Routing Tradeoffs inside a d-dimensional Torus with applicability to CAN

MINA GUIRGUIS AZER BESTAVROS IBRAHIM MATTA
{msg, best, matta}@cs.bu.edu
Computer Science Department
Boston University
Boston, MA 02215, USA

***Abstract*— Overlay networks have evolved as powerful systems enabling the development of new applications ranging from simple file sharing applications to more complex applications for managing Internet traffic. Content Addressable Network (CAN) [1] is one such network where nodes (peers) are organized in a d-dimensional torus. Nodes maintain state for their immediate neighbors and a request is routed inside the network through the neighbor that is closest to the destination. This process is repeated until the request reaches its destination. In this paper, we consider routing tradeoffs between space and time; Space in terms of state maintained at each node and time in terms of the average path length experienced as requests get routed inside the network. Our findings motivate the importance for nodes to maintain state, not just for their immediate neighbors, but also for a few Long Range Nodes (LRNs). These LRNs will allow longer jumps inside the space, reducing the average path length. We evaluate the effect of having these long jumps through comparing different setups that store the same amount of state. Based on this, we propose a new dynamical scheme where nodes update their LRNs in order to adapt to the nature of requests. This has significant implication when some nodes become popular in hot-spot zones. We validate our findings through simulations.**

***Index Terms*— Peer-to-Peer, Overlay Networks, Grid Computing and CAN.**

I. INTRODUCTION

Over the past few years, overlay networks have received a lot of attention in the networking research community and many new applications were made possible through these powerful networks. Recent studies [2], [3] show that the majority of Internet traffic is attributed to file sharing applications such as [4], [5], [6], [7], and many others. In general, overlay networks can be classified into structured versus unstructured networks. In unstructured networks, such as Gnutella [4] and KaZaA [5], a request for a particular object is flooded through the network until there is a hit otherwise the request fails.[1] This, in return, raises an issue of scalability in addition to the generation of a lot of traffic [8]. In [9], the authors estimate the traffic generated by Gnutella's search mechanism to be 330 TB/month, based on 50,000 nodes where 95% of any pair of nodes are less than 7 hops away from each other. Of course, the above limitations have prompted research in the area of structured overlay networks where scalability and robustness can be achieved through the use of Distributed Hash Tables (DHTs). CAN [1], Chord [10], Pastry [11] and Tapestry [12], for example, rely on the presence of DHTs to locate content and thus they provide scalable guarantees on the number of hops it takes to answer a query.

Our main goal in this work is to explore the tradeoffs between space (state maintained at each node) and time (average path length) when routing is performed inside a d-dimensional torus. This has significant impact on CAN [1] like structures where nodes maintain state for their immediate neighbors. Our results suggest that, with the same amount of state (or even less), a careful selection of neighboring nodes, can reduce the average path length experienced by the requests as they get routed inside the network. We explain this below.
When nodes route requests inside the network, each node selects the next hop that is closest to the destination and forwards the request to such a node. This process is repeated until the request reaches the destination. In this paper, we propose different ways for decreasing the logical routes inside the network. This is achieved by allowing the nodes to store state for few Long Range Nodes (LRNs). The motivation of having LRNs is to allow for long jumps inside the network. We are further motivated by the cheap local computation of the next hop, compared to the time taken to send a request to the next hop.

There is work that studied the benefit of long-range contacts inside a network. This work started by a paper in social sciences by Stanley Milgram in 1967 [13], where he found that individuals using local information are collectively very effective at actually constructing short paths between two points in a social network. Kleinberg in [14] modeled "The Small World phenomenon", showing that having correlation between local structure and long-range contacts provides fundamental clues for finding short paths inside the network. He showed that when long-range contacts are chosen by a process that is related to the zone coordinates in a specific way, then long jumps will be useful and nodes would be able to use them. He proved this for a decentralized algorithm capable of finding short paths with high probability. This

[1] Flooding is usually limited to several hops. In Gnutella [4], for example, the Time To Live (TTL) field in a request packet is set to 7 and decremented by each node before the packet gets discarded.

decentralized algorithm provides progressive closing-in on the destination as each new node routes a given request. If a network does not possess this decentralized algorithm, long jumps may exist but the nodes, operating at the local level only, may not be able to find them. In CAN, as nodes route requests, there is this ”closing-in to the destination” property that was described by Kleinberg, hence, having long range jumps would be beneficial, as we show in this paper.

