An Expresway over Chord in Peer-to-Peer Systems

Hathai Tanta-ngai


October 18, 2005

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada
An Expressway over Chord in Peer-to-Peer Systems

Hathai Tanta-ngai
Faculty of Computer Science
Dalhousie University
Halifax, Nova Scotia, Canada
hathai@cs.dal.ca

Abstract

We introduce an auxiliary coarse-grained routing layer (an expressway) on Chord, a DHT based structured peer to peer system. With the assumption that nodes in the system have different resource capacities such as storages and bandwidths, powerful (high connectivity and bandwidth) nodes can join the expressway to perform fast routing. First we design the logical structure of an expressway overlay. Then we explore the advantages of physical properties to superimpose the expressway on powerful nodes such as nodes nearby gateway routers.

We focus on the design and analysis of the logical structure of the expressway overlay. Expressway nodes maintain more routing entries that can forward requests with a longer per hop distance. The expressway defers the “last mile” fine-grained routing to the underlying system. Instead of using periodically update, we propose event-based notifications for membership and routing entry management on the expressway. Our initial experimental results show that the average logical path length of the system
when over 20% of nodes join the expressway with a forwarding power of 4 is about the same as when every node joins the expressway. The logical routing path lengths of the system improve up to \((1 - 2\left(\frac{p-1}{p}\right)\log_p 2) \times 100\%\) for an expressway with a forwarding power of \(p\). The system requires \(O(\log^2 r)\) messages to update all expressway routing entries when a node joins or leaves the system, where \(r\) is the number of expressway nodes.

By exploring the advantages of node abilities in both logical and physical networks, we expect the expressway will help forwarding requests faster by reducing routing path lengths and using high bandwidth channels.
1 Introduction

The proliferation of computing power and internet technology increases numerous world-wide internet accesses. More powerful computing devices, more internet connections, and higher network bandwidths enable direct collaboration among end users. The demands of end users for data sharing and content distribution applications drive the interests in building peer-to-peer (P2P) systems.

In P2P systems, end users form direct communications and collaborate among each other without having a centralized server. Every end user node in the system plays an equal role in providing and requesting services. P2P systems are dynamic and autonomous. Nodes join and leave the system according to their interests. To maintain the system connectivity, each node keeps contacts of a set of other nodes in the system forming a network of P2P systems. There are two kinds of P2P systems; unstructured and structured. In unstructured P2P systems, nodes maintain their contacts randomly based on their interests or on heuristics knowledge. In structured P2P systems, nodes are assigned identifiers, IDs, in a predefined system ID space and maintain their contacts based on a system of predefined rules.

Finding information or services in a large-scale P2P system is nontrivial due to the complexity of constructing the global state. In this report, the notion of “hop” refers to the number of nodes in the system that a request or a message is sent along the path to the destination (logical hop). Structured P2P systems based on a Distributed Hash Table (DHT) such as Chord [12], CAN [10], Pastry [11], and Tapestry [16] provide scalable and efficient routing algorithms to locate the node responsible for a service in large-scale P2P systems. Given a service key such as a file name or a URL, Chord, Pastry, and Tapestry can resolve the service location in $O(\log n)$ hops where $n$ is the number of nodes in the system. These systems are based on the assumption that every
node has uniform storage capacity and uniform bandwidth.

In P2P systems, although every node is equal in functionality, resources that each node can contribute to the system can vary. With the assumption of uniform resources, the routing performance of the system is limited to the minimum resources that any node can provide.

A number of techniques have been proposed to improve the routing performance by allowing nodes to manage different amounts of resources. The topology-aware expressway [13] and Brocade [15] create an additional overlay, called an expressway, that consists of powerful nodes on top of an existing system. The expressway nodes forward messages on behalf of nodes in the underlying system that register with them. While an expressway node can route messages to a destination faster than the underlying system, the expressway node creates a single point of failure between an expressway and the underlying system. eCAN [14] creates shortcuts to a different ID space to improve the routing performance. However, the shortcuts are created along with the evolution of the system, where nodes are assigned additional resources according to their placement in the ID space and not according to the availability of their resources. SmartBoa [6] allows nodes to maintain different sizes of routing tables, however the tables are mainly used for the first routing hops. If there is no limit on resources, each node can keep in contact with every other node in the system. In this case each node can forward a request to the destination in a constant number of hops [5], but each node has to keep track of every other node in the system.

We explore the possibility of improving the routing performance in a structured P2P system with limited resources to improve the overall system response time to locate services, with the assumption that nodes can provide different amount of resources according to their availability.

We state the objective of this project in Section 2 and describe background and re-
lated work in Section 3. Our expressway structure, its expressway routing algorithm, and its functionality are described in Section 4. The theoretical analysis of the expressway is discussed in Section 5. We discuss the results from initial simulations of the expressway in Section 6. The theoretical comparisons of our expressway with others are discussed in Section 7. Finally, conclusions and future work are given in Section 8.

2 Objective

We develop a P2P routing environment where nodes can contribute different amounts of resources. We expect powerful nodes that have significant resources to help improving the average routing path length of the whole system, while nodes with lower resources can benefit with small extra effort. Our new routing environment is built on the notion of an expressway with the concern of removing a single point of failure between the expressway and the underlying system.

As the existing systems with limited uniform resources already provide efficient routing algorithms with $O(\log n)$, we expect to improve the system average routing path length, but in the same asymptotic order of the existing systems. Even with a reduction of a single hop on an overlay routing of a P2P system, we reduce the number of messages sent over the internet.

This report focuses on the logical design of the expressway overlay and its performance.
3 Background and Related Work

3.1 Finding Services in P2P Systems

A straightforward approach to find services in a P2P system is using a centralized directory to provide mapping between services and their locations; such an approach is used in Napster [9]. Unfortunately, the centralized directory faces scalability and reliability problems as it creates a bottleneck in the community. To eliminate these problems, nodes in the system collaborate among each other to locate services without using the centralized directory.

3.1.1 Unstructured P2P Systems

Unstructured P2P systems like Gnutella [4] use a broadcast technique to search for services. Nodes issue a request to all of nodes in their contacts. Upon receiving a request, nodes return the location of the requested service to the requester if they have information of the requested service’s location. Otherwise they continue broadcasting the request to other nodes. Gnutella also faces with scalability problems because broadcast messages consume high bandwidth and require many nodes to cooperate in finding a service’s location. Limiting the number of hops for forwarding requests helps reduce the load on the system, but a request might not be able to find the service’s location within the predefined hop limit.

Creating a set of powerful nodes, called supernodes, to work on behalf of other nodes, such as in the FastTrack protocol [3], can remove the bottleneck and reduce broadcast messages. Each generic node connects to a supernode to send search requests and to publish services. The supernode collaborates with other supernodes to answer requests for its connected generic nodes. The broadcast messages, hence, are
limited to a set of supernodes. Nonetheless, searches require the collaboration of many supernodes, flood messages across networks, and might not get results within the pre-defined hop limit.

Another approach is to use heuristic search, as is done in Freenet [1, 2]. Each node keeps statistics about nodes that can answer seen queries. When a node receives a request, it tries to answer the request if the node has related information. Otherwise, the node forwards the request to a node having high probability of answering that request. When replying, the address of the node that can answer the request is included in the reply message. This address is used in updating the statistics about answering similar requests by the node in the future. The heuristic search requires less bandwidth and achieves higher scalability than the previous approaches. However, due to the limited storage, nodes can not keep statistic information for all queries. The information of unpopular services might be removed and these services may finally become unreachable. Searching algorithms in an unstructured P2P system causes the system to face its limitations, which include efficiently locating services and guaranteeing that locations of existing services will always be found under non-failure conditions. The next generation of P2P systems, namely structured P2P systems, solves these limitations.

3.1.2 Structured P2P Systems

In structured P2P systems [10, 11, 12, 17], each node is assigned a randomly uniformly distributed identifier, e.g. the hash of its IP address, mapped to a system predefined ID space. Each endpoint service is defined with a *key* that is also hashed to an ID in the system predefined ID space, called a *key ID*. A node publishes or requests a service using its key ID and routes the message to the node responsible for the given ID, which provides the meeting point between a node providing a service and a node seeking the service. The node responsible for the key ID can either keep the location
of the service or provide the service itself, such as through caching. Each system has a predefined distributed hash table (DHT) function to map between a set of IDs and the nodes responsible for those IDs. Each system also has a deterministic routing algorithm to route a request with a given ID to the responsible node. When a node joins, it transfers its responsibility from the previous responsible node. When a node leaves, it transfers its responsibility to the next responsible node in the system.

