

A peer-to-peer expressway over Chord

Hathai Tanta-ngai, Michael McAllister*

Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada

Received 2 February 2006; accepted 6 February 2006

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. We design the logical structure of an expressway overlay.

We focus on the design and analysis of the logical structure of the expressway overlay that is parameterized by a characteristic called the forwarding power p . Expressway nodes maintain more routing entries that can forward requests with a longer per hop distance than the underlying system. The expressway defers the “last mile” fine-grained routing to the underlying system. We also propose an event-based notification for membership and routing entry management on the expressway. Our initial experimental results of an expressway with a forwarding power of 4 show that the average logical path length of the system when over 20% of nodes join the expressway is about the same as when every node joins the expressway. The theoretical analysis shows that the logical routing path lengths of the system improve up to $1 - 2(\frac{p-1}{p}) \log_p 2$. 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.

© 2006 Elsevier Ltd. All rights reserved.

Keywords: Structured peer-to-peer system; Overlay routing; Peer-to-peer routing; Heterogeneous resources

1. Introduction

We develop a P2P hierarchical routing environment in a structured Peer-to-Peer (P2P) system where nodes can contribute different quantities of resources. Our routing environment is built on the notion of *an expressway*—an overlay of powerful nodes that provide fast routing performance. We allow powerful nodes to contribute more resources on the expressway to improve the routing performance for all nodes, while nodes with limited resources benefit with small extra effort.

To find information or services in a large-scale P2P system, nodes forward a request of a service among each other until the location of the service is resolved. We count one “logical hop” each time a node forwards a request to another node. In this paper, “hop” refers to the “logical hop” not the physical hop that a node forwards a request in the physical network, unless explicitly stated otherwise.

Structured Peer-to-Peer (P2P) systems based on a Distributed Hash Table (DHT) such as Chord [1], Pastry [2], Tapestry [3], and CAN [4] provide scalable and efficient overlay routing to locate services in large-scale P2P systems.

* Corresponding author. Tel.: +1 902 494 3151; fax: +1 902 492 1517.

E-mail addresses: hathai@cs.dal.ca (H. Tanta-ngai), mcallist@cs.dal.ca (M. McAllister).

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, and CAN requires $O(n^{1/d})$ hops to resolve the service, where n is the number of nodes in the system and d is the dimension of the CAN space. Moreover, with unlimited resources, each node can keep the contact of every other node in the system. Then each node can forward a request to its destination in a number of constant hops [5]. Although they demonstrate an efficient routing performance, these systems assume that every node has uniform storage and bandwidth.

With uniform resources, the routing performance of the system is limited by the minimum resources that any node can provide. In P2P systems, although every node is equal in functionality, the resources contributed by each node can vary. We explore a possibility to take advantage of node heterogeneity to improve routing performance in overlay networks.

Currently, powerful nodes, which possess more resources, contribute extra resources primarily through virtual nodes [1]. By increasing the number of virtual nodes, the routing path length of the system increases as the number of active identifiers (IDs) increases. Moreover the impact of virtual nodes in the aspect of improving the routing performance is not clearly defined.

With *SmartBoa* [6], a non-uniform resources approach, nodes maintain different sizes of routing tables. The extra entries are used to plan the first hop. The later hops are routed using a technique similar to Chord. As a result, *SmartBoa* only provides a fast first hop to forward a request. Furthermore, there is no experimental result or theoretical analysis of its performance and maintenance cost.

Previous work on expressways assign nodes to the expressway based on their IDs in the P2P system [7] or have the expressways act as directories where expressway nodes relay requests on behalf of other nodes [8,9]. In the former case, nodes have no control over whether or not they are on the expressway. In the latter case, a system requires registrations between nodes in the underlying system and expressway nodes. Expressway nodes also need to publish/unpublish nodes in their registration on the expressway overlay. Moreover, an expressway node creates a single point of failure between an expressway and nodes in the underlying system. Failure on an expressway node disables the accessibility of its registered nodes to the expressway.

We propose an expressway where nodes join and leave the expressway voluntarily depending on their resource availability. Expressway nodes do not behave as a representative of a set of nodes, instead they have more knowledge than nodes in the underlying system. Expressway nodes route a request among each other with a longer forwarding distance closer to the destination until the request reaches an area where there is no expressway node in the routing path. Then, they defer the last routing hops to the underlying system.

As our expressway routes requests in the same ID space as its underlying system, requests that are deferred from the expressway can continue to be delivered to the destination with the distance they left from the expressway without restart routing messages in a new ID space. There is no single point of failure in our expressway. The expressway is designed as an auxiliary overlay where the underlying system guarantees the routing correctness.

We create an expressway over Chord [1] 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 [2,10]. Moreover, by justifying the role of an expressway as an auxiliary overlay and using the underlying system to guarantee the lookup correctness, we propose a new notification method for the expressway over Chord to maintain its membership changes.

The introduction of the expressway does not aim to overcome other techniques that aim to optimize the degree-diameter properties of structure P2P graphs, such as *de Bruijn graph* or *Trie* [11], instead we want to extend the limitation of uniform design to allow nodes to contribute their resources differently to improve overlay routing performance.

We focus on design and analysis of an expressway overlay in term of its performance and effects on the overlay network. The initial simulation shows that when over 20% of nodes join the expressway with a forwarding power of 4, the average logical path length of the system is about the same as when every node joins the expressway. On average, an expressway with a forwarding power of 4 improves the routing performance by up to 21.64% over the underlying system. The system generates $O(\log_p^2 r)$ messages among expressway nodes to update related expressway routing tables according to a single membership change, where r is the number of nodes on the expressway.

