Three Hops Overlay Routing

Patrick Lee
University of Toronto

Hans-Arno Jacobsen
University of Toronto

Abstract

Improvement in datacenter networks will enable architects to built larger distributed systems. However, standard distributed system architectures and their resulting communication patterns are incapable of supporting systems much larger than the current limits.

This paper describes the design and evaluation of an overlay topology and messaging infrastructure that can be used for creating datacenter-scale distributed systems. The topology provides sufficient scalability to accommodate future datacenter growth. Theoretical analysis suggests its fault resiliency and network capability are unmatched by other common network topologies. The messaging infrastructure build on top of this topology supports incremental scaling, node failure recovery and flexible routing. Properties of the topology ensure that the overlay has constant bounded delay and bandwidth usage, which are useful for planning incremental deployment while guaranteeing performance. Finally, evaluation of our overlay prototype reveals usage trends that best matches the characteristics of our overlay.

1 Introduction

The increased reliance on cloud-based distributed systems (e.g., BigTable[9], PNUTS[10], Dynamo[12], Cassandra[25], MapReduce[11] and Dryad[21]), henceforth referred to as services, pressures services to scale beyond their existing capacities. However, the scalability of services are hindered by their architectures and resulting communication patterns. To understand this issue, we must first examine the underlying network architectures and their implications on service designs.

Tree-like physical network topologies dominate current datacenter network designs[16, 19, 17]. While such physical networks are simple to deploy, they suffer from severe oversubscription due to the lack of interconnecting links between switches. Traffic generated by services that routes through multiple tree levels may experience heavy network congestion, which result in service performance degradations. To cope with these concerns, services are deployed within their own clusters or pods, each occupying an unique subtree partition[18]. As a direct consequence of this arrangement, services are rarely designed to scale beyond low thousands of servers, a small number relative to the total number of servers available to a typical cloud datacenter[15].

Given this environment, two classical approaches thrive in the service design space: the (hierarchical) master-and-slave approach and the direct peer-to-peer approach. GFS[14], BigTable[9] and MapReduce[11] are prime examples of (hierarchical) master-and-slave designs using star (or tree) topology communication patterns. Unsurprisingly, the master node is a single point of failure and a potential bottleneck for operations that require the master's coordination. While careful planning can reduce the reliance on the master node, it is unlikely that these techniques are adequate to improve scaling by (one to two) orders of magnitude. Dynamo[12] and Cassandra[25] are two examples of direct peer-to-peer designs which use complete graph topology communication patterns. Because each node may interact with any other node within the system, accurate global membership states must be maintained by each node to deal with failures. This requirement places a heavy burden on both the network and the nodes. Therefore, direct peer-to-peer designs quickly become impractical when we wish to deploy services spanning, potentially, an entire datacenter.

Recent advancements in physical network topologies, such as VL2[16], DCell[17] and Dragonfly[23], will enable future datacenters to overcome the flaws of current physical network designs. This in turn enables service architects to design larger services. Since the aforementioned standard practices are incapable of fully utilizing a datacenter, services must be structured around an alternative approach that uses a different communication pattern.

At first glance, Internet-scale distributed hash table (DHT) overlays (e.g., Chord[38], Pastry[35], and CAN[32]) seem like ideal building blocks for datacenter-scale services. Despite the wide adoption of consistent hashing[22], service architects have consistently avoided the use of Internet-scale DHT overlays; this is clearly evident in the designs of Dynamo[12], Cassandra[25] and Memcached[2]. We argue the design choices which enabled these overlays to scale to their targeted size prevent them from being viable solutions for building services.

Firstly, because Internet-scale DHT overlays are designed to hypothetically support millions to billions of nodes[1], N, each node can reasonably maintain at most $O(log(N))$ entries within its forwarding table, and require an average of $O(log(N))$ hops to forward a message. As a consequence, each message consumes $O(log(N))$ times more bandwidth; latency also increases by $O(log(N))$ times[2]. Since most service level agreements have strict latency budgets (e.g., PNUTS[10] must complete a typical request within a latency budget of 50-100 ms), an unbounded logarithmic growth in latency and bandwidth usage makes it difficult to plan for incremental scaling while guaranteeing performance.

Secondly, the communication patterns of common functionalities, such as event notification, content distribution and replication, often do not conform to the overlay's DHT-based routing pattern. While it is possible to implement other routing paradigms, such as multicast, on top of DHTs through the use of rendezvous points (e.g., Scribe[36]), such implementations are highly inefficient since they require the use of twice the average number of forwarding hops to deliver a message.

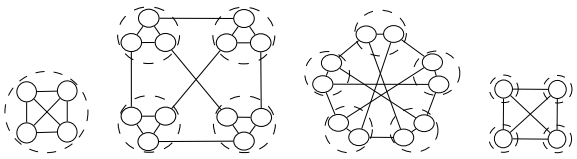
These observations motivate us to develop a new overlay abstraction intended for datacenter-scale services. This overlay must have a small yet constant diameter to minimize the message forwarding overhead, and to ensure a constant upper bound for the routing delay[3] and bandwidth usage. The overlay must be fault tolerant since failures are the norm in cloud datacenters consisting of commodity servers. It must also support arbitrary routing paradigms without relying on mechanisms, such as rendezvous points, that aggravate network congestion and introduce additional delays due to extra forwarding hops.

Our proposed solution is a hierarchical hybrid peer-to-peer overlay that satisfies all of the aforementioned conditions. The overlay is based on a topology class that has a constant diameter of three and exhibits superior fault resiliency and network capacity when compared with other common network topologies. Scalability is ensured since each node within a N node overlay only maintains $O(N^{\frac{1}{3}})$ states. Embedded within the overlay, each node forms an unique broadcast tree of height three. This is exploited to create a simple generic multicast algorithm. Thus, arbitrary routing paradigms (e.g., DHT, publish/subscribe, and point-to-point) can be implemented using this multicast algorithm by placing filters at appropriate nodes along the forwarding path.

Section 2 presents the topology design along with scalability and fault resiliency analysis from a theoretical perspective. Section 3 describes the construction of the overlay. Section 4 explains how the overlay is used to facilitate efficient message routing. Section 5 examines implementation issues and presents an evaluation of our overlay prototype. Finally, Section 6 discusses related work.

2 Topology Design

Intuitively, our topology is a two-level fractal structure where each level forms a complete graph. Given a set of nodes, we partition the nodes into equal-sized sets, called virtual node groups. Each virtual node group forms a complete graph by connecting all the nodes within the same virtual node group. We then consider each virtual node group as a single vertex, called a virtual node, and connect all the virtual nodes to form yet another complete graph. In many respects, our topology is similar to a super-peer network[42], where each virtual node can be view as a super-peer. However, unlike super-peer overlays, the responsibility for connecting the virtual node to other virtual nodes is evenly distributed among the nodes within the virtual node group. Figure 1(b) and 1(c) are two examples of the proposed topology. Since each node is at most one inter-virtual node edge and two intra-virtual node group edges aways from any other node, our topology has a constant diameter of three.