Examples of these systems are Chord [12], Pastry [11], Tapestry [16], and CAN [10]. Chord, Pastry, and Tapestry map nodes in a one-dimensional ID space. Chord maps the responsibility of each ID to the node having the closest ID higher than or equal to the given ID circularly. In Pastry, the node having ID closest to a given ID is responsible for that ID. Tapestry maps the responsibility of each ID using surrogate routing which routes a request to the node responsible for the request ID, called root node. The surrogate routing routes the request with prefix routing closer to the request ID one digit at a time. If there is no digit match for the request in the node’s contact list, the request is forwarded to a node having to closest digit to the request ID. When the request reaches a node where itself is the only node that the message can be forwarded, that node is designated as the root node of the requested ID.

Each node in these systems maintains $O(\log n)$ routing entries plus additional extra tables, which are different among the systems. These systems can resolve a request of a given ID to its responsible node in $O(\log n)$ hops.

Unlike the Chord, Pastry and Tapestry, CAN maps nodes in a $d$-dimensional virtual coordinate space. Each node has a virtual identifier (VID) that represents its zone. CAN maps a VID to a point in the coordinate space. The node that owns the zone where the point lies is responsible for that VID. Each node in CAN maintains a routing table of size $2d$ and requires $O(dn^{1/d})$ hops to resolve a request of a given VID.

The main advantages of these systems are their abilities to locate services efficiently
and to guarantee finding available services under non-failure conditions. The systems have high scalability and reliability as each node maintains only information in its responsibility. The routing is distributed among nodes in the network with deterministic paths. As a result, finding services in structured P2P systems consume less bandwidth than in unstructured P2P systems. The systems adapt to dynamic environments and provide good load-balancing.

3.2 Chord

Chord [12] is a well-known DHT-based structured P2P system. Chord’s nodes are defined in a one-dimensional $m$-bit ID space and are organized as a uni-directional ring. The node that is responsible for a service key ID is the node having the closest ID higher than or equal to the key ID circularly, called the key’s successor node. The key’s successor node is responsible for maintaining locations (IP addresses) of nodes that provide services associated with that key. Nodes can publish and find a service location by using a lookup function to route the request to the service key’s successor node. Each node maintains the location of the closest node whose ID is higher than its own ID circularly, called the node’s successor. Nodes then can route a lookup request with a given key ID by forwarding the request to their successors until the key’s successor node is reached. We use the terminology “node $x$” and “key $k$” interchangeably with “node with ID $x$” and “key with ID $k$" respectively.

To provide efficient routing, each node maintains a routing table of $m$ entries called the finger table. Each entry $i$ of a finger table at node $x$ holds the ID and the location of the closest node whose ID is higher than $x$ and at a distance of at least $2^{i-1}$ from $x$ circularly, where $1 \leq i \leq m$. Figure 1 shows a structure of Chord from node 7’s
Chord: 6-bit ID space (64 IDs); 18 peers

Figure 1: A node 7 (N7)’s view of Chord and its finger table.

view and its finger table. The solid arrows point to the closest IDs of each entry in N7’s finger table. The extended dash arrows from each solid arrow point to the contact nodes in each entry.

Nodes resolve a lookup request for a key $k$ by routing the request to the node in their finger tables whose ID immediately precedes $k$ until the key’s successor is reached. A node returns its successor as the key’s successor node if the successor ID is greater than or equal to the key ID. Figure 2 shows a routing path when node 7 requests for a service with key 59, which has node 63 as the key successor node.

Since each node maintains finger table entries at a distance of a power of two around the circular ID space, each node forwards a request at least half of the remaining distance between the node and the destination node. As a result, a request is resolved within $m$ steps or $\log_2 N$, where $N$ is the number of ID in the ID space. However, in a system with $n$ nodes, after $2(\log_2 n)$ hops the distance between the current forwarding
- N7 seeks for a service having key ID 59, which is mapped to N63.

![Figure 2: A routing path in Chord.](image_url)

request and the target is reduced to at most $2^{m - 2 \log_2 n}$, which is $2^m / n^2$. With the assumption that nodes are uniformly randomly distributed across the ID space, the probability that there is a node in this interval is $1/n$. As a result, Chord resolves a request within $O(\log_2 n)$ hops, for an $n$-node system. On average, Chord needs $(\frac{1}{2}) \log_2 n$ hops to resolve a request [12].

When a node joins the system, it contacts its successor by issuing a lookup request with its own ID. The lookup function returns the location of its immediate successor. Then, the new node transfers all the locations of services corresponding to its ID from its successor. When a node leaves the system, the node transfers the locations of services that it maintains to its successor. Other nodes in the system learn about joining or leaving nodes by periodically updating their finger tables. Failed nodes are also detected and fixed as parts of the periodical update. Chord needs $O(\log^2 n)$ hops for each node to update its finger table by using the lookup function to identify a node that
corresponds to each entry.

3.3 Routing Improvement over Structured P2P Systems

Although, the generic structured P2P systems can resolve a lookup request in \(O(\log n)\) messages. A number of investigations have been conducted to reduce the number of lookup messages to improve the lookup’s response time such as expressway routings, an irregular sized first-hop routing, and one hop routing.

The eCAN [14] was proposed to improve CAN’s routing performance by modifying the routing table with different forwarding spans, called an expressway. The expressway partitions the entire CAN-coordinate space into zones of different forwarding span. The expressway partitions are created as a snapshot of the evaluation of CAN zones. Nodes that join the system first keep contacts with each other as they develop their new zones. Later they use their contacts as an expressway to forward messages directly without passing through their neighbour zones, which are used in CAN. The drawbacks of eCAN are that an expressway node is assigned according to the evolution of zones that the node belongs to, not the capabilities of the node itself, and the improvement of the expressway is limited by the evolution of the CAN zones.

Two other existing expressways are the topology-aware expressway [13] and Brocade [16]. Both use powerful nodes to built an auxiliary overlay on top of the structure P2P systems. The expressway nodes relay requests on behalf of other nodes and act as directories of nodes in the underlying P2P systems that are physically located near them. The topology-aware expressway builds its overlay based on Autonomous-System-level topology extracted from BGP report or based on the “landmark numbering strategy”. Each expressway node periodically advertises a summarization of nodes in its physical proximity to its neighbour expressway nodes and routes the messages
based on a distance-vector-based route advertisement. Brocade builds its expressway overlay as its underlying system consisting only expressway nodes in a smaller ID spaces. Both the topology-aware expressway and Brocade reduce the network traffic for routing messages, but the expressway nodes themselves may create bottlenecks between the expressway and the underlying system. Moreover, the topology-aware expressway may suffer under a dynamic environment as the distance-vector routing takes time to converge routing information to a stable state and might be face with the count-to-infinity problem.

SmartBoa [6] uses heterogeneity in node bandwidths to provide flexibility for each node to adjust the size of its routing table and its updating cost differently. Nodes are divided into different levels according to their bandwidths. The higher bandwidth nodes take care of higher costs of updating routing information indicated by their routing-table size. When the membership of nodes in the system changes, the updated status messages are multicasted from nodes having higher bandwidth to nodes having lower bandwidth according to contacts in their routing tables. To route a message to the node responsible for a service, a node uses the contacts in its routing table for the first hop. The larger routing-table size the longer the first jump can be. The later hops are routed using a technique similar to Chord. The routing table in SmartBoa is used for the local node to route requests only for its first hop. As a result, SmartBoa only provides a fast first hop to forward a request.

One Hop Lookups for P2P overlay [5] was introduced to reduce a number of hops for a lookup request down to one hop. Each node in the system maintains in contact with every other node in its routing table. The correctness of the routing tables is maintained by using event-base notifications, which are distributed to every node hierarchically. The system ID space is divided into equal-size intervals. The node having its ID in the mid-point of each interval is called the leader. Each ID interval is
recursively divided into smaller intervals and has sub-leaders of sub-intervals. A notification message is propagated from the leader in the top hierarchy down to leaders in the smallest intervals, then the messages are sequentially forwarded to other nodes in the smallest intervals. As each node has contacts of every other node, the lookup request can be directly forwarded to the destination in one hop. In this system, each node keeps track of every other node, which creates high maintenance cost. The leaders, which are decided by the system, might create a bottleneck when nodes having low performance are selected to be the leaders.

4 Our Approach: an Expressway

We design a logical structure of the expressway by allowing nodes that have significant resources to maintain more routing table entries and handle more messages in the system. We built the expressway on top of a structured P2P system with one-dimensional ID space such as Chord [12]. Expressway nodes maintain routing entries with longer per hop distance to route requests in the same ID space as in the underlying system. Our expressway differs from previous ones in that it does not contain a directory of other nodes and the population of expressway nodes does not depend on a node’s ID. Nodes voluntarily join and leave the expressway and stay in the underlying system depending on their resource availability.