Paper Outline: The rest of the paper is organized as follows: Section II summarizes the CAN architecture outlining the preliminaries for routing in a d-dimensional torus. Section III describes the Static Model, where each node stores few LRNs and keeps them fixed at all times. We show different alternatives on how we can choose a single LRN and what is its marginal utility. Section IV describes a Dynamic Model, where the LRNs are chosen and replaced as nodes route requests. We show that the average path length could be significantly decreased when hot-spot phenomena occur. In Section V, we present relevant related work. We derive our conclusions along with future work in Section VI.

II. Preliminaries

In this section, we summarize the CAN architecture and we refer the reader to [1] for a detailed description.

CAN [1] is a distributed system that divides a virtual *d*-dimensional Cartesian coordinate space among *n* individual nodes (peers), where each node stores a region of the hash table in the form of (key,value) pairs. To store or retrieve a pair $(k1, v1)$, key $k1$ is mapped onto a point p in the coordinate space using a deterministic uniform hash function. Then the node who owns the zone where p lies, will be the one responsible for the pair $(k1, v1)$. Requests for a particular key are routed by intermediate nodes towards the CAN node whose zone contains that key. Each CAN node will only maintain state for its $2d$ neighbors (2 nodes along each dimension). *The higher the value of d, the shorter is the path between two nodes but with correspondingly more information state needed to be maintained at each node.* This is the fundamental trade-off that we will consider in this paper. It is important to note that these paths are logical in the sense that they don't reflect anything about the underlying Internet (IP) topology, since a single logical CAN hop could mean several physical IP network hops.

The n nodes can be organized in different ways based on our choice of the dimension d. In particular, $n = m^d$ where m is the number of nodes that can be projected per dimension (we refer to m as the base). Since the torus wraps around, no two nodes can be more than $\frac{m}{2}$ hops away from each other, per dimension. Thus, the maximum distance between any two nodes is $\frac{m.d}{2}$. Figure 1 illustrates how all the nodes are distributed as a function of the distance from a given source node.

The top level in Figure 1 represents any source node at $level = 0$. The next level represents the set of nodes the can be reached in one hop from the source (immediate neighbors). The third level, represents the set of nodes that can be reached in 2 hops and so on. The maximum number of nodes that can reached are at distance $d.(\frac{n^{\frac{1}{d}}}{4})$ which is in the middle level. Then the number of nodes per set that can be reached will start to decrease as we increase the number of jumps since the torus will start to wrap around. Figure 1 is important, because the distribution is not symmetric in the sense that if a node i chooses a node j uniformly at random, it is more likely to reach it at a distance $d.(\frac{n^{\frac{1}{d}}}{4})$ away from it, because there is larger number of nodes at that level. We will see how this fact is used to drive our intuition about the results later on.

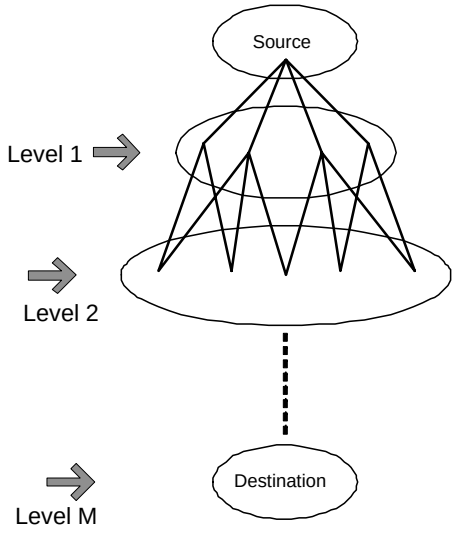

Fig. 1. Distribution of nodes at different levels.

An illustrative Example: Let's choose the number of nodes equal to 625 and we will organize them in a 4 dimensional space with a base of 5, as $5^4 = 625$. Assume we start at Node A [2,1,3,0] and we want to route a request to Node B [3,2,1,4]. Since A is only aware of its immediate neighbors, it routes the request to one of its neighbors along the direction to the destination. So in this case it routes to Node [3,1,3,0]. One possible route would be starting at A [2,1,3,0] to [3,1,3,0] to [3,2,3,0] to [3,2,2,0] to [3,2,1,0] to B [3,2,1,4] for a total of 5 hops. [2]

III. A Static Model

In this section, we slightly change the CAN architecture by allowing each node, in addition to maintaining state for its immediate $2d$ neighbors, to maintain state for a fixed number of Long Range Nodes (LRNs) denoted by k, for a total of $2d + k$ nodes. These are the nodes that can be reached in a single hop. In this section, we only consider the case where the k LRNs will be chosen independent from the requests being generated and will be kept unchanged during the lifetime of the

[2]Notice that there are other possible routes between node A and Node B. These can be utilized when nodes go down.

nodes.[3] The routing algorithm is similar to the one described in the original CAN paper [1]; when a node receives a request and it is not the destination, it computes the distance between the destination and every node of its $2d+k$ nodes for which it maintains state. Then it selects the node that is as close to the destination as possible in terms of the Cartesian distance. This process is repeated until the request reaches its destination.

