Hierarchical P2P Multicast

Runze Wang Tejaswi Tanikella Jake Sutton

ABSTRACT

We present a hierarchical multicast network which we believe can offer comparable performance to that achieved by Narada with significantly less information stored in each node. The three critical design goals for the hierarchical multicast network were scalability, decentralization and efficiency. The key insight behind hierarchical multicast is to exploit the inherently clustered distribution of nodes geographically around the world, and the fact that communication can be made more efficient by maximizing utilization of intra-cluster links and minimizing the use of inter-cluster links.

1 INTRODUCTION

IP multicast was first proposed and implemented in 1988 to efficiently utilize link bandwidth by relying on routers to maintain membership information and forward traffic [13]. However, the design required each of the router to maintain state for each member node, which is difficult to scale. In the modern internet, most internet providers do not support this functionality in their network to reduce the overhead on the routers and to hide the inner network topology from other operators. Application level multicast protocols have been proposed to solve the problem using multicast overlays. Our protocol is one such solution.

Any protocol that tries to build upon global information to build a multicast tree will eventually be limited by CPU or memory bounds. We have thus decided not to build using global information, but by building a sub-optimal tree using local information. An overlay on top of Chord can connect nodes in a highly scalable manner. But in large distributed networks Chord exhibits poor connection characteristics. Chord's poor performance is due to it's assumption of homogeneity of link latencies. In real networks, this assumption breaks down, causing very large latencies and inefficient packet paths.

In this paper we try to create a hybrid over Chord by introducing hierarchy to establish homogeneity in link latencies at each level. We then leverage it in a case where scalability and link clustering are inherent to the system. Finally, we evaluate it using a Azure VM cluster, and disuss our observations.

2 RELATED WORK

Multiple application level solutions have been proposed, including Narada (2002) [14]. In Narada, nodes form a mesh and share link information, in order to build a tree to multicast data. In small and mid sized networks, Narada provides an optimal solution to multicast data which is resilient to link and node failures. But, the overhead of computing the complete tree from the mesh increases as the number of nodes increases, and does not allow Narada to scale.

Hierarchical peer-to-peer networks have been evaluated previously, generally looking to leverage physical proximity information of the network to increase performance over naive Chord. One example is HPM, a scalable and efficient method to build a network of hierarchical rings, each of which is physically and logically close to each other, with support for inter- and intra-ring routing [1]. Simulations show HPM achieves comparable results to Chord. In contrast to other peer-to-peer overlay networks, physical proximity is used to build HPM. The authors are working toward using HPM to support multicast [2]. HPM is a host-based peer-to-peer network.

Other solutions have been proposed that require a degree of centralization like DCCast and QuickCast. Quickcast has the capability to efficiently transfer data between datacenters using multicast trees [16], by selectively partitioning the receivers and fairly allocating bandwidth between them. It is efficient for large numbers of receivers. In contrast to the other papers, QuickCast can be deployed on a global scale, but is only suitable for transfer between networks controlled by SDN.

3 DESIGN

In this section we describe the Hierarchical Peer to Peer Multicast protocol that implements multicast at the application level. The following were our design objectives:

(1) *scalable*: The design must be scalable and even nodes with limited computing capabilities must be able to join and function within the network.

(2) *decentralized*: The design must be completely decentralized and the forwarding must take place in a completely distributed manner.

(3) *efficienct*: The design must minimize the use of high-latency links and maximize usage of low-latency links.

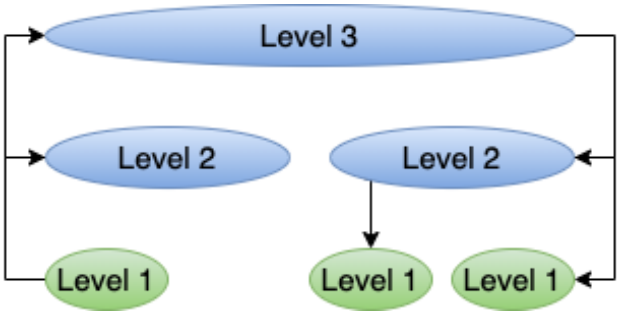

Figure 1: A hierarchical multicast. A single node in a Level 1 Cluster forwards to every node in its Level 1, Level 2, and Level 3 cluster.