(a) c=3, i=0, N=4 (b) c=3, i=1, N=12 (c) c=3, i=2, N=10 (d) c=3, i=3, N=4

Figure 1: Topology examples

Note that we can obtain different topologies by varying the fractal level (in particular, a fractal level of one yields a complete graph). The diameter of these topologies is a function of the fractal levels, k,

$$diameter(k) = \begin{cases} 1, & k = 1 \\ 2 \cdot diameter(k-1) + 1, & k > 1. \end{cases}$$

Clearly, we must select a fractal level that balances our scalability needs and latency requirements. In situa-

[1] One to four orders of magnitude larger than the largest datacenters

[2] Assuming the increased bandwidth requirement does not cause serve network congestion

[3] Provided the network is not heavily congested

tions where the number of edges per node is bounded from above (e.g., when creating a physical router topology[17]), higher fractal levels may be used to improve the topology's scalability. However, this improvement comes at the expense of increasing the topology's routing latency and complexity, and reducing the topology's fault resiliency. Because our aim is to develop an application level overlay, our design is not severely restricted by connection limitations. The main limitation is the need to efficiently maintain neighboring states. Therefore, we can afford to use only two fractal levels.

2.1 Maximizing Scalability

To reduce the stress on any individual node in terms of managing its neighbors' states, each node should maintain the same number of connections, c, thus forming a fully connected c-regular graph. The overlay's size can then be expressed as the trade off between the number of connections used within each virtual node group and the number of connections used between virtual node groups. More formally,

$$\begin{aligned} N &= OverlaySize(c,i) \\ &= (\#\ of\ nodes\ per\ virtual\ node) \times (\#\ of\ virtual\ nodes) \\ &= (c-i+1)(i(c-i+1)+1) \end{aligned}$$

where i is the number of connections used by a single node to connect its virtual node group to other virtual node groups. Figures 1(a)-1(d) illustrate this trade off for c = 3 and i = 0, 1, 2 and 3, respectively. In general, selecting i from the two ends of the range will yield topologies that are more fault resilient yet less scalable, while selecting i from the middle of the range will yield topologies that are more scalable but less fault resilient. For now, our focus is on improving scalability. We will address the issue of fault resiliency in Section 2.4.

Keeping the total number of connections per node, c, fixed, we can find the optimal trade off by maximizing the overlay's size with respect to i,

$$i^* = \arg\max_{0 \le i \le c} OverlaySize(c,i)$$

This amounts to finding the local maxima at i^*, which can be obtained by solving $\frac{\partial}{\partial i} OverlaySize(c,i) = 0$ using the quadratic equation, resulting in

$$i^* = \frac{4(c+1)-\sqrt{4c^2+32c-8}}{6} \approx \frac{c+1}{3}$$

Thus, each node should allocate $\lfloor \frac{c+1}{3} \rfloor$ number of connections for connecting to other virtual node groups, and the remaining $c - \lfloor \frac{c+1}{3} \rfloor$ number of connections to handle neighbor nodes within the same virtual node group. For c = 3, the optimal topology is Figure 1(b).

For the rest of the paper, we will use $i^* = \frac{c}{3}$ and express the overlay's size as

$$N = OverlaySize(c) = \left(\tfrac{2c}{3}+1\right)\left(\tfrac{c}{3}\left(\tfrac{2c}{3}+1\right)+1\right)$$

to simplify analysis.

2.2 Suboptimal Topology

The topology we have discussed up until now is the optimal topology given a fixed number of connections per node. However, since we cannot always partition nodes into equal sized virtual node groups and node failures can occur, it is not always possible to construct the optimal topology. Here, we briefly discuss the cost of maintaining a suboptimal topology, where every virtual node is still fully connected to every other virtual node and every node within a virtual node group is still fully connected to every other node within the same virtual node group, but not all of the virtual node groups contain the optimal, $\frac{2c}{3}+1$, number of nodes.

In order to preserve a complete graph at the virtual node level, the inter-virtual node connections must be redistributed evenly within the suboptimal virtual node group. The cost for a suboptimal virtual node group, with respect to the total number of extra connections required, can be express as

$$\begin{aligned} cost &= (\#\ of\ inter\ virtual\ node\ connections\ for\ an \\ &\quad optimal\ group) - (\#\ of\ inter\ virtual connections \\ &\quad that\ can\ be\ handled\ by\ the\ suboptimal\ group) \\ &= \tfrac{c}{3}\left(\tfrac{2c}{3}+1\right) - \left(\tfrac{c}{3}+f\right)\left(\tfrac{2c}{3}+1-f\right) \end{aligned}$$

where f is the number of failures, or missing nodes, within the virtual node group relative to the virtual node group's optimal size. Simple algebraic manipulation reveals that a suboptimal virtual node group does not suffer from additional cost when $f < \frac{c}{3}+1$. Thus, it is in our interest to ensure each virtual node group retains the majority number of nodes relative to the optimal size[4].

2.3 A Realistic Look at Scalability

To put the scalability of our overlay into context, we compare its size as a function of the number of connections maintained by each node, to some publicly available figures in Table 1. Comments on the relative size are included whenever appropriate. All figures, with the exception of Google's and Microsoft's datacenters, represent the total number of servers deployed across multiple datacenters by each company as of 2009.

Table 1 shows that our overlay can comfortably accommodate the scale of today's largest datacenters. While we anticipate datacenters will continue to grow in size, we do not expect datacenters to grow significantly (i.e., multiple orders of magnitude) larger than the current largest datacenters[5] (e.g., Microsoft's Chicago datacenter). This is mainly due to the physical limitations,

[4] This condition can be relaxed for large topologies since a slight increase in the total connection cost will have little impact when averaged over a large group

[5] The current industrial trend is to deploy numerous *micro datacenters* close to major population centers to minimize end-user application latency[15]

c	N	Relative Size Comparison
10	200	—
20	1,386	—
30	4,431	—
40	10,220	iWeb 10,000 [30]
50	19,686	Peer1 10,277 [30]
60	33,661	Facebook 30,000 [30]
70	53,040	Akamai 48,000 [30], a Google datacenter (2005) 52,200 [29]
80	78,786	Rackspace 56,671 [30],
90	111,691	Intel 100,000 [30]
100	152,660	Microsoft's Chicago datacenter (Phase I, 2009) 140,000 [13]
110	202,686	—
120	262,521	—
130	333,080	Microsoft's Chicago datacenter at maximum capacity 300,000 [28]

Table 1: Overlay size comparison

such as power availability, and the risks, such as power outage, involved in placing a large amount of computing resources within a single location. Research in improving the utilization of existing resources will also play a major role in reducing the need to enlarge datacenters. To this end, our overlay succeeds in providing sufficient scalability to handle future datacenters growth.

2.4 Fault Resiliency and Network Capacity

To investigate the fault resilience property of our topology, we simulated failures within fixed size topologies and examined the percentage of nodes that belong to the largest remaining connected component. For our purpose, link failures are not considered. Node failures are assumed to occur independently and all remaining nodes are assumed to be correct (i.e., omission and Byzantine failures are not considered). Each simulation is repeated 50 times and the average is reported.

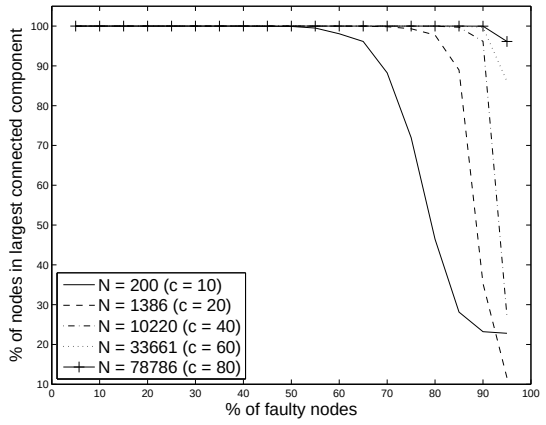


Figure 2: Stability in the presence of node failure

Figure 2 summarized. The figure show that our proposed topology can suffer up to 50% node failures without degradation in the topology's connectivity. Furthermore, as the size of the topology increases, the risk of partitioning the topology also decreases. In other words, the topology becomes more fault resilience as its size grows. With the exception of the complete graph topology, this last behavior is not exhibited in other well known network topologies.