The expressway routes a request with a coarse-grained routing to other expressway nodes and defers the last hops fine-grained routing to the underlying system. Expressway nodes maintain extra cost only for their expressway routing tables. They do not maintain extra responsibility for keys. Mapping a key to the node responsible for the key does not change from the underlying routing system. We uses the underlying system to guarantee lookup correctness and maintain system connectivity. We create an
expressway over Chord [12] to demonstrate the expressway functionality and the expressway routing performance as Chord provides a simple interface and has similar characteristics as the other DHT-based structured P2P systems [11, 17].

4.1 An Expressway over Chord

4.1.1 Expressway Structure

The expressway nodes form a ring as in Chord. Nodes join the expressway with their own IDs and keep their statuses in the underlying system to maintain the system connectivity and their responsibilities of keys. Each expressway node maintains a pointer to its expressway successor, expSucc, its expressway predecessor, expPred, and an additional routing table called an expressway finger table.

![Expressway Structure Diagram](image_url)

An expressway with a forwarding power of 4 over Chord.

<table>
<thead>
<tr>
<th>Entry</th>
<th>The interval ID</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,0)</td>
<td>(8, 9)</td>
<td>N12</td>
</tr>
<tr>
<td>(2,0)</td>
<td>(9, 10)</td>
<td>N12</td>
</tr>
<tr>
<td>(3,0)</td>
<td>(10, 11)</td>
<td>N12</td>
</tr>
<tr>
<td>(1,1)</td>
<td>(11, 15)</td>
<td>N12</td>
</tr>
<tr>
<td>(2,1)</td>
<td>(15, 19)</td>
<td>N17</td>
</tr>
<tr>
<td>(3,1)</td>
<td>(19, 23)</td>
<td>N21*</td>
</tr>
<tr>
<td>(1,2)</td>
<td>(23, 39)</td>
<td>N37*</td>
</tr>
<tr>
<td>(2,2)</td>
<td>(39, 55)</td>
<td>N46*</td>
</tr>
<tr>
<td>(3,2)</td>
<td>(55, 7)</td>
<td>N56*</td>
</tr>
</tbody>
</table>

- A suffix '*' denotes an expressway node contact.

Figure 3: An expressway with a forwarding power of 4 over Chord in a 6-bit ID space system from an expressway node 7 (N7)’s view.

For an expressway in a system with an $m$-bit ID space, an expressway node $x$
maintains the contacts of expressway nodes in the interval \([x, x + p^{i+1}]\) where \(0 \leq i \leq (\log_p 2^m) - 1\). For each interval, the expressway finger table of node \(x\) contains \(p - 1\) entries that divide the interval \([x, x + p^{i+1}]\) into \(p\) equal subintervals. The value \(p\) is called the forwarding power of the expressway. The expressway with a forwarding power of \(p\) forwards a request at a distance of \(p^i\) per hop. Chord is a system with a forwarding power of 2.

An expressway finger table entry is denoted as entry \((a, i)\), where \(1 \leq a \leq (p - 1)\) and \(0 \leq i \leq (\log_p 2^m) - 1\). For an expressway node \(x\), its expressway finger table entry \((a, i)\) maintains the ID and address of the closest expressway node whose ID higher than \(x\) in the interval \([x + ap^i, x + (a + 1)p^i]\). The expressway finger table contains both expressway and non-expressway nodes. Non-expressway nodes in the expressway finger table provide contact points to defer requests to the underlying system in ID ranges where no expressway nodes exist. If there is no expressway node in the \([x + ap^i, x + (a + 1)p^i]\) interval, \(x\) stores the closet node \(y\) whose ID is higher than or equal to \(x + ap^i\), where \(y\) could be a non-expressway node or an expressway node.

Figure 3 shows an example of an expressway with a forwarding power of 4 over Chord in Figure 1 (Section 3.2) from node 7 (N7)’s view. Expressway finger table entries \((1, 0), (2, 0), \ldots, (2, 1)\) hold non-expressway nodes, because there is no expressway node whose ID is in their interval. Expressway finger table entries \((3, 1), (1, 2), (2, 2),\) and \((3, 2)\) holds the closest expressway nodes whose ID higher than node 7 in each ID interval, which are 21, 37, 46, and 56 respectively.

### 4.1.2 Routing over the Expressway

Routing over the expressway is similar to the prefix routing in Chord [12]. The expressway performs high-level routing and uses the underlying P2P system to guarantee the correctness. As nodes join the expressway with their own ID and keep their sta-
tuses in Chord, routing a lookup request to expressway nodes whose IDs are closer to the key ID result in directing the request closer to the key’s successor in the underlying system.

An expressway node, N7, seeks for a service having key 59, which is mapped to N63.

Figure 4: A routing path over the expressway with a forwarding power of 4.

Expressway nodes resolve a lookup request for a key $k$ by routing the request over the expressway to the node $x$ whose ID immediately precedes $k$ in their expressway finger tables. If $x$ is an expressway node, the request continues in the expressway. If $x$ is a non-expressway node, there is no expressway node between $x$ and the destination and the request is forwarded to the underlying system via $x$ to the destination. Expressway nodes defer requests to the underlying system only when the next hop is a non-expressway node in their expressway routing tables. Hence, non-expressway nodes route requests only in the area where no expressway nodes exist before the destination.
Figure 4 shows a routing example when node 7 seeks for a service having key 59 over the expressway in Figure 3. Node 63 is responsible for key 59. Node 7 routes the request to node 56 whose ID immediately precedes 59 in node 7’s expressway finger table (see Figure 3). As the closest node whose ID preceding 59 in node 56’s expressway finger table is non-expressway node 58, the request is deferred to the underlying system via node 58. Node 58 returns its successor as the key’s successor node, node 63.

Expressway nodes offer an additional function, `expSuccLookup`, that maps the request ID of an `expSuccLookup` to the closest expressway node whose ID is higher than or equal to the requested ID circularly, called the ID’s `expSucc`. The `expSuccLookup` function is used to build expressway entry points and expressway finger table entries, as detailed in Section 4.1.3 and 4.1.5 respectively. Expressway nodes route the `expSuccLookup` request similar to routing a lookup request, except that the `expSuccLookup` request is not deferred to the underlying system. Instead, expressway nodes forward the `expSuccLookup` request to the expressway node whose ID immediately precedes the `expSuccLookup` request’s ID in their expressway routing tables until the request reaches the expressway node preceding the ID’s `expSucc`, at which the expressway node returns its `expSucc` as the ID’s `expSucc`.

### 4.1.3 Accessing the Expressway

Expressway nodes can automatically access the expressway as they are part of it. Non-expressway nodes need to have one or more contact points to expressway nodes to forward their lookup requests over the expressway. Each non-expressway node maintains additional routing entries called `expressway entry points` in their routing table. The number of expressway entry points can vary in each non-expressway node.
A non-expressway node defines its expressway entry points at different distances in the ID space. An expressway entry point $i$ of a non-expressway node $x$ with a distance $d_{x,i}$ keeps the location of the closest expressway node whose ID is higher than $x$ by at least $d_{x,i}$. The expressway node for each entry is found by issuing an \texttt{expSuccLookup} request with ID $x + d_{x,i}$ to a known expressway node—such as an expressway contact of $x$’s successor. Moreover, to keep their expressway entry points up-to-date, non-expressway nodes periodically refresh these entry points and update their entry points when they detect faulty entries, as detailed in Section 4.1.8.

The different expressway entry points give choices to non-expressway nodes. Non-expressway nodes can insert a request into the expressway at the entry point whose ID precedes and is closest to the key ID, which knows more about the existence of nodes near the key’s successor.

4.1.4 Synchronization among expressway nodes

Expressway nodes ensure the correctness of their expressway successors by executing a stabilization protocol, denoted as the \texttt{expStabilize} protocol, which is similar to Chord’s stabilization protocol [12]. Unlike a periodic stabilization protocol in Chord, the expStabilize protocol is called when an expressway node learns that there is a membership change in the system such as when a node joins or leaves the expressway. By issuing the expStabilize protocol, an expressway node not only ensures the correctness of its expSucc contact, but also becomes aware of the correctness of its previous contact when its expSucc contact is updated. Moreover when an expressway node learns that the relationship with its predecessor changes, the expressway node updates its predecessor contact and notifies its old predecessor to run the expStabilize protocol. As a result, the protocol will perform synchronization along the path from the old predecessor to the new predecessor.
Algorithm 1 \( x.expStabilize(z) \)

Ensure: \( x.expSucc.expPred = x \) and \( x.expSucc \in (x, z] \)

if \( z == \) null then
    \( z = x.expSucc.notifyExpPred(x) \)
end if

if \( z == x \) then
    do nothing
else if \( z \in (x, x.expSucc]) \) then
    if \( (z \in (x, x.expSucc)) \) then
        temp = x.expSucc
        x.expSucc = z
        z.notifyExpSucc(temp)
    end if
end if

x.expStabilize(null)

else if \( (x.expSucc \in (x, z)) \) then
    x.expStabilize(null)
    x.expSucc.expStabilize(z)
end if

Algorithms 1, 2, and 3 show pseudocode of the expStabilize protocol, namely the expStabilize, notifyExpPred, and notifyExpSucc functions. The expStabilize function is called with an input parameter \( z \), where \( z \) is an expressway node. If \( z \) is null, the synchronization is only performed between \( x \) and \( x \)'s expSucc, otherwise the synchronization will be called to ensure the relationship of expSucc and expPred along the path between \( x \) to \( z \).