The main contributions of this work are (1) a logical design of an expressway overlay on a structured P2P system that does not require registration and publishing of nodes in the underlying system to the expressway and (2) an

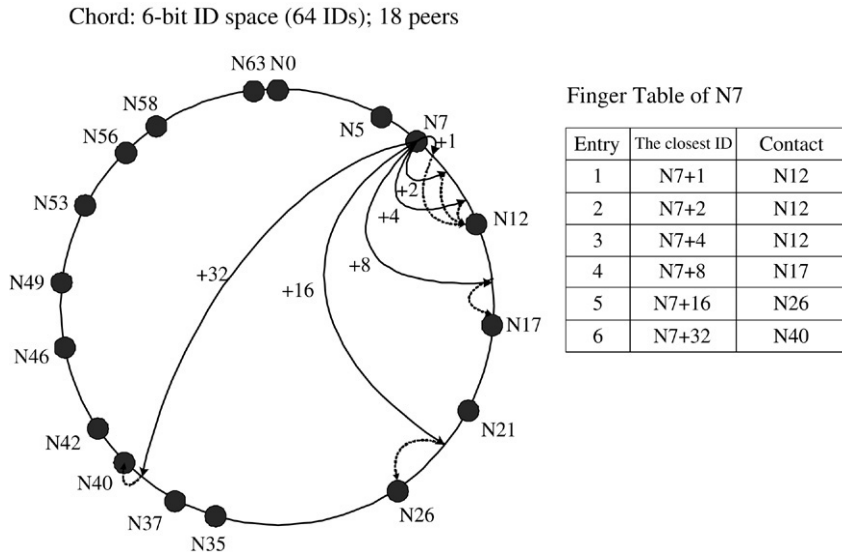


Fig. 1. Node 7 (N7)'s view of Chord and its finger table.

event-based notification for maintaining membership in the expressway, which is introduced as an alternative to a periodic update that is used in the underlying system.

The outline of this paper is as follows. Section 2 describes Chord's structure and functionality. Our expressway structure, its routing algorithm, and its functionality are provided in Section 3. Section 4 provides an analysis of the expressway routing performance. The results from initial simulations are discussed in Section 5. The theoretical comparisons of our expressway with others are discussed in Section 6. Finally, conclusions and future work are given in Section 7.

2. Chord

Chord [1] provides a scalable lookup service for a large-scale P2P system. Nodes are defined in a one-dimensional m -bit ID space and are organized as a uni-directional ring. Each service is defined with a *key* that is hashed into the same ID space as the node IDs. Each node maintains a set of service locations—IP addresses of nodes that provide services. Chord provides a lookup function that maps a key to the node that maintains the service's location. Each node publishes or locates a service by mapping the service's key to the responsible node.

The node that is responsible for a key is the node having the smallest ID equal to or greater than the key circularly, called the key's *successor*. 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 route each lookup request to their successors until the key's successor is reached. We use the terminology "node x " and "node with ID x " interchangeably.

To provide efficient routing, each node maintains a routing table of m entries called the *finger table*. Entry i at node x holds the ID and the address of the closest node whose ID is higher than x and at a distance of at least 2^{i-1} from x circularly, where $1 \leq i \leq m$. Fig. 1 shows node 7 (N7)'s view of Chord and its finger table. To resolve a lookup request for a key k , nodes route the request to the node in their finger tables whose ID immediately precedes k until the key's successor is reached. For a system with n nodes, the request can be located within $O(\log n)$ hops. On average, Chord expects $\frac{1}{2} \log_2 n$ hops to resolve a lookup request.

When nodes join or leave the system, they notify their successors and transfer the locations of services for which they are responsible to their successors. While in the system, each node periodically polls the nodes in its finger table to maintain connectivity.

3. An expressway over Chord

We present a logical design and analysis of an expressway over Chord. An expressway consists of nodes that have significant resources and bandwidth. The expressway provides a short-cut to forward requests to expressway

An expressway with a forwarding power of 4 over Chord.

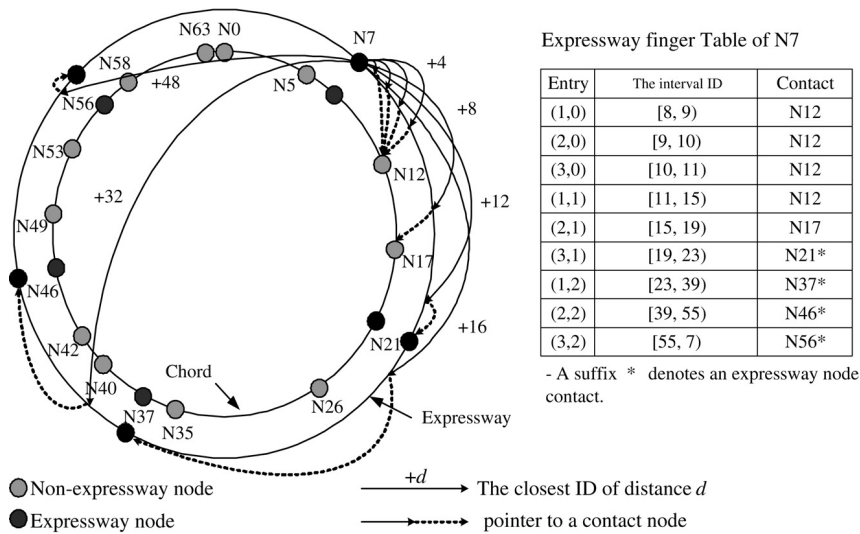


Fig. 2. An expressway with a forwarding power of 4 over Chord in a 6-bit ID space system from an expressway node 7 (N7)’s view.

nodes alone with a longer per hop distance than non-expressway nodes. At a point where there are no expressway nodes along the path to the destination, requests are delivered to the underlying routing system, which ensures routing correctness.

3.1. 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, and its expressway predecessor. In the context of the expressway, we will refer to “expressway successor” and “expressway predecessor” as “successor” and “predecessor” respectively. Each expressway node keeps an additional routing table called an *expressway finger table* to route requests on the expressway and forward the last hops to the underlying system. Row i of the expressway finger table for node x contains $p - 1$ entries that divide the interval $[x, x + p^i]$ 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.

For an expressway with a forwarding power of p , each expressway node maintains an expressway finger table with $(p - 1) \log_p 2^m$ entries. 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 is 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 are promoted in expressway finger tables to provide contact points to defer requests to the underlying system. 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 $x + ap^i$ in the underlying system, where y could be a non-expressway node or an expressway node.