A. Impact of Long Range Nodes

To demonstrate the impact of having LRNs on the average path length, we wrote a simulator for the CAN architecture. Figure 2 illustrates the average path length as a function of k . These k LRNs are chosen uniformly at random and the average path length is computed over 100,000 requests generated uniformly at random. Increasing the number of requests beyond 100,000 does not change the average path length significantly. The curves in Figure 2 correspond to two setups; one is for 279,936 nodes with $d = 7$ and $m = 6$; and the other is for 390,625 nodes with $d = 8$ and $m = 5$.
When k is chosen to be zero, this would be the base case where nodes do not maintain state for LRNs. Clearly, the higher the value of k, the shorter is the average path length between any two nodes, because it is more likely for a request to utilize one or more LRN while being routed.

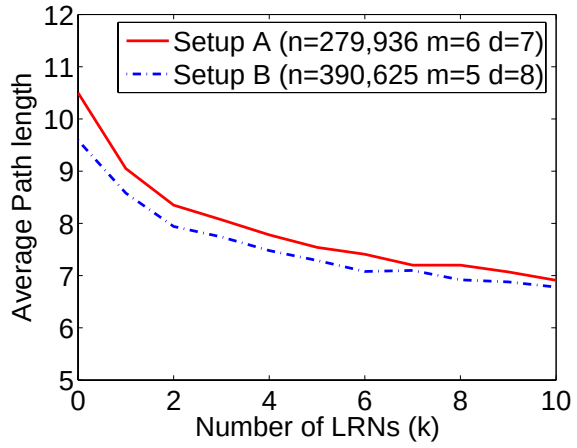

Fig. 2. Average path length as a function of additional Long Range Nodes. Setup A corresponds to 279,936 nodes with $d = 7$ and $m = 6$. Setup B corresponds to 390,625 nodes with $d = 8$ and $m = 5$.

As illustrated in Figure 2, for as small values for k as 2, we can get up to 20% shorter path length. Then the improvement increases but at a slower pace. In other words, the significance of Figure 2 lies in the *marginal utility for the addition of a single LRN*. The presence of the first few LRNs (one or two nodes) have the most profound impact on reducing the average path length. Beyond which, the diminishing return effect kicks in. The locations of the LRN(s) also play an important role on the average path length. Indeed, a LRN that allows long jumps in a zone to which no other LRN is pointing, is favorable. A good choice of a LRN is the one that provides reachability to a *fresh* zone. As the number of LRNs increases, such zones will start to overlap reducing the marginal gain. In order to be precise when assessing the impact of LRNs, we need to compare two setups that require to maintain *the same amount of state*. So far, nodes have been maintaining more state. Next, we look at the marginal utility of a single LRN based on its location.

[3]In case of a node going off-line, another node will replace it as described in [1].

B. Marginal Utility for a single LRN

In this part, we turn our attention to how to choose a single LRN. In particular, we explore the trade-offs between time and space for different schemes for the choice of a single LRN.

Scheme I (Random): A simple scheme is to choose the LRN uniformly at random.

Scheme II (Max-Distance): Our second alternative is to choose the LRN to be as far as possible from the selecting node. Ties between nodes that are at the same maximum distance will be broken arbitrarily. Since two nodes can not be more than $\frac{m.d}{2}$ hops away from each other, we choose the LRN to be at that exact maximum distance from the selecting node.

Scheme III (Hybrid-Distance): The above scheme would not allow a particular request to benefit from multiple long jumps as it gets routed inside the network since all the jumps are of the same maximum distance. In this scheme, we allow each node to choose its LRN with different number of hops. The main goal of this approach is allow a given request to use multiple longer jumps as it gets routed.