Algorithm 2 \( x.notifyExpPred(z) \)

Ensure: \( x.expPred \in [z, x) \)

if \( (z \in (x.expPred, x) \) or \( (x.expPred = \) null)) then
    temp = x.expPred
    x.expPred = z
    if \( (temp != \) null) then
        temp.expStabilize(x.expPred)
    end if
end if
return x.expPred

When an expressway node \( u \) calls the expStabilize function with a null pa-
rameter, $u$ calls $\text{notifyExpPred}$ (Algorithm 2) to notify $u$’s expSucc, $x$, that $u$ might be $x$’s expPred. If $x$’s expPred is $null$ or if $x$ learns that $u$ is its expPred, $u \in (x.\text{expPred}, x)$, then $x$ updates its expPred to $u$ and notifies its previous expPred to run the $\text{expStabilize}$ function with parameter $u$ to stabilize the path to $u$. At the end, $x$ returns its current expPred to $u$. Hence $u$ will get back the updated expPred, $z$, of its expSucc, $z \in [u, u.\text{expSucc})$. After that, $u$ learns if there is a new expressway node in between $u$ and $u$’s expSucc, $z \in (u, u.\text{expSucc})$, then $u$ updates its expSucc and synchronizes with its new expSucc, otherwise the synchronization is done as $u$ is an expPred of $u$’s expSucc.

As expressway nodes do not run the $\text{expStabilize}$ function periodically, $u$ needs to make sure that there is a node aware of its previous expSucc. Node $u$ notifies its new expSucc to be aware of its previous expSucc with the $\text{notifyExpSucc}$ function, which updates the expSuccs along the path until the previous expSucc of $u$ is included in the expSucc path. The pseudocode of the $\text{notifyExpSucc}$ function is shown in Algorithm 3. When a node $x$ is notified that $z$ might be its expSucc, if $z \in (x, x.\text{expSucc})$ then $x$ will update its expSucc to $z$ and will notify $z$ that the previous $x$’s expSucc might be $z$’s expSucc. If $x.\text{expSucc} \in (x, z)$, then $x$ continues to notify its expSucc to be aware of $z$. At the end, $x$ runs the $\text{expStabilize}$ function to ensure the correctness between $x$ and its expSucc.

When an expressway node $u$ calls the $\text{expStabilize}$ function with parameter $z$, $u$ assumes that $z$ might be expPred of $u$’s expSucc. The expressway node $u$ synchronizes with $z$ as its does when the $\text{expStabilize}$ function is called with $null$. However, there might be a case when $z$ is not in between $u$ and $u$’s expSucc, $u.\text{expSucc} \in (u, z)$, such as when an expressway node keeps a wrong contact and notifies to the wrong node. In this case, $u$ calls $\text{expStabilize}$ function with $z$ along its expSucc path until reaches the expressway node $x$ where $z \in (x, x.\text{expSucc})$ at which
Algorithm 3 $x.notifyExpSucc(z)$

**Ensure:** There is a node $x$ such that $x.expSucc = z$

```plaintext
if ($x.expSucc == z$) then
    do nothing;
else
    if ($z \in (x, x.expSucc))$ then
        temp = $x.expSucc$
        $x.expSucc = z$
        $x.expSucc.notifyExpSucc(temp)$
    else if ($x.expSucc \in (x, z)$) then
        $x.expSucc.notifyExpSucc(z)$
    end if
    $x.expStabilize(null)$
end if
```

point its propagation ends and $x$ synchronizes with $z$. As a result, the synchronization is performed along the path from $u$ to $z$. Figure 5 shows the relationship of an expressway node before and after expressway node $N15$ calls the $expStabilize(null)$.

By calling the $expStabilize$ function, an expressway node $x$ is guaranteed that $x$ is the expPred of $x$’s expSucc. The $expStabilize$ function can only solve conflicts for the expressway node having ID in the interval of $x$ and its expSucc. The synchronization will be run along the path from an expressway node $u$ to an expressway node $z$ only when there is an expressway node $y$ updates its expPred from $u$ to $z$.

According to [7], the expressway ring is in a *weakly ideal* state, if for all nodes $x$ in the system, $x$ is an (expressway) predecessor of $x$’s (expressway) successor. The expressway ring is in the *strongly ideal* state, if for all (expressway) nodes $x$ there is no (expressway) node $y$ such that $y$ having ID in between $x$ and $x$’s (expressway) successor.

We cannot guarantee that after running the expStabilize protocol, the expressway ring will be in weakly ideal state. To achieve the weakly ideal state, each expressway node should periodically issue the $expStabilize$ function. Moreover, to guarantee
that the expressway ring is in the strongly ideal state, each expressway node should run Chord’s idealization protocol[7] on the expressway ring. The idealization protocol requires $O(r^2)$ rounds to make an arbitrary connected expressway ring becomes strongly ideal, where $r$ is the number of expressway nodes in the system. Our synchronization protocol face with the problem when there are nodes in between $x$ and $x$’s expSucc, that can not be reached by any node in the synchronization path—nodes inside a loop situation, such as N7 in figure 5(g). When N15 runs the expStabilize protocol, the result is shown in figure 5(h), where N15 keeps the wrong expPred’s contact. In this case the conflict will be solved when N7 calls the expStabilize function.

For our choice of design, as the expressway is an auxiliary overlay and the lookup correctness is guaranteed by the underlying system, expressway nodes will run the expStabilize function only when they learn that their contacts are wrong. In ad-
dition, with our synchronization protocol, each expressway node is always aware of updating other expressway nodes when an expressway node relationship is changed. As a result the nodes inside a loop situation should rarely happen. However, the probability that this situation will occur and the effect of incorrect contacts on the routing performance need to be further analyzed.

4.1.5 Joining and Leaving the Expressway

When a node \( x \) joins the expressway, \( x \) needs to notify other expressway nodes of its existence. First, \( x \) calls \( \text{expSuccLookup} \) with its own ID to find its \( \text{expSucc} \). After that \( x \) adds itself to the expressway by updating its \( \text{expSucc} \) contact and setting its \( \text{expPred} \) to \( \text{null} \). Then \( x \) calls the \( \text{expStabilize} \) function with \( \text{null} \) to synchronize with its \( \text{expSucc} \), which will then synchronize the path between the previous \( \text{expPred} \) of \( x \)’s \( \text{expSucc} \) and \( x \) as shown in Figure 5 when N15 is the new join node.

After new expressway node \( x \) finds its \( \text{expSucc} \) and runs the \( \text{expStabilize} \) protocol, \( x \) initializes its expressway finger table by issuing \( \text{expSuccLookup} \) requests with the smallest ID for each expressway finger entry interval ID to its \( \text{expSucc} \). If the returned expressway node is not in the interval ID of that entry, \( x \) finds the closest node for that entry by issuing a lookup request with the same ID. After that, \( x \) notifies other expressway nodes that need to update their entries as \( x \) joins the expressway as detail in Section 4.1.6.

When an expressway node \( x \) leaves the expressway, it notifies its \( \text{expPred} \) \( u \) and \( \text{expSucc} \) \( v \). Upon receiving the notification that \( x \) is leaving, \( u \) updates its \( \text{expSucc} \) to \( v \) if its current \( \text{expSucc} \) is equal to \( x \) and then runs the \( \text{expStabilize} \) protocol with \( v \). Similarly, upon receiving the notification from \( x \), \( v \) updates its \( \text{expPred} \) to \( u \) if its current \( \text{expPred} \) is equal to \( x \) and waits for the synchronization from \( u \). The synchronization then resolves the conflict of relationship among \( \text{expSucc} \) and \( \text{expPred} \) of the accessible
nodes along the path from $u$ to $v$. After that, $u$ notifies the other expressway nodes that have $x$ in their expressway finger table entries to update their contacts according to $u$’s expSucc, as detail in Section 4.1.6.

There is no key responsibility transferred when expressway nodes join or leave the expressway because the mapping function of a key to the key’s successor is not changed. The expressway only helps to forward requests and is not responsible for the key mapping of other nodes.