Fig. 2 shows an example of an expressway with a forwarding power of 4 over Chord from node 7 (N7)’s view. Expressway finger table entries (1, 0), (2, 0), . . . , (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 is higher than node 7 in each ID interval, which are 21, 37, 46, and 56 respectively.

3.2. Routing over the expressway

Routing over the expressway is similar to prefix routing in Chord [1]. 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

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

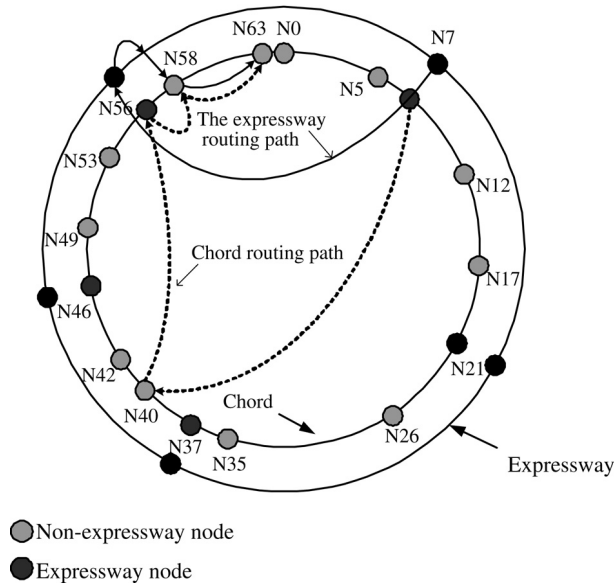


Fig. 3. A routing path over the expressway with a forwarding power of 4 vs. Chord.

and maintain their status in Chord, routing a lookup request to expressway nodes whose IDs are closer to the key ID results in directing the request closer to the key’s successor in the underlying system. Such routing also provides a capability to skip forwarding requests to non-expressway nodes, which have lower bandwidth.

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, then 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 hops are non-expressway nodes in their expressway routing tables. Hence, non-expressway nodes route requests only in the area where no expressway node exists before the destination.

Fig. 3 shows a routing example from node 7 to key 59 over the expressway in Fig. 2 versus the routing path in Chord. Node 63 is responsible for key 59. The dashed arrows show Chord’s routing path, while the solid arrows show the expressway’s routing path. With the expressway routing, node 7 routes the request to node 56 whose ID immediately precedes 59 in its expressway finger table (see Fig. 2). Now the closest node whose ID precedes 59 in node 56’s expressway finger table is non-expressway node 58 and the request is deferred to the underlying system via node 58. Node 58 returns its successor, node 63, which is the key’s successor.

Expressway nodes offer an additional function, `expSuccLookup`, that maps the request ID of an `expSuccLookup` request 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 Sections 3.3 and 3.5 respectively. Expressway nodes route the `expSuccLookup` request similar to routing a lookup request, except that the request is not deferred to the underlying system.

3.3. 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 `expSuccLookup` request with ID $x + d_{x,i}$ to a known expressway node—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 3.8.

The different expressway entry points give choices to non-expressway nodes. Non-expressway nodes can choose to 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.

3.4. Synchronization among expressway nodes

Expressway nodes ensure the correctness of their expressway successors by executing a stabilization protocol, denoted as the `expStabilize` protocol, which is similar to Chord's stabilization protocol [1]. We design an event-based synchronization on an expressway overlay to avoid polling in a stable state. Unlike a periodic stabilization protocol which is used 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 $x.expStabilize(z)$ as run at node x

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

```

Algorithms 1 to 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 .

When an expressway node u calls the `expStabilize` function with a `null` parameter, u calls `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`, $u \in (x.expPred, x)$, then x updates its `expPred` to u and notifies its previous `expPred` to run the `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`, $z \in [u, u.expSucc)$. After that, u learns if there is a new expressway node in between u and u 's `expSucc`, $z \in (u, u.expSucc)$, 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`.

Algorithm 2 $x.\text{notifyExpPred}(z)$ as run at node x

```

Ensure:  $x.\text{expPred} \in [z, x)$ 
if ( $z \in (x.\text{expPred}, x)$  or ( $x.\text{expPred} = \text{null}$ )) then
   $\text{temp} = x.\text{expPred}$ 
   $x.\text{expPred} = z$ 
  if ( $\text{temp} \neq \text{null}$ ) then
     $\text{temp}.\text{expStabilize}(x.\text{expPred})$ 
  end if
end if
return  $x.\text{expPred}$ 

```

Algorithm 3 $x.\text{notifyExpSucc}(z)$ as run at node x

