Building Low-maintenance Expressways for P2P Systems

Zhichen Xu, Zheng Zhang[1]
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2002-41
March 1st , 2002*

Recent P2P systems, represented by Oceanstore and PAST, offer an administration-free and fault-tolerant storage utility. Nodes in these systems collectively contribute towards a storage space, in a self-organizing fashion. While elegant from a theoretical perspective, they can be improved in three important areas: (i) low maintenance cost; (ii) the ability to make discriminative use of the nodes in the system that has different capacity and resource constraints; (iii) the ability to adapt to the underlying network conditions and the applications' needs.

In this paper, we explore "expressways" as an auxiliary mechanism to deliver high routing performance, leaving the baseline infrastructure concentrating on tasks such as efficient resource utilization and simple management. The basic ideas of our proposal are to divide the total space in the overlay network into topology areas of different spans; we then select candidates in these areas to collectively construct a balanced, self-organized expressway system. By archiving the relevant system information (e.g. physical coordinates of the nodes) as objects stored on the system itself, we can further tune the expressway dynamically to fit both physical network conditions and application needs.

Using CAN as the target system, we show that expressways can easily boost its routing performance from $O(N^{1/d})$ to O(ln N), while keeping CAN's low maintenance cost. Our simulation studies show that our techniques to tune the expressways toward network conditions reduce the routing latency to about 50% over the default "random strategy" while staying within 2-3 times of the optimal. Furthermore, our simple heuristics to adjust the expressways toward application behavior can further reduce the physical latency up to 19%. Our mechanisms are robust and flexible, irrespective to the physical distribution of the participating nodes.

* Internal Accession Date Only
Approved for External Publication
[1] Authors are in alphabetic order

Building Low-maintenance Expressways for P2P Systems

Zhichen Xu and Zheng Zhang[1]
HP Labs, Palo Alto

Abstract

Recent P2P systems, represented by Oceanstore and PAST, offer an administration-free and fault-tolerant storage utility. Nodes in these systems collectively contribute towards a storage space, in a self-organizing fashion. While elegant from a theoretical perspective, they can be improved in three important areas: (i) low maintenance cost; (ii) the ability to make discriminative use of the nodes in the system that has different capacity and resource constraints; (iii) the ability to adapt to the underlying network conditions and the applications' needs.

In this paper, we explore "expressways" as an auxiliary mechanism to deliver high routing performance, leaving the baseline infrastructure concentrating on tasks such as efficient resource utilization and simple management. The basic ideas of our proposal are to divide the total space in the overlay network into topology areas of different spans; we then select candidates in these areas to collectively construct a balanced, self-organized expressway system. By archiving the relevant system information (e.g., physical coordinates of the nodes) as objects stored on the system itself, we can further tune the expressway dynamically to fit both physical network conditions and application needs.

Using CAN as the target system, we show that expressways can easily boost its routing performance from $O(N^{1/d})$ to $O(\ln N)$, while keeping CAN's low maintenance cost. Our simulation studies show that our techniques to tune the expressways toward network conditions reduce the routing latency to about 50% over the default "random strategy" while staying within 2-3 times of the optimal. Furthermore, our simple heuristics to adjust the expressways toward application behavior can further reduce the physical latency up to 19%. Our mechanisms are robust and flexible, irrespective to the physical distribution of the participating nodes.

1 Introduction

Peer-to-Peer (P2P) represents a class of systems that utilize distributed resources and perform critical functions in a decentralized manner. Comparing with traditional client/server systems, some benefits of P2P include improved scalability and reliability, elimination of hot-spots surrounding big servers, better resource utilization, and lower cost-of-ownership.

While P2P holds the promise of a general-purpose paradigm shift in computing, the most popular service they offer is information placement and lookup. Recent systems, represented by CAN [9], Chord [12], Oceanstore [14] and PAST [10], offer an administration-free and fault-tolerant storage utility. Nodes in these systems collectively contribute towards a shared storage space, in a self-organizing fashion. In these systems, there is a consistent binding between objects to nodes. Locating an object is reduced to the problem of routing to the destination node from the node where the query is submitted. An object can always be retrieved as long as the hosting nodes can be reached. In such systems, the random generation of node IDs and document keys allow even distribution of the data objects amongst the hosting nodes. The logical overlay of these systems provide some guarantee with respect to the number of logical hops that need to be traversed to locate an object.

While elegant from a theoretical perspective, they can be improved in three important areas: (i) achieving low maintenance cost; (ii) the ability to make discriminative use of the nodes in the system that have different capacity; (iii) the ability to adapt to the underlying network conditions and the applications' needs. For example, both Oceanstore and PAST use a routing scheme derived from Plaxton's proposal [14]. Nodes joining and leaving in these systems has high impact on maintenance cost, because they affect logarithmic number of other nodes. In addition, Plaxton's proposal was originally proposed for a centrally administrative environment: Web caching. Nodes in such systems cannot independently discover their neighbors in a decentralized fashion [9]. In Pastry and Tapestry, downstream links point to nodes close-by when the routing table is constructed. However, the nodes that are examined to put in the routing tables represent only a small set of candidates. In CAN, when a new node joins the overlay, it joins a node that is close to it in IP distance.This can result in uneven node distribution, which can complicate data placement and create "hotspots". Chord, in contrast, does not take the physical topology into consideration when constructing the overlay. Instead, it uses heuristics when it's beneficial to take more logical hops but each with smaller latency. The choices are, therefore, limited to entries in the routing tables.

We address these problems using auxiliary data structures called "expressways". These are analogous to the real world expressways, which greatly relax the constraint of physical distance, allowing people to focus on factors that matter the most. Expressways have the following attributes: they are *auxiliary* mechanisms, not a wholesale replacement of local routes; they have *greater distance span* per "hop", they are (usually) of *high bandwidth*, and they are constructed to accommodate commuters' needs. However, real expressways are centrally coordinated in construction, and they respond poorly to traffic congestions. If expressways

1. Authors are in alphabetic order. This work was done while Zheng Zhang was at Hewlett-Packard Laboratories

are to be attempted on the P2P systems, they must be *self-organized* and *self-maintained*, and *adaptive* to changing network conditions. Without achieving these goals, we would have violated the core spirit of P2P.

The basic idea of our proposals are the follows: (i) We first divide the total space in the overlay network into topology areas of different routing spans, with the topology areas of the smallest span correspond to the default P2P overlay. (ii) For a given topology area, there exist flexibility as to what nodes can represent it. These nodes then collectively form expressways at different levels in a self-organizing way. (iii) Next, we archive relevant system information such as physical coordinates, node capacity, and load information (or even application-level hints) as objects stored inside the system itself, in a way that's easy to update and retrieve. (iv) Last, we combine (ii) and (iii) to tune the expressway dynamically according to both network conditions as well application behavior.

For networks that delivers routing performance less than O(*log* N), our expressway provides a way to boost its performance to O (*log* N), while keeping the maintenance cost close to that of the default system. In addition, the techniques we have proposed to tune toward physical network conditions and application should be applicable to other P2P systems as well. In systems such as Pastry [10], different routing span has already been built in, conditions (i) and (ii) are already satisfied. However, (iii) and (iv) can be applied to derive better routing decision in accord to physical conditions and application needs.