	Setup 1	Setup 2
Base (m)	4	5
Dimension (d)	7	6
Number of Nodes (n)	16384	15625
Base Case k=0	6.9827	7.1911
Scheme I (Random) $k = 1$	6.1243	6.3498
Scheme II (Max-Distance) $k = 1$	5.9069	6.2489
Scheme III (Hybrid-Distance) $k = 1$	6.2369	6.5604

TABLE I

AVERAGE PATH LENGTH FOR TWO DIFFERENT SETUPS WITH DIFFERENT CHOICES FOR A SINGLE LRN. THE DATA IS COMPUTED BASED ON 100,000 REQUESTS

Table I depicts the average path length according to the different schemes for the choice of a single LRN. The base case, without the presence of LRNs, is provided here for comparison.

Clearly, all schemes reduce the average path length compared to the base case. However, the improvement is more pronounced when each node selects its LRN to be at the maximum distance from itself (i.e., scheme II). The improvement is about 15%. This choice insures that with a single jump, the node can get to a completely fresh zone because we are sort of dividing the space into two parts, and assigning a LRN inside the further half. Schemes I and III

perform almost the same because they both use short jumps as well as longer ones, however, the short ones do not have a strong impact on reducing the average path length.

An important result to note is the benefit of a single LRN, which is demonstrated when one compares the base case for Setup 1 against using scheme II in Setup 2. These two setups have almost the same number of nodes, so it is fair to compare them. For a node in Setup 1, it stores state for 14 nodes and the average path length observed by requests is 6.9. However, for a node in Setup 2, it stores state for 13 nodes (2×6 neighbors + 1 LRN) and requests observe an average path length of 6.2 with scheme II. So by having nodes store state for a single LRN, we achieve *shorter average path length* with *less state being maintained at each node*. This fact can be leveraged in CAN [1] where instead of moving to a higher dimension to reduce the average path length, it is better to store few LRNs (one or two) and maintain the same (or less) amount of state.

IV. A Dynamic Model

In the previous section, we assumed that requests are generated uniformly at random across the whole CAN topology. In practice, hot-spot phenomena do occur when some particular nodes become extremely popular. When such phenomena occur, it is quite beneficial for other nodes to reach such popular zones as quickly as possible through the minimum number of hops. This fact drives our idea that we would allow nodes to *dynamically adapt* their LRNs to the nature of the requests. As the number of requests destined to the popular node increases, the location of the LRN becomes more important. The static model, ignores this case so if a node fails to have its LRN near that popular node, such LRN will not be beneficial and it is better for the node to replace it.

In this section, we consider the impact of maintaining a single LRN, which will be replaced dynamically as nodes route requests. Given a request for a node to route, the node will examine the distance between its LRN and the destination, if such a distance is bigger than a certain threshold called **Update Threshold**, the node would replace its LRN by another node that is at half the distance between its old LRN and the destination. So if we have a popular node, as more requests arrive to this node, its LRN will start to point closer and closer to the hot-spot, until its LRN will be the popular node itself[4]. It is important to note that this convergence depends on two parameters; the update threshold and the probability that a request will be destined to the hot-spot node. We denote this latter probability by p. To avoid frequent updating of the LRN on each request, a node can simply toss a coin and with some probability it updates its LRN and with the remaining probability it keeps it as is.

We generated a single hot-spot phenomenon and we ran several experiments with different update thresholds and with different probabilities p. With probability p the requests are destined to the hot-spot node and with the remaining probability they are destined to another node at random. Figure 3 illustrates the average path length for different update thresholds as the probabilities for requests to be destined to the single hot-spot node changes. The setup is for 15,625 nodes with m chosen to be 5 and d is equal to 6. This setup has an average path length of 7.1 when k=0 and an average path length of 6.2 when k=1. It is clear that as the probability increases, the average path length decreases almost linearly. Having small update threshold reduces the path length for a fixed probability value, but the higher is the overhead for updating the LRN. When 50% of all the requests are destined to a single hot-spot node and the update threshold is set to 1, the average path length is 3.5. A 50% improvement compared to the base case when k is equal to 0 and 43% improvement when k is chosen to be 1 according to scheme II. As the probability p increases the improvement increases and with p = 0.9, the improvement is 78% and 75%, respectively.

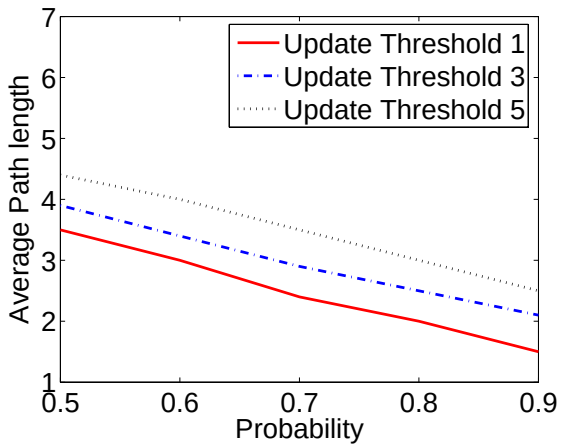

Fig. 3. Average path length for a particular node in a system with 15,625 nodes with base = 5 and dimension = 6. This system had average path length of 7.1 when k=0 and average path length of 6.2 when k=1. The data is based on the generation of 100,000 requests.