```

Ensure: There is a node  $x$  such that  $x.\text{expSucc} = z$ 
if ( $x.\text{expSucc} == z$ ) then
  do nothing;
else
  if ( $z \in (x, x.\text{expSucc})$ ) then
     $\text{temp} = x.\text{expSucc}$ 
     $x.\text{expSucc} = z$ 
     $x.\text{expSucc}.\text{notifyExpSucc}(\text{temp})$ 
  else if ( $x.\text{expSucc} \in (x, z)$ ) then
     $x.\text{expSucc}.\text{notifyExpSucc}(z)$ 
  end if
   $x.\text{expStabilize}(\text{null})$ 
end if

```

As expressway nodes do not run the `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 `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 `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 `expStabilize` function to ensure the correctness between x and its `expSucc`.

When an expressway node u calls the `expStabilize` function with parameter z , node u assumes that z might be `expPred` of u 's `expSucc`. The expressway node u synchronizes with z as it does when the `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 the `expStabilize` function with z along its `expSucc` path until it reaches the expressway node x where $z \in (x, x.\text{expSucc}]$ at which point its propagation ends and x synchronizes with z . As a result, the synchronization is performed along the path from u to z . Fig. 4 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 which updates its `expPred` from u to z .

Liben-Nowell et al. [12] define a ring to be in a *weakly ideal* state, if for all nodes x in the system, x is an (expressway) predecessor of x 's (expressway) successor. They define a ring to be in a *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 a weakly ideal state. To achieve the weakly ideal state, each expressway node should periodically issue the `expStabilize` function.

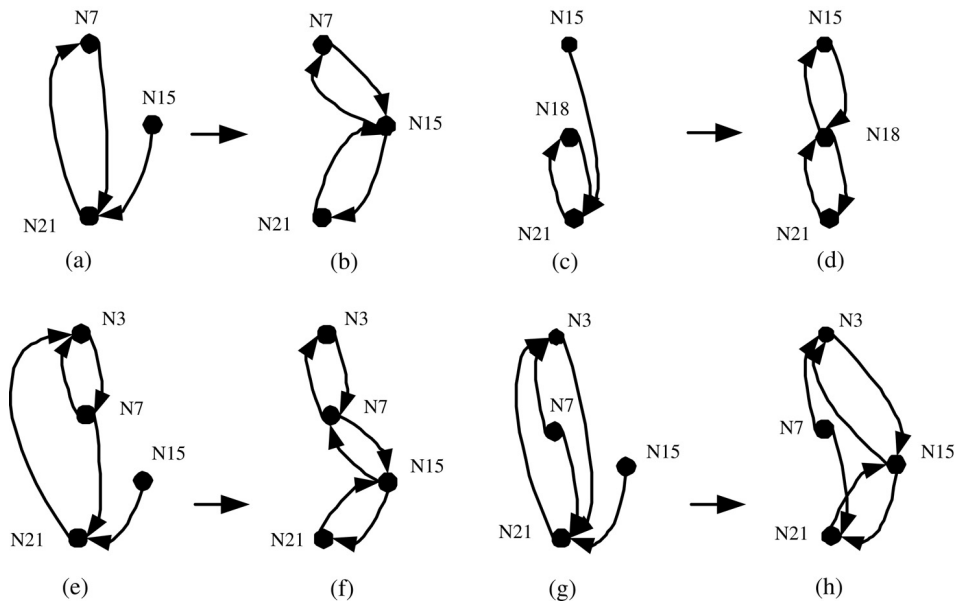


Fig. 4. The relationship of expressway nodes before and after N15 calls the `expStabilize(null)`, which is changed from (a, c, e, and g) to (b, d, f, and h) respectively.

Moreover, to guarantee that the expressway ring is in the *strongly ideal* state, each expressway node should run Chord's idealization protocol [12] 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 is faced with the problem when there are nodes in between x and x 's `expSucc` that cannot be reached by any node in the synchronization path—nodes inside a loop situation, such as N7 in Fig. 4(g). When N15 runs the `expStabilize` protocol, the result is shown in Fig. 4(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 addition, 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.

3.5. 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 `expSuccLookup` with its own ID to find its `expSucc`. After that x adds itself to the expressway by updating its `expSucc` contact and setting its `expPred` to `null`. Then x calls the `expStabilize` function with `null` to synchronize with its `expSucc`, which will then synchronize the path between the previous `expPred` of x 's `expSucc` and x as shown in Fig. 4 when N15 is the new join node.

After new expressway node x finds its `expSucc` and runs the `expStabilize` protocol, x initializes its expressway finger table by issuing `expSuccLookup` requests with the smallest ID for each expressway finger entry interval ID to its `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 detailed in Section 3.6.

When an expressway node x leaves the expressway, it notifies its `expPred` u and `expSucc` v . Upon receiving the notification that x is leaving, u updates its `expSucc` to v if its current `expSucc` is equal to x and then runs the `expStabilize` protocol with v . Similarly, upon receiving the notification from x , v updates its `expPred` to u if its current `expPred` is equal to x and waits for the synchronization from u . The synchronization then resolves the conflict of

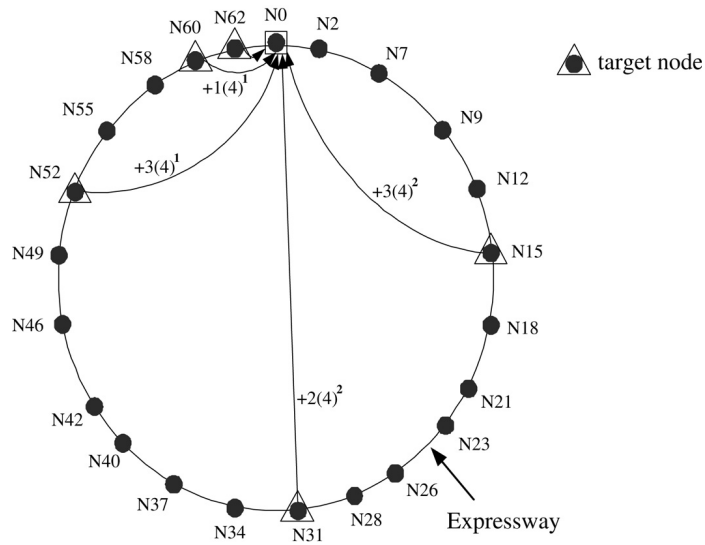


Fig. 5. The target nodes that should be notified when N0 joins or leaves the expressway.

relationship among expSucc and 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 detailed in Section 3.6.

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.

3.6. Notifying other expressway nodes to update their expressway finger tables

As mention in Section 3.4, 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 is 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 expressway finger entry (a, i) . We call these 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 $0 \leq i \leq (\log_p 2^m) - 1$. Fig. 5 shows the target nodes that should be notified when N0 joins 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 Fig. 6 shows the target ID $t(a, i)$ and the target ID interval according to the expressway in Fig. 5 as node N0 joins the system.

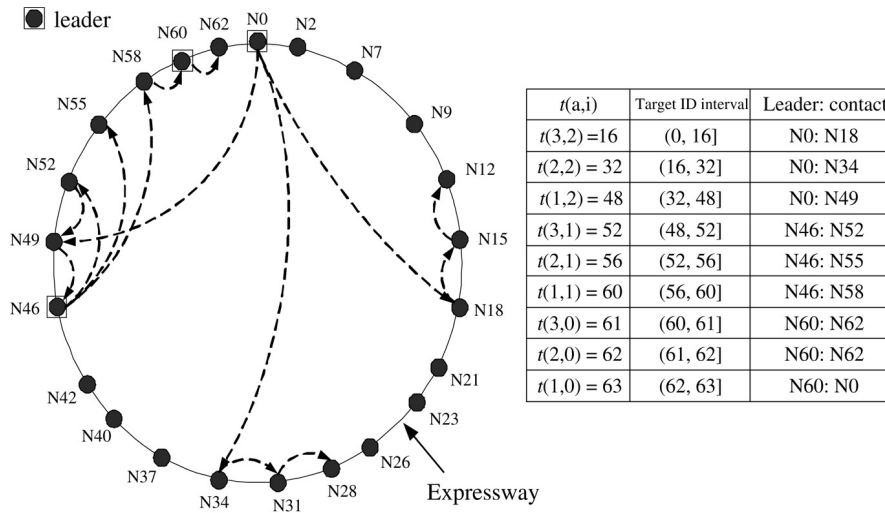


Fig. 6. A notify distribution when membership changes as N0 joins the expressway.

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 Fig. 6, column “leader: contact” represents the leader and the expressway contact for each target ID $t(a, i)$ of N0 according to the knowledge in the leader’s expressway finger 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 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 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.

Fig. 6 shows an example of an expressway node N0 joining the system and notifying other expressway nodes of its existence. N0 starts the notification distribution as a leader¹ l_3 , which notifies N18, N34, and N49 with the target ID 16, 32, and 48 respectively. Since N18, N34, and N49 overshoot according to the notified target IDs, they forward the notification back to their expPred s. N15 and N31 update their expressway finger tables, so they continue to forward the notification to their expPred s, which are not affected by N0, then the notification is terminated. N46 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). N46 notifies N52, N55, and N58 for the target ID 52, 56, and 60 respectively. N52 updates its expressway finger and forwards the notification to N49, which is affected by the N0, so the notification stops. N55 updates its expressway finger table but does not forward the notification to N52 or N58 because N52 and N58 are not in the target ID interval of 56. By getting the notification for the target ID 60 from N46, N58 is not affected by N0 but its ID is between 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.