In this paper, we make the following contributions:

- We explored the idea of using auxiliary data structures (expressways) for P2P overlay networks using CAN as an example. We show that the expressways for CAN [9], can easily improve performance from $O(N^{1/d})$ to O(ln N), with only a moderate storage overhead. The only states that need to be maintained accurately are those provided by the basic CAN already.
- We describe a simple algorithm for constructing expressway. The basic idea is to take snapshots when the system is growing. These sequences of snapshots preserve a set of topologies with different spans, and serve as the default expressway system.
- Using the default expressway system as a flexible framework, we developed several techniques to tune the expressway to fit network conditions and the application behavior. This is achieved by taking advantage of the archival capability of the P2P overlay network for storing the system's relevant information onto itself.
- We evaluate our techniques with simulation studies. The results show that our technique to tune the expressway toward physical network conditions can improve the average physical routing distance over our default random algorithm by about 50%, while stay within 2-3 range of the optimal. The simulation studies also show that our simple heuristics to tune the expressway to fit the application behavior can further reduce physical latency to up to 19%.
- In addition, our mechanisms are robust and flexible. For instance, our experiments reveal that good routing performance can be reached irrespective to the physical distribution of the participating nodes, while still preserving even storage distribution.

The remainder of the paper is organized as follows. Section 2 describes the basic algorithms for constructing and routing with the expressways and their evaluation. Section 3 describes advanced issues such as tuning the expressways to fit the underlying network conditions and application needs. In Section 4, we discuss how to build expressways for other P2P systems. We discuss related work in Section 5 and conclude in Section 6.

2 Constructing Expressway and Routing with Expressway

This section starts with a short description of CAN [9], which is designed for peer-to-peer file sharing and large-scale storage management systems [6, 1], followed by an overview of expressways for CAN and then the algorithms for building and routing with the expressways. We choose CAN primarily because of several nice properties it possesses. For example, CAN is able to scale to unlimited number of nodes, it has a very simple routing algorithm, it is self-configured, and it has low maintenance cost which is particularly important in a dynamic environment. We also believe that CAN's O ($dN^{1/d}$) routing performance makes it an ideal candidate to study the effect of expressway system. Our philosophy is to start with something simple yet extensible, and augment the basic scheme in a need-to basis. At the end of this section, we analyze the expressway system and evaluate them with simulation.

2.1 CAN

CAN stands for *content-addressable network*. It abstracts the problem of data placement and retrieval over large scale storage systems as hashing that maps "keys" onto "values" [4]. CAN organizes the logical space as a d-dimensional Cartesian space (a d-torus). The Cartesian space is partitioned into zones, with one or more nodes serve as owner(s) of the zone. An object key is a point in the space, and the node owns the zone that contains the point owns the object. Routing from a source node to a destination node boils down to routing from one zone to another in the Cartesian space. Node join corresponds to picking a *random* point in the Cartesian space, routing to the zone that contains the point, and split the zone with its current owner(s). Node

departure amounts to having the owner(s) of one of the neighboring zone take over the zone owned by the departing node. In CAN, two zones are neighbors if they overlap in all but one dimension along which they abut each other.

2.2 Overview of Expressway

Like the real-world expressway, the expressways for CAN augment CAN's routing capacity with routing tables of increasing span. To build expressways, the entire Cartesian space is partitioned into zones of different spans with the smallest zones correspond to the CAN zones, and any other zones are called e*xpressway zones*. Consequently, each node owns a CAN zone and is also a resident of the expressway zones that enclose its CAN zone.

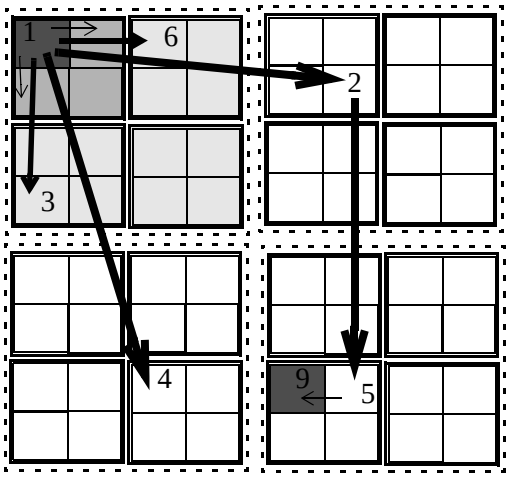

Figure 1: Expressways for CAN

These expressway zones and the CAN zone are recorded in each node in a data structure we call the *total routing table*, $R_T = <R_0, R_1, \ldots, R_L>$. R_L corresponds to *x*'s *default* routing table and is what CAN already builds. Each R_i (*i=0* to *L-1*) is called an *expressway routing table* that has larger span than the default routing table. The smaller the *i*, the larger the span, and routing with a smaller *i* is said to occur in a higher level of expressway. Each R_i contains the node's *i*-th largest enclosing zone and the set of neighbor zones (*expressway zones*) of the similar span (a formal definition of the routing table will be given in Section 2.3). For each neighboring expressway zone, the expressway routing table keeps the addresses of one or more nodes in that zone.

Figure 1 illustrates the expressways with an example. The CAN zones are at level 3, and each of the CAN zone is 1/64 of the entire Cartesian space. In this example, four neighboring CAN zones make one level-2 expressway zone and four level-2 zones make a level-1 zone. For example, node 1 owns a CAN zone (the zone with dark shade to the up-left corner), and it is also resident of the level-2 and level-1 expressway zones that enclose the CAN zone. The total routing table of node 1 consists of the default routing table of CAN (represented by the plain arcs) that link only to node 1's immediate CAN neighbors and the expressway routing tables (represented by the thick arcs) links to one node in each of node 1's neighboring expressway zones at level 2 and level 1. Figure 1 also shows how node 1 can reach node 9 using expressway routing.

It should be pointed out that, among the total routing table, only the default routing table needs to be maintained, which is guaranteed by the basic CAN infrastructure. For the rest, what really matters are the topologies of the zones recorded in the total routing tables. The topologies are stable; whereas the node responsible for these zones can be changed on the fly. In Section 3, we show how we can make use of this property to tune the expressway toward network conditions and application behavior.

2.3 Building Expressway

The preceding section serves to introduce the concept of expressway and the intuition behinds it. The challenge is how to construct the expressways at various levels dynamically as the nodes join and leave. There exist several choices. The algorithm we will describe below is called *evolving snapshot*. As we will see later, this algorithm establishes the basic topology of the expressways.

The idea behind the evolving-snapshot algorithm is quite simple. At regular intervals of system growth, snapshots are taken. A snapshot is simply a "frozen" copy of a current routing table. Formally, the routing table of a node *x*, *x.R*, includes the nodes' current zone, denoted by *x.R.Z*, and the set of neighboring zones, $x.R.N_d$ on each of the *d* dimensions, and the addresses of one or more residents for each neighboring expressway zone. This frozen routing table is then pushed onto *x*'s *total routing table*, $x.R_T$.

"Snapshots" seem to imply some global coordination. However, this is not the case: by the very nature of CAN, the total Cartesian space is uniformly populated. Thus, each node takes snapshot independently by observing its zone size, with which it may infer as to what stage the system has grown. When *x's current zone*, $x.R_L.Z$, shrinks to a target size, $x.R_{L-1}.Z/K$, it takes a new snapshot by incrementing *L* and cloning R_L out of R_{L-1}. We call *K* the span of expressway (*K* can vary from level to level in practice).

Table 1 summarizes the notation used:

Initially, there is only one node in the system. Its total routing table is $R_T=<R_0, R_1>$, and $R_0.Z$ is the entire Cartesian space. When a node *y* splits with *x*, it inherits *all* entries of *x*'s total routing table other than *x*'s current routing table ($x.R_L$), and makes its default routing table, $x.R_L$ according to the CAN algorithm. As the system evolve, a node takes snapshots at regular interval, accumulating those "frozen"

R_T	Total routing table: R_T = <R_0, R_1, …, R_L>
$R_i.Z$	The zone of the node when R_i is taken
$R_i.N_d$	The set of neighboring zones when R_i is taken; each element in the set includes the topology (zone coordinates) and the nodes that are currently representing it
L	Total level of expressways that this node is aware of currently
K	The span of the expressway
m	$Log_K N$, where N is number of nodes in the system

Table 1: Summary of Notations

routing tables in the past, each with decreasing span, in its total routing table.