To enhance our understanding of this behavior, we focus on two theoretic metrics used for analyzing fault resiliency and network capacity: the bisection width[26] and the bottleneck degree[17]. The bisection width is the minimum number of links that can be removed in order to divide a graph into two equal sized partitioned networks. By definition, larger bisection width implies stronger fault resiliency. The bisection width also provides an upper bound for a graph's overall network capacity. The bottleneck degree is the maximum number of active forwarding paths that uses a link in an all-to-all communication model. In other words, it is the maximum degree of multiplexing required for any particular link within the topology. In terms of fault resiliency, the bottleneck degree indicates the relative importance of any individual link; the lower the bottleneck degree, the less likely that a path is effected due to an arbitrary link failure. In terms of network capacity, because a lower bottleneck degree implies a smaller amount of multiplexing within each link, therefore, traffic is more spread out over all the links and individual paths can achieve higher throughput.

Graph Structure	Degree	Diameter	Bisection Width	Bottleneck Degree
Complete	$N-1$	1	$\frac{N^2}{4}$	1
Star	–	2	$\frac{N-1}{2}$	N
Ring	2	$\frac{N}{2}$	2	$\frac{N^2}{8}$
2D Torus	4	$\sqrt{N}-1$	$2\sqrt{N}$	$\frac{N\sqrt{N}}{8}$
Tree	–	$2\log_{d-1} N$	1	$N^2\left(\frac{d-1}{d^2}\right)$
FatTree	–	$2\log_2 N$	$\frac{N}{2}$	N
Hypercube	$\log_2 N$	$log_2 N$	$\frac{N}{2}$	$\frac{N}{2}$
Butterfly *	4	$2l$	$\frac{N}{l+1}$	$O(Nl)$
de Bruijn	d	$\log_d N$	$\frac{2dN}{\log_d N}$	$O(N\log_d N)$
DCell	$k+1$	$< 2\log_d N - 1$	$\frac{N}{4\log_d N}$	$< N\log_d N$
THOR	$\Theta(\sqrt[3]{N})$ †	3	$\Omega(N^{\frac{4}{3}})$ ‡	$O(N^{\frac{2}{3}})$ £

* $N = (l+1)^l$

† $\sqrt[3]{\frac{27N}{5}} < c < \sqrt[3]{\frac{27N}{4}}$

‡ Bisection width $> \frac{1}{81}\left(\frac{27N}{5}\right)^{\frac{4}{3}}$

£ Bottleneck degree $< \frac{4}{9}\left(\frac{27N}{4}\right)^{\frac{2}{3}}$

Table 2: Topology comparison

Table 2 compares our topology to other common network topologies. It is an extension of the table presented by Guo et al.[17]. Within the table, N represents the topology size, d represents a fixed degree and k represents the number of levels used within the DCell topology. Our topology occupies the last row of the table, under the heading of THOR. The star topology was also added for the sake of completeness with respect to the

discussion of overlay topologies.

Our topology is the only topology besides the complete graph which has super-linear growth in bisection width and sub-linear growth in bottleneck degree. Therefore, we conclude that the abnormal growth in bisection width and bottleneck degree, relative to other topology structures, are the main contributing factors for the behavior where our topology becomes more fault resilience as its size grows. From the network capacity stand point, we can also conclude that our topology will have better performance in terms of maximum capacity and load distribution when compared with other topologies.

3 Overlay Construction

In this section, we describe the mechanisms used for overlay construction and maintenance. For simplicity we assume that a node will not be reassigned to another group once it has joined the overlay[6], and node departures will not cause a decrease in the number of groups.

3.1 Membership Service

Centralized services such as Chubby[7], Autopilot[20] and Zookeeper[1] have demonstrated their usefulness in reducing the design complexity of large-scale production systems. Borrowing this idea, we choose to simplify our overlay design through the use of a centralized membership service. Similar to other centralized services, our membership service is a single point of failure and must be replicated to ensure high availability.

The membership service is used for assigning node identifiers when nodes first join the overlay. The membership service also provides look up functionality for finding members within a specific group. By maintaining the node information at a centralized location, we can accurately keep track of statistics such as the total number of nodes within the overlay and the expected number of connections per node. These statistics significantly reduce the complexity of deciding where a node should be assigned to, and when a new group should be created.

To ensure the membership service can scale with the overlay, we need to minimize our dependency on the membership service. Hence, we choose not to rely on the membership service for managing connections. This lessens the membership service's storage requirement since there are far fewer nodes than connections. Furthermore, since the membership service is not heavily involved in the failure recovery process, it does not need to constantly maintain an up-to-date view of the entire overlay. Thus, interactions between the membership service and the nodes can be cut down dramatically.

[6] A node can join a new group by leaving and reentering the overlay.

Node Assignment Policies. The datacenter's physical network has a major impact on the overlay's performance. As such, node assignment policies should account for the underlying network's characteristics and the service's requirements. For instance, in existing datacenters, bandwidth availability between racks is scarce relative to bandwidth availability within a rack[19, 14]. Services that have a high bandwidth requirement can assign nodes from the same group into the same rack to minimize communication cost, but at a cost of increased failure correlation within a group and reduced fault resiliency within the overlay due to potential router failures. In contrast, services that require high availability can spread nodes from the same group onto separate racks, but at the cost of reduced bandwidth availability. Because our implementation was not designed for a specific workload or physical network topology, we choose to assign nodes based on a first come first serve policy.

3.2 Establishing Inter-group Connections

Once a node decides to create a inter-group connection, it must select a connection partner from the external virtual node group such that the number of connections is evenly distributed amongst the nodes within that group. To deterministically achieve this goal, we must rely on either consensus or a coordinator node to perform the selection. However, neither option is palatable since the communication overhead of consensus is very high for large quorum groups and the coordinator approach can create a significant workload on the coordinator node[7].

Instead, we opted to use a randomized load balancing algorithm described by Mitzenmacher[31] to select the connection partner[8]. When a node wishes to establish an inter-group connection with another group, it looks up a randomized subset of that group's members from the membership service. Because the returned candidate list may include faulty nodes, the initiating node usually requests more than two candidates at a time. Upon receiving the candidate list, the initiating node creates temporary connections with every node in the list and requests load information from each node. It then waits for responses from the candidate nodes, up to a maximum time window. Finally, from the set of candidates that respond in time, the initiating node selects the node with the least load as its connection partner and then closes all other temporary connections between the two groups.

[7] Leader election is also a costly operation due to potential failures.

[8] In retrospect, parallel randomized load balancing[3] would have worked better, but this does not effect the correctness of the overlay.

3.3 Re-balancing Inter-group Connections

Significant imbalance of inter-group connection assignments can occur when a new node joins an existing group, or occasionally, by chance due to randomized load balancing. Periodically, each node checks its local topology states for imbalance in inter-group connections. Within each group, if there is a substantial disproportion between the node with the most number of inter-group connections, referred to as the old node, and the node with the least number of inter-group connections, referred to as the new node, then the old node will attempt to transfer an inter-group connection[9] to the new node. We will refer to the node on the other end of the inter-group connection as the external node.