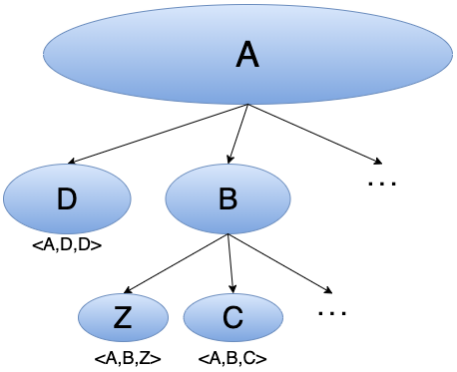

Figure 2: Hierarchy of Network. Upper layer nodes shadow lower layer nodes

Mesh networks built using Chord succeed in being scalable, and the number of nodes that can function within a network is theoretically unbounded. Chord's inability to use inter node distances while building the mesh network makes it a poor choice for a multicast overlay network. The hierarchical network described below inherits the scalable and decentralized nature from Chord, while improving upon link usage efficiency.

3.1 Network Description

Naive multicast over a Chord overlay guarantees a upper limit on the number of hops, but treats all hops as equal latency links. The path from the source node to the end point has hops with heterogeneous latencies, resulting in poor end point latencies. The hierarchical network solves this by moving the smaller latency hops into lower levels and restoring homogeneity in the upper levels.

We divide the network into hierarchies, based on thresholds tailored to the network. These thresholds dictate the choice of joining a particular cluster to declare creation of a new cluster. Each level of the hierarchy is made up of clusters, which represent a set of nodes which are have inter-node latencies within that level's assigned threshold. At each level inter cluster latencies are greater than the threshold declared at that level. Each node has a separate set of finger tables associated with each level. Nodes in a cluster share the same higher level finger tables. Each node has a node ID made up of smaller IDs, one for each level it is a part of and is assigned as it would be in a Chord network. Each node uses the level's finger tables to forward packets. The levels can be visualized as separate Chord rings functioning independently and connected by common nodes.

To build and maintain the network, we use the Join and GetFingerEntry RPC calls. We cluster using inter-node latencies measured using the Join call and use the information to join an existing cluster or to create a new cluster. The GetFingerEntry RPC is used to fill the new node's finger tables.

3.1.1 Hierarchical Node ID.

Each Node is assigned an ID, which consists of multiple hash values for Chord rings at each level it is a part of. The hash value is generated like Chord, making sure to uniformly distribute all the nodes in the ring space. When a node joins a cluster, it copies the cluster's node ID up to the current level. i.e. if a node Z joins a cluster {A,B,C} at level 2, it's new ID would become {A,B,Z} as shown in Figure2. This ensures that nodes within a cluster share the hash values at all upper levels, hence appear as the same node at higher levels.

3.1.2 Join the Network.

A new node joins a network by sending out a Join Request to a node. The RPC call includes the level the new node wants to join and timestamp data, so both nodes can calculate the communication latency. This information is used by the new node to decide if it should join the cluster. The RPC response also contains the finger table of the responding node so the new node can move on to finding a cluster at lower levels or can begin filling it's finger tables. The node sends out join requests to multiple levels trying to find it's clusters. Each time it successfully joins a cluster, it updates its Node ID information, continues filling its finger tables at that level and sends out join requests repeating the process to find clusters at all levels. If it fails to find a cluster, i.e. all the clusters are greater than the threshold values, it creates its own singleton cluster and adds itself to the ring at the current level.