Figure 2 explains the concept of snapshots and total routing table, with *K=4*. Independent of *d*, the evolution of a CAN system can be thought as building a binary tree since each new node will split with a random existing node. Each newly added link of the tree leads to a new node join in. At any given point of time, the leaves are the existing nodes in the system. The oval attached to each link in the figure represents the original CAN routing tables of the node since the node's inception. Ovals framed by a box correspond to routing tables that have undergone snapshots. The total routing table of a node can be found by walking down the tree from the root towards the node, picking up the snapshot routing tables along the path. The tree rooted by *x* when it joined the system is called *x*'s expressway tree (see Section 2.5 for a detailed description of the properties of an expressway tree).

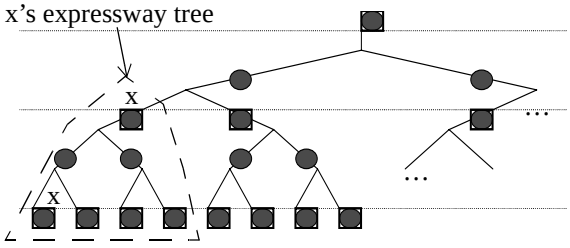

Figure 2: Snapshots and total routing table. Links represent nodes; ovals are CAN routing tables, and those framed with a box are snapshots.

Table 2 shows the actions that need to be taken when a node joins, in addition to what CAN already does. The new node inherits total routing table from the node being split, and then both nodes test to see if its current zone has shrunken to *1/K*-th of its last snapshot and, if so, a new snapshot is taken. As can be seen, the algorithm introduces minimum changes to CAN's existing task.

Procedure for a node *y* joins node *x*

y.R_T = <*x*.R_0,…*x*.R_{L-1}, y.R_L>
Repeat procedure for testing for new snapshot

Procedure for testing for new snapshot

```
// executed by both x and y
If (R_L.Z ≤ R_{L-1}.Z/K) {
    R_{L+1} = R_L
    R_T = <R_0, R_1, …., R_L,R_{L+1} >
    L = L+1
}
```

Table 2: Procedures for a new node join

2.4 Routing

The routing protocol is very simple: if the destination point is within the node's current zone ($R_L.Z$), we already reach the destination. Otherwise, it iterates through the total routing table, starting from the one with the largest span, until it finds a routing table R_i such that $R_i.Z$ does *not* cover the destination point. Figure 3 illustrates this. R_i is the first routing table whose space does not cover the destination, and the message will be routed according to R_i, to one of R_i's expressway neighbors.

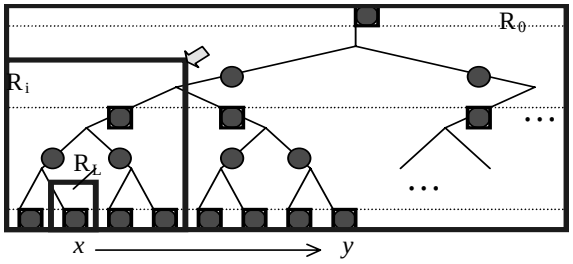

Figure 3: Routing using expressway by finding a routing table with largest span.

Table 3 shows the pseudo-code for the basic algorithm, where, *d* is the dimension of the Cartesian space; `pt` is the destination point that we want to route to. $R_i.Z.L_j$ and $R_i.Z.U_j$ denote the lower and upper bounds of $R_i.Z$ along the j-th dimension.

2.5 Tuning towards load-balance

As we will prove later, expressway built using the above algorithm has O(*ln* N) routing performance. Before we do that, however, we must resolve one critical issue.

Route with Expressway

```
If (pt ∈ R_L.Z) Return;
For (i=0; i ≤L; i++)
   If (pt ∉ (R_i.Z )
     Route using R_i;
```

Route with R_i

```
For (j=0; j<d; j++)
   If (pt < R_i.Z.L_j || pt > R_i.Z.U_j) {
     Route to x∈ R_i.N_j that is closest to pt
     Break;
   }
```

Table 3: Pseudo-code for routing with Expressway

Expressways in real life have the attribute that, they are of high bandwidth. Assuming a uniform distribution of traffic, it can be shown that expressway at level *i* needs to handle *K* times more traffic on average than expressway at level *i-1*. *K* is the expressway span introduced in Table 1. Thus the problem we have to deal with is load-balance among the nodes that participates in expressway routing, since the default algorithm has less number of nodes handling traffic in high level expressways.

Considering *x*'s *expressway tree* (see Figure 2), the evolving snapshot algorithm makes sure that any *y* that is a descendent of *x* has inherited all the routing tables of *x* all the way from the root of the whole system till *x* becomes the root of its expressway tree. As a result, it can be shown that *y* has the same capability as *x*, either when forwarding a packet outside *x*'s expressway tree, or when handling a packet destined to a node inside this tree.

Therefore, achieving load balance only requires that a node *z* that routes to *x* in expressway routing, to have the option of replacing *x* with *any* node inside the *x*'s expressway tree. Let $z.R_i$ be the routing table being used that elects to route to *x*, let *X* represent the respective expressway zone in which *x* is a resident. We simply pick a point *pt* in *X* and routes to it, whoever owns that zone now replaces *x* in $z.R_i$. There are variations on how *pt* is selected, the simplest one being picking any random point inside *X*.

Because the number of nodes to replace *x* is proportional to the size of *x*'s expressway tree being used, this implies that the number of nodes handling expressway routing at a level corresponding to *x*'s root position in the tree is proportional to the number of residents inside x's expressway tree. Thus, the algorithm will automatically balance the load among expressways. The difference between load distributions of using the non-load balanced algorithm versus the load-balanced version can be significant. Figure 4 shows our simulation results reporting the number of messages each node has to forward with the default routing table and the routing table constructed using random selection.

In the following sections, we will refer this load-balance scheme via random expressway neighbor selection, the *default random algorithm*.

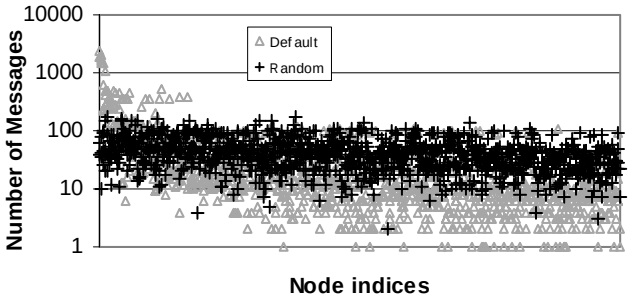

Figure 4: For the load-balanced algorithm the distribution of messages each node has to handle is uniform whereas for the non-load-balanced algorithm the message distributions are skewed. (This is with 2-d CAN of 1024 nodes.)

2.6 On-demand Maintenance

We now discuss how the expressways are maintained. For node join, nothing extra needs to be done as the evolving snapshot algorithm relies on the system growth. A newly joined node lazily updates its expressway neighbors for the expressway zones recorded in its total routing table, and such maintenance is proportional to the total levels of expressways, which is O ($\log_K$ N).

Handling node departure is slightly more complicated. Let *x* be the node that leaves. The default CAN already handles the event such that one of *x*'s neighbor will take over *x*'s responsible zone. However, *x* can be the expressway neighbor of multiple nodes; its expressway functions need to be maintained by some replacement nodes. We can do this with two steps in a demand-driven manner.

First, when another node *y* later attempts to route to node *x* that has departed, the request will time-out and *y*'s routing algorithm may retract and use an expressway of smaller reach. Note *y*'s routing without using any expressway will always work, reflecting our overarching guideline that the expressway is only an auxiliary system. Next, *y* picks up a point in the space of *x* recorded in the failed routing table, and route to it. This will always succeed at node *z* whose zone contains that point. *z* must be a descendent of *x* and thus inherits *x*'s routing capability. *z* is now the replacement of *x* to repair *y*'s snapshot.