To prevent oscillating migration behavior due to over aggressive re-balancing within the group, the new node will only accept a single inter-group connection at a time. To ensure consistency between groups, we need to avoid the scenario where both the old node and the external node concurrently attempt to shed the same connection. Therefore, the old node must first acquire a lease from the external node, which temporarily prevents the connection from migrating within the other group, before the old node can transfer the connection to the new node. To prevent live-locks, at most one lease can be held by the old node and the external node at any given time; lease disputes are resolved by group priorities.

Once the old node acquired a lease, it directs the new node to connect to the external node, bypassing the mechanism described in Section 3.2. The new node will then notify all members within the group of this event. The old node continues to use its existing connection as an uni-directional channel to forward already queued messages. However new traffic between the two groups are diverted to the new node. Once the backlog queue is drained, the old node will finally closes its connection.

3.4 Node Arrivals

For a large service, the performance impact of adding a few nodes to the service is negligible. This reason, along with economy of scale, dictates that incremental scaling should be done in batches of noticeable size. On the other hand, service administrators may wish to replenish nodes that were brought offline either due to failure or maintenance. This is usually done in batches, although it can also be done one node at a time. For the purpose of our discussion, we will consider single node arrivals and batches arrivals separately.

Single Node Arrivals. Based on our assumption that nodes will not be reassigned to other groups and the analysis in Section 2.2, we should not create a new virtual node group for the case where a single node joins to the overlay; otherwise, the new node must bear the entire burden of connecting to every other virtual node group. Upon arrival, the membership service will assign the node to an existing virtual node group with the least number of members. The node joins the virtual node group by connecting to every other member in the group. The node can then rely on the connection re-balancing mechanism described in Section 3.3 to redistribute the inter-group connections within the group.

Multiple Node Arrivals. If the overlay can accommodate the new batch of nodes without creating new groups, or if the batch size of the new nodes is smaller than half the optimal group size, then the membership service will assign all new nodes to existing groups. Otherwise, the membership service will create as many groups as needed in order to accommodate all of the nodes, and the new nodes are then distributed among all of the groups such that each group ends up with roughly the same number of nodes. New nodes assigned to existing groups will join their groups by using the procedure stated in the previous paragraph. As for the nodes assigned to the newly created groups, a node first joins the group by connecting to all members within the group. It then establishes inter-group connections with candidates from a bootstrap list provided by the membership service, using the mechanism presented in Section 3.2; the bootstrap lists are generated such that each node within the same group will handle a partition of the external group set.

3.5 Node Departures

Following Hamilton[18]'s advice, our overlay does not distinguish between normal shut down and node failures. This design choice greatly reduces the implementation complexity and ensures the recovery mechanism will function correctly during failures. Because reactive recovery schemes are prone to create positive feedback cycles when the network is congested[34], we choose to implement a periodic recovery scheme. Our recovery scheme was designed to handle node failures and temporary inter-group link failures.

A node checks its local topology states for failures at regular intervals. If the node detects that it is managing the least number of connections within the local group and the local group is disconnected from an external group which has a lower group priority, then the failure recovery protocol is triggered.

Upon initializing the recovery process to reestablish communication with a targeted group, the node sends out lease requests to its neighbor within the local group to indicate it will commence the recovery process and other

[9] The least used connection is selected to minimize disruption.

members should temporary ignore the targeted group. The node then waits for a short interval to ensure no other member attempts to recover the same group; during the waiting period, if the node receives a lease request from another member for recovering the same group and that member has a higher priority than itself, then its lease is revoked and the recovery protocol is aborted. Once the node confirmed that it is the designated recovery handler, the node communicates with the membership service and asks for a subset of candidate nodes from the disconnected group. Finally, the node reestablishes inter-group connection with one of the returned candidates, using the method described in Section 3.2.

4 Message Routing

Efficient message routing in unstructured graph-based overlays is generally a challenge. Since graph-based overlays inherently contain cycles, cycle detection and prevention mechanisms are needed to avoid circulating messages infinitely. Depending on the overlay's topology properties, such mechanisms may introduce high overhead to the routing cost. Furthermore, designing a routing scheme that uses efficient (i.e., shortest) paths can be difficult. For these reasons, tree-based overlays are usually preferred over unstructured graph-based overlays for message dissemination. Although message dissemination for tree-based overlays is extremely simple, drawbacks such as unbalanced node workloads and poor fault-resiliency make tree-based overlays unattractive for large-scale service deployment.

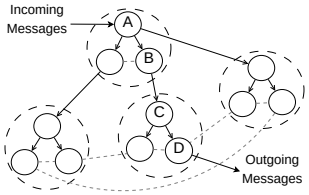


Figure 3: An embedded dissemination tree

By utilizing additional information associated with our overlay hierarchy, message routing within our overlay is no more complicated than tree-based message routing. In particular, we observe there is exactly one shortest path between any pair of nodes, and this shortest path will only involve nodes that belong to either the source or destination virtual node group. Furthermore, when the source and destination virtual node groups differ, the shortest paths between a source node and all of the nodes within the destination virtual node group will share a partial path, diverging only on the last hop within the destination virtual node group. Thus, for any arbitrary node within the overlay, there exists an embedded dissemination tree with depth of three, rooted at that node. Figure 3 illustrates the dissemination tree for an arbitrary node, A, embedded with the topology from Figure 1(b). The notion of a central node associated with tree-based overlays does not exist in our overlay because each node is the root node of its own unique dissemination tree; hence, unlike tree-based overlays, the message forwarding workload is evenly distributed among all the nodes.

4.1 Shortest Path Routing

Algorithm 1 Generic multicast algorithm

```
multicast(msg):
    for all n ∈ interestedLocalNeighbors(msg) do
        send(msg) to neighbor n
    end for
    for all g ∈ interestedNeighboringGroups(msg) do
        send(msg) to neighboring group g
    end for
    for all c ∈ interestedClients(msg) do
        deliver(msg) to client c
    end for

receive(msg) from sender x:
    if msg.sourceNode ∈ localVirtualNodeGroup then
        for all g ∈ interestedNeighboringGroups(msg) do
            send(msg) to neighboring group g
        end for
    else if x ∉ localVirtualNodeGroup then
        for all n in interestedLocalNeighbors(msg) do
            send(msg) to neighbor n
        end for
    end if
    for all c ∈ interestedClients(msg) do
        deliver(msg) to client c
    end for
```

Algorithm 1 illustrates a generic multicast algorithm using the shortest path dissemination trees embedded within our overlay. $InterestedLocalNeighbors$ is a function that returns a subset of the members within the same virtual node group that are directly or indirectly interested in a message. A node becomes indirectly interested in a message if any virtual node group that it is connected to is interested in the message; it simply acts as a forwarding node if it is not directly interested in the message. $InterestedNeighboringGroups$ is a function that returns a subset of the virtual node groups that are interested in the message and are directly connected to the node. Similarly, $InterestedClients$ is a function that returns a subset of clients that are interested in the message and are directly connected to the node. Returning to our example in Figure 3, node A will initiate message forwarding by executing the $mutlicast$ function; node B will forward the message to node C by executing the first branch of the if statement within the $receive$ function; and node C will forward the message to node D by executing the second branch of the if statement within the $receive$ function.