3.1.3 Filling Finger Tables.

Each node begins with the knowledge of a single known node to which it sends out the initial request, and then continues it's traversal, finally finding its position in the network. Since

```
JOIN(hierarchy, known_node):
    //add rootNode to known nodes.

    FOR each known_node:
        send a JoinReq.
        receive a JoinResp.
        add JoinResp.FingerEntries
                        to known_nodes.

        IF RTT < threshold:
            join the cluster
            JOIN(hierarchy -1, known_node)

        ELSE:
            add_to_finger_table(known node)

    IF unable to join any cluster:
        create_new_cluster()
        //stop searching for clusters
```

Figure 3: Psuedo-code for Node Join

each ring it joins is an independent Chord ring, the node fills its finger tables and maintains it's successors as implemented by the Chord protocol.

In the same way that Chord finger tables maintain nodes in a ring, the finger table contains "subclusters" in the ring. For better fault tolerance, nodes can use an additional RPC to fetch a node address from the subcluster. This makes sure that nodes have different addresses pointing to the same cluster.

3.1.4 Forwarding.

The forward RPC, along with the payload also transmits a limit and sender Node Id, which is used to prevent forwarding loops within the hierarchical network. The payload is forwarded within the Hierarchical Network with the following rules:

(1) *Inter-hierarchy Forwarding:* The payload must never be forwarded to higher levels than on the one it received the packet on. If the payload is received from a peer at particular level, the payload should only forwarded within the level and to lower levels that the node is a part of.

(2) *Intra-hierarchy Forwarding:* The payload is forwarded to all nodes in the current level within the limit. The limit is set by partitioning the ring into sections using the Finger Table entries as shown in Figure 5.

The forwarding within a Chord ring using limits is explained in Figures 4, 5, and 6.

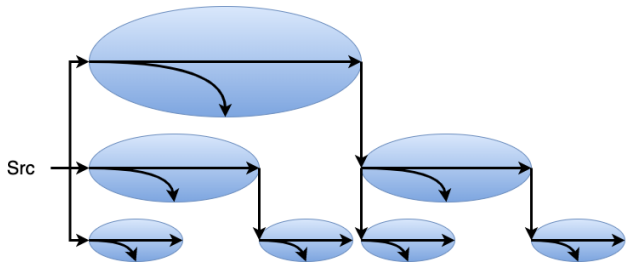

Figure 4: Forwarding in the hierarchy with limitation to the destination of each packet

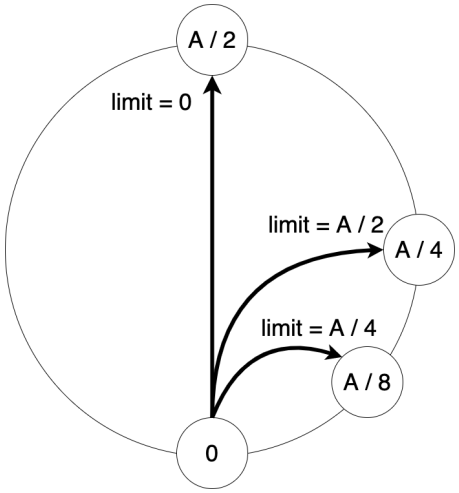

Figure 5: Forwarding within a Chord Ring. Each node can forward the payload to all Nodes in the ring within $log(N)$ steps, using the property that Chord's finger tables provide.

```
Forward(packet, hierarchy, limit):
    // Receive a packet
    if packet received from higher level:
        limit = selfID.

    for each entry in Finger Table within limit:
        Forward(packet, hierarchy,
                min(limit,next entry))
        // forward to finger entries
```

Figure 6: Psuedo-code for Forwarding

The above rules build a tree along which the payload is forwarded, which minimizes the use of high-latency links and maximizes usage of low-latency links.