¹ As the expressway is created in 6-bit ID space with a forwarding power of 4, the leader starts with the level $\log_4 2^6 = 3$.

Table 1
Theoretical performance analysis

	Chord	Expressway
Lookup (hops)	$O(\log_2 n)$	$O(\log_p r)$
Avg. case lookup (hops)	$(\frac{1}{2}) \log_2 n$	$(\frac{p-1}{p}) \log_p r$
#Routing entries	$\log_2 2^m$	$(p-1) \log_p 2^m$
#Notification cost (join/leave expressway)	–	$s + (p-1) \log_p^2 r$

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

3.7. Updating non-expressway entries in an expressway finger table

In addition to expressway nodes, there are non-expressway nodes that are promoted into the expressway finger table. 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 cannot be reached.

3.8. Handling faulty routing entries

In a dynamic environment, nodes may leave the expressway or the system, or nodes 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.

4. Theoretical performance analysis

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

4.1. Routing performance

We analyze the routing performance of the expressway with a forwarding power of p when using Chord as the underlying system. We 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 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 points to a non-expressway node whose ID is in the area where the expressway node ignores its existence. In this case, an expressway node forwards the request to the next closest expressway node in its contacts with a lower distance. If this case is

repeated multiple times until it compensates for 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 5), this case rarely occurs.

Next, we analyze the routing performance of the expressway when every node is on the expressway. The analysis is similar to that of Chord [1] and leads us to understand the limitation of the improvement provided by the expressway itself. We have [Lemmas 1 and 2](#).

Lemma 1. *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 forwards 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 peer 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 . 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 the request and z is reduced by a factor of p^{-1} in each hop, and is at most 2^m initially, where m is the number of bits in the ID space, then within $\log_p 2^m$ hops the distance will be 1, meaning we have arrived at 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 the request and z is $2^m/r$, which contains one node in the distance. Hence we reaches z and the request is resolved, which leads to [Lemma 1](#). \square

Lemma 2. *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 1](#), we have that the expressway can resolve all requests within $\log_p n$ hops. Based on the analysis by Loguinov et al. [11], any path from source to destination on the expressway of length j is formed by drawing j unique elements from a set of size p^i where $0 \leq i \leq (\log_p r) - 1$. As a result, each expressway node can reach exactly $\binom{\log_p r}{j}$ 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 Eq. (1). Hence the expected path length of the expressway is given by Eq. (2), which is equal to $(\frac{p-1}{p}) \log_p r$. As a result, we have [Lemma 2](#).

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

$$\sum_{j=1}^{\log_p r} j \binom{\log_p r}{j} \left(\frac{p-1}{p}\right)^j \left(\frac{1}{p}\right)^{(\log_p r)-j} = \left(\frac{p-1}{p}\right) \log_p r. \quad (2)$$

\square

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

4.2. 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.² 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 3](#).

Lemma 3. *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 $O(s + (p - 1) \log_p^2 r)$, where s is the number of target nodes that are sequentially notified and r is the number of expressway nodes.*

Proof. Supposing 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 1](#), 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 levels 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 a 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 $(p - 1) \log_p^2 r$ 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 $O(s + (p - 1) \log_p^2 r)$, where s is the number of target nodes that are sequentially notified, which leads to [Lemma 3](#). \square

Comparing with periodic updates, if at every update interval each expressway node updates one of its expressway finger entries, the maintenance cost of the whole system will be $O(r \log_p r)$ messages, and if updating all expressway finger entries the system requires $O(r \log_p^2 r)$, given that $O(\log_p r)$ messages are required to refresh each entry. If the updating interval of all entries is more often than the half-life of the system³ [12] and if $p \ll r$, then the notification requires less maintenance cost than periodic updates of all expressway finger tables.

5. 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 table. We test the routing performance and the maintenance cost of the system when nodes join.

5.1. 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 requests originating at expressway nodes versus at 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.

[Figs. 7](#) and [8](#) show the average path length per request over 10,000 requests originated from an expressway node and from a non-expressway node respectively. Each point in the graph is an average over a set of experiments; the standard deviations for each point lies between 1.03 and 1.64 hops. The average path length when the percentage

² An expressway node with a forwarding power of 4 in a 128-bit ID space system need approximately 4 kilobytes additional storage.