In our algorithm, there might be *d* neighbors in each expressway level that uses *x* as expressway neighbors. Thus, on average the total number of nodes that will update their routing table entries is the product of the total number of expressway levels and the number of neighbors at each level, which is O (d.$\log_K$N). In any case, we repair the expressways in the background. We can keep the addresses

of multiple nodes that belong to each neighboring expressway zone. When one node is detected to have departed, another node is used instead. We then use the process described in the previous paragraph to replace the departed node.

2.7 Analysis

We now analyze the routing performance of CAN with expressway in place. For simplicity, we assume a uniform distribution of the CAN nodes in the Cartesian space.

Let m be $log_K\ N$, therefore there are m-1 levels of expressways. The total routing table has depth of m. As a result, the storage for routing table increases m-folds. For $K=4$ and $N=2^{20}$ (i.e., a million nodes), m is 10. Since each routing table is only a few hundred bytes (for $d=2$ it's 160 bytes, assuming perfect alignment).[2] Storage increase is, therefore, not a concern.

As shown in Figure 5, routing using expressways involve at most m levels of CAN-like routing. In the first step, the routing algorithm will iterate through the routing table of the source node to get down to the level where the expressway zone does not cover the destination; it will then use CAN routing to reach an expressway zone that covers the destination. This process continues until the destination is reached. Thus there are $(log_K N)*(d/3)K^{1/d}$ hops in the worst case. The first factor in the equation is the number of levels the routing will travel, whereas the second being the average routing hops in any level. To simplify our analysis, we ignored the fact that the top-level expressway can use torus for routing and has in average (d/4) $K^{1/d}$ hops, we use (d/3) $K^{1/d}$ hops in stead. [3]

The bigger the K, the less levels of expressways there are, but the trade-off is that routing at each level is more expensive. Solving this routing cost for optimal value of K, it turns out of $K=e^d$ gives the best routing performance. Below, we show how $K=e^d$ is derived.

2. For each neighbor it needs 32 bytes to represent the zone and 8 bytes to record the address of the node that is responsible for the corresponding neighboring expressway zone, assuming 64 bit node address.

3. We assume that both the source and the destination follow a uniform distribution in the Cartesian space. Therefore, the average distance is 1/3. For example, for a 1-dimension CAN, the formula is $\int_0^1 \left(\int_0^x (x-y)dy + \int_x^1 (y-x)dy \right) dx$, which equals to 1/3. x and y are the positions of the source and destination respectively.

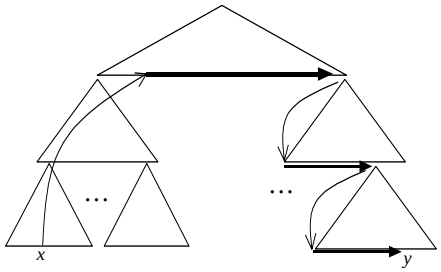

Figure 5: Routing with Expressways

Let $f(x) = \frac{d}{3} \cdot x^{1/d} \cdot \log_x N$, then

$f(x)' = (d/3).\left(\frac{1}{d} \cdot x^{\frac{1}{d}-1} \cdot \log_x N + x^{\frac{1}{d}} \cdot \frac{\ln N}{-x \cdot (\ln x)^2} \right)$.

Let $f(x)'=0$, we have $\frac{1}{d} \cdot x^{\frac{1}{d}-1} \cdot \frac{\ln N}{\ln x} = x^{\frac{1}{d}-1} \cdot \frac{\ln N}{(\ln x)^2}$.

Simplify the above equation, we get $\frac{1}{d} = \frac{1}{\ln x}$.

Thus the optimal K is e^d, and the optimal routing performance is:

$$f(x) = \frac{d}{3} \cdot (e^d)^{\frac{1}{d}} \cdot \frac{\ln N}{\ln(e^d)} = \frac{d}{3} \cdot e \cdot \frac{\ln N}{d} = \frac{e}{3} \cdot \ln N \ .$$

The fact that optimal routing using expressway is achieved independent of the choice of d is somewhat an interesting and beneficial result. In CAN, lower dimension simplifies node join and leave considerably, but sacrifices fault-tolerance because less "parallel" routes are available. Therefore, we can choose a low d such as 2 to simplify maintenance, while preserving reasonable fault-tolerance capability and still achieving optimal performance.

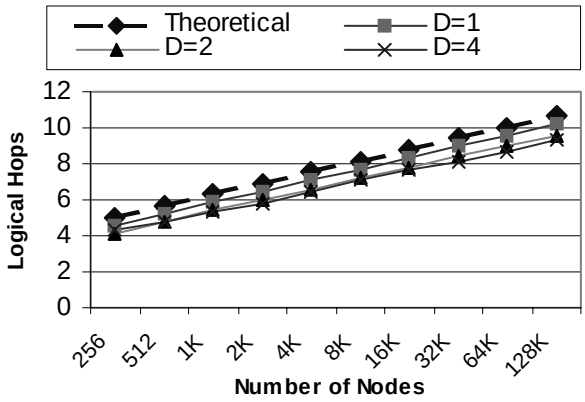

Figure 6: Expressway performance compared with theoretical values

In Figure 6, we compare the theoretical performance of using the expressway ($e \ln N / 3$), with the results

obtained using simulation for d=1, 2, and 4, up to 128 K nodes. We can see that the simulation results matches well with the theoretical values. The small deviations are resulted from two factors. First, we have ignored that fact that in some cases the torus can be used. Second, the theoretical optimal values of *K* are not always achievable given that we always does 2-way splitting when a node joins the system.

Figure 7 shows that expressway with the lowest dimension (d=1) easily outperforms the basic CAN up to d=5.

Last, in practice, for a reasonable small *K*, we can fully connect the *K* nodes that are in the same expressway zone and are at the same level of the expressway. This renders the routing performance O(log_K N) and tunable by varying *K*.

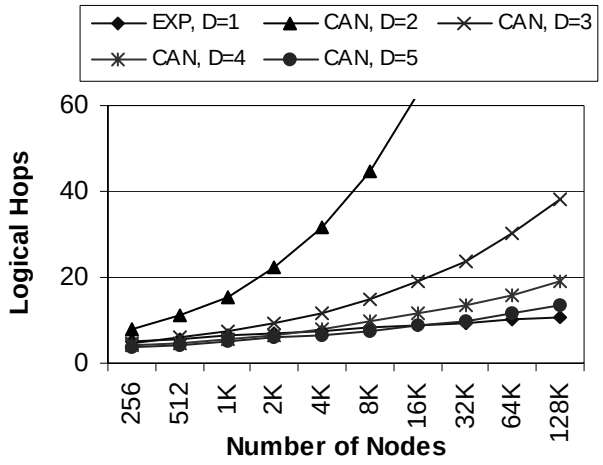

Figure 7: Expressway performance compared with CAN with different d

3 Tuning towards best performance

So far, we have discussed how the expressways can be constructed, maintained, and deliver O (*ln* N) or O(log_KN) routing performance. As have been illustrated through our algorithms to balance the load, the important feature is that the topology is stable whereas the representatives for each expressway zone can be changed on-the-fly. This flexibility is key to tackle advanced issues such as tuning towards both network conditions and application behavior.

3.1 Tuning towards Network Conditions

One important problem we must tackle is to utilize the underlying network topology to best suit the overlay network. This is of particular importance to CAN because the dilemma it is facing: as nodes join the Cartesian space at random point, it is guaranteed that documents (whose keys are also produced randomly), are evenly distributed across all nodes. Attempts to distribute the nodes according to their physical locations ran the risk of uneven storage distribution, since the partitions of the space will reflect the network topology and are unlikely to be uniform. One possible solution would be using the physical distribution of nodes to "skew" the distribution of documents themselves. However, in a dynamic environment where nodes come and go, coupling document distribution with node distribution creates unreasonable dependencies.

