhead with a slightly higher average number of contacts.

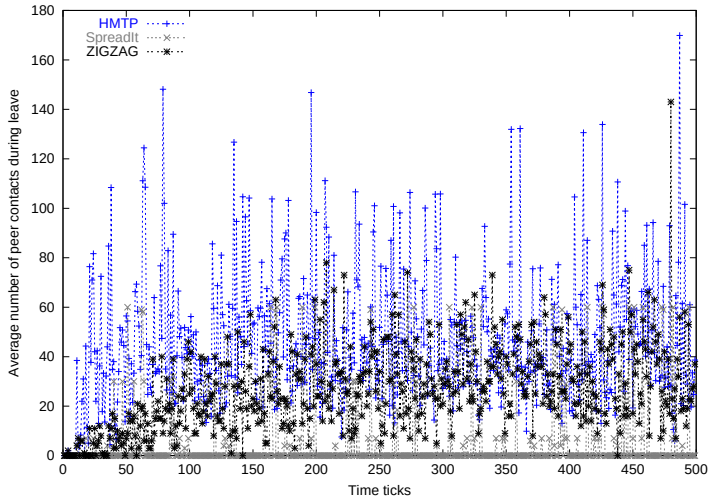

Figure 5.21: 100K leave overhead

Similarly to the 10K simulation the HMTP overlay has the highest leave control overhead in the 100K simulation as well. HMTP's leave control overhead is higher than in the 10K simulation. Figure 5.21 shows the average number of peer contacts during leave events. The SpreadIt and ZIGZAG overlays have a lower leave control overhead.

Robustness

Figure 5.22 shows the SpreadIt overlay again having a considerable glitch ratio. However, while SpreadIt exhibits similar glitch ratio tendencies from the 10K simulation, HMTP does not. HMTP's has significant spikes in glitch ratio not exhibited in the 10K simulation setup.

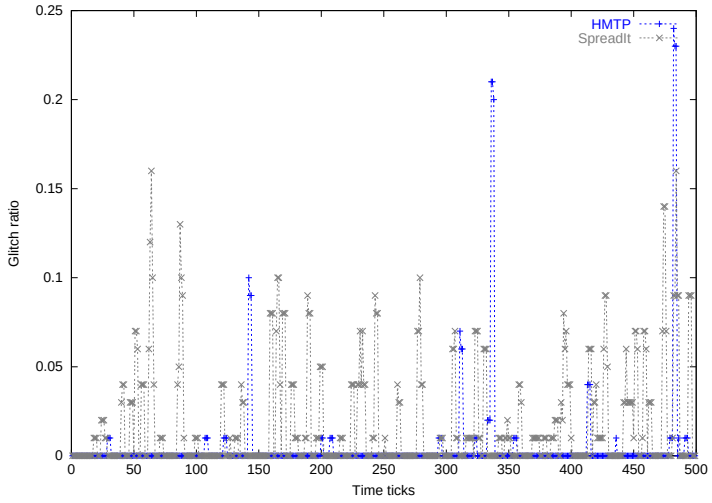

Figure 5.22: 100K glitch ratio

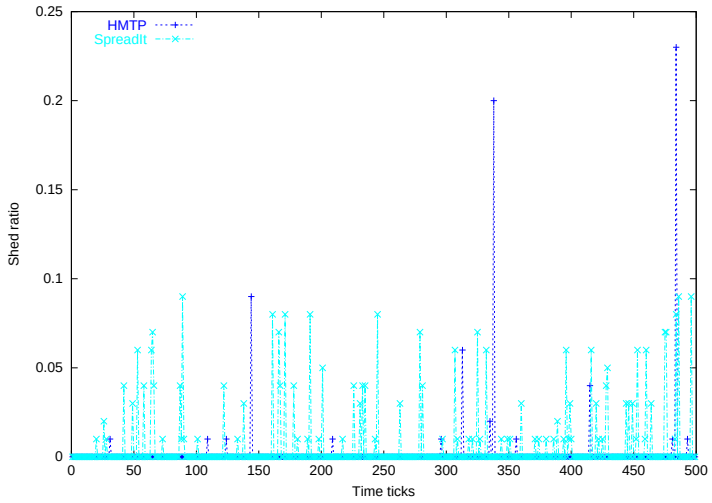

Figure 5.23: 100K shed ratio

Figure 5.23 shows the shed ratio for the SpreadIt and HMTP overlays. Similarly to the 100K glitch ratio results, the shed ratio is evidently problematic in the HMTP overlay. Notice how big spikes in glitch ratio at ticks 145, 335 and 490 perfectly coincide with HMTP node count drops observed in Figure 5.17.

5.3.3 Other simulations

Besides the previously described 10K and 100K simulations we have conducted numerous other simulations. More specifically, we were intrigued by the HS and LS BRITE topology parameters and their effect on simulation results. The HS and LS parameters are used to specify the planes that host the physical network topology generated by BRITE. HS represents the size of the high-level plane while LS represents the size of the low-level plane. When generating a topology BRITE assigns nodes to a plane divided into HS $\times$ HS squares. Each HS square of the plane is in turn subdivided into smaller LS $\times$ LS squares. Finally, each node is assigned one LS square. Each LS square can only hold one node.

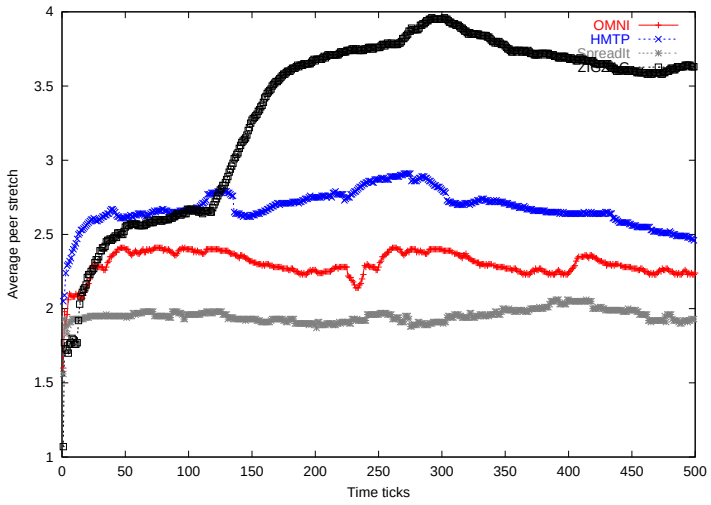

Figure 5.24: Stretch

Varying the HS and LS parameters did not have a major impact on the simulation results except in terms of stretch. However, the stretch result impact is easily interpreted. If the HS and LS parameters are set such that generated nodes are compacted closely together, the stretch results might not be as obvious as in a case where the nodes are spread apart from one another. We have used the same HS and LS parameters for our 10K and 100K simulations. We have set HS to 1000 and LS to 100. Figure 5.11, depicting stretch results for 10K, shows larger stretch results variations between the four algorithms than in 100K simulation stretch results depicted in Figure 5.18. We repeated the 100K simulation while

setting HS to 10000 and LS to 1000. As anticipated, the stretch results, shown in Figure 5.24, are easier to interpret.