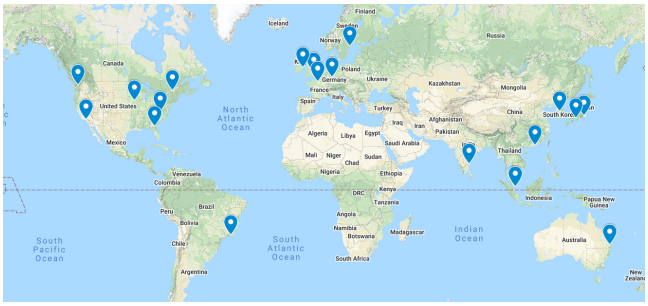

Figure 7: Azure Servers distributed across the globe. From visual analysis, the servers are geographically clustered, and we observe that this is reflected in latency measurements. Hierarchical multicast exploits clustering to build an efficient network.

4 INTER-DATACENTER MULTICAST

It's scale and inherent clustering, makes inter-datacenter multicast an interesting application for hierarchical multicast . Firstly, multicast among VMs across data centers, needs a solution that can scale to very large numbers while keeping low computation overhead. Protocols like Narada, which compute the multicast tree at each end point, cannot scale well in such large environments. Any node failure requires each end point to recompute the entire tree, which becomes prohibitive. Secondly, the VMs inherently have a very clustered network at a global scale, which makes creating a clustered network a good way to interconnect them.

To minimize the number of intercontinental links used in forwarding, we build a hierarchical structure over the existing topology to cluster nodes within the same continent. For any intercontinental communication the upper layer (contains nodes from different continents) will provide an efficient route. And after the packet is in the target continent, lower layer nodes will take over to distribute the packet to all destinations using low latency links.

5 EVALUATION

Our test bed consists of 20 Azure VMs, each in a different regions provided by Microsoft Azure. We compare the performance of two Hierarchical Networks with Narada and Chord. Since Narada builds a tree with the complete mesh information, it's solution is optimal in terms of latencies and link usage. Chord with no latency information forms a ring network with suboptimal link usage. The Hierarchical network's performance is expected to be lower than Narada but better than Chord.

Separate programs for Narada and Hierarchical Multicasting were built using Golang. Data was received and forwarded using RPC, which was built by using gRPC. Concurrent processing is built into Go, making it an ideal language for networking applications. The code was built to exploit concurrency while forwarding data. The main thread controls the core data structures and spawns threads to forward data to other hosts.

Using the latency measurements, we produced the network topology for each of the networks and recorded them as a configuration file. Each node reads the network topology from a configuration file and populates it's data structures. One of the nodes, designated as the initiator, starts multicasting data. For the evaluation, we have sent a small payload of 400 Bytes, every 5 Seconds. Since the RPC framework was built on TCP, we were not able to evaluate the performance of a continuous data stream. Pushing large payloads caused large delays, since gRPC waits for the complete message before calling the registered RPC receive function. These issues were identified, but were not modified due to the lack of time. Re-implementing RPC using a different framework or RPC over UDP might fix the issue, which we note as an additional task. The Logger was also built using Go, with the same principles to exploit concurrency.

We evaluate four networks, described below, in increasing order of expected performance.

(1) Naive Chord. (a one layer Hierarchical Network)

(2) Two layer Hierarchical Network, a network built with two layers of hierarchical clustering.

(3) Three Layer Hierarchical Network, a network built with three layers of hierarchical clustering.

(4) Narada(a minimum spanning tree)

To gather information for evaluating these networks, additional data is added to the packet payload which records the path the packet takes. On receiving a packet, each node adds the current timestamp and it's identifier and sends the data to a central logger. The logs gathered by the central logger have been used for the following evaluation.

5.1 Latency

The first metric is latency for the packet to reach end nodes. The latencies have been presented in the figures. We observe, as expected Naive Chord performs very poorly, with maximum latency reaching 2.5seconds. The mean latency of the Hierarchical Networks are comparable to that of Narada. It is important to note that the maximum latency value is in fact less than that of Narada. The reason for which is use of long links for the first hop by the Hierarchical Networks compared to Narada, which delays it. This will be discussed later in more detail.