³ The interval that the number of nodes increases to double its size or decreases to half its size.

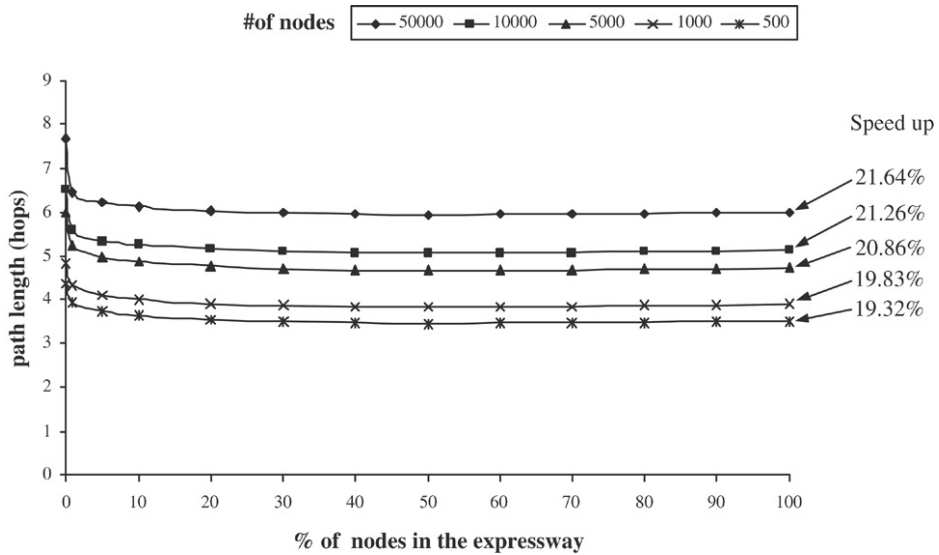


Fig. 7. The average path length required to resolve a lookup request issued from an expressway node.

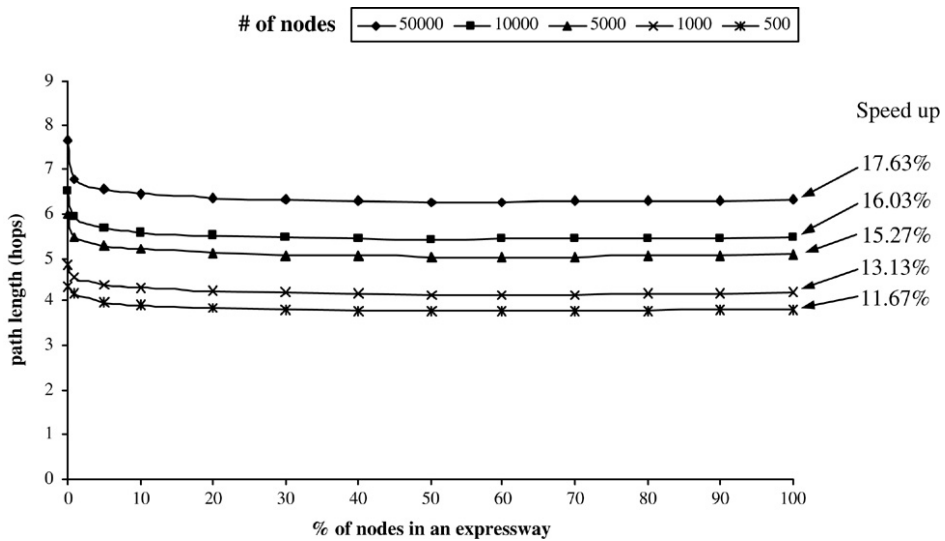


Fig. 8. The average path length required to resolve a lookup request issued from a non-expressway node.

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 originating 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.

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. For an expressway with a forwarding power of p , a distance to the destination is about n/rp , where n and r are the number of nodes and expressway nodes respectively. Hence, the system required $O(\log_2 \frac{n}{rp})$ hops in the underlying system. As a result, for an expressway with a forwarding power of 4, when the number of expressway nodes is over 12.5% of the Chord nodes, theoretically the system requires on average 1 hop in the underlying system.

The raw data shows that the system achieves the minimum average path length when about 50% of nodes join the expressway, which is not easily noticed in Figs. 7 and 8. This behaviour happens because expressway nodes keep

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

ExpSize	#Avg. notification messages	SD.
500	42.63	7.12
2 500	59.40	10.47
5 000	67.27	11.94
10 000	76.68	13.60
15 000	82.89	14.73
20 000	87.58	15.75
25 000	91.71	16.87
30 000	95.49	18.16
35 000	99.13	19.57
40 000	102.50	20.50
45 000	105.79	21.95

contacts of non-expressway nodes in their expressway routing table entries where there is no expressway node existing 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 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 Fig. 7. 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 Fig. 8.

5.2. Maintenance cost

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 into each placement. We measured the number of messages required to notify the other nodes when a new node joined in each placement. 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 is $O(\log^2 r)$ where r is the number of expressway nodes. In the current implementation, each node does not have the knowledge of the predecessor of the join node. As a result, every time a node updates its expressway finger table, it forwards the notification messages to its neighbours without the knowledge that they might not be in the target nodes.

6. Expressway comparison

Other expressway techniques that are related to our work are Brocade [9] and the topology-aware expressway [8]. Currently we focus on a logical hop analysis. The topology-aware expressway is built based on physical properties such as a physical distance of nodes. So, we choose to compare our expressway with Brocade as it is the closest

⁴ For $p = 4$: $(1 - 2(\frac{p-1}{p}) \log_p 2) = (1 - 2(\frac{4-1}{4}) \log_4 2) = 25\%$ (see Section 4.1).

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

	Brocade	Our expressway
Forward a request to an expressway node (hops)	1	1
Routing a request on an expressway (hops)	$O(\log_p r)$	$O(\log_p r)$
Relaying a request from an expressway node to the destination [best, worst] (hops)	$[2, O(\log_2 n)]$	$[\frac{1}{2} \log_2 \frac{n}{rp}, O(\log_2 \frac{n}{r})]$

n is the number of Chord nodes; r is the number of expressway nodes; p is the forwarding power of an expressway.

expressway for a logical analysis. The comparison between our expressway with the topology-aware expressway will be left as future work. We analyze Brocade and our expressway comparison based on the nodes dynamically joining and leaving the system. We define a system to be less dynamic if the half-life of nodes in the system is longer than the periodic update interval of the nodes' statuses. On the other hand, we define a system to be highly dynamic if the half-life of nodes in the system is less than the periodic update interval of the nodes' status.

6.1. Brocade versus our expressway

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. We divide the 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. Then we conclude with the overall comparison. Table 3 presents the theoretical routing performance between Brocade and our expressway.

We assume that both expressways route requests with the same forwarding power of p and that the Brocade overlay is analyzed as it was implemented using Chord. The difference between Brocade and our expressway is that Brocade requires a registration of non-expressway nodes in its physical network with nearby expressway nodes and requires that expressway nodes publish and unpublish their list of non-expressway nodes over the expressway overlay. In addition, Brocade nodes do not promote non-expressway nodes in the expressway routing tables. We analyze Brocade and our expressway in an n -node system with r expressway nodes.

Both Brocade and our expressway 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. Our non-expressway nodes forward requests to the expressway nodes whose IDs are closest to the IDs of their expressway entry points. 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. Brocade and our expressway achieve a different logical routing path length when a request is relayed from an expressway overlay to the destination.

To relay a request from an expressway overlay routing to the destination, the Brocade expressway node that keeps the published information of the destination forwards the request to the destination expressway node x to which the destination node is registered. After that, x forwards the request to the destination. In the best case when all the published information is up-to-date, the system requires 2 hops to forward a request from an expressway overlay to the destination. In the worst case when the published information is outdated, the destination does not belong in the register lists of the destination expressway node and the request is forwarded using the underlying routing system. As a result, in the worst case, the system requires $O(\log_2 n)$ hops to forward a request to the destination.