4.1.6 Notifying other expressway nodes to update their expressway finger tables

As mention in Section 4.1.4, the expressway maintains its membership by event-based notification. To maintain the expressway finger table according to membership changes in the system, expressway nodes that need to be updated should be notified. In an expressway with a forwarding power of $p$, each expressway node $x$ keeps the contacts of the closest expressway node whose ID higher than $x$ in the interval $[x + ap^i, x + (a + 1)p^i]$ in its expressway finger table entry $(a, i)$. The expressway nodes that need to be notified according to the joining or leaving node $y$ are the expressway nodes $x$ such that $y$ is the closest node whose ID is higher than $x + ap^i$ in its an expressway finger entry $(a, i)$. We call this expressway nodes target nodes. In other words, the target nodes are expressway nodes $x$ such that $x \in (y.\text{expPred} - ap^i, y - ap^i]$, where $1 \leq a \leq (p - 1)$ and $0 \leq i \leq (\log_p 2^m) - 1$. Figure 6 shows the target nodes that should be notified when N0 joins or leaves the expressway with a 6-bit ID space and a forwarding power of 4. The arrows show expressway finger table entries of the target nodes that are affected by N0.

To notify the target nodes about its existence, node $y$ does not have contacts of all target nodes, so we create a hierarchical notification distribution. The target nodes are divided into $(\log_p 2^m)$ hierarchical levels according to their IDs. The notification is
distributed to the target nodes from level \((\log_p 2^m) - 1\) down to level 0 or until we reach the expPred of \(y\). In each level \(i\), the expressway node having the highest ID less than or equal to \((y - p^i)\) is defined as a leader node \(l_i\) that is responsible for forwarding the notification to the next level.

The notification is generated from the joining node or the expPred of the leaving node as the leader, \(l_{\log_p 2^m}\), for the first notification level. Since expressway nodes are uniformly randomly distributed across the ID space, there might not be an expressway node in some levels. Each leader then learns which notification level \(i\) it should notify from the distance of its expSucc and the changing node \(y\) such that \(i = \lfloor \log_p (y - x_{\text{expSucc}}) \rfloor\). At each notification level \(i\), the leader generates \(p - 1\) notifications to expressway nodes with the target ID \((y - ap^i)\), denoted \(t(a, i)\), where \(1 \leq a \leq (p - 1)\). The notification with the target ID \(t(a, i)\) will be distributed among target nodes having ID in the interval \((y - (a + 1)p^i, y - ap^i]\). The table in Figure 7 shows the target ID \(t(a, i)\) and the target ID interval according to the expressway in Figure 6 as node N0.
joins the system.

As each peer does not have the global information of the system of which expressway nodes exist in the system, the leader node $x$ generates a notification to each target ID $t(a, i)$ according to the knowledge in its expressway finger table. For each notification to the target ID $t(a, i)$, the leader $x$ forwards the notification to the expressway node in its expressway finger entry $(b, j)$ where $t(a, i) \in [x + bp^j, x + (b + 1)p^j]$. If the expressway finger entry $(b, j)$ does not contain the contact of an expressway node, $x$ forwards the notification to the closest expressway node having ID higher than $t(a, i)$.

As shown in the table in Figure 7, column Leader:contact represents the leader and the expressway contact for each target ID $t(a, i)$ according to the knowledge in the leader’s expressway finger table.

![Figure 7: A notify distribution when membership changes as N0 joins the expressway.](image)

<table>
<thead>
<tr>
<th>$t(a, i)$</th>
<th>Target ID interval</th>
<th>Leader: contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(3,2)=16$</td>
<td>$(0, 16]$</td>
<td>N0: N18</td>
</tr>
<tr>
<td>$(2,2)=32$</td>
<td>$(16, 32]$</td>
<td>N0: N34</td>
</tr>
<tr>
<td>$(1,2)=48$</td>
<td>$(32, 48]$</td>
<td>N0: N49</td>
</tr>
<tr>
<td>$(3,1)=52$</td>
<td>$(48, 52]$</td>
<td>N46: N52</td>
</tr>
<tr>
<td>$(2,1)=56$</td>
<td>$(52, 56]$</td>
<td>N46: N55</td>
</tr>
<tr>
<td>$(1,1)=60$</td>
<td>$(56, 60]$</td>
<td>N46: N58</td>
</tr>
<tr>
<td>$(3,0)=61$</td>
<td>$(60, 61]$</td>
<td>N60: N62</td>
</tr>
<tr>
<td>$(2,0)=62$</td>
<td>$(61, 62]$</td>
<td>N60: N62</td>
</tr>
<tr>
<td>$(1,0)=63$</td>
<td>$(62, 63]$</td>
<td>N60: N0</td>
</tr>
</tbody>
</table>

When an expressway node $z$ gets the notification message of target ID $t(a, i)$ from an expressway node $x$ about the joining or leaving of node $y$, node $z$ checks if its expressway finger table needs to be updated. If $z$ is not affected by the changing node $y$ and if $z \in (x, t(a, i))$, then $z$ continues to forward the notification to expressway
nodes having ID closest to the target ID according to the knowledge in its expressway finger table in the same way as the leader. If \( z \) is not affected by the changing node \( y \) and if \( t(a, i) \in (x, z) \), which means the notification is overshoot, then \( z \) sends the notification backward to its expPred. If \( z \) needs to be updated, the notification reaches a target node where at least one of its expressway finger entries needs to be updated according to \( y \). If \( z \)'s expSucc and expPred are in the \( t(a, i) \) interval, they might be the target node as well. Consequently, \( z \) forwards the notification to its expSucc and expPred until they are not affected by \( y \). Upon getting the notification, an expressway node \( z \) also checks that if it is the leader—the closest node having ID less than or equal to \( t(a, i) \)—then \( z \) will generate the notification to the next level.

Figure 7 shows an example of an expressway node \( N_0 \) joining the system and notifying other expressway nodes of its existent. \( N_0 \) starts the notification distribution as a leader \(^1\) \( l_3 \), which notifies \( N_{18}, N_{34}, \) and \( N_{49} \) with the target ID 16, 32, and 48 respectively. Since \( N_{18}, N_{34}, \) and \( N_{49} \) overshoot according to the notified target IDs, they forward the notification back to their expPreds. \( N_{15} \) and \( N_{31} \) update their expressway finger tables, so they continue to forward the notification to their expPreds, which are not affected by \( N_0 \), then the notification is terminated. \( N_{46} \) does not update its expressway finger table but it is the leader \( l_2 \) of level 2, which is responsible for forwarding the notification to the next level (level 1). \( N_{46} \) notifies \( N_{52}, N_{55}, \) and \( N_{58} \) for the target ID 52, 56, and 60 respectively. \( N_{52} \) updates its expressway finger and forwards the notification to \( N_{49} \), which is affected by the \( N_0 \), so the notification stops. \( N_{55} \) updates its expressway finger table but does not forward the notification to \( N_{52} \) or \( N_{58} \) because \( N_{52} \) and \( N_{58} \) are not in the target ID interval of 56. By getting the notification for the target ID 60 from \( N_{46}, N_{58} \) is not affected by \( N_0 \) but its ID is between

\[^1\]As the expressway is created in 6-bit ID space with a forwarding power of 4, the leader starts with the level \( \log_4 2^6 = 3 \)
46 and 60, so N58 forwards the notification to N60. N60 then updates its expressway finger table and, as the leader \( l_1 \), N60 forwards the notification to N62. N62 is the expPred of node 0, so the notification distribution is done.

4.1.7 Updating Non-expressway Entries in an Expressway Finger Table

In addition to expressway nodes, there are non-expressway nodes that are promoted into expressway finger tables. Expressway nodes learn the statuses of these non-expressway nodes with a periodic update. The expressway nodes can poll at the same period as in the underlying system. The expressway nodes might set the period of the update time according to how well they want to learn about the underlying system. An expressway node also updates its non-expressway entry when it learns that the non-expressway node in its contact has failed or its contact cannot be reached.

4.1.8 Handling Faulty Routing Entries

In a dynamic environment, nodes may leave the expressway or the system or may fail without notice. An expressway finger table entry or an expressway entry point that refers to such a node is called a faulty entry.

When node \( x \) uses a faulty entry that refers to an expressway node to route a request to a non-expressway node \( y \), node \( y \) may ignore the request or send a “back-off” message to \( x \). If \( y \) ignores the request it will be identified as a failed expressway node. When \( x \) detects that a node fails or when \( x \) receives the “back-off” message, it reroutes the request to the next preceding node. If there is no preceding node, \( x \) defers the request to the underlying system by using its underlying routing table. Then, \( x \) refreshes the faulty entry in its expressway finger table by issuing an expSuccLookup request for that entry.
5 Theoretical Performance Analysis

We analyze the performance and cost of the expressway and compare with Chord to examine benefits of the expressway on the underlying system. Table 1 summarizes the analysis of the expressway. The details are discussed in the following sections.