By varying the behaviors of $interestedLocalNeighbors$, $interestedNeighboringGroups$ and $interestedClients$, different message routing schemes can be implemented. For example, global broadcast is implemented by modifying $interestedLocalNeighbors$ to return a set containing all other members within the virtual node group and $interestedNeighboringGroups$ to return a set containing all virtual node groups that the node is connected to. In subsequent sections, we will demonstrate how this simple idea can be used to support different message routing paradigms. The described routing paradigms are implemented in our overlay through the use of object polymorphism.

4.1.1 Publish/Subscribe

For the purpose of our discussion, we will consider the publish/subscribe paradigm as a generalization of all forms of dynamic multicast routing schemes. We focus on topic-based publish/subscribe routing since it is one of the most successful paradigms in use by reconfigurable multicast systems. To illustrate this point, many popular Google applications, including YouTube, rely on an internal topic-based publish/subscribe service[33] to facilitate data exchange. However, our discussion is also general applicable to content-based publish/subscribe routing, an active area of research[4, 8, 27, 40, 41].

When a node becomes interested in a topic, it subscribes to the topic by broadcasting a subscription message to all the nodes within the overlay. Upon receiving a subscription message, the node adds the topic to the subscription set. Publication messages can then be disseminate to selected sets of nodes by comparing the publication topics against the subscription set.

In our topic-based publish/subscribe matching implementation, the subscription set is maintained in three tree-based maps[10] (topic $\rightarrow$ set of interested recipients): one for the clients, one for the local neighbors, and one for the external groups. Note that Algorithm 1 does not uniquely distinguish subscriptions with the same topic when the subscriptions originated from the same external group. Therefore, in practice, topics are aggregated within each group and subscription messages are broadcast out of a group only when that group's aggregated topic set changes.

4.1.2 Point-to-point Communication

Point-to-point channels can be easily implemented by routing the messages based on the destination node's id. However, location-dependent addressing schemes promote static network assignment and fragmentation of resources, and hence should be avoided[15]. As an alternative, point-to-point channels can be implemented on top of the topic-based publish/subscript mechanism. When a client that uses point-to-point communication joins the overlay, it subscribes to a topic that represents its canonical location-independent address. A peer can then communicate with that client by publishing messages with the client's canonical address as the topic.

[10] For systems that deal with large amount of topics, Bloom filters[6] are more appropriate data structures than maps.

4.1.3 Distributed Hash Table

Consistent hashing forms the foundation for distributed hash table based systems. In the original variant[22], each node in the system is assigned an unique yet random position value in a ring of integers modulo n, typically $\mathbb{Z}_{2^{128}}$ or $\mathbb{Z}_{2^{160}}$. An object identified by a key can then be located within the system by mapping the key to a hash value and finding the node with the closest value.

A two-level consistent hashing scheme follows naturally from the hierarchy of our overlay. Each node is assigned to a set of random position values which are unique relative to the other members within the same virtual node group, thus forming a local ring within each group. Similarly, each virtual node is assign to a set of random position values which are unique relative to other virtual nodes, thus forming a global ring that spans the entire overlay. In our implementation, we assumed the set size is the same for each (virtual) node. By making this assumption, each node can compute all position values using the local topology information without any coordination, thereby reducing management overhead.

To locate an object within our overlay, a message is first routed to the appropriate group identified by consistent hashing using the global ring. Once the message has reach the appropriate group, it is routed to the correct node by applying consistent hashing on the group's local ring. Note that if both levels of consistent hashing share a hash value generated by a single hash function, then partitioning at the global ring will create unusable segments within the group's local ring, which results in uneven load distribution. Therefore, each level must apply a different hash function to ensure pairwise independence. In practice, we generate the two hash values by splitting a single hash value computed from a cryptographic strength hash function into two equal halves.

Load Balance Comparison. Because of the importance of distributed hash table in the development of storage (e.g., Dynamo[12] and Cassandra[25]) and caching (e.g., Memcached[2]) services, we must ensure the proposed two-level consistent hashing scheme performs no worse than the standard consistent hashing in terms of load balancing.

c	N	Hashing scheme	p†	Points maintained within each node		Normalized partition size handled by each node		
				Total #	Rel. size	Avg.	Std. dev.	90th %
30	4,431	Standard	1	4,431	1.000	2.26e-4	2.27e-4	5.23e-4
		2 Levels	38	8,816	0.9948	2.26e-4	4.9e-5	2.90e-4
		2 Levels	1	232	0.0262	2.26e-4	3.67e-4	5.72e-4
60	33,661	Standard	1	33,661	1.000	2.97e-5	2.99e-5	6.85e-5
		2 Levels	78	67,236	0.9987	2.97e-5	4.8e-6	3.61e-5
		2 Levels	1	862	0.0128	2.97e-5	5.11e-5	7.67e-5
90	111,691	Standard	1	111,691	1.000	8.95e-6	8.91e-6	2.064e-5
		2 Levels	118	223,256	0.9994	8.95e-6	1.15e-6	1.046e-5
		2 Levels	1	1892	0.0085	8.95e-6	1.557e-5	2.327e-5
120	262,521	Standard	1	262,521	1.000	3.81e-6	3.81e-6	8.79e-6
		2 Levels	158	524,876	0.9997	3.81e-6	4.3e-7	4.37e-6
		2 Levels	1	3,322	0.0063	3.81e-6	6.53e-6	9.89e-6

† The number of assigned points per (virtual) node

Table 3: Consistent hashing schemes load comparison

Within our topology, the total number of position values maintained by each node grows cubically, $O(overlay\ size) = O(c^3)$, for standard consistent hashing, while the same number grows quadratically, $O(\#\ of\ virtual\ node) + O(\#\ of\ node\ within\ the\ group) = O(c^2) + O(c) = O(c^2)$, for the two level scheme. Moreover, because each position value within the two level scheme can be represented with half as many bits compare to the standard scheme, hence, we can store twice as many position values for the two level scheme using the same amount of storage. For a fair comparison, we need to ensure that both schemes use roughly the same amount of storage for maintaining routing information. Therefore, to compensate for the differences, we evenly distributed twice the number of points used by the standard consistent hashing scheme between the routing table for the virtual nodes and the routing table for the nodes. For simplicity, the same number of points are assigned to each virtual node and node.

We compared the standard consistent hashing scheme using a $\mathbb{Z}_{2^{160}}$ ring against the two level scheme using two levels of $\mathbb{Z}_{2^{80}}$ rings, with hashes generated by a SHA1 function. Our results are summarized in Table 3. Note that $Rel.\ size$ indicates the amount of storage used by a scheme relative to the storage required by the standard scheme for the same overlay size, and the partition sizes are normalized such that $\sum partitionSize_i = 1$.

As a baseline, our results for standard consistent hashing agree with the theoretical analysis from Karger et al.[22], with the majority of partitions no larger than roughly double the average partition size. More importantly, the two level hashing scheme routinely has a smaller standard deviation and 90 percentile when using the same amount of storage. Hence, the two level consistent hashing scheme is better at spreading the load distribution than regular consistent hashing. Also, at the cost of slight increase in load imbalance, significant storage saving can be achieved through the two level consistent hashing scheme because its storage growth is one polynomial degree lower than standard consistent hashing.