We were also interested to see how large a topology network and how large an overlay we can create. We were able to construct a half-a-million network topology using a two-level, top-down hierarchical model where both the AS and the router level utilize the power-law adhering Barabasi-Albert generation model. We have only tried this simulation with the ZIGZAG overlay. We used the Poisson random distribution with an average rate of 125 peer joins per simulation tick for the join distribution. We used the exponential distribution with λ having the value of 0.01 to model the duration distribution. Just as we did for all other ZIGZAG simulations we used a single bandwidth type having a large uplink outdegree. This combination of input parameters resulted in a ZIGZAG overlay reaching over 13,000 nodes. The results of this simulation are reviewed in Section 5.3.7.

Having provided a detailed summary of the simulations we have conducted along with the accompanying results, we can now delve into the interpretation of these simulation results. We summarize the complete simulation results for each overlay algorithm individually and identify trends and trade offs between the various metrics. Furthermore, we try to pinpoint causes for these results, trends and trade offs.

5.3.4 SpreadIt

Of the four tree-first overlay algorithms we have reviewed, the SpreadIt overlay algorithm is the simplest in terms of its overlay construction and is the easiest to implement. However, the simplicity comes with an attached cost. The SpreadIt overlay achieves acceptable scores in all metrics except for the robustness metrics of the glitch and shed ratio. In the following discussion, we provide a brief overview of SpreadIt's performance for the metrics we have recorded.

SpreadIt stretch and stress, as depicted in Figure 5.25 and 5.26, display little distinction between the 10K and 100K simulation results. Both stretch and stress results are modest compared to the other three overlay algorithms we have reviewed.

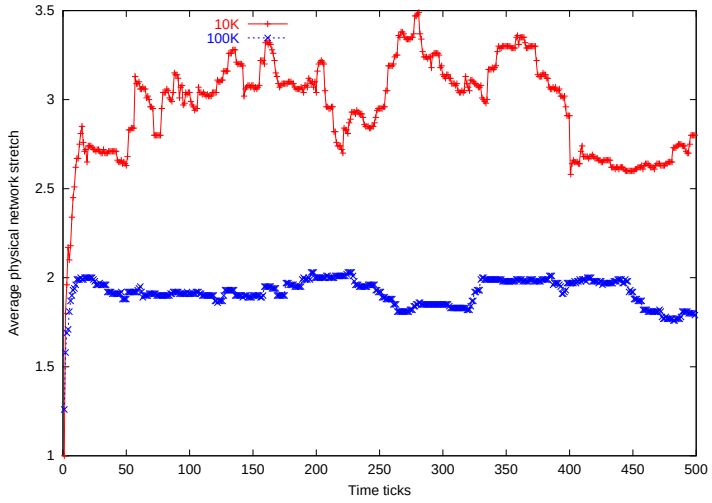

Figure 5.25: SpreadIt stretch comparison

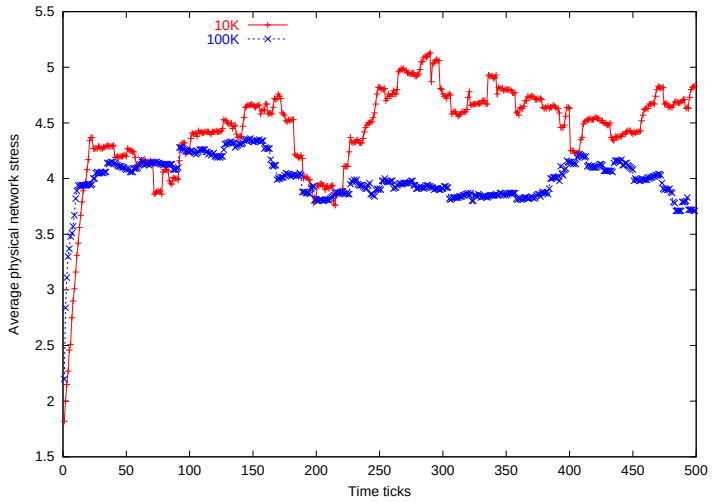

Figure 5.26: SpreadIt stress comparison

As shown in Figure 5.27 and 5.28 SpreadIt join and leave control overhead is fairly consistent for both 10K and 100K simulations. Join and leave control overhead is admirable compared to that of HMTP, OMNI, and ZIGZAG overlay algorithms.

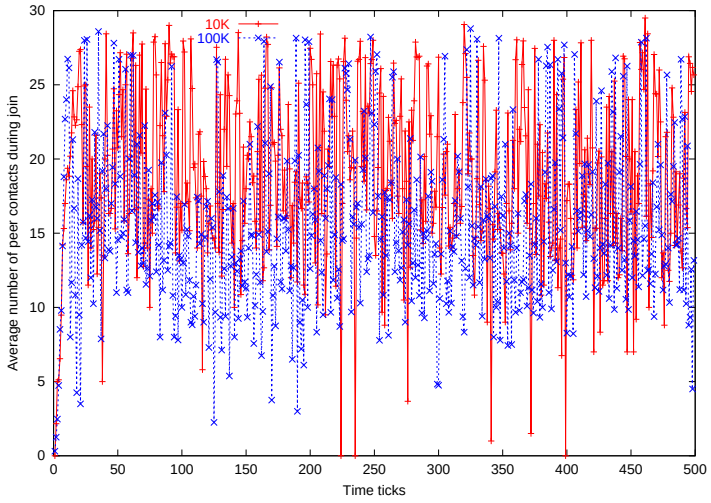

Figure 5.27: Join control overhead comparison

However, SpreadIt's excellent join and leave control overhead results do not come as a surprise. Namely, since the SpreadIt overlay was unable to grow beyond a thousand nodes in both 10K and 100K simulation, join and leave control overhead was never an issue to begin with.