In our expressway, expressway nodes promote non-expressway nodes into 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 placed around the ID space, the distance where a request needs to be forwarded in the underlying system to the destination is n/rp . In the best case when expressway nodes do not contain any faulty routing entries, the system requires $O(\log_2 \frac{n}{rp})$ hops to forward a request in the underlying system to the destination. On the other hand, the expected path length to defer a request from the expressway routing overlay to the destination is $\frac{1}{2} \log_2 \frac{n}{rp}$. In the worst case, all the promoted non-expressway entry are faulty entries and the expressway node forwards the request using its underlying finger table. As our expressway nodes join the expressway with their own IDs and maintain their memberships in the underlying system, the system requires $O(\log_2 \frac{n}{r})$ hops to forward a request in the underlying system to the destination. The number of hops required to forward a request to the destination in our system varies proportionally with n/rp . For example, for an expressway with a forwarding power of

Table 4
The maintenance and overhead cost of Brocade versus our expressway

	Brocade	Our expressway
# Routing entries	$(p - 1) \log_p 2^m$	$(p - 1) \log_p 2^m$
# Registered node entries	n/r	–
# Published non-expressway node entries	n/r	–
Periodically update registered nodes	$(n/r)/\text{period}$	–
Published/unpublished registered nodes	$O(\log_p r)/\text{node}$	–
Periodically update promoted non-expressway nodes	–	$O(\log_2^2 \frac{n}{r})/\text{period}$
Building up expressway routing entries	–	$O((p - 1) \log_p^2 r)$
Notification when membership changes*	–	$O(s + (p - 1) \log_p^2 r)$
Periodically update expressway* routing entries	$O(r(p - 1) \log_p^2 r)/\text{period}$	–
Finding an expressway entry (for non-expressway nodes)	DNS-like	logical lookup $O(\log_p r)$

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

4 and when 25% of nodes join the expressway, the system requires 1 hop to forward a request from the expressway to the destination. All non-expressway nodes are promoted in the expressway routing tables. This argument is supported by our experiments, which show that when 20% of nodes join the expressway, the performance of the system is equivalent to when every node joins the system.

In conclusion, our expressway can resolve a request with fewer hops than Brocade when the proportion of n , r , and p make the system perform best, specifically when $\frac{n}{rp} \leq 2$. When the system is less dynamic, most of the non-expressway node statuses are up-to-date. Brocade is more attractive than our expressway because when the published information is correct, Brocade requires a constant number of hops to relay a request to the underlying system. When the system is highly dynamic, most of the non-expressway information is not up-to-date. Our expressway is more attractive than Brocade as our expressway’s worst case upper bound is better than Brocade. Moreover, our expressway requires less maintenance cost than Brocade, which we will discuss in the following section.

Table 4 shows a 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 requires the same number of routing entries to maintain the contacts of other nodes, namely $(p - 1) \log_p 2^m$ entries. In addition, each Brocade expressway node requires on average n/r entries to keep the contacts of registered nodes and an additional n/r entries on average to keep information of non-expressway nodes that are published for the expressway routing. Brocade’s expressway nodes need to periodically monitor the existence of their registered nodes and to publish/unpublish the status of their registered nodes to the expressway overlay. Each published/unpublished information requires $O(\log_p r)$ hops to resolve the location of the responsible expressway nodes. In our expressway, promoted non-expressway nodes in expressway routing tables are needed to be monitored. For each expressway node, there are on average $O(\log_2 \frac{n}{r})$ promoted node entries and each entry requires $O(\log_2 \frac{n}{r})$ messages to check the correctness of a promoted node entry. As a result, each expressway node requires $O(\log_2^2 \frac{n}{r})$ messages per update period to keep promoted node entries up-to-date.

Our expressway maintains membership changes using an 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 needs to build its expressway routing table when it first joins the expressway which requires $O((p - 1) \log_p^2 r)$ messages. Brocade expressway nodes maintain their memberships by using periodic updates. Each expressway node requires $O((p - 1) \log_p^2 r)$ messages per period to update all entries. In total, a system with r expressway nodes requires $O(r(p - 1) \log_p^2 r)$ messages per period. Brocade’s new expressway nodes do not need to build their expressway routing entries when they join the expressway as their expressway routing entries will be built in part by periodic updates. Although our expressway requires overhead on building expressway routing tables,

the cost of building routing tables is equal to one round of periodic updates of expressway routing entries in Brocade. After building their routing tables, our expressway nodes only update their entries when a node joins or leaves the expressway. We expected that when an expressway overlay is less dynamic, our expressway with an event-based notification will require less maintenance cost than a periodic update, as discussed in Section 4.2.