We conclude our discussion of load balancing by stating that our choice of position points allocation (i.e., using the same number of points per node for each level) is suboptimal with respect to load distribution. While the optimal trade off between the number of points assigned to the global ring and the number of points assigned to the local ring can be found by solving an optimization problem that minimizes the variance within both ring levels, constrained by twice the maximum number of points used in the original consistent hashing scheme, this is beyond the scope of our paper.

4.2 Dealing with Failures

When the overlay suffers from a failure, a virtual node group may become temporarily disconnected from another virtual node group. Hence, we need a solution to work around the fact that messages cannot travel directly between the two disconnected groups during brief recovery periods. To minimize disruption, messages should not be dropped whenever possible.

One obvious solution to deal with temporary failures is to maintain within each node a message backlog queue per disconnected group, and resume message forwarding once the link between two groups has been reestablished. However, a drawback for this approach is that it may require a huge amount of storage to handle accumulated messages. Additionally, when the message incoming rate is roughly equal to the maximum outgoing rate, the backlog can introduce significant delays after the link has recovered because the node is unable to drain the queued messages quickly enough. Capping the maximum queue length fixes both issues, but it is an unattractive alternative since this could lead to a high message drop rate. Thus, we did not rely on buffering to handle failures[11].

Instead of buffering messages, we make use of path redundancy within our topology to route messages. In our current implementation, the overlay supports message forwarding through a single intermediate virtual node group: if the destination group is disconnected from the source group, then an arbitrary group that is connected to the source group is selected for the duty of forwarding the message to the destination group; if the intermediate group is also disconnected from the destination group, then the message is dropped. Note that as the overlay size increases, the probability that two groups sharing the same neighboring node from a third group decrease; therefore, two independent paths become less likely to share a common intermediate node as the overlay grows in size. Hence, when failures are independent, the successful message delivery rate of our scheme improves as the overlay scales out. This should not come as a surprise since our fault resiliency simulation from Section 2.4 shows similar results. Note that we can view

[11] However, buffering is used to deal with high volume traffic bursts.

this routing scheme as a direct application of RON[5]; in particular, if we collapse each group into a single vertex, then it should be obvious that the traffic pattern generated by this scheme matches that of RON.

Our RON-styled routing scheme can be easily adapted to support applications that require guaranteed fault resiliency and timely message delivery. By ensuring that no two paths share a common initial link and messages are routed through at most one intermediate forwarding group, we observe that each path has a maximum length of five and has a disjoint set of intermediate nodes. Based on these observations, we can implement a form of mesh-based routing[37]: regardless of failure, a source node broadcasts a message to all of its neighbors; if the neighboring node belongs to the same group, then the node selects an arbitrary neighboring group as the intermediate group to relay the message to its destination; otherwise, the neighboring node belongs to an intermediate group and hence should forward the message directly to its destination. Because each node is c-connected, mesh-based routing through our overlay can handle up to $c-1$ intermediate node failures[12].

5 Implementation and Evaluation

Implementation. The overlay nodes are implemented in C++ using a simplified version of the staged event-driven architecture[39]. Dynamic resource controllers were not included in our system since performance predictability is jeopardized by dynamic resource reallocation. Instead, thread pool allocation and batching factor polices are statically specified. This reduces the implementation complexity and improves repeatability in our experimental evaluations.

The resulting design resembles the architecture of a combined input-output-queued switch, with two control planes and a single forwarding plane. One control plane is in charge of the overlay's topology management and implements the mechanisms described in Section 3. The second control plane is similar to a traditional router control plane which deals with building and updating route matching engines. Route matching engines are paradigm specific; their complexity ranges from simple look up tables to complicated state machines. Applications can implement different routing paradigms and dynamically register new route matching engines to the nodes through the use of the factory pattern. Our prototype currently supports broadcast, topic-based publish/subscribe and DHT routing as described in Section 4. Finally, normal message traffic is routed through the forwarding plane. Since all three planes share the same senders, outgoing messages are prioritize based on the sending plane.

Because our prototype was created as a proof-of-concept, the implementation design was kept simple with very few optimizations. This design approach works fairly well in general, however, one particular shortcoming of oversimplification was observed during our experiments. Due to the effects of multicasting, a message may be handled by multiple senders, each running on a different thread. In our current implementation, to ensure the messages are properly cleaned up after the sending stage, each message is replicated at the route matching stage and unique copies of the message are passed to the senders. This results in unnecessary high memory usage as well as wasted CPU cycles due to memory copying. In retrospect, memory copying overhead can be eliminated through the use of reference counting with the support of a sophisticated lock management.

Evaluation. To enhance our understanding of the overlay's routing capability, we deployed and tested our overlay on various workloads. We are particularly interested in finding the trends in maximum sustainable throughput as the overlay grows since this is a good predictor for the maximum service rate that can be achieved as a service scales out[13]. While evaluations were performed on all implemented routing paradigms, we focus on the results from broadcast routing as they represent the worst case performance. In our experiments, the increase in latency due to additional forwarding hops was negligible.

Our experiments were conduced on a 22 nodes[14] cluster connected by a 1 Gbps Ethernet switch. The overlays were deployed such that the nodes are evenly divided amongst the servers, with each node connecting to a sink client. As a baseline, messages are broadcast from a single source client. Since messages are routed along the shortest path dissemination tree, this is equivalent to broadcasting within a tree-based overlay with similar branching factor[15]. This baseline is compared to the multiple sources case where messages are broadcast from every node within the overlay, with each node connecting to a source client. In all cases, the broadcast rate is increased until the nodes can no longer keep up with the broadcast rate and messages begin to accumulate. To observe the effects of varying message payload size, each experiment is repeated using 100 bytes, 1,000 bytes and 10,000 bytes payloads.

Figure 4 shows the average maximum sustainable broadcast rate per source client. Note the throughput rate for this particular figure is in logarithmic scale. As ex-

[12] We cannot guarantee link failure resiliency since different links may share a common physical link in the underlay.

[13] Absolute rates convey less information for our purposes because absolute rates are optimization dependent.

[14] Linux 2.6, 2×dual-core Xeon 5160 @ 3.00GHz, 4GB RAM

[15] By using the same implementation and node configuration, implementation biases are eliminated.

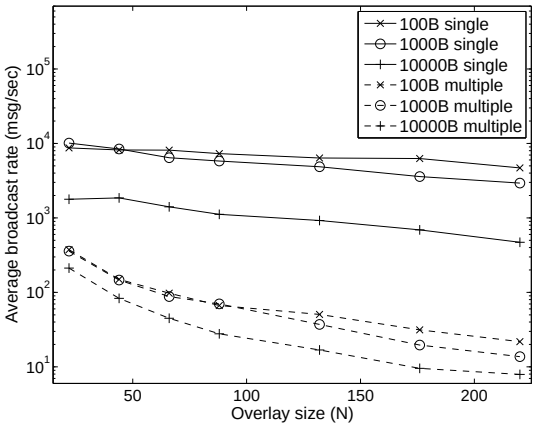


Figure 4: Average maximum sustainable broadcast rate per source client

pected, the average broadcast rate decreases as the overlay grows due to the increase in branching factor, and the average broadcast rate for the single source case is much high than the multiple sources case because resources are not shared. Observe the broadcast rate for the single source case suffers from exponential decay while the same fate is not shared by the multiple sources case. These trends suggest that sending from multiple sources becomes more beneficial as the overlay grows.