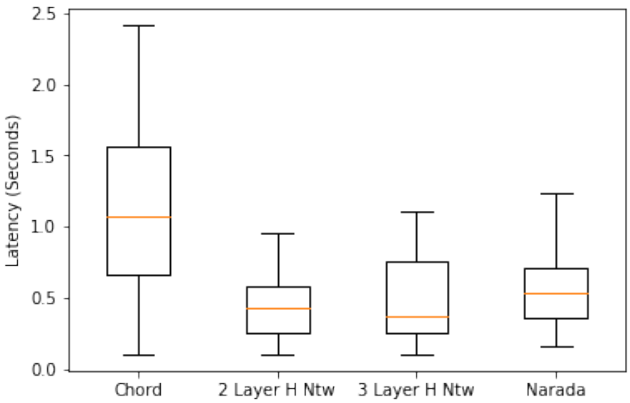

Figure 8: Latencies measured for all the networks.

Narada's CDF shows more than 90% of the node latencies are within 0.9 seconds, but a few outliers receive packets after 1.2 seconds. This re-establishes our earlier observation that Narada's mean is higher than both the Hierarchical networks.

5.2 Hop Quality

The Second evaluation metric is the quality of single link hops chosen to reach the endpoints. Narada ensures that long links are used less often and shorter links used more often. Similarly the hierarchical network reduces inter-cluster communication (the longer links). Based on analysis of our logs, the following histograms were plotted which present inter node single hop link latencies.

The Link Usage Histograms (Figure 10) present the latency of one hop links and demonstrate that the Hierarchical Networks and Narada use links similarly. All the algorithms deliver data in 19 hops, but the choice of the links causes difference in latencies.

5.3 Network Stretch

Network stretch, defined as ratio of Multicast latency to the average latency on the IP network [15], allows us to evaluate the quality of the path chosen during multicast. Our measurements indicate both the hierarchical clustering networks provide lower stretch than Narada. Chord, as expected, fares poorly with the maximum stretch reaching 8, maximum stretch for Narada was 3.5, while the maximum in case of the hierarchical networks was 2. The reasons for lower stretch in the hierarchical networks is discussed Section 6.

6 DISCUSSION

In this section we summarize the comparison of hierarchical multicast, Chord, and Narada, and compare all of them to unicast performance.

- *Latency* - We have shown that hierarchical multicast achieves latencies significantly better than Naive Chord and comparable to Narada on our datacenter topology.
- *Hop Quality* - We count the link usage over high latency and low latency links. The result is that our architecture has performance comparable to Narada.
- *Stretch* - Both the hierarchical network and Narada show comparable Stretch values. But the maximum stretch of the hierarchical network is lower than that of Narada due to way Narada builds its tree. This has been further discussed below.

The reason for the hierarchical network's slightly better performance in the latency and stretch metrics is the greater accessibility by all nodes to longer links. In our observation Narada's tree had a few heavily connected nodes (Canada and Japan), and the rest had less than 2 links. Hence the packet has to reach the heavily connected node before it can be forwarded over the long links. This moves the payload away from the end point before it can be sent in the right direction. The hierarchical network on the other hand lets the multicast initiator to send packets over long links in the first hop.

For example in Narada, the packet from a node on the west coast of the US (henceforth US-West1) reached a node in Australia (henceforth Aust) by travelling through Canada and Japan. In the Hierarchical Network with two layers, US-West1 forwarded it to Aust thorough Japan. This occurs because Australia and Japan are within the same cluster, to which US-West1 forwards. In the Hierarchical Network with three layers, Aust forms a singleton cluster at the highest layer. Hence payload is directly sent from US-West1 to Aust at the highest level. This demonstrates the clear benefit of the greater accessibility by all nodes to longer links.

For stretch analysis, Figure 11 shows that there are bins with stretch less than 1. We believe the reason we found stretch values less than one is that we collected statistics on different days and at different times of day, and latencies may have changed over time. Note that all stretch values less than one correspond to single hop links. While the stretch values are not derived from latencies taken at the same instant, they can be used to compare trends.