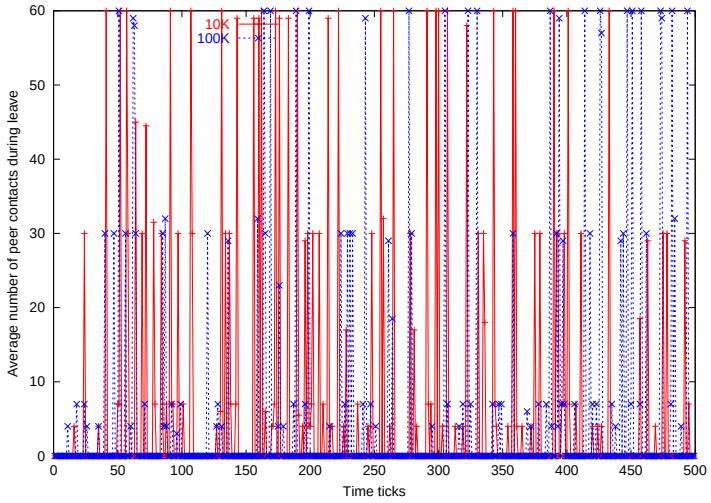

Figure 5.28: Leave control overhead comparison

The robustness metrics of the SpreadIt overlay exhibit problematic tendencies. As shown in Figure 5.29 and 5.30 the SpreadIt glitch and shed ratio are unacceptable. In both 10K and 100K simulations, roughly 10% of the SpreadIt nodes in the overlay were experiencing glitches while roughly 5% of the SpreadIt nodes were shed at each simulation tick.

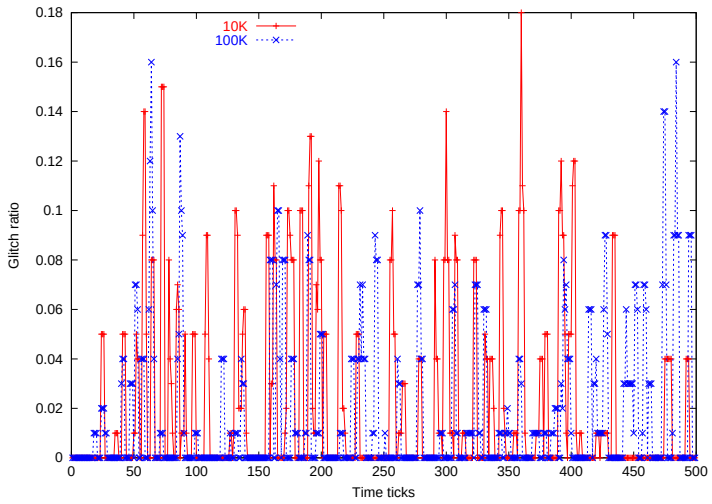

Figure 5.29: SpreadIt glitch ratio comparison

One of the biggest advantages of SpreadIt is the simplicity of join and leave algorithms. It would not be hard to implement a full-scale SpreadIt overlay, just as Deshpande, Bawa and Garcia-Molina actually did. However, as we have seen in our simulations results, such a SpreadIt overlay implementation seems only applicable to small-scale media streaming.

We suspect that SpreadIt overlays can be easily enhanced to address the above-mentioned issue. One may be able to improve the original SpreadIt join algorithm by including additional state tracking for each node. For example, HMTP employs a stack of previously discovered potential parents to allow each node more possibilities in a search for a new parent. Similar information may be exploited in SpreadIt to improve scalability.

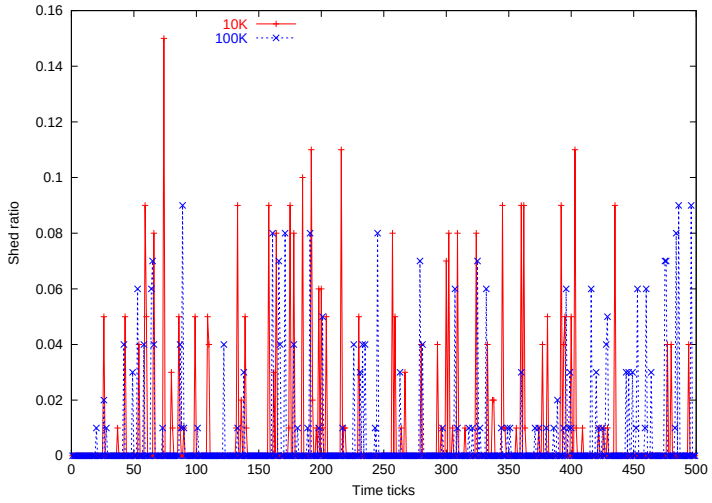

Figure 5.30: SpreadIt shed ratio comparison

5.3.5 HMTP

The HMTP overlay algorithm is an elegant and simple tree-first overlay algorithm that manages to accomplish respectable results according to the metrics that we employed. HMTP's stress and stretch results are among the best we have recorded.

As depicted in Figure 5.31 HMTP's stretch results exhibit no specific correlation between the 10K and 100K simulation setups. However, only the OMNI overlay in its 100K stretch results exhibits better stretch than HMTP.

HMTP is a stellar performer when it comes to stress. In both simulation scenarios we observed similar stress results that rarely reached values higher than 2.5. To put these results in perspective, ZIGZAG and OMNI have stress values that vary between 5 and 7 in both simulation setups.

HMTP exhibits seemingly good control overhead characteristics as well. Join control overhead scales sub-linearly with the number of peer nodes in the overlay. The average number of peer join contacts for the 10K simulation was 65, and for the 100K simulation it was 75. In addition, HMTP's leave control overhead scales sub-linearly with the number of nodes in the overlay. The average number