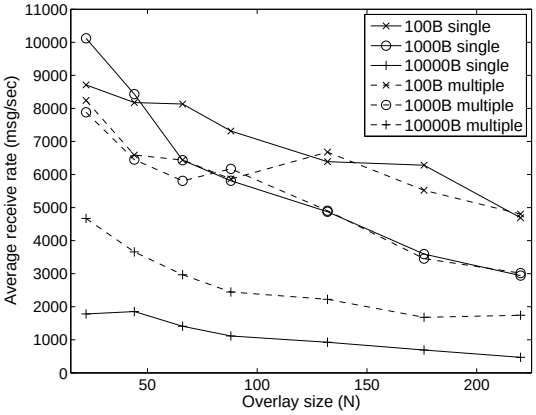


Figure 5: Average maximum sustainable receive rate per sink client

Figure 5 displays the average maximum sustainable receive rate per sink client. Because the total broadcast rate is statistically equivalent to the average receive rate per client, it is not included in our presentation. When message sizes are small, there is no significant performance difference between single source and multiple sources. This is attributed to a combination of high system call overheads and heavy context switching due to explicit thread yielding, which are used to relieve thread starvation caused by unfair mutex acquisition. In fact, for very small messages, sending from a single source may have a slight advantage over sending from multiple sources. However, when the messages are large, sending from multiple source is clearly superior to sending from a single source.

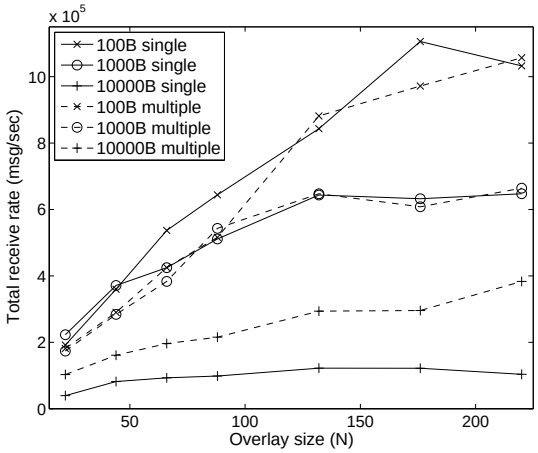


Figure 6: Total maximum sustainable receive rate

The total maximum sustainable receive rate is presented in Figure 6 and 7. Note that the values presented in Figure 7 included bytes used by the overlay's message headers. In general, these two figures reconfirm the results from Figure 5. Figure 7 also illustrates that sending messages in large batches helps improve overall throughput, regardless of sources. However, batching may not be possible for dynamic routing paradigms.

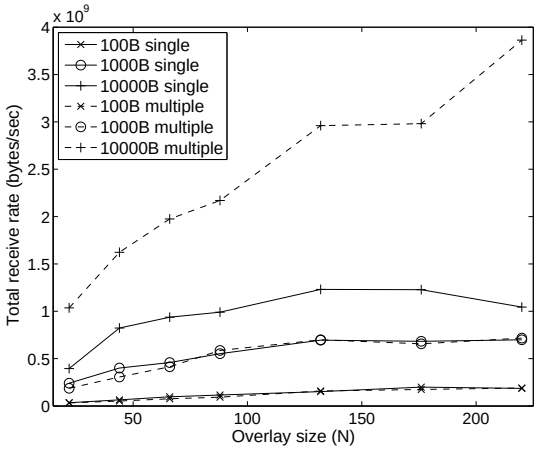


Figure 7: Total maximum sustainable receive rate (in bytes)

For all experiments, the throughput variance, both across time and between nodes, of the multi-source case is smaller than the single source case. This suggest the multi-source case is more capable of smoothing the burstiness of forwarding traffic than the single source case. This should not come as a surprise since the workload is evenly distributed across the entire overlay.

6 Related Work

Our topology was originally inspired by the small-world phenomenon as described by Kleinberg[24]. Because an Euclidean distance grid poorly models a datacenter network, we replace the grid with cliques of neighborhoods. Although our topology originates from the small-world phenomenon, its local clustering coefficient[26] is unusually large when compared with typical small-world networks[16]. Therefore, it is unclear if our topology can be classified as a small-world network.

In our initial design, each node has exactly one inter-virtual node connection. A scalability comparison against Pastry[35] reveals that our initial design was inferior to Pastry[17]. Following this analysis, efforts were directed toward optimizing our topology with respect to its size. As a result, our topology becomes four times more scalable than Pastry[18]. For the same diameter, the de Bruijn graph[26] is more scalable than our topology; however, our overlay's fault resiliency, its ease of incremental scaling, and its flexible routing paradigm support are not reproducible with the de Bruijn graph.

Interestingly, our topology is more closely related to physical router topologies than peer-to-peer overlay topologies. Specifically, our design choices are similar to DCell[17] and Dragonfly[23], both targeted towards improving the scalability of datacenter physical networks. With the exception of the fault resiliency theoretical analysis, which was influence by DCell, our work was independently developed from both approaches.

DCell[17] relies on low radix mini-switches and redundant server ports to construct a recursively defined topology. The lowest level of DCell, $DCell_0$, is formed by connecting servers to a mini-switch. Higher level of DCell, $DCell_i$, are formed by connecting the servers between k $DCell_{i-1}$ networks to form a complete graph. If the DCell topology is drawn using servers as vertices and logically connected edges instead of physical wiring edges, then their topology is equivalent to ours as initially described in Section 2. Since DCell is physically restricted by low degree of connectivity, it must rely on higher fractal levels in order to improve scalability. Because DCell is designed to replace existing router network architectures, its routing strategy focuses on point-to-point unicast routing. Due to the lack of automatic failure recovery within DCell, it must resort to a more complicated routing scheme for dealing with failure, requiring the use of the Dijkstra shortest path algorithm.

In contrast, the Dragonfly topology[23] is a hypothetical topology designed to minimize wiring cost. It relies on high radix routers to construct a router topology core, with terminals connecting to the core from the edges. Ignoring the router ports used for connecting the terminals, their topology core is roughly equivalent to our optimized topology, with each group within the Dragonfly topology core having one less router node than the groups within our topology. If we introduce an additional term to our $OverlaySize$ function to account for the terminals, then we can optimized the $OverlaySize$ with respect to this new term and get a closed form solution by using the cubic formula. The resulting assignment policy is as follows: $c - 2\lfloor\frac{c+1}{4}\rfloor$ ports should be used for inter group connections, $\lfloor\frac{c+1}{4}\rfloor$ ports should be used for intra groups connections, and the remaining $\lfloor\frac{c+1}{4}\rfloor$ ports should be used for connecting the terminals, where c is the radix of the routers. Therefore the "$a = 2p = 2h$" policy suggested by Kim et al. is a suboptimal policy[19]. Similar to DCell, Dragonfly focuses on point-to-point unicast routing. Although Kim et al. did not deal with routing in the face of failures, their use of the Valiant's algorithm for the purpose of load balancing is equivalent to our RON-style routing. Finally, because the two topologies are basically equivalent, our theoretic analysis is also applicable to the Dragonfly core; the bisection width and the bottleneck degree within the Dragonfly core is asymptotically the same as our topology. Hence, we expect the Dragonfly topology to have higher network capacity than other physical network topologies, including trees and DCell.