We first consider a naive solution. Suppose *X* is a neighboring expressway zone recorded in node *y*, *y* randomly selects a set of replacement nodes, *E(X)* in *X*. *y* then measures RTTs to the nodes in *E(X)*. A subset of *E(X)* now serve as the representatives of *X* in $y.R_i$. At the time of routing, *y* chooses from *E(X)* with probability inversely proportional to the candidate's RTT to *y*. The problem here is that the size of *X* gets exponentially larger at higher level of expressway, where coming up a representative *E(X)* set is impractical.

3.1.1 Locating Closest Nodes using Coordinate Maps

We now describe a solution that is based on nodes' positions in the coordinate space. Eugene *et al* [4] described an algorithm to map any given node into a Cartesian space by measuring against a set of landmarks. The distance between two nodes in that space closely reflects their Internet distance. Karp *et al* [5] and Li *et al* [7] also argued that physical distance, measured by services such as GPS, can approximate network distance in the wireless setting. Thus, we can substitute a node's network coordinates with its coordinate in a Cartesian space.

Procedure for testing for new snapshot

// execute by new and old
obtain *p*
if (new node) {
  for (i=1; i≤L; i++) {
    map *p* to *p'* in $R_i.Z$
    store <R_i.Z, *id*, *p*> in $z \in R_i.Z$ that owns *p'*
  }
}
If ($R_L.Z \leq R_{L-1}.Z/K$) {
  $R_{L+1} = R_L$
  $R_T = <R_0, R_1, \ldots, R_L, R_{L+1}>$
  L = L+1
  Map *p* to *p'* in R_L.Z
  Store <R_i.Z, id, *p*> in $z \in R_i.Z$ that owns *p'*
}

Table 4: Procedures for a new node join

The basic idea is to build maps of network coordinates, and utilizing the archival capability of the P2P system itself to store and maintain the maps in the various expressway zones in CAN. When such maps are available, any node *y*

Procedure Locating Closest Node of x in Z

Let *pt* be *x*'s network coordinate;
Map *pt* to *pt'* in Z;
Route to the node *y* in Z that owns pt';
If (*y*'s summary is not empty)
  Return summary
Else
  Define a TTL to search outside *y*'s summary range.

Table 5: Procedures for locating the physically closest node in a zone

can find a resident close to it in a given expressway zone *Z* by consulting the appropriate map. There are many ways to build a distributed map. Our map is composed of grids, each contains residents whose coordinates fall inside the boundary defined by the grid. This is just like a real map. What is different though, is that we use the expressway zone's CAN partition as the grid itself.

The goal here is to devise a hash function that can map positions in the network coordinate space to points in a CAN zone while at the same time preserve the distance relationship among the positions in the coordinate space. Suppose we want to store the information of a node *n*, whose position in the coordinate space is *p*, onto the CAN zone *Z*, and the hash function maps *p* to *p'* in *Z*, we store the triple <Z, *n*'s address, *p*> as an object in the node that owns *p'*. There are three cases:

- If the dimension of the coordinate space equals to the dimension of the CAN we build and if *Z*'s range is [*l*, *u*] along dimension *i*, for a node *n* with coordinate p_i along the *i*-th dimension, *n* is mapped to a point *p'* in Z with coordinate $l+p_i*(u-l)$ in the *i*-th dimension.
- Storing coordinates that have a lower dimension than the CAN is straightforward, and we can simply set the extra dimensions to zero.
- When network coordinates have a higher dimension than CAN, what we can do is to fold multiple dimensions in the coordinate space into one dimension of the CAN zone. For example, to fold *d* dimensions of the coordinate space onto [*l*, *u*] of one dimension of a CAN zone, one such function is $l + \frac{1}{d} \cdot \sum_{i=0}^{d-1} p_i \times (u - l)$.

Any given node *x* participates at all level of expressways, for other nodes that are physically close to *x* to select *x* as expressway neighbor, *x*'s network coordinate needs to be published in maps corresponding to those expressway zones. Therefore, there is one map for each expressway zone in the system. It follows that each node will appear in a maximum *of* $log_K N$ such maps. Assume that the total number of nodes *N* is 2^{20} and *K* is 4, this number is 10. We believe that this is not a big issue.

The original node join procedure is slightly modified, as shown in Table 4. The existing node being split only needs to publish its location in the new snapshot, if one is to be taken; whereas a new node must publish its location in all the expressway zones in which it is a resident. Now, when a node is looking for candidates in an expressway zone *Z* that is close to it, it uses its own coordinate and index into *Z*'s map. As shown in Table 5.

It is not guaranteed that there are nodes recorded in the targeted. To increase the probability that a node can locate the candidates that are physically closest to it in a given zone, there are a number of techniques we can use. First of all, a node's coordinates can be published not only in the designated grid, but also in a set of neighboring grids defined by a radius. This not only improves the availability of the map, but also allows a summary of nodes living in the grids close-by to be built. Alternatively, we can store the network coordinates onto a sub space of the "host zone". The net effect is a condensed map, which increases the chance of a node being able to locate the closest node in the expressway zone of interest. For example if the subspace is 0.5 of the size of the zone, the map stored on one node include information about approximately two nodes assuming a uniform distribution of the nodes in the Cartesian space. We define the ratio of map size to the size of the hosting zone, *condense rate* of coordinate map.

Clearly, the two mechanisms exhibit a trade-off: the replication and summary improve the possibility as well availability of the map service, but this is achieved at the expense of additional communication and storage costs; whereas the condensed map could possibly create hot spots for map queries. Last, we note this two techniques can be combined to further increase the probability of locating physically close-by nodes.

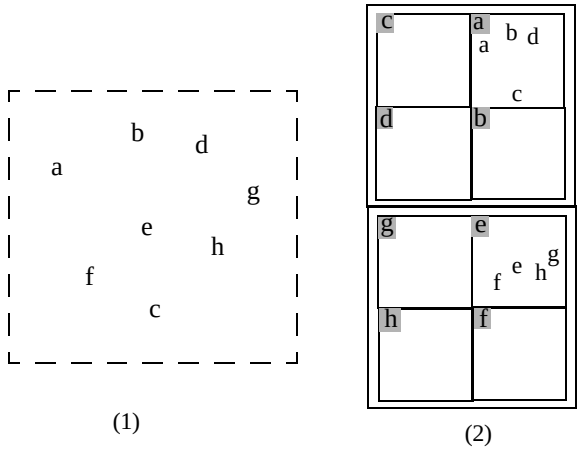

Figure 8: Storing and retrieving coordinate maps

Figure 8-1 depicts 8 nodes (*a* to *h*) and their network coordinates. These nodes are distributed in a 2-d CAN as shown in Figure 8-2, where *a-d* and *e-h* correspond to two neighboring expressway zones. Each node's CAN zone are those small squares with owner's ID in shaded box. Without using coordinate map, each node simply randomly pick one node from the neighboring zone as its expressway neighbor. For instance, *a* can select either *e, f, g* or *h*, without considering physical locality. With coordinate map with condense rate of 1/4, we can do much better. In this case, *a-d* publish their positions in the grid owned by *a*, where *e-h* publish in the grid owned by *e*. Now when *a* selects its expressway neighbor, it uses its own network coordinate and consults the map of its neighbor which is stored in *e*, and find that *e* is physically closest. Thus *a* uses *e* as its expressway representative for the zone that comprise *e-h*. Likewise, *c* will select *f*.

Because of the dynamic nature of the system, a node should periodically check the target expressway zone's map to see whether more favorable nodes are available. The accuracy of the map can be lazily maintained. In the most reactive case, departed nodes are deleted from the map only when they are selected as expressway replacements and later found un-reachable. Alternatively, each owner of the map grid can periodically poll the liveliness of the nodes. The most proactive measure is to update the map when a node is departed. In the following evaluation, we will use the condensed map as the default algorithm.