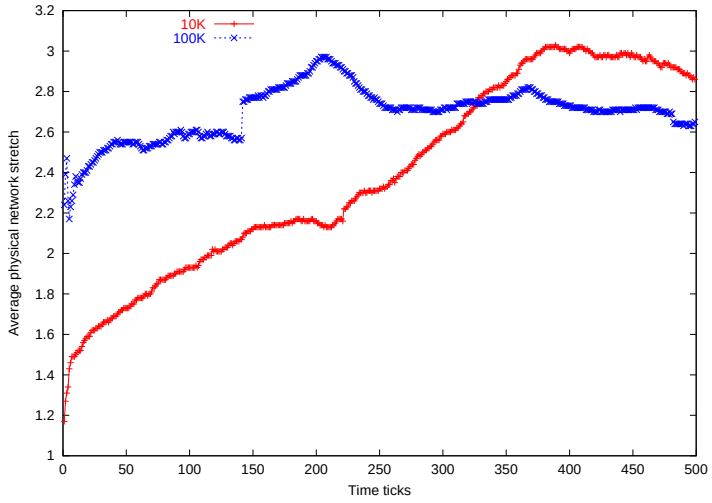

Figure 5.31: HMTP stretch comparison

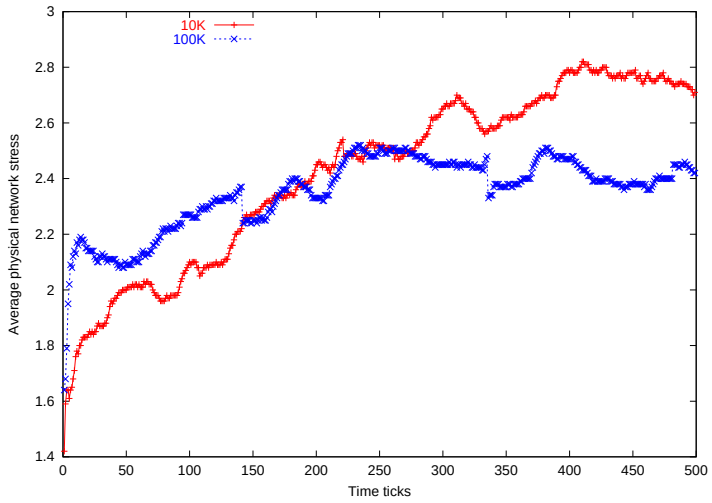

Figure 5.32: HMTP stress comparison

of peer leave contacts for the 10K simulation was 46 and for 100K it was 48. There is, however, an interesting tendency for certain peer crashes or leaves to cause very high jumps in control overhead. Figures 5.14 and 5.21 illustrate this finding.

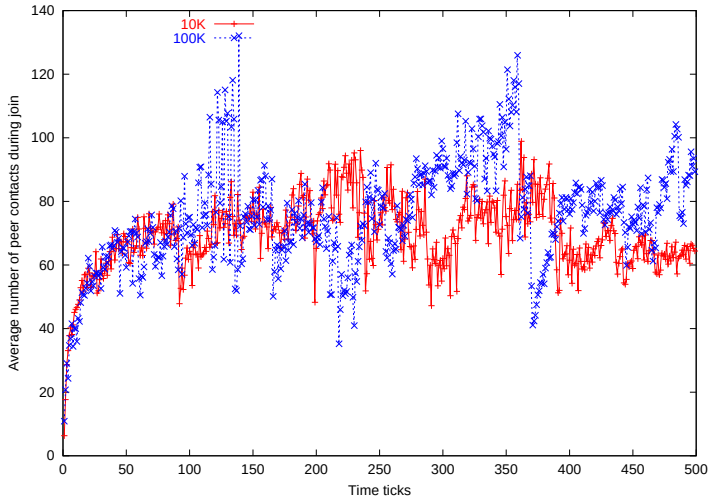

Figure 5.33: HMTP join control overhead comparison

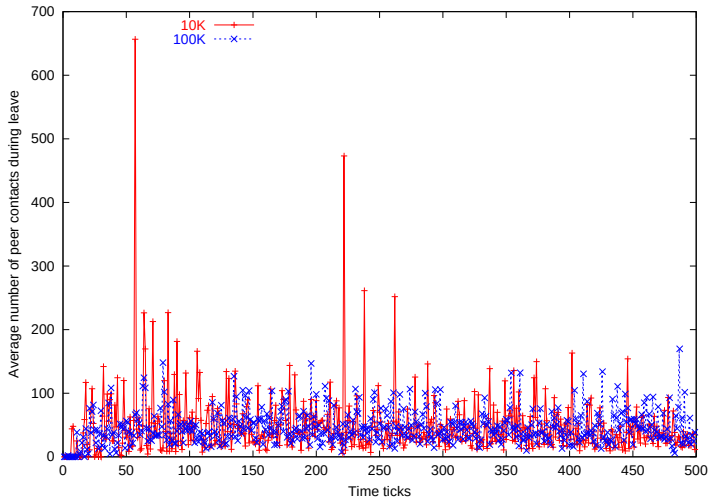

Figure 5.34: HMTP leave control overhead comparison

HMTP's robustness performance is affected by random jumps in glitch ratio and shed ratio. There are no obvious patterns to shed and glitch ratio in each individual simulation, nor between the different simulation setups.