Next we compare the overhead that non-expressway nodes require to find their expressway contacts. Brocade uses a DNS-like system to locate the closest expressway node for each non-expressway node. In our expressway, a 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. Even though our non-expressway nodes require more messages to find expressway contacts for their expressway entry points, the cost to find each entry is equal to one lookup. In addition, we do not have to build an additional DNS-like system for non-expressway nodes to find their expressway contacts. 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 has their contacts on the expressway overlay. By having more expressway entries, our expressway removes the single point of failure each non-expressway node has to enter the expressway overlay, which appears in Brocade.

Based on the assumption that the underlying system is highly dynamic and the expressway overlay is less dynamic, our expressway requires less overhead and maintenance cost than Brocade. Our expressway is more attractive than Brocade as Brocade has a higher upper-bound routing path when information of non-expressway nodes is outdated. However, in our expressway, if the expressway routing entries are nominated by promoted non-expressway nodes, the expressway will require a higher overhead in periodic updates for these entries and a higher routing path length in the underlying system. Hence, our expressway is better than Brocade when the proportion of n , r , and p satisfies, $\frac{n}{rp} \leq 2$. For a system with less dynamic and fewer expressway nodes, $r < \frac{n}{2p}$, Brocade is more attractive as it provides constant routing hops to forward a request from the expressway to the destination.

7. Conclusions and future work

We propose an expressway with a hierarchical overlay in P2P systems where nodes can provide different resources and can seamlessly collaborate between each level. Instead of using periodic updates, we propose an event-based notification to maintain membership of expressway nodes. We allow nodes with significant 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 resources.

Our expressway design takes advantage of nodes that can provide significant resources to decrease routing path length in terms of logical 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 forward a request to the destination in the logical ID space with fast forwarding channels of expressway nodes, and our expressway skips routing to non-expressway nodes in the edge of physical networks.

We design, analyze and implement a logical expressway 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. The event-based notification requires $O(s + (p - 1) \log_p^2 r)$ messages to distribute among expressway nodes for a single membership change where there are r nodes in the expressway, s nodes to notify, and the expressway has a forwarding power of p . The experimental result shows that the distribution cost diminishingly increases as the number of expressway nodes in the system increases.

We show theoretical comparison of our expressway with Brocade. Our expressway routing performance is comparable with Brocade. Our design does not require the registration process, publish/unpublish non-expressway node information and periodic updates for expressway maintenance and overhead costs as it does in Brocade. When $\frac{n}{rp} \leq 2$, expressway nodes are less dynamic, and when non-expressway nodes are highly dynamic, our expressway is more attractive than Brocade.

In the future, we will study the effect of a physical network on expressway overlay design parameters such as p and r , and the system physical routing path length and latency. Comparison of our expressway performance on physical networks with Brocade and the topology-aware expressway will be further investigated. We can also extend our expressway to a multi-level expressway; however, the number of layers and additional costs need to be justified.

Acknowledgments

We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC), IBM, and Precarn Inc. The first author is also supported by the Royal Thai Government Scholarship. Finally, we thank Nick Cercone for his contributions in improving and refining this paper.

References

- [1] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup protocol for internet applications, *IEEE/ACM Transactions on Networking* 11 (1).
- [2] A.I.T. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Springer-Verlag, 2001, pp. 329–350.
- [3] B. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, J.D. Kubiatowicz, Tapestry: A resilient global-scale overlay for service deployment, *IEEE Journal on Selected Areas in Communications* 22 (1).
- [4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, A scalable content-addressable network, in: *Proceedings of the ACM SIGCOMM*, 2001, pp. 161–172.
- [5] A. Gupta, B. Liskov, R. Rodrigues, One hop lookups for peer-to-peer overlays, in: *Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operation Systems*, 2003.
- [6] J. Hu, M. Li, W. Zheng, D. Wang, N. Ning, H. Dong, Smart-Boa: Constructing p2p overlay network in the heterogeneous internet using irregular routing tables, in: *Proceeding of the 3rd International Workshop on Peer-to-Peer System, IPTPS'04*, 2004.
- [7] Z. Xu, Z. Zhang, Building low-maintenance expressways for P2P systems, Technical Report, Internet System and Storage Laboratory, HP Laboratories Palo Alto, 2002.
- [8] Z. Xu, M. Mahalingam, M. Karlsson, Turning heterogeneity into an advantage in overlay routing, in: *Proceeding of The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM*, 2003.
- [9] B. Zhao, Y. Duan, L. Huang, A. Joseph, J. Kubiatowicz, Brocade: Landmark routing on overlay networks, in: *Proceedings of 1st International Workshop on Peer-to-Peer Systems, IPTPS*, 2002.
- [10] B.Y. Zhao, J.D. Kubiatowicz, A.D. Joseph, Tapestry: An infrastructure for fault-tolerant wide-area location and routing, Technical Report, University of California at Berkeley, 2001.
- [11] D. Loguinov, A. Kumar, V. Rai, S. Ganesh, Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience, in: *SIGCOMM'03, Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ACM Press, New York, NY, USA, 2003, pp. 395–406.
- [12] D. Liben-Nowell, H. Balakrishnan, D. Karger, Analysis of the evolution of peer-to-peer systems, in: *Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing*, 2002.