3.1.2 Evaluation of Algorithm

We have considered several options to study the effect of our algorithms. Using a full topology generators such as gt-itm or BRITE [8, 13] is possible and could derive meaningful statistics such as latency in ms. However, reasonable conclusion can only be drawn until a large number of topologies are tried. At the end, we decide to report latency as the "distance" between network coordinates, and vary the distributions of the nodes instead. The two cases we consider are:

- Uniform: here the nodes spread evenly geometrically.
- "Ring": in this case, the nodes uniformly distributed in a ring-shaped area around the edges of a 2-dimension Cartesian space. We produce this ring-shaped distribution to mimic the fact that nodes, in practice, are distribute over the edge of the Internet.

We choose CAN with d=2 to give a reasonable fault-tolerance capability. We compare our algorithm, varying the map condense rate from 0 to 1, against the random algorithm described in Section 2.5, and the "ideal" case where the network coordinates maps perfectly to the overlay network. As explained earlier, the "ideal" case will certainly gives the best routing performance, but is impractical because the resulting uneven storage distribution (and thus loads from application requests).

Figure 9 and Figure 10 depict the ratios of the average latencies of CAN with the help of the network coordinates maps to those of "ideal" CAN. In both cases, we use expressway routing. In practice, the latency should be the sum of *physical_distance* and *logical_hops* * *per_hop_overhead*. To be conservative, we have set *per_hop_overhead* to zero.

The observations that we can make from the two figures include the following. (i) The larger the number of the nodes in the system, the further away is the performance from the "ideal" case. This is simply because the accumulation of the deviations of the individual hops. (ii) The larger the number of nodes the better improvements over the random case. This because the average number of logical hops is larger when there are more nodes in the system. Consequently, there is a better chance to locate physically closest nodes. As we will show later, the earlier logical hops typically have a larger span and hence a larger candidate pool to select the next hop node. (iii) The smaller the condense rate, the better the performance. This is expected, because more candidates can be found. We note that, after condense rate reaches 0.2, reducing the rate further does not give as dramatic an improvement. (iv) Our technique performs fairly consistently over different node distributions. In fact, when the nodes follow a non-uniform (ring-shaped) distribution, which is closer to reality, the improvements over the default random algorithm are more significant. Comparing with the default random algorithm, the improvements are close to 50%, and stay within 2-3 range of the "ideal".

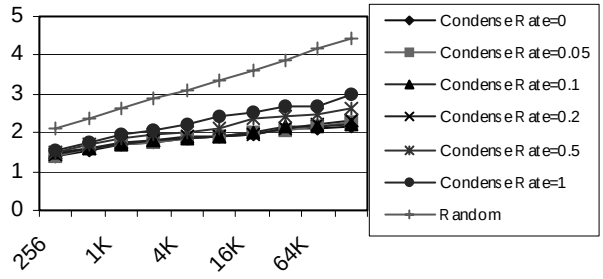

Figure 9: Ratio of physical latencies of CAN using coordinate maps to those of "ideal" CAN. The coordinates of the nodes follow a uniform distribution. (X axis shows the number of nodes and Y axis shows ratio to the "ideal" case)

We found that these results quite encouraging. We acknowledge that "certain percentage of reduction" is not very intuitive because we use distance as a measure of latency, and plan to use one real topology to derive results with more meaningful latency metrics.

To find out to what extent the map has helped us, we plot average latency data per-hop in Figure 11, comparing with the "ideal" case. The figure shows an interesting obser-

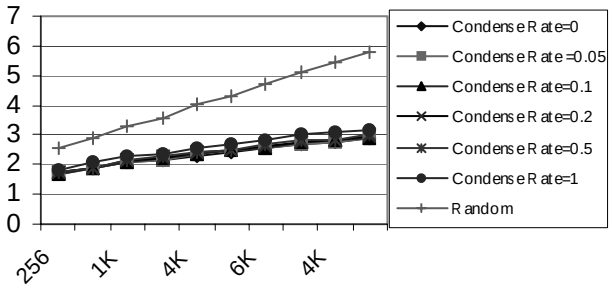

Figure 10: Ratio of physical latencies of CAN with maps to those of "ideal" CAN. The coordinates of the CAN nodes follow a ring-shaped distribution. (X axis shows the number of nodes and Y axis shows ratio to the "ideal" case)

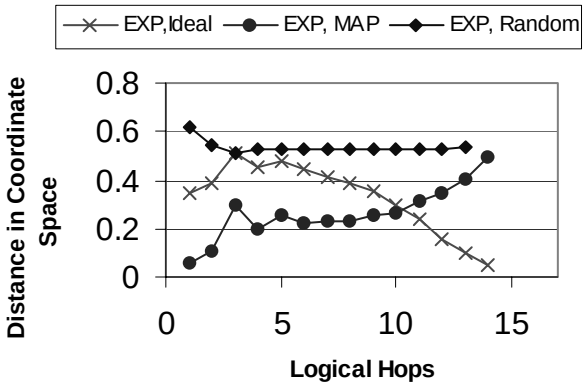

Figure 11: Comparing with the case that uses coordinate map and the "ideal" case. (d=2, and N=4096, and the physical coordinates of the nodes are uniformly distributed in the entire space)

vation, that is, with expressway way, the physical distance of each logical hop increases (see the curve that is marked with round dots), whereas for the "ideal" case, the reverse is true (i.e., physical distances of each logical hop decreases. See the curve marked with cross). This is because with expressway routing, earlier logical hops tend to cover expressway zones that have larger span (see Figure 5). When locating the closest nodes in a neighboring expressway zone, the larger the neighboring zone, the better chance it can find a node that is close by. Whereas in the "ideal" case, where the logical topology perfectly matches the physical topology, the span of the logical hops reflect the physical distance.

This observation leads us to believe that there is still room to improve. An optimal solution should have per hop distance close to that of using the map at beginning of the routing, while close to that of "ideal" at later stages. This is possibly only when node distribution can be intelligently controlled, so that close-by physical nodes can be located irrespective to logical distance.

3.2 Route around congested node

Optimizing routes according geographic vicinity, as described in the previous section, is not sufficient. Indeed, the Internet is a dynamic environment where the flow of traffic is unpredictable. An important topic for any P2P networks is to deal with congestions so as to deliver service with the best effort, end to end. An extensive treatment of the problem is beyond the scope of this paper and remains as a focus for our future work. In this section, we only outline our principles and sketch out several techniques.

Avoiding congestion involves at least two critical components. First, the congestion must be detected and reported; second, alternative routes should be evaluated. For the first task, we employ two techniques: explicit congestion notification for congestion avoidance and active probing for congestion discovery. With explicit notification, each node observes its own forwarding capacity. If it finds that it is about to be congested, it will notify the upper stream neighbors explicitly, so that the neighbors can route to an alternative node that can possibly detour around the congested node.

In some cases, a node can be too swamped to do explicit notification. To handle this situation, a node probes its routing neighbors (including expressway neighbors) periodically, giving a time window within which a response must be provided. Failing that is an indication that either the neighbor has departed, or it is too loaded and congestion has occurred. In order to make sound replacement decision, active probing is done periodically to all the expressway candidates in the target expressway zone. The expressway candidates can be obtained by extending the previous algorithm to return not just closest node, but a set of them recorded in the summary, or in the grid from the condensed map.

In addition to the two techniques just described, we can also integrate congestion control as part of the process of locating the most appropriate expressway neighbor in a given zone. To do so, each node will periodically update its map with information about its load. As before, failure to update this information can indicate that: the node becomes congested, or it has departed the system.