Undoubtedly, one of the best aspects of the HMTP overlay is its original

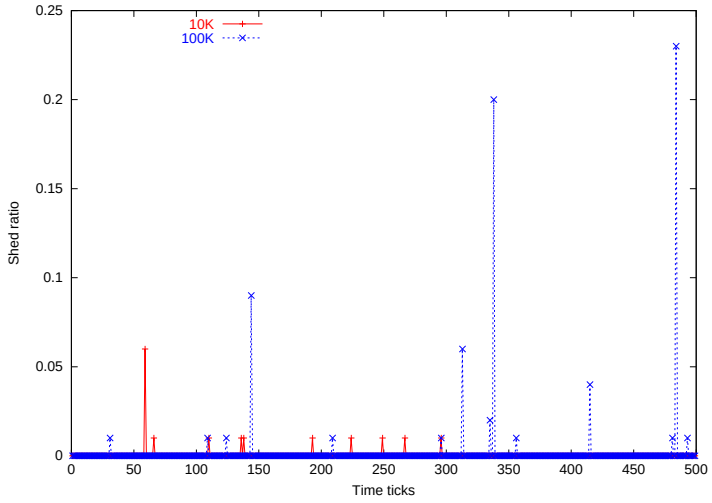

Figure 5.35: HMTP shed ratio comparison

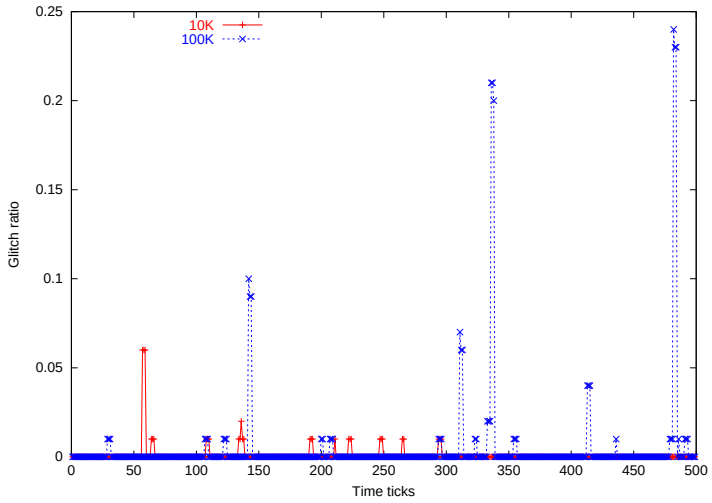

Figure 5.36: HMTP glitch ratio comparison

and simple join algorithm. We have shown that HMTP's join algorithm is a primary factor that leads to impressive stress and stretch results. Rejoin and tree improvement algorithms are equally appealing as well. In addition, such admirable stress and stretch results are not achieved at the expense of join and leave control overhead. Join and leave control overhead are reasonably good and

scale sub-linearly with the number of nodes in the overlay.

One of HMTP's weaker aspects is the peculiar spikes in leave control overhead. Although the average number of peer leave contacts grows sub-linearly with the number of nodes, there is a tendency for high and relatively frequent jumps in leave control overhead. We suspect that there are two major reasons that may be causing high oscillations in leave control overhead. Firstly, since the leaf to non-leaf ratio is 2:1, there is a high number of parent nodes that by crashing or leaving the overlay can potentially cause spikes in leave control overhead. Secondly, HMTP's rejoin algorithm allows each orphaned node to individually search for a new parent. When a high upstream bandwidth capable node that serves many peers leaves or crashes, each orphaned node runs the rejoin algorithm individually, thus, ultimately resulting in high spikes in leave overhead. Recall that the number of contacts by the orphaned nodes is included in the leave control overhead of the node that left or crashed.

5.3.6 OMNI

As the objective of the OMNI overlay is to minimize the average tree latency, one would expect that OMNI should perform above average when it comes to stretch. Indeed, our assumptions were confirmed by our simulations. OMNI has a relatively good stretch performance in both simulation setups. The stretch results have a tendency to be relatively stable during the duration of simulations. In the 10K simulation only HMTP has a better stretch, while in the 100K OMNI has the best results.

In the results overview of HMTP we have seen that the stress and stretch results have a tendency to correlate. As will be discussed subsequently, ZIGZAG exhibits the same stress and stretch correlation. Poor results in stress tend to be mirrored by poor results in stretch while good results in stress tend to be mirrored by good stretch results. Interestingly enough, however, OMNI, which has better than average stretch results, exhibits as bad stress as ZIGZAG. In both simulation scenarios we can observe the progression of similar stress results. While ZIGZAG exhibits the worst stress among all overlays, OMNI's stress is approximately only 10% better across both simulation setups.

As previously indicated, we have not measured join and leave control overhead

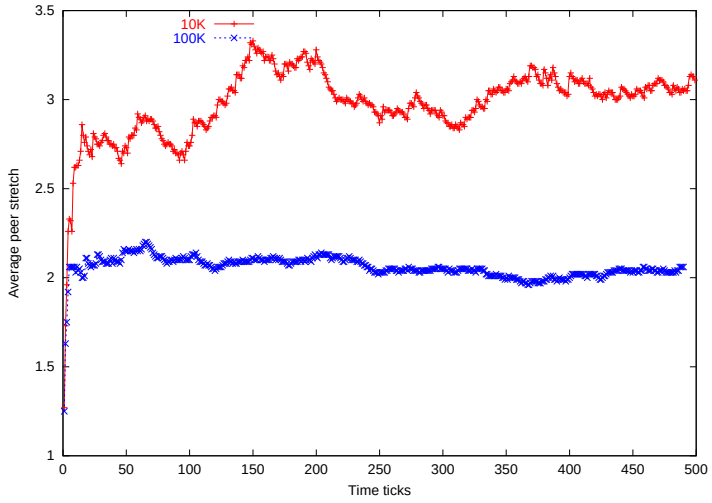

Figure 5.37: OMNI stretch comparison

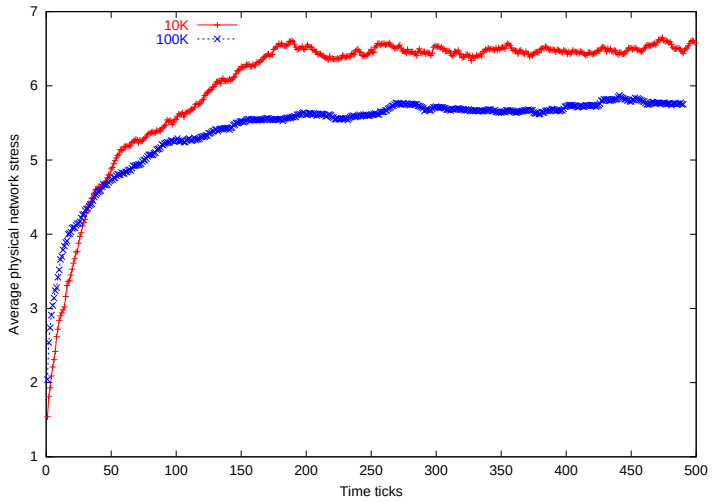

Figure 5.38: OMNI stress comparison

for the OMNI overlay since Banerjee et al. [12] intentionally left the join and leave algorithm details unspecified. Robustness measurements were not taken either for the same reason.

One of OMNI's better aspects is, as its design requirements specify, keeping

the aggregate tree latency as low as possible. We were able to reproduce and confirm the simulation results conducted by Banerjee et al. Although we have not taken any control overhead measurements for the OMNI overlay, we suspect that OMNI would have join control overhead very similar to HMTP. Both HMTP and OMNI use a variation of a depth-first search algorithm when joining a new node. OMNI's MSN join algorithm is rather convoluted but undoubtedly contains aspects of depth-first search. However, OMNI's leave algorithm is conceptually different from HMTP's. While HMTP allows each immediate orphaned node to search for a new parent individually, OMNI employs an coordinated rejoin algorithm. Although very elegant and simple, the individual rejoin algorithm employed by HMTP exhibits some troubling aspects in our simulations. We believe that OMNI's coordinated rejoin algorithm would perform better in leave control overhead.