7 Conclusions

In this paper, we presented an overlay messaging infrastructure targeted at service deployment within a datacenter. Our topology offers network capacity and fault resiliency unmatched by other topologies (with the exception of the complete graph); our overlay supports incremental scaling and flexible routing. Despite the benefits offered by our overlay, it is not a drop-in replacement that can solve the scalability and communication issues of all existing services. When our overlay is used carelessly (for instance, if our overlay is used in a hierarchical master-and-slaves configuration), our overlay will perform no better than a tree-based overlay. For best results, a service build on top of our overlay infrastructure should follow a peer-to-peer model and ensure that the request workload is evenly distributed across all nodes. Routing patterns should also be customized to take full advantage of the overlay's multicasting effect.

[16] As the size of the graph increases, the local clustering coefficient approaches $\frac{8}{9}$ for our topology, whereas the local clustering coefficients often decay to zero for small-world networks[26].

[17] By bounding Pastry's diameter to three and selecting the number of connections maintained by each node, we can estimate the maximum number of nodes that can be packed into Pastry's topology.

[18] Our topology is optimal, within a constant factor of $\frac{4}{27}$, with respect to the Moore Bound[26].

[19] The optimal policy should be "$(a-1) = 2p = 2h$"

References

[1] Apache zookeeper! http://hadoop.apache.org/zookeeper/.

[2] Memcached - a distributed memory object caching system. http://memcached.org/.

[3] ADLER, M., CHAKRABARTI, S., MITZENMACHER, M., AND RASMUSSEN, L. Parallel randomized load balancing. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing* (New York, NY, USA, 1995), ACM, pp. 238–247.

[4] AGUILERA, M. K., STROM, R. E., STURMAN, D. C., ASTLEY, M., AND CHANDRA, T. D. Matching events in a content-based subscription system. In *PODC '99: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1999), ACM, pp. 53–61.

[5] ANDERSEN, D., BALAKRISHNAN, H., KAASHOEK, F., AND MORRIS, R. Resilient overlay networks. *SIGOPS Oper. Syst. Rev. 35*, 5 (2001), 131–145.

[6] BRODER, A., AND MITZENMACHER, M. Network applications of bloom filters: A survey. In *Internet Mathematics* (2002), pp. 636–646.

[7] BURROWS, M. The chubby lock service for loosely-coupled distributed systems. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation* (Berkeley, CA, USA, 2006), USENIX Association, pp. 335–350.

[8] CARZANIGA, A., AND WOLF, A. L. Forwarding in a content-based network. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2003), ACM, pp. 163–174.

[9] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2006), USENIX Association, pp. 15–15.

[10] COOPER, B. F., RAMAKRISHNAN, R., SRIVASTAVA, U., SILBERSTEIN, A., BOHANNON, P., JACOBSEN, H.-A., PUZ, N., WEAVER, D., AND YERNENI, R. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow. 1*, 2 (2008), 1277–1288.

[11] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (2008), 107–113.

[12] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev. 41*, 6 (2007), 205–220.

[13] FRIED, I. Inside one of the world's largest data centers, November 2009. http://news.cnet.com/8301-13860_3-10371840-56.html.

[14] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The google file system. *SIGOPS Oper. Syst. Rev. 37*, 5 (2003), 29–43.

[15] GREENBERG, A., HAMILTON, J., MALTZ, D. A., AND PATEL, P. The cost of a cloud: research problems in data center networks. *SIGCOMM Comput. Commun. Rev. 39*, 1 (2009), 68–73.

[16] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev. 39*, 4 (2009), 51–62.

[17] GUO, C., WU, H., TAN, K., SHI, L., ZHANG, Y., AND LU, S. Dcell: a scalable and fault-tolerant network structure for data centers. *SIGCOMM Comput. Commun. Rev. 38*, 4 (2008), 75–86.

[18] HAMILTON, J. On designing and deploying internet-scale services. In *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–12.

[19] HOELZLE, U., AND BARROSO, L. A. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[20] ISARD, M. Autopilot: automatic data center management. *SIGOPS Oper. Syst. Rev. 41*, 2 (2007), 60–67.

[21] ISARD, M., BUDIU, M., YU, Y., BIRRELL, A., AND FETTERLY, D. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (New York, NY, USA, 2007), ACM, pp. 59–72.

[22] KARGER, D., LEHMAN, E., LEIGHTON, T., PANIGRAHY, R., LEVINE, M., AND LEWIN, D. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (New York, NY, USA, 1997), ACM, pp. 654–663.

[23] KIM, J., DALLY, W. J., SCOTT, S., AND ABTS, D. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News 36*, 3 (2008), 77–88.

[24] KLEINBERG, J. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing* (New York, NY, USA, 2000), ACM, pp. 163–170.

[25] LAKSHMAN, A., AND MALIK, P. Cassandra - a decentralized structured storage system. In *LADIS '09: Proceedings of the 3rd ACM SIGOPS International Workshop on Large-Scale Distributed Systems and Middleware*.

[26] LOGUINOV, D., KUMAR, A., RAI, V., AND GANESH, S. Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2003), ACM, pp. 395–406.

[27] MACHANAVAJJHALA, A., VEE, E., GAROFALAKIS, M., AND SHANMUGASUNDARAM, J. Scalable ranked publish/subscribe. *Proc. VLDB Endow. 1*, 1 (2008), 451–462.

[28] MILLER, R. Microsoft: 300,000 servers in container farm, May 2008. http://www.datacenterknowledge.com/archives/2008/05/07/microsoft-300000-servers-in-container-farm/.

[29] MILLER, R. Google unveils its container data center, April 2009. http://www.datacenterknowledge.com/archives/2009/04/01/google-unveils-its-container-data-center/.

[30] MILLER, R. Who has the most web servers?, May 2009. http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/.

[31] MITZENMACHER, M. D. *The power of two choices in randomized load balancing*. PhD thesis, 1996. Chair-Sinclair, Alistair.

[32] RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2001), ACM, pp. 161–172.

[33] REUMANN, J. Pub/sub at google. *CANOE Workshop '09* (August 2009).

[34] RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2004), USENIX Association, pp. 10–10.

[35] ROWSTRON, A. I. T., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg* (London, UK, 2001), Springer-Verlag, pp. 329–350.

[36] ROWSTRON, A. I. T., KERMARREC, A.-M., CASTRO, M., AND DRUSCHEL, P. Scribe: The design of a large-scale event notification infrastructure. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication* (London, UK, 2001), Springer-Verlag, pp. 30–43.

[37] SNOEREN, A. C., CONLEY, K., AND GIFFORD, D. K. Mesh-based content routing using xml. *SIGOPS Oper. Syst. Rev. 35*, 5 (2001), 160–173.

[38] STOICA, I., MORRIS, R., LIBEN-NOWELL, D., KARGER, D. R., KAASHOEK, M. F., DABEK, F., AND BALAKRISHNAN, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. 11*, 1 (2003), 17–32.

[39] WELSH, M., CULLER, D., AND BREWER, E. Seda: an architecture for well-conditioned, scalable internet services. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2001), ACM, pp. 230–243.

[40] WHANG, S., BROWER, C., SHANMUGASUNDARAM, J., VASSILVITSKII, S., VEE, E., YERNENI, R., AND GARCIA-MOLINA, H. Indexing boolean expressions. In *VLDB 2009* (2009), Stanford InfoLab.

[41] YAN, T. W., AND GARCÍA-MOLINA, H. Index structures for selective dissemination of information under the boolean model. *ACM Trans. Database Syst. 19*, 2 (1994), 332–364.

[42] YANG, B., AND GARCIA-MOLINA, H. Designing a super-peer network. *Data Engineering, International Conference on 0* (2003), 49.