All these provisions may still be insufficient. In the cases when the system is heavily loaded, the only possibility to eliminate congestion is to control the sending rate of the sources; the traditional congestion control techniques (such as the slide-window protocol in TCP) can be employed.

3.3 Tuning according to application needs

The other side of the coin of delivering optimal performance has to do with tailoring the routing infrastructure according to application behaviors.

Reducing per-hop physical distance does not necessarily reduce the total delay. Consider the routing from *x* to *d* where *d* is the final destination. The next physically closest expressway node is *y*. There could exist *y'* that belongs to the same expressway zone of *y* and is physically further to *x*, but so happens to take fewer logical hops to *d*. To account for communicating patterns of the application, we do the following. First, in the total routing table of node *x*, for each neighboring expressway zone, instead of only keeping the address of the node that is physically closest to *x*, we keep the addresses of multiple nodes for each neighboring expressway zone. Second, when route to a particular destination, we make an attempt to balance the per-hop physical distance and total number of logical hops to achieve overall smallest latency. We developed a simple heuristics, that is, when choosing a candidate y in node x's total routing table to route from *x* to destination *d*, we try to minimize the terms

physical_distance(x,y)

+ *ratio_to_ideal* ideal_distance*(y, d)

The term *physical_distance*(x,y) gives the physical distance between *x* and its expressway neighbor *y*, the function *ideal_distance*(y,d) estimates the distance between (y,d) assuming that the logical topology perfectly matches the physical topology, and the term *ratio_to_ideal* gives the ratio of the average physical distance using map to that of the "ideal" case.

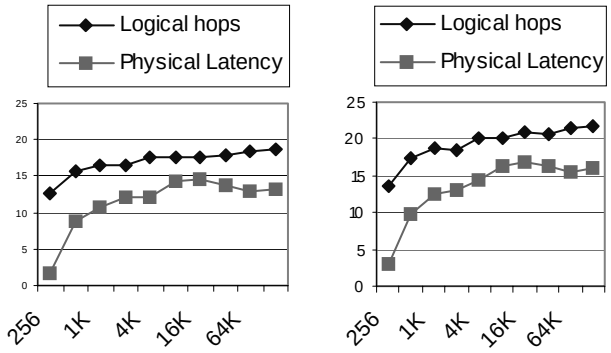

Figure 12: Percentage of improvement over IP optimization alone. The figure to the left shows the results with K entries for each expressway zone and the figure to the right shows results with 2K entries for each expressway zone. The nodes follow a uniform distribution. (X axis shows the number of nodes and Y axis shows percentage of improvement)

Figure 12 to Figure 13 demonstrates the improvement with this optimization. They are for the uniform and ring topology, respectively. In each figure, there are two pairs of curves, using *K* and *2K* entries. We report the percentage of improvement over both logical hops and latency comparing to optimizing towards network conditions alone. In our experiments we set *ratio_to_ideal* to 2 and 2.5 respectively

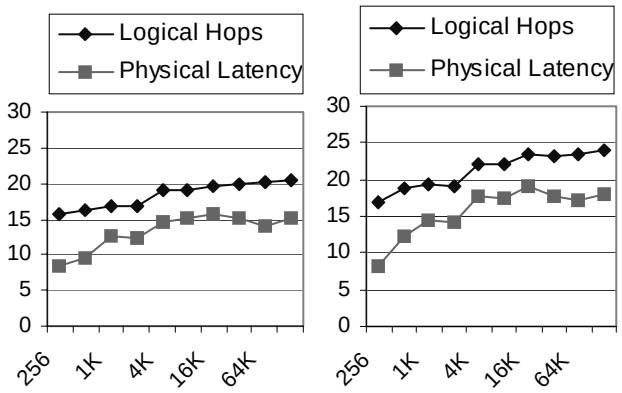

Figure 13: Percentage of improvement over IP optimization alone. The figure to the left shows the results with K entries for each expressway zone and the figure to the right shows results with 2K entries for each expressway zone. The nodes follow a uniform distribution. (X axis shows the number of nodes and Y axis shows percentage of improvement)

for the two different node distributions. (Recall that Figure 9 and Figure 10 have shown that the physical distance using map is approximately 2 - 3 times the case when logical topology perfectly matches the physical topology.)

What can be seen is that this simple optimization is rather effective, reducing physical distance up to 19% percent. In addition, retaining more nodes to represent an expressway zone can lead to better performance. Like before, our technique performs fairly consistently regardless of the nodes distributions, and the improvements for the ring-shaped distribution is slightly better.

Of course, all there are achieved at the expense of additional storage cost. For a d-dimensional CAN, assuming perfect neighbor alignment, the number of bytes for each expressway level can be computed using the formula $2d*(16d + (8d+1)e)$, where 16d bytes are used to represent the lower and upper bounds of an expressway neighbor along each dimension. For each candidate in a neighboring expressway zone, *8d* bytes are used to record the center of the node's CAN zone, and 8 bytes are used to records its physical distance to the current node. *e* is the number of such candidates. For example, when *e* is 4 and *d* is 2, the number of bytes at each level is 400 bytes. It requires approximately 4K bytes if there are 10 expressway levels, which should not be a major concern.

To further improve the routing performance, some storage can be set aside at each node to cache routes based on runtime behavior. Application level routing cache has been used in Freenet [2], where newly discovered nodes are inserted into routing table. Simulation results in [2] indicates that routing performance improve over time, approaching that of O(logN). The resulting network as a whole exhibits the power-law pattern, where a small number of nodes evolve to have high connectivity over time. This

P2P Systems	Routing Perf.	Maintenance Cost	Physical Network Conditions	Application Behavior	Storage Utilization
Chord/CFS	O(log N)	O(1) immediate, $O((\log N)^2)$ lazy	Server selection heuristics	Caching	Virtual servers
Tapestry/ Oceanstore	$O(\log_{(2^b)} N)$	O(1) immediate, O(log N) lazy	Select nodes that are close to current node when overlay is constructed	Attenuated Bloom filters	Replica management
Pastry/PAST				-	Replica diversion/file diversion
CAN	$O(d\ N^{1/d})$	O(d)	landmark ordering Overlay = physical network	caching/replication	split zone with node that has largest volume
CAN w. Expressway	O (ln N)	O(d) immediate O(ln N) lazy	Coordinate map	heuristics to balance physical distance with logical hops	Even node distribution

Table 6: Comparison with other P2P systems, N is the total number of nodes in the system

isn't necessarily desirable: a power-law network is vulnerable to attacks on these well-connected nodes. Furthermore, that would mean that the overall loads in the system may not be balanced. We believe application level routing would remain as an interesting research topic.

4 Building Expressway for Other P2P systems

It should be pointed out that it is not our goal to build expressways that out performs O(log*N*) logical hops in general. Rather, we believe that the most important attributes of a P2P system are to reduce maintenance cost, and to provide venues to tune the system to suit the application needs and the underlying network conditions. Such that, in practice, it beats O(log N).

The central concepts of our proposals are the followings:

1. Identify topologies with different routing span, and utilize such information as a foundation to construct expressways.
2. Having a default routing mechanism such that for a given target topology area, there exist flexibility as to who can be chosen as the representative of the area.
3. Archive relevant system information such as physical coordinates and load information as objects stored inside the system itself, in a way that's easy to update and retrieve.
4. Combine (2) and (3) to tune the expressway dynamically according to both network conditions as well application behavior.

The evolving snap algorithm provides a general means to establish the basic expressway structure.[4] For networks that delivers routing performance less than O(log N), our expressway provides a way to boost its performance to O (log N), while keeping the maintenance cost of the default system.

In addition, the techniques we have proposed to tune toward physical network conditions and application should be applicable to other P2P systems as well. In systems such as Pastry, different routing span has already been built in, condition (1) and (2) are already satisfied. Though it is conceivable that another layer of expressway can still be built on top. However, 3) and 4) can be applied to derive better routing decision in accord to physical conditions and application needs.