5.3.7 ZIGZAG

The ZIGZAG overlay algorithm is the most complex overlay algorithm out of the four tree-first overlay algorithms that we have reviewed. Besides the poor stress and stretch results, ZIGZAG excels in every other metric that we have recorded. In the following paragraphs we will try to pinpoint the causes for those results.

ZIGZAG's stretch results manifest similar trends between the 10K, 100K and 500K simulation setups. ZIGZAG's stretch performance has a tendency to be the worst out of the four overlays.

In all three simulation scenarios we can observe similar stress results tendencies. ZIGZAG exhibits the worst stress among all overlays in all simulation setups. ZIGZAG's performance is the worst in the 10K simulation while performing only slightly better in 100K and 500K simulations.

ZIGZAG exhibits excellent control overhead characteristics. Join overhead scales sub-linearly with the number of peer nodes in the overlay and the network size. The average number of peer join contacts for the 10K simulation was 92; for the 100K simulation it was 125; finally, for the 500K the average number was 126.

If we count only non-leaf node crashes or leaves the average number of peer

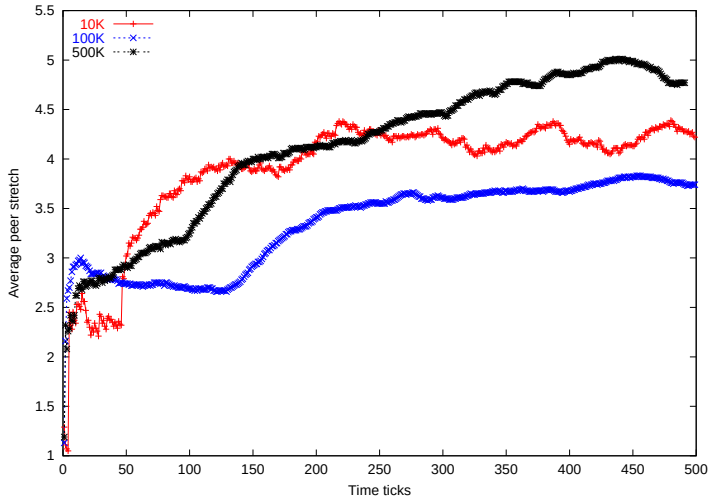

Figure 5.39: ZIGZAG stretch comparison

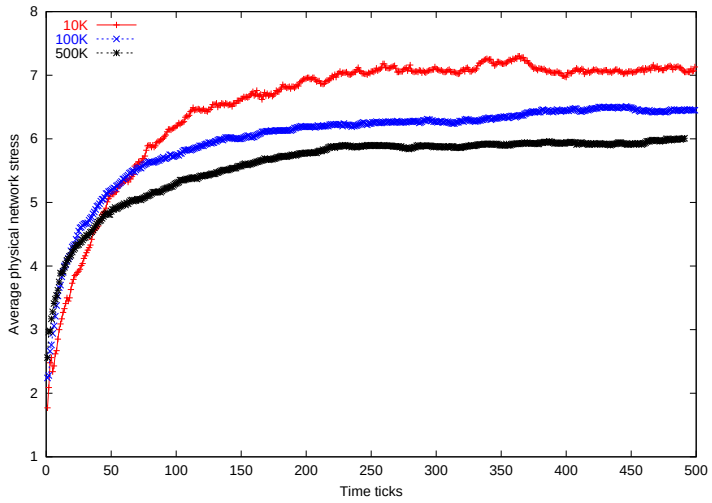

Figure 5.40: ZIGZAG stress comparison

leave contacts for the 10K simulation was 15, 28 for the 100K, and 51 for the 500K. However, when leaf node crashes or leaves are included then ZIGZAG leave control overhead scales sub-linearly with the number of nodes in the overlay.

Recall that we did not take any robustness measurements for ZIGZAG since

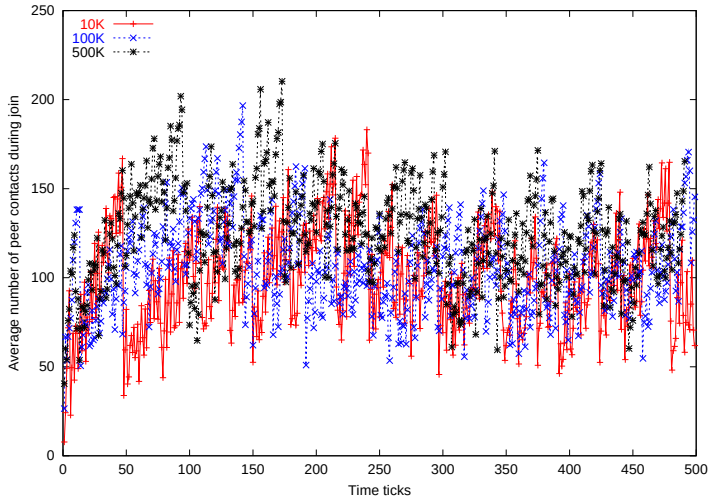

Figure 5.41: ZIGZAG join control overhead comparison

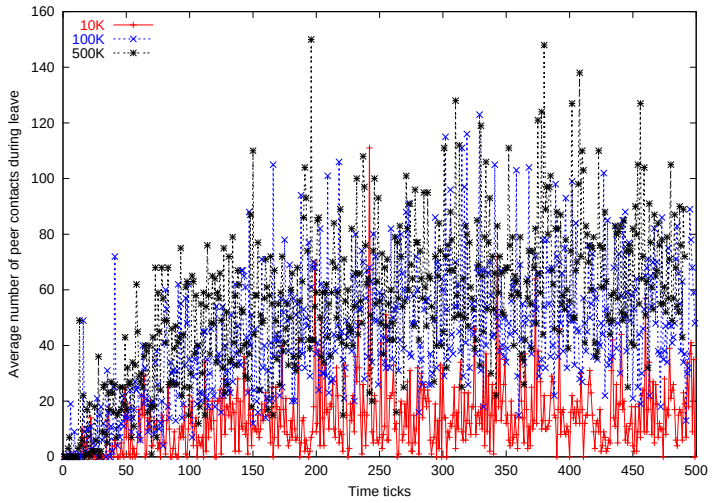

Figure 5.42: ZIGZAG leave control overhead comparison

we assigned essentially an unlimited uplink bandwidth to all ZIGZAG overlay nodes.