Table 1: Theoretical performance analysis

<table>
<thead>
<tr>
<th></th>
<th>Chord</th>
<th>Expressway</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookup (hops)</td>
<td>$O(\log_2 n)$</td>
<td>$O(\log_p r)$</td>
</tr>
<tr>
<td>Avg. case lookup (hops)</td>
<td>$(\frac{1}{2})\log_2 n$</td>
<td>$(\frac{p-1}{p})\log_p r$</td>
</tr>
<tr>
<td>#Routing entries</td>
<td>$\log_2 2^m$</td>
<td>$(p-1)\log_p 2^m$</td>
</tr>
<tr>
<td>#Notification cost</td>
<td>-</td>
<td>$O(s + (p-1)\log_p^2 r)$</td>
</tr>
</tbody>
</table>

n is the number of Chord nodes
r is the number of expressway nodes
m is the number of bit in the ID space
p is the forwarding power of an expressway
s is the target nodes the sequentially notified

5.1 Routing performance

We analyze the routing performance of the expressway with a forwarding power of $p$ when using Chord as the underlying system. We have a conjecture that at each routing hop, an expressway node can forward a request with the distance equal to or greater than a non-expressway node. The conjecture can be true if all the entries of the Chord finger tables are maintained in the expressway finger tables such as the expressway with a forwarding power of a power of 2. With this constraint, expressway nodes will have contacts of nodes that have distances equal to or greater than the contacts of non-expressway nodes. As a result, expressway nodes have more knowledge about other nodes in the system than non-expressway nodes.
Expressway nodes prefer to keep contacts of other expressway nodes in their expressway finger table entries rather than non-expressway nodes. The expressway nodes can skip forwarding requests to intermediate non-expressway nodes in the routing paths, therefore, at each routing hop, expressway nodes can forward a request faster than non-expressway nodes. However, by lacking complete information of nodes in each ID interval (ignore non-expressway nodes in some conditions), an expressway finger entry is not useful if the routing destination is the non-expressway node whose ID is in the area where the expressway node ignore its existence. In this case, an expressway node forward the request to the next closest expressway node in its contracts with a lower distance. If this case is repeat multiple times until compensate the benefit of fast routing on the expressway in the previous hops, the expressway itself will degrade the routing performance. However, according to our experiment (Section 6), this case is hardly occur.

Next, we analysis the routing performance of the expressway when every node is on the expressway. The analysis leads us to understand the limitation of the improvement provided by the expressway itself. We have Lemma I and II.

Lemma I: With the assumption that nodes are regularly placed around the ID space ring and every node is on the expressway, the expressway with a forwarding power of \( p \) can resolve a request with the maximum path length of \( \log_p r \), where \( r \) is the number of expressway nodes.

Proof. Suppose that node \( x \) wishes to resolve a lookup request for a key \( k \). Let \( z \) be the node that immediately precedes \( k \)'s successor. We analyze the number of hops required to forward the request to reach \( z \).

Assume that \( x \neq z \), then \( x \) forward its request to the closest predecessor of \( k \) in its expressway finger table. Consider the entry \((a, i)\) such that the expressway node \( z \) is
in the interval \([x + ap^i, x + (a + 1)p^i]\). Since this interval is not empty, \(x\) will contact its \((a, i)\) expressway finger entry, the first node \(f\) in this interval. Because \(f\) and \(z\) are in the interval \([x + ap^i, x + (a + 1)p^i]\), \(f\) is closer to \(z\) than to \(x\), or equivalently. As each expressway node contains \(p - 1\) entries which divide each interval into \(p\) equal subintervals. The distance between \(f\) and \(z\) is at most \(p^{-1}\) of the distance between \(x\) and \(z\).

If the distance between the node handling for the request and \(z\) is reduced in a factor of \(p^{-1}\) in each hop, and is at most \(2^m\) initially, where \(m\) is the number of bit in the ID space, then within \(\log_p 2^m\) hops the distance will be one, meaning we have arrived \(z\). With the assumption that nodes are regularly placed around the ID space ring and every node is on the expressway, after \(\log_p r\) hops, the distance between the node handling for the request and \(z\) is \(2^m / r\), which contain one node in the distance. Hence we reaches \(z\) and the request is resolved, which leads to Lemma I.

\[\square\]

**Lemma II:** With the assumption that nodes are regularly placed around the ID space ring and every node is on the expressway, the expected path length of the expressway with a forwarding power of \(p\) is \((\frac{p-1}{p})\log_p r\), where \(r\) is the number of expressway nodes.

**Proof.** Suppose that node \(x\) wishes to resolve a lookup request for a key \(k\). We analyze the expected path length required to resolve the request.

From Lemma I, We have the expressway that can resolve all requests within \(\log_p r\) hops. Based on the analysis in [8], any path from source to destination on the expressway of length \(j\) is formed by drawing \(j\) unique elements from a set of \(p^i\) where \(0 \leq i \leq (\log_p r) - 1\). As a result, each expressway node can reach exactly \((\log_p r)\) expressway nodes at shortest hop \(j\). Since the probability that each hop is selected is \((\frac{p-1}{p})\), the probability mass function of the routing path length \(j\) is given by a binomial
distribution whose $j^{th}$ term is shown in Equation 1. Hence the expected path length of
the expressway is given by Equation 2, which is equal to $(\frac{p-1}{p})\log p r$. As a result, we
have Lemma II.

$$\binom{\log p r}{j} \left(\frac{p-1}{p}\right)^j \left(\frac{1}{p}\right)^{\log p r - j}$$

(1)

$$\sum_{j=1}^{\log p r} \binom{\log p r}{j} \left(\frac{p-1}{p}\right)^j \left(\frac{1}{p}\right)^{\log p r - j} = \left(\frac{p-1}{p}\right)\log p r$$

(2)

As the average path length of Chord is $\frac{1}{2} \log_2 n$ [12], an expressway with a forwarding power of $p$ will improve the performance for system routing path lengths up to
$(1 - 2(\frac{p-1}{p})\log_2 2) \times 100\%$, when $r$ is equal to $n$.

### 5.2 Maintenance cost

With the assumption that an expressway is failure free and that there is no node that leaves the system without notifying its expPred and its expSucc, the extra cost that an expressway node needs to consider is the maintenance cost of its expressway routing table. For an expressway with an $m$-bit ID space and a forwarding power of $p$, each expressway node needs to maintain $(p-1)(\log p \ 2^m)$ expressway finger table entries. As the expressway only maintains additional storage for expressway finger tables and their predecessors and successors, the cost of storage is negligible compared to the current computer performance \footnote{an expressway node with a forwarding power of 4 in a 128-bit ID space system need approximately 4 K bytes additional storage.}. The maintenance that we need to consider is the maintenance to keep the expressway finger table up to date, which is the notification distribution cost.
With the assumption that the expressway nodes are regularly placed around the ID space ring, we analyze the notification distribution cost of an expressway when a node joins or leaves, which leads to Lemma III.

**Lemma III:** With the assumption that the expressway nodes are regularly placed around the ID space ring, the notification distribution cost when a node joins or leaves an expressway with a forwarding of \( p \) is \( O(s + (p - 1)\log^2_p r) \), where \( s \) is the number of target nodes that are sequentially notified and \( r \) is the number of expressway nodes.

**Proof.** Suppose that the expressway nodes are regularly placed around the ID space ring, we analyze the notification distribution cost of an expressway when a node joins or leaves.

From Lemma I, the maximum path length of the expressway is \( \log_p r \), where \( r \) is the number of expressway nodes in the system. Then the number of notification level will be \( \log_p r \) levels. For each level, there are \( p - 1 \) target IDs, hence there are \( (p - 1)\log_p r \) target IDs that need to be notified when membership changes. For each target ID, as we do not have complete knowledge of the expressway nodes in the system, each target ID in a level requires at most \( \log_p r \) hops to find the target node. Each notification distribution requires \( (p - 1)\log^2_p r \) messages to find the first node in the target ID interval. Then the notification is sequentially distributed to other nodes in the same target ID interval. As a result the cost for the notification distribution is \( O(s + (p - 1)\log^2_p r) \), where \( s \) is the number of target nodes that are sequentially notified, which leads to Lemma III.