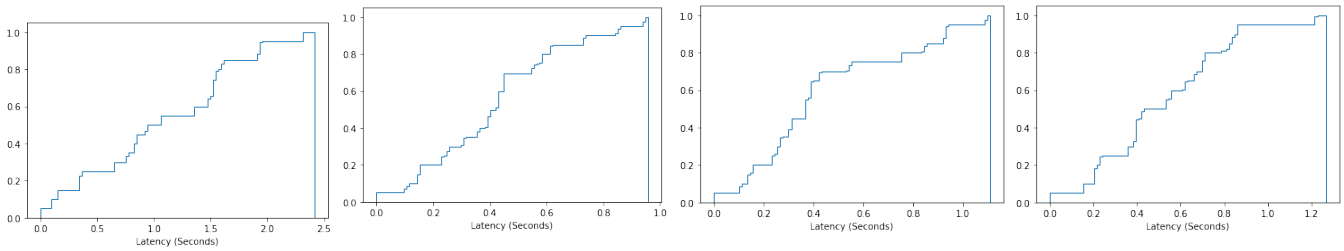

Figure 9: CDF of latency using (from left to right) Chord, the Three-Layer Hierarchical Network, the Two-Layer Hierarchical Network and Narada.

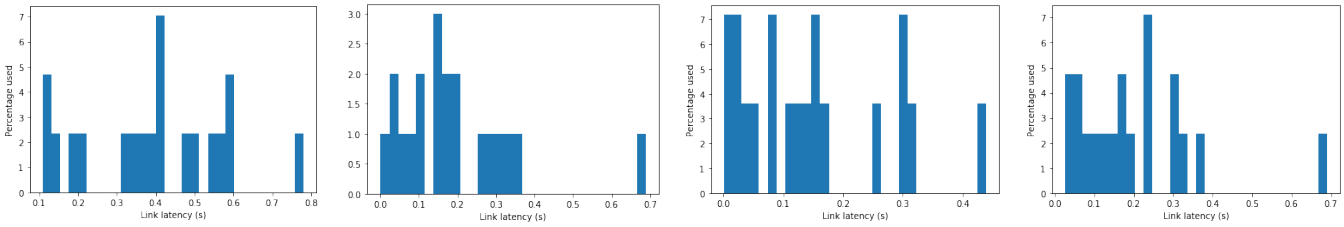

Figure 10: Link Usage in (from left to right) Chord, the Two-Layer Hierarchical Network, the Three-Layer Hierarchical Network, and Narada.

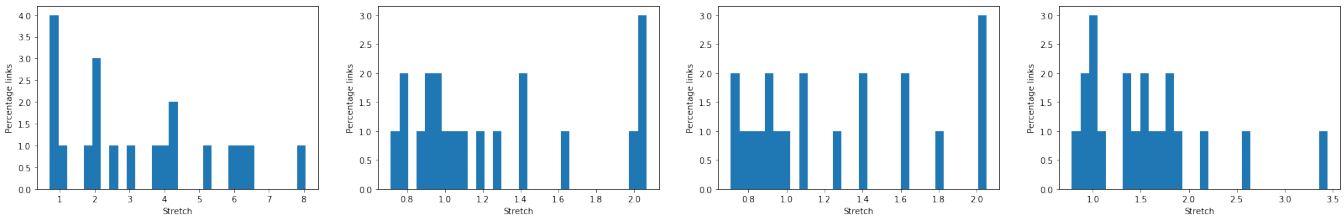

Figure 11: Stretch in (from left to right) Chord, the Two-Layer Hierarchical Network, the Three-Layer Hierarchical Network, and Narada.

7 CONCLUSION

This paper's main contribution has been to measure the effectiveness of an hierarchical network which inherits Chord's scalability while allowing link inequalities.

Based on the evaluation so far, the solution does provide multicast services comparable to Narada. We have identified two areas where further work is necessary. Firstly, in addition to multicasting, Narada can also handle random link/node failures. Fault tolerance is not been evaluated in this paper.