Overall, ZIGZAG is one of the best if not *the* best tree-first overlay algorithm among the ones that we have reviewed. Through a series of empirical simulations

described previously in Section 5.2.3 we have verified the claims of Tran et al. [41] about ZIGZAG's performance. ZIGZAG's logical clustering organization leads to excellent control overhead tendencies. Join overhead is growing sub-linearly with the number of nodes, which is crucial for overlay scaling to thousands of peer nodes. We have also verified that ZIGZAG indisputably creates very low-depth overlay trees but not at the expense of significantly overloading certain peer nodes. We have observed roughly the same average outdegree as well as the maximum peer outdegree for all three simulation setups. The maximum node outdegree does not grow beyond a hundred nodes and the average outdegree has constantly lingered between 8 and 9.

One of ZIGZAG's weaker points is its stress and stretch performance. In fact, we suspect that the reasons for poor stretch and stress results are intertwined. We believe that there are two major factors that contribute to the poor stretch results.

The first factor is ZIGZAG's ratio of leaf to non-leaf nodes. An interesting aspect of the ZIGZAG overlay is that the majority of nodes are leaves. In fact, the ratio of leaf nodes to non-leaf nodes varies between 8:1 and 10:1 while this ratio for HMTP was 2:1. Recall that most of the leaves are guaranteed to be at the same depth and that the height of the tree is bounded by $O(\log_k N)$ where N is the number of nodes in an overlay and k is a parameter that was set to 5 in our simulations. In all three simulations the leaves were at a depth of between 4 and 5. Given that the vast majority of the nodes have a depth of $O(\log_k N)$, it is not hard to conjecture that ZIGZAG's stretch results might be less than impressive.

The second factor that contributes to the poor stretch results is the poor stress results, which are in turn caused by the join algorithm. In order to have good stress results, an overlay tree should closely match the underlying physical network. This, in turn, requires that a newly joined node be solely guided to its new parent by the network distance metric. The HMTP join algorithm, for example, does this guidance very well. Recall that the ZIGZAG join algorithm joins new nodes strictly at the bottommost layer clusters. The vast majority of the nodes that serve no other nodes are in the bottommost layer clusters. As the join algorithm pushes a joining node towards the leaves it takes the network delay into account. However, the join algorithm has only $O(\log_k N)$ levels to refine that network delay. Recollect that the ZIGZAG overlay tree is a relatively balanced tree as well. ZIGZAG's overlay does not closely match the underlying physical

network and therefore ZIGZAG's stress performance cannot be impressive. Those suspicions were confirmed in the performed simulations.

Chapter 6

Conclusion

In recent years we have witnessed the emergence of numerous P2P overlay algorithms. While many of these overlay algorithms focus on the sharing and searching of distributed computer resources, there is a subset of overlay algorithms that attempt to solve the problem of large-scale Internet multimedia streaming. Before the emergence of large-scale Internet multimedia streaming algorithms there was a void left by IP Multicast and Content Delivery Networks (CDNs) that needed to be filled. The current buoyancy of Internet streaming overlay algorithms has been a result of unsuccessful IP Multicast deployment and the high cost associated with CDNs. The renewed interest in more scalable, distributed and decentralized overlay networks has contributed to the emergence of Internet streaming overlay algorithms as well.

6.1 Overview

However, as new P2P overlay algorithms emerge, the trend wherein each research team uses their own makeshift simulators continues and thus, we believe, jeopardizes the progress of this interesting research field. As far as we know, we are the first to provide a generic and unified tree-first overlay simulator for multimedia streaming. Until now, all tree-first overlays have utilized individual, ad-hoc simulators which do not provide the opportunity to compare multiple overlay algorithms in the same simulation environment. Through the implementation of four leading tree-first overlay algorithms, we have demonstrated that our simulator is easily adaptable to various tree-first overlay algorithms. Furthermore,

our simulator provides a set of API extensions which allows new overlay algorithms to be added seamlessly. Our simulator uses network topologies generated by the power-law adhering BRITE topology generator. The BRITE topology generator has recently emerged as one of the more promising universal topology generators. We have also shown that our simulator is easily customizable. Our simulator can accept one of a multitude of predefined join, duration and peer bandwidth distributions. In addition, new join, duration and peer bandwidth distributions can easily be contributed. Recall that simulator parameters include a set of overlay metrics as well. We have provided a rich set of metrics that are readily available to others. Additional metrics can be defined and transparently added to our simulator. Just as we have allowed customization of any simulation input parameter, multiple data output formats can be used simultaneously. We have provided XML, Excel, and Gnuplot simulation output formats. Additional output formats can be defined and added effortlessly. All our simulation graphs were generated using the Gnuplot output format. By using our simulator, research teams can readily compare the algorithm that they are developing with a set of already developed algorithms. Thus an overlay research team is able to cost-effectively receive immediate feedback about the algorithm being developed. Finally, we have shown the ability to large-scale simulations in our simulator using a BRITE generated topology network of more than half a million nodes and an overlay with tens of thousands of peer nodes.

The generic simulator allowed us to simulate four leading tree-first overlay algorithms with the exact same input parameters in the same simulation environment. After verifying that our implementations of the SpreadIt, HMTP, OMNI, and ZIGZAG overlay algorithms are reasonable representations of their original counterparts, we administered a detailed comparison of these overlay algorithms in our simulator. We believe that we were the first to have an opportunity to gain a unique insight by performing those simulations. We were able to spot non-apparent trade-offs between various metrics as well as gain extraordinary insight in certain aspects of all four overlays. We describe these findings at length in Section 5.3.4, 5.3.5, 5.3.6, and 5.3.7. We also believe that we were the first to formally introduce two new robustness metrics: glitch ratio and shed ratio.

6.2 Future directions

A generic overlay simulator presents an essential component in a P2P overlay researcher's toolbox. The simulator should play a key role when comparing an algorithm in development against a set of existing overlay algorithms. Having theoretical aspects of the overlay confirmed empirically is the next best thing to developing the full-blown Internet version of the overlay. We believe that besides focusing on theoretical aspects of the overlay algorithms, special attention should be given to empirical aspects as well. A simulator provides immediate empirical feedback about an overlay's prospects. A generic simulator allows P2P overlay researchers to combine and experiment with multiple combinations of various overlay characteristics at a much lower cost than before.

A simulator is also an important tool in identifying tradeoffs between various overlay metrics. As we have seen in Section 5.3.4, 5.3.5, 5.3.6, and 5.3.7, we were able to provide insights into some of the metric tradeoffs for the four overlays with which we have experimented. We believe that we were the first to provide such an insight as a result of our simulator. However, it is evident that further studies into tradeoffs between various overlay metrics is fundamental.