The above results are limited to a single hot-spot node. In the case of multiple hot-spot nodes, each node can maintain a single LRN per each hot-spot node. This ensures that popular requests will always utilize shorter average path lengths through the appropriate LRN, in addition to reducing the total traffic through the network.

V. Related Work

There is a lot of work that looked at improving routing in overlay networks as well as reducing traffic generated by search mechanisms in file sharing applications. This work can be classified based on the overlay structures, being present or not. In unstructured overlays, introducing hierarchical mechanisms indeed improves the search mechanisms. In [5], some nodes act as super-nodes and requests are only flooded between these super-nodes. Since each super-node maintains state for nodes directly connected to it and will be able to answer queries on their behalf, the average path length can

[4]Notice that this algorithm does not point immediately to the hotspot, because in case of multiple hotspots, we would like to have the LRN at an equal distance between them.

be reduced. Despite the efforts in reducing the average path length, unstructured networks do not scale very well [8]. This has prompted research in the area of structured overlays as in [1], [10], [11], [12]. In structured overlays, and specifically in CAN [1], the authors discuss how nodes can carefully choose their neighbors that are physically close at the IP layer. This can be achieved by the idea of land-marking where nodes measure their RTTs to different landmarks and order their list and when they join the CAN, they only join in that portion of the coordinate space associated with this landmark ordering. This method reduces the physical delay per CAN hop. In this paper, we were interested in reducing the number of logical hops. This method can be combined with our approach to expedite the search mechanism. On the other hand, hot-spot phenomena occur when one or more nodes become popular due to the keys they are responsible for. Research has addressed hot-spots through content replication and caching [15], [16], [17]. This indeed alleviates the contention on the nodes involved. In this paper, we addressed how to dynamically choose the LRN in order to reduce the average path length. Our technique can be used in conjunction with content replication and caching.

VI. Conclusions and Future Work

In this paper, we have shown how routing can be improved in CAN like systems. Maintaining state for Long Range Nodes allow long jumps inside the space. We have demonstrated that the average path length can decrease with *less* state being maintained at each node. The key idea is to let each node have its LRN point to a fresh zone in the torus. This fact has significant implication on CAN [1] where instead of moving to a higher dimension to reduce the average path length, it is more efficient to maintain state for one or two LRNs. We proposed a dynamic model for updating the LRNs. The key is to adapt to the nature of requests which improves routing significantly when hot-spot phenomena do occur. As for future work, we would like to choose the LRNs, not just based on CAN distances but also based on the underlying IP topology.

Acknowledgment: We thank John Byers for his comments and fruitful discussions on this work.

References

[1] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of ACM SIGCOMM*, 2001.

[2] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Internet Measurement Workshop*, 2002.

[3] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble and H. Levy, "An Analysis of Internet Content Delivery Systems," in *Proceedings of OSDI*, 2002.

[4] "Gnutella," http://gnutella.wego.com/.

[5] Sharman Networks Ltd, "KaZaA Media Desktop 2001," http://www.kazaa.com.

[6] "Morpheus," http://www.morpheus.com/.

[7] "Limewire," http://www.limewire.com/.

[8] J. Ritter, "Why Gnutella Can't Scale. No, Really," *2001*.

[9] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.

[10] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," in *Proceedings of ACM SIGCOMM*, 2001.

[11] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[12] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," *Computer Science Division, U. C. Berkeley- Tech. Rep.*, 2001.

[13] S. Milgram, "The small world problem," in *Psychology Today 1,61*, 1967.

[14] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," in *Proceedings of The Thirty-second Annual ACM Symposium on Theory of Computing*, May 1999.

[15] D. Rubenstein A. Stavrou and S. Sahu, "A Lightweight, Robust P2P System to Handle Flash Crowds," in *Proceedings of the IEEE ICNP*, 2002.

[16] P. Maniatis T. Stading and M. Baker, "Peer-to-Peer Caching Schemes to Address Flash Crowds," in *Proceedings of the First International Peer To Peer Systems Workshop*, 2002.

[17] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," in *Proceedings of ACM SIGCOMM*, 2002.