Comparing with periodic updates, if at every update interval each expressway node updates one of its expressway finger entry, the maintenance cost of the whole system will be \( O(r\log_p r) \) messages, and updating all expressway finger entries the system requires \( O(r\log^2_p r) \), given that \( O(\log_p r) \) messages are required to refresh each entry.
If the updating interval of all entries is more often that the half-life of the system\(^3\) [7] and if \(p \ll r\), then the notification requires less maintenance cost than periodic updates of all expressway finger tables.

6 Experimental Results

Initial routing simulations have been done to validate the effectiveness of our expressway over Chord. The expressway has a 32 bit ID space with a forwarding power of 4. Non-expressway nodes maintain the expressway entry points at the same distance as each entry in their finger tables. We test the routing performance and the maintenance cost of the system when nodes join.

6.1 Routing Performance

The routing performance of an expressway is tested in different static P2P environments where we varied the number of nodes populating the Chord ID space from 500 to 50,000 nodes. The routing performance of a request originated at expressway nodes and non-expressway nodes are tested separately. For each populating size, we vary the number of expressway nodes from 1% up to 100% of Chord nodes for a test on expressway nodes and vary the number of expressway nodes from 1% up to 99% of Chord nodes for a test on non-expressway nodes. For each test, we routed a randomly generated request in 10,000 random node placements for each configuration. Then we counted the number of hops per request and projected the average number of hops as an average path length of a lookup request.

Figures 8 and 9 show the average path length per request over 10,000 requests originated from an expressway node and from a non-expressway node respectively with a

\(^3\)The interval that the number of nodes increases double of its size or decreases half of its size.
standard deviation between 1.03 and 1.64 hops. The average path length when the percentage of nodes in the expressway equals zero shows the average path length of the underlying system, Chord. The results from tests from expressway and non-expressway nodes show the same behaviour, but requests originated from non-expressway nodes have a higher average path length because non-expressway nodes need an extra hop to forward their requests to the expressway. As the number of nodes in the expressway increases, the average path length required to resolve a request decreases.

When the number of expressway nodes is over 20% of the Chord nodes, the average path length of a lookup request is comparable to when every node joins the expressway. The raw data shows that the system achieves the minimum average path length when about 50% of nodes joining the expressway, which is not easily notice in the Figure 8 and Figure 9. This behaviour happens because expressway nodes keep contacts of non-expressway nodes in their expressway routing table entries where there is no
Figure 9: the average path length required to resolve a lookup request issued from a non-expressway node

expressway node exists in that entry interval. Consequently, when the number of expressway nodes is about 50\%, all non-expressway nodes between an expressway node and its expSucc are kept in its expressway finger table. Hence all nodes in the system can be reached by expressway nodes. Adding more expressway nodes in the system will not improve the average path length of the system, but will increase the number of hops on the expressway itself.

According to the theoretical analysis, an expressway with a forwarding power of 4 will reduce the average path length of the system up to 25\%\(^4\). However, this maximum improvement is not shown in our experiment because, in the theoretical analysis, we assume nodes are regularly placed around the ID space ring. In the simulation, the minimum distance between two nodes can be less than the expected minimum distance, as a result the maximum path length is greater than \(\log_p r\). When the number of nodes in

\[^4\text{For } p = 4: \left(1 - 2\left(\frac{2-1}{p}\right)\log_p 2\right) \times 100 = \left(1 - 2\left(\frac{4-1}{4}\right)\log_4 2\right) \times 100 = 25\% \text{ (see Section 5.1).}\]
the system increases, the standard deviation of minimum distance between two nodes decreases. As a result, the system improvement increases closer to the theoretical analysis when the number of nodes in the system increases.

For an expressway node, the expressway can improve the routing performance up to 21.64% in a system with 50,000 nodes as shown in Figure 8. The improvement of requests from non-expressway nodes also increases as the number of nodes in the system increases, but less than the improvement of requests from expressway nodes because the requests are not originated on the expressway, which is up to 17.63% in a system with 50,000 nodes as shown in Figure 9.

### 6.2 Maintenance cost

Table 2: The average number of notification messages when a node join an expressway

<table>
<thead>
<tr>
<th>ExpSize</th>
<th>#avg. notification messages</th>
<th>SD.</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>42.63</td>
<td>7.12</td>
</tr>
<tr>
<td>2500</td>
<td>59.40</td>
<td>10.47</td>
</tr>
<tr>
<td>5000</td>
<td>67.27</td>
<td>11.94</td>
</tr>
<tr>
<td>10000</td>
<td>76.68</td>
<td>13.60</td>
</tr>
<tr>
<td>15000</td>
<td>82.89</td>
<td>14.73</td>
</tr>
<tr>
<td>20000</td>
<td>87.58</td>
<td>15.75</td>
</tr>
<tr>
<td>25000</td>
<td>91.71</td>
<td>16.87</td>
</tr>
<tr>
<td>30000</td>
<td>95.49</td>
<td>18.16</td>
</tr>
<tr>
<td>35000</td>
<td>99.13</td>
<td>19.57</td>
</tr>
<tr>
<td>40000</td>
<td>102.50</td>
<td>20.50</td>
</tr>
<tr>
<td>45000</td>
<td>105.79</td>
<td>21.95</td>
</tr>
</tbody>
</table>

The number of notification messages when a node joins the expressway is tested as the maintenance cost of the expressway. We tested the notification cost on a system with 50,000 nodes and varied the number of expressway nodes from 500 to 45,000 nodes on the expressway. We randomly generated 10,000 node placements for each
configuration and added one node in to each placement. We measured the number of messages requires to notify others node when a new node joined in each placements. The test is done only on the system with 50,000 nodes because the notification is only distributed among expressway nodes. The results of the experiment are shown in Table 2. The average number of notification messages increases in $O(\log^2 r)$, where $r$ is the number of expressway nodes. In the current implementation, each node does not have the knowledge the predecessor of the join node. As a result, every time a node updates it expressway finger table, it forwards the notification messages to its neighbours without the knowledge that they might not be in the target nodes.

7 Expressway Comparison

Other expressway techniques that are similar to our work are Brocade [15] and the topology-aware expressway [13]. Brocade and the topology-aware expressway were proposed to reduce routing latency of each routing hop. We compare our expressway with the other two expressways with the assumption that expressway nodes are superimposed on the same physical nodes. The main properties of expressway nodes are nodes that have significant resources such as storage, connectivity and bandwidth, and sit nearby gateway routers.

7.1 Brocade versus Our Expressway

Brocade [15] uses a structured P2P overlay (Pastry) as an expressway structure. Non-expressway nodes register themselves to expressway nodes in their physical proximity. Expressway nodes then publish the IDs of the registered non-expressway nodes as their objects in the expressway overlay. Routing requests over Brocade is the same as in the underlying system with a smaller search space. The source expressway node
uses an overlay routing to route a request to the destination’s expressway node. Then the destination’s expressway node relays the request to the final destination using the underlying routing system.

Table 3 shows a routing performance comparing between Brocade and our expressway. We divide an analysis into 3 parts: 1.) when a request is forwarded to an expressway, 2.) when a request is routed on an expressway, and 3.) when a request is relayed to the underlying system to the destination. As our expressway and Brocade use logical overlay routings and with the assumption that expressway nodes in both system locate at the same place in physical network, we analysis routing performance in logical hops. Moreover, we assume that both expressways can forward requests with a forwarding power of $p$ and Brocade overlay is analyzed as it was implemented using Chord; an expressway over Chord with a forwarding power of $p$ and no non-expressway node promoted.

Table 3: The routing analysis of our expressway versus Brocade in logical hops

<table>
<thead>
<tr>
<th></th>
<th>Brocade</th>
<th>our expressway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward a request to an expressway node (hops)</td>
<td>$O(\log_p r)$</td>
<td>$1$</td>
</tr>
<tr>
<td>Routing a request on an expressway (hops)</td>
<td>$O(\log_p r)$</td>
<td>$O(\log_p r)$</td>
</tr>
<tr>
<td>Relaying a request from an expressway node to destination (hops)</td>
<td>$2 + c^*$</td>
<td>$O(\log_2 \frac{n}{r_p})$</td>
</tr>
</tbody>
</table>

$c^*$ depends on the refresh rate

By using a structured P2P overlay as an expressway structure, routing over Brocade and our expressway are similar except that the techniques that are used to relay requests to and from the expressway are different. Both use only one hop to forward a request to the expressway. However, Brocade’s non-expressway nodes forward requests to the closest expressway nodes in their physical proximity, but our non-expressway nodes forward requests to the expressway nodes whose IDs closet to IDs of their expressway
entry points. As a result, Brocade’s first hop aims to forward a request to the closest expressway node in physical distance, while our expressway aims to advance a request closer to the request’s destination in the logical ID space.