As new overlay algorithms for multimedia streaming are being developed we believe it would be invaluable to compare them to the four algorithms already implemented in our simulator. We believe we have made a compelling case for overlay research teams to use our simulator. We have made our simulator publicly available [6] and encourage research teams to use it. Although we encourage others to implement new algorithms in our simulator, we have plans of our own. For example, we would like to implement a recently released ZIGZAG algorithm variation [42] and compare it to the original ZIGZAG algorithm.

We are continuing to explore and experiment with additional overlay metrics. We would like to understand tradeoffs between various metrics better. We believe it would invaluable to identify and formally present tradeoffs between the defined overlay metrics. Even trivial additional metrics can sometimes be very useful to empirically verify assumptions observed in simulations. For example, we have assumed that ZIGZAG's overlay has a higher leaf to non-leaf ratio than the other three overlay algorithms. After implementing the leaf to non-leaf metric we have found that leaf to non-leaf ratio differences between ZIGZAG and other

algorithms are rather significant.

Long running simulations are inevitable for large-scale overlays. Most of the simulations that we ran lasted not longer than a day. However, some lengthier simulations ran for a week. On several occasions we had to restart simulations that were interrupted due to power failures. We believe that a backup and restore of individual simulation runs is a crucial functionality for such circumstances. The simulator could backup the simulation state in a specified frequency of safe-points. Individual simulations could be restarted from any of the backup safe-points, thus enabling long running large-scale simulations.

We will continue to experiment with variations and modifications of the simulation setups of the four overlay algorithms. More specifically, we would like to see the effect of varying the k parameter in ZIGZAG overlays as well as varying the temperature parameter and probability of performing random-swap operations for OMNI overlays. We are also interested in exploring how simple modifications to the SpreadIt overlay would affect its performance.

BIBLIOGRAPHY

[1] Colt. http://hoschek.home.cern.ch/hoschek/colt/.

[2] Gnutella. http://gnutella.wego.com.

[3] Javasim network simulator. http://www.javasim.org.

[4] myns simulator. http://www.cs.umd.edu/~suman.

[5] ns network simulator. http://www-nrg.ee.lbl.gov/ns/.

[6] p2pns. http://p2pns.sourceforge.net/.

[7] p2psim. http://www.pdos.lcs.mit.edu/p2psim/.

[8] Ssf network simulator. http://www.ssfnet.org.

[9] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the 32nd Annual Symposium on Theory of Computing*, pages 171–180, Portland, May 2000. ACM.

[10] S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols, 2002.

[11] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, pages 205–220, Pittsburgh, PA, August 2002. ACM.

[12] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, and Samir Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of IEEE INFOCOM 2003*, pages 1521–1531, San Francisco, April 2003. IEEE.

[13] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science Magazine*, 286:509–512, 1999.

[14] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[15] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[16] Y. Chawathe, S. McCanne, and E. Brewer. An architecture for Internet content distribution as an infrastructure service, 2000.

[17] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, June 2000. ACM.

[18] S.E. Deering. Host extensions for IP multicasting, 1989. Internet RFC 1112.

[19] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over a peer-to-peer network. Technical report 2001-31, Stanford University, 2001.

[20] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, January/February 2000.

[21] D.Waitzman, C.Partridge, and S.E Deering. Distance vector multicast routing protocol. Request for comments - 1075, IETF Network Working Group, 1988.

[22] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of ACM SIGCOMM*, pages 251–262, Cambridge, MA, September 1999. ACM.

[23] Paul Francis. Yoid: Your own Internet distribution, April 2000. http://www.aciri.org/yoid.

[24] Lixin Gao. On inferring autonomous system relationships in the Internet. In *Proceedings of IEEE Global Telecommunications Conference*, pages 387–396, San Francisco, November/December 2000. IEEE.

[25] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet map discovery. In *Proceedings of IEEE INFOCOM 2000*, pages 1371–1380, Tel Aviv, Israel, March 2000. IEEE.

[26] ISI ns-2 online manual. *The Network Simulator ns-2: Tips and Statistical Data for Running Large Simulations in NS.* http://www.isi.edu/nsnam/ns/ns-largesim.html.

[27] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of USENIX Symposium on Operating System Design and Implementation*, San Diego, October 2000. The USENIX Association.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 4598:671–680, May 1983.

[29] David Liben-Nowell, Hari Balakrishnan, and David Karger. Observations on the dynamic evolution of peer-to-peer networks. *Lecture Notes in Computer Science*, 2429:22–33, 2002.

[30] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: Universal topology generation from a user's perspective. Technical Report BUCS-TR-2001-003, Boston University, January 2001.

[31] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in Internet topologies. *ACM Computer Communication Review*, 30:18–28, April 2000.

[32] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter p2p networks. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pages 492–499, Las Vegas, October 2001. IEEE.

[33] Vern Paxson and Sally Floyd. Why we don't know how to simulate the Internet. In *Proceedings of the 29th conference on Winter simulation*, pages 1037–1044, Atlanta, December 1997. IEEE.

[34] Dimitris Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, pages 49–60, San Francisco, March 2001. The USENIX Association.

[35] S. Raman, S. McCanne, and S. Shenker. Asymptotic scaling behavior of global recovery in SRM. In *Proceedings of SIGMETRICS/PERFORMANCE 98, Joint International Conference on Measurement and Modeling of Computer Systems*, pages 90–99, Madison, Wisconsin, June 1998. ACM.

[36] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM 2001*, pages 161–172, San Diego, August 2001. ACM.

[37] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001. ACM.

[38] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, January 2002.

[39] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the ACM SIGCOMM 2001*, pages 149–160, San Diego, 2001. ACM.

[40] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: Degree-based vs structural. In *Proceedings of ACM SIGCOMM*, pages 147–159, Pittsburgh, PA, August 2002. ACM.

[41] D. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM 2003*, pages 1283–1292, San Francisco, April 2003. IEEE.

[42] Duc A. Tran, Kien A. Hua, and Tai T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications*, 22:121–133, January 2004.

[43] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1611–1622, December 1988.

[44] Ellen Zegura. Thoughts on router-level topology modeling, January 2001. The End-to-end interest mailing list.

[45] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of IEEE INFOCOM 2002*, pages 1366–1375, New York, June 2002. IEEE.

[46] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, Port Jefferson, New York, June 2001. ACM.