5 Related work

Recently, there are much work focusing on building decentralized routing, and data placement and retrieval. We compare them with CAN with expressway in terms of routing performance, maintenance cost, ability to tune the system toward underlying network conditions and application behavior (see Table 6 for a summary).

Both Tapestry [14] and Pastry [10] are based on Plaxton's proposal [14], where node IDs and the routing tables are divided into levels each with *b* bits. In level *i* of a routing table, it keeps the addresses of the nodes whose IDs match up to the *i*-th level with the current node but differ in the *i+1*-th level. To reduce the overall routing latency, the nodes kept in the routing tables are those that are physically "closest" to the current node while meeting the criterion.

4. The only requirement is that there are ways to find out the logical closeness among nodes – this is the case in CAN, Chord, Pastry and Tapestry.

In both Tapestry and Pastry, the closest node are picked when a new node joins. For instance, in Pastry, a new node *X* first contacts a node *A* that is physically close to it. It have *A* route to *X*'s ID. Along the way, it picks up the level *i* routing table from the *i*-th node encountered. Because *X* is close to *A* and the routing tables of *A* and all the nodes along the path are constructed such that they are close to the respective node. Therefore, the routing neighbors picked up by *X* will be reasonably close to X. In a second phase, *X* requests the state from its routing neighbors with an attempt to locate better candidates. Tapestry also keeps two backup neighbors for each routing entry. An introspective optimization process measures the latencies to primary and backup neighbors. It may use higher-level neighbors for lower level neighbors if they are closer.

Besides ensuring random distribution of node IDs and document keys, in Tapestry, documents are replicated on multiple *root* nodes, and each root may have multiple *floating replicas*. Replica management is used to adjust the number and locations of floating replicas to service the access request more efficiently. In PAST, balanced document distribution is achieved through *replica diversion* and *file diversion* [10].

To fit application behavior, Tapestry employs a*ttenuated Bloom filters*. The first Bloom filter records the objects contained locally, whereas the *i*-th Bloom filter is the union of all Bloom filters for all the nodes at distance *i* through any path from the current node.

In Chord (and CFS [3]), the routing performance is O(log N). The maintenance cost is O(1) immediately by just using the predecessor's finger table. It takes O $(\log N)^2$ messages to repair the routing tables in the background. To take advantage of physical network, Chord employs some server selection heuristics which may take more logical hops to achieve overall smaller physical latency. Chord employs caching to take advantage of application behavior. It achieves even storage utilization via the notion of *virtual server* [3].

The routing performance of CAN is O(d $N^{1/d}$), where *d* is the dimension of the Cartesian space and the maintenance cost is in the order of O(d). To take advantage of the physical network conditions, the overlay is built to reflect the physical network via landmark ordering. This, however, can cause uneven node distribution. CAN takes advantage of the application behavior via caching data copies and on-demand lazy replication. Even storage utilization is achieved by split zone with a neighbor with largest volume and replication.

In our proposal, the routing performance of CAN with expressway is O(ln N). The default CAN routing table needs immediate repair with a cost of O(d). As explained in Section 2.6, lazy maintenance requires O(ln N) messages on average. We use the default expressway topology as a framework to tune towards both network conditions and application needs (Section 3.1 and Section 3.3). These techniques preserve the random distribution of the nodes in the logical space and the document space, ensuring even storage utilization. It should be noted that our techniques do not preclude the use of any other orthogonal techniques such as replication and caching.

We now offer some high-level remarks. First, the finger tables of Chord can be thought as expressways, and a 1-d CAN with expressways has the same level of performance as Chord O(Log *N*). However, we can use higher dimension CAN [9] (e.g., d=2) to achieve higher fault tolerance capability. Second, the notion of *multiple realities* in CAN remotely relates to expressway in that the snapshots were realities of the system in the past. The difference is that the expressways do not have to be perfect CANs. Third, routing in PAST and Tapestry resembles the expressways in that, in routing table, entries at a lower level has a larger span than those at a higher level. However, PAST and Chord force a rigid logical structure (e.g., fixed number of routing levels). This makes the routing distance independent of the actual system size. In contrast, our scheme ensures that small system simply has less number of expressways and routing can take fewer hops. We note that for PAST, it is conceivable to allocate relative larger number of bits for each routing level when the system is of a relative small size.

Fourth, in the existing P2P networks, our contribution of using the archival nature of the system to store and retrieve relevant system information to gain performance advantage has been unique. Our results are robust, maintaining uniform node distribution irrespective to their physical distribution. We use CAN as the underlying platform and inherited some of its desirable features, including unlimited number of nodes and objects. In our experience, CAN's ability to better deal with dynamic nature of nodes departure and join as well as the its simple routing algorithms are fundamental properties.

Last, self-archiving of system information has been explored in researches in other area. For instance, GLS [7] organizes the node space into grids the resemble CAN zones and their higher-order grid resembles our expressway zones. However, their goal is to assign appropriate number of location servers for each mobile node, rather than efficient routing, and the grids are pre-defined geographic regions.

6 Status and future work

Our ongoing research focuses on a few areas. We are in the process of gaining more in-depth understanding of expressways properties; we are developing better algorithms to more efficiently establish expressway routes that tune towards application needs and the underlying physical networks. What we have shown is one way of using the archival capability of the P2P system. It is conceivable to extend

idea of coordinate maps to also store application-level hints, such as the storage capacity of the node, or even the type of applications that has been installed (e.g., to transcode the content along the way); when choosing a expressway neighbor, we choose the ones that meet the application-level semantic as well. We are building a prototype of CAN with expressways and looking into extending the expressway idea to support application-level multicast for applications such as streaming. Last, we are in the process of building a distributed file service over CAN with expressway.

1. References

[1] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an existing set of Desktop PCs. In *Proceedings of SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[2] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong. Freenet: a distributed anonymous information storage and retrieval system. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, **LNCS 2009**, ed. by H. Federrath. Springer: New York (2001).

[3] F. Dabek, M. Frans Kaashoek, D. Karger, R. Morris, I. Stoica. Wide-area cooperative storage with CFS. *ACM SOSP 2001*, Banff, October 2001.

[4] T. S. Eugene, Ng, H. Zhang. Towards global network positioning. *ACM SIGCOMM Internet Measurement Workshop 2001*. San Francisco, CA.

[5] B. Karp and H. Kung. Greedy Perimeter Stateless Routing. In *Proceedings of ACM Conf. on Mobile Computing and Networking (MOBICOM)*, Boston, MA, 2000.

[6] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An Architecture for Global-scale Persistent Storage. *In Proceedings of ASPLOS 2000*, Cambridge, Massachusetts.

[7] J. Li, and J. Jannotti, D.D. Couto, D. Karger, and R. Morris. A Scalable Location Service for Geographic Ad-hoc Routing. In *Proceedings of ACM Conference on Mobile Computing and Networking (MOBICOM)*, Boston, MA, 2000.

[8] A. Medina, A. Lakhina, I. Matta, J. Byers. BRITE: Universal Topology Generation from a User's Perspective. *Technical Report BUCS-TR-2001-03*. Computer Science Department, Boston University.April 2001.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A scalable content-addressable network. *SIGCOMM'01*, August 27-31, 2001, Scan Diego, CA.

[10] A. Rowstron and P. Druschel,. PAST: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. SOSP'01.

[11] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a Case for Informed Internet Routing and Transport, *IEEE Micro*, pp. 50-59, v 19, no 1, January 1999.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kasshoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM'01*, August 27-31, San Diego, CA.

[13] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an INternetwork. *IEEE Infocom'96*, San Francisco, CA, May 1996.

[14] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph. Tapestry: an infrastructure for fault-toleratnt wide-area location and routing. *Technical Report UCB/CSD-01-1141*. EECS.