On the expressway overlay, both Brocade and our expressway use the same routing algorithm to route requests closer to the destination in the logical ID space which required $O(\log_p r)$ hops where $r$ is the number of expressway nodes. Moreover, as expressway nodes in both systems are superimposed on the same physical nodes, we expected the physical routing path lengths of Brocade’s and our expressway are about the same.

To defer a request to the underlying system, Brocade publishes all non-expressway nodes in its expressway overlay. A request is routed on the expressway until it reaches the expressway node that is responsible for keeping the location of the destination expressway node, $x$, that is expected to be the expressway node nearby the destination in the underlying system. Then the request is forwarded to $x$ and then the destination. Since the system is dynamics, if the refresh rate of the expressway knowledge is not synchronous with the underlying system, the destination might not locate in the same administrative domain as the destination expressway node (when a node joins or leaves the system, nodes that responsible for related service keys are changed). The request is routed to the final destination using the underlying system routing. As a result, Brocade requires 2 and a few more hops to locate the final destination when a request is deferred to the underlying system.

In our expressway, expressway nodes promote non-expressway nodes to keep in their routing tables. When a request reaches the area where there is no expressway node to the destination, the request is deferred to the underlying system. With the assumption that nodes are regularly place around the ID space, the distance where a request need to be forwarded in the underlying system to the destination is $(n/rp)$. Hence, the
system requires $O(\log_2 \frac{n}{rp})$ hops to forward a request in the underlying system to the destination.

Table 4: The maintenance and overhead cost of Brocade versus Our Expressway

<table>
<thead>
<tr>
<th></th>
<th>Brocade</th>
<th>our expressway</th>
</tr>
</thead>
<tbody>
<tr>
<td># Routing entries</td>
<td>$(p - 1)\log_p 2^m$</td>
<td>$(p - 1)\log_p 2^m$</td>
</tr>
<tr>
<td># Registered node entries</td>
<td>$n/r$</td>
<td>-</td>
</tr>
<tr>
<td># Published non-expressway node entries</td>
<td>$n/r$</td>
<td>-</td>
</tr>
<tr>
<td># non-expressway node entries that need periodically update</td>
<td>-</td>
<td>$O(\log_2 \frac{p}{n})$ (only promoted nodes)</td>
</tr>
<tr>
<td>Periodically update registered nodes</td>
<td>yes</td>
<td>-</td>
</tr>
<tr>
<td>Building up expressway routing entries</td>
<td>-</td>
<td>$O((p - 1)\log_p 2^m r)$</td>
</tr>
<tr>
<td>Notification when membership change*</td>
<td>-</td>
<td>$O(s + (p - 1)\log_p 2^m r)$</td>
</tr>
<tr>
<td>Periodically update expressway* routing entries</td>
<td>$O(r(p - 1)\log_p 2^m r)/\text{period}$</td>
<td>-</td>
</tr>
<tr>
<td>finding expressway node</td>
<td>DNS-like</td>
<td>logical lookup $O(\log_p r)$</td>
</tr>
</tbody>
</table>

* the number of total messages in the system (not per node).

Table 4 shows the maintenance and overhead cost comparison between Brocade and our expressway. For an $m$-bit ID space, an expressway with a forwarding power of $p$ for both Brocade and our expressway require the same number of routing entries to maintain contacts of other nodes, which is $(p - 1)\log_p 2^m$ entries. In Brocade, each expressway nodes require on average $n/r$ entries to keep contracts of registered nodes and additional on average $n/r$ entries to keep information of non-expressway nodes which are published for expressway routing. Moreover, the registered nodes need to be periodically updates to monitor their existence in the system. In our expressway, non-expressway nodes are promoted in expressway routing tables. Hence, we do not require extra space for keeping non-expressway entries. However, the promoted non-expressway nodes in routing table entries, which are on average $O(\log_2 \frac{p}{n})$ entries, need
to be periodically updated.

Our expressway maintains membership changes using event-based notification. When a node joins or leaves, the system requires \( O(s + (p - 1)\log_p^2 r) \) messages to be distributed among expressway nodes. In addition, an expressway node need to build up its expressway routing table when it first joins the expressway which requires \( O((p - 1)\log_p^2 r) \) messages. On the other hand, Brocade expressway nodes maintain their memberships by using periodically up date. Each expressway node requires \( O((p - 1)\log_p^2 r) \) messages per period to update all entries. In total the system with \( r \) expressway nodes requires \( O(r(p - 1)\log_p^2 r) \) messages per period. Brocade new expressway nodes do not need to build up their expressway routing entries when they join the expressway as their expressway routing entries will be builded in part of periodical updates.

Next we compare the overhead that non-expressway nodes require to find its expressway contacts. Brocade uses DNS-like system to locate the closest expressway node for each non-expressway node. In our expressway, non-expressway node uses the expressway routing to locate an expressway contact for each expressway routing entry, which requires \( O(\log_p r) \) hops per entry. Our expressway is more scalable for a non-expressway node to add additional expressway contacts as expressway nodes do not publish information of each node-expressway node that have their contacts in the expressway overlay.

7.2 **Topology-Aware-Expressway versus Our Expressway**

In the topology-aware-expressway [13], non-expressway nodes also register themselves with expressway nodes nearby in the similar way as Brocade, but its expressway structure is different. The expressway nodes in the topology-aware-expressway
keep contacts of other expressway nodes that are physically nearby. The distance-vector-base route advertisement is used in the expressway to advertise registered non-expressway nodes. A summarization technique is proposed to reduce the routing state maintenance. Instead of advertise each non-expressway node individually, nodes that are close to each other in P2P ID space are aggregated to be advertised together. To route a request, each expressway node forwards the request to its neighbour expressway node that has a non-expressway node having ID closer to the requested ID. When no such a node exist, the expressway node relays the request to the underlying system.

With its expressway structure design, the topology-aware-expressway will forward requests among expressway nodes in their physical proximity. However, a number of hops that is required for a request to be routed to the destination when the request is relayed to the underlaying system depends on the summarization advertisement of registered non-expressway nodes. For broaden summarization, longer hops are needed in the underlying routing system. With distance-vector-base advertisement, the amount of routing state maintenance will be increased dramatically as the summarization is finer-grained. As identifier of nodes in the underlying system is randomly-uniformly distributed, summarization might not be efficient. Experiments [13] show that topology-aware outperform an expressway with an overlay structure. However, the maintenance cost of the system in a dynamic environment is still an open issue.

8 Conclusions and Future Work

We proposed an expressway with a hierarchical overlay in P2P systems where nodes can provide different resources and can seamlessly collaborate between each level without having a single point of failure at the expressway. We divide the tasks for nodes with higher resources to provide fast coarse-grained routing by maintaining a
large routing table. Nodes in the underlying system take care of the small routing table entries but maintain the connectivity and guarantee the lookup correctness for the fine-grained routing. The expressway creates a routing environment where nodes can contribute different amounts of resource. As long as both expressway and the underlying system use the same distance metrics to map between nodes and key’s responsibility, our expressway can be replaced by other P2P system topologies.

Our expressway design takes advantages of nodes that can provide significant resources to improve routing path length in both logical and physical distance. In the logical ID space, the expressway node keeps additional routing tables that can forward a request with a longer per hop distance. In the physical network, expressway nodes are superimposed on nodes that have high connectivity and bandwidth that sit nearby the gateway routers. As a result, our expressway can fast forward a request to the destination in the logical ID space with fast forwarding channels of expressway nodes, and our expressway skip routing to non-expressway nodes in the edge of physical networks.

We design, analyze and implement an expressway logical overlay on Chord. Initial routing simulations show that the expressway with a forwarding power of 4 reduces the average lookup path length up to 17.63% for non-expressway nodes and 21.64% for expressway nodes. When the number of expressway nodes is over 20%, the system can achieve about the same average path length as when all nodes are on the expressway. We also proposed a new notification method to maintain the membership of expressway nodes instead of using periodical updates as in the underlying system. The notification requires $O(s + (p - 1)\log_p r)$ messages to distribute among expressway nodes for a single membership change.

We show theoretical comparisons of our expressway with Brocade and the topology-aware-expressway. Our expressway routing performance is comparable with Brocade. However, our design does not require the registration process and periodical updates
for expressway maintenance and overhead cost as it does in Brocade. Based on the experiment in [13], the topology-aware-expressway routing outperforms our expressway. However, with the design of our expressway that does not require periodical update, we expect that we can improve performance of our expressway overlay in logical hops up to $O(1)$ with low maintenance cost. With this expectation, it is interesting to be further investigate the performance of logical expressway routing versus the topology-aware-expressway.

In the future, we will study the effect of the physical network on expressway overlay design parameters such as $p$ and $r$, and the system physical routing path length and latency. We can extend an expressway as a muti-level expressway, however, the number of layer and additional cost need to be justified.

**References**