Secondly, choosing thresholds with mathematical reasoning needs to be explored. By creating a heuristic that produces a near optimal set of thresholds for a given network, we will be able to guarantee certain properties of the network and better suit the needs of the applications.

REFERENCES

[1] Amad, Mourad, et al. "HPM: A Novel Hierarchical Peer-to-Peer Model for Lookup Acceleration with Provision of Physical Proximity." Journal of Network and Computer Applications, Academic Press, 13 July 2012, www.sciencedirect.com/science/article/pii/S1084804512001580.

[2] Amad, Mourad, et al. "Application Layer Multicast Based Services on Hierarchical Peer to Peer Architecture." Applied Mechanics and Materials, vol. 892, Trans Tech Publications, Ltd., June 2019, pp. 64–71. Crossref

[3] Dangelo, Mirko, and Mauro Caporuscio. "SA-Chord: A Self-Adaptive P2P Overlay Network." 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), 2018, doi:10.1109/fas-w.2018.00035.

[4] Lakshminarayanan, Karthik, et al. “End-Host Controlled Multicast Routing.” Computer Networks, vol. 50, no. 6, 2006, pp. 807–825., doi:10.1016/j.comnet.2005.07.019.

[5] Livadas, C., “An Evaluation of Three Application-Layer-Multicast Protocols’, Sep. 25, 2002, Laboratory for Computer Science, MIT.

[6] L. Dai, Y. Cui and Y. Xue, "On Scalability of Proximity-Aware Peer-to-Peer Streaming," IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications, Barcelona, 2007, pp. 2561-2565.

[7] Jianjun Zhang, Ling Liu, C. Pu and M. Ammar, "Reliable peer-to-peer end system multicasting through replication," Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings., Zurich, Switzerland, 2004, pp. 235-242.

[8]Maymounkov P., Mazières D. (2002) "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." In: Druschel P., Kaashoek F., Rowstron A. (eds) Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg

[9] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. 2002. "Scalable application layer multicast." In Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM ’02). Association for Computing Machinery, New York, NY, USA, 205–217.

[10] Rao, Ananth and Lakshminarayanan, Karthik and Stoica, Ion and Shenker, Scott. "Flexible and Robust Large Scale Multicast Using i3" EECS Department, University of California, Berkeley, 2002.

[11] D. Chen, Z. Tan, G. Chang and X. Wang, "An Improvement to the Chord-Based P2P Routing Algorithm," 2009 Fifth International Conference on Semantics, Knowledge and Grid, Zhuhai, 2009, pp. 266-269.

[12] F. Chao, H. Zhang, X. Du and C. Zhang, "Improvement of Structured P2P Routing Algorithm Based on NN-Chord," 2011 7th International Conference on Wireless Communications, Networking and Mobile Computing, Wuhan, 2011, pp. 1-5.

[13] S. E. Deering. 1988. Multicast routing in internetworks and extended LANs. In Symposium proceedings on Communications architectures and protocols (SIGCOMM ’88). Association for Computing Machinery, New York, NY, USA, 55–64.

[14] Yang-hua Chu, S. G. Rao, S. Seshan and Hui Zhang, "A case for end system multicast," in IEEE Journal on Selected Areas in Communications, vol. 20, no. 8, pp. 1456-1471, Oct. 2002.

[15] S. P. Ratnasamy, P. Francis, M. Handley, R. Karp, andS. Shenker, “A Scalable Content-Addressable Network”.InProc. of ACM Special Interest Group on Data Commu-nication (ACM SIGCOMM ’01),pages 161–172, Aug.2001.

[16] M. Noormohammadpour, C. S. Raghavendra, S. Kandula and S. Rao, "QuickCast: Fast and Efficient Inter-Datacenter Transfers Using Forwarding Tree Cohorts," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 225-233.

[17] Region to Region Latency. https://www.azurespeed.com/Azure/RegionToRegionLatency.