**MIT-LCS-TR-897**

# Improving Application-level Network Services with Regions

Ji Li

May 2003

# Improving Application-level Network Services With Regions

by
Ji Li

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2003, in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science and Engineering

## Abstract

The underlying premise of the Region Project is that the concept of a region should be a new architecture capability in networking. A region is an entity that encapsulates and implements scoping, grouping, subdividing, and crossing boundaries of sets of entities. It is a powerful tool for managing the increasingly complex demands on the Internet and its successors, and thus should be made into an explicit, first-class component of the network architecture. Autonomous Systems and peer-to-peer networks can be viewed as two simple forms of existing regions. In this work, we explore the utility of informing members in one region of the membership of those same entities in different regions. Specifically, we improve peer-to-peer networks with information derived from Autonomous Systems. This thesis makes three notable contributions. Firstly, we provide a general peer-to-peer simulation framework for different optimization schemes. Secondly, we achieve performance improvements in the lookup, caching and replication of peer-to-peer system. Finally, we enhance our overall understanding of regions through the simulation, as well as their utilities to improve system performance.

Thesis Supervisor: Karen R. Sollins
Title: Principal Scientist

## Acknowledgments

First, I would like to thank my advisor, Karen Sollins, for her expert guidance and advice for both my research and my professional life.

Second, my research has benefited from the members of the Advanced Network Architecture (ANA) research group at LCS. They have provided useful thoughts and comments. Thank you all.

Third, I acknowledge Ben Leong, Dah-Yoh Lim and Chuang Hue Moh for valuable discussions on the implementation.

This work was funded in part under NSF Grant ANIR-0137403, "Regions: A new architectural capability in networking" and NSF Grant CCR-0122419, "The Center for Bits and Atoms". It was also funded in part by a gift from the Cisco University Research Program.

Lastly, I wish to dedicate this thesis to my family. They have been a constant source of support and love. I hope I have made them proud.

# Contents

# Chapter 1
# Introduction

The Internet, in reality, is more complex than a small collection of interconnected hosts or LANs. The IP-level structure of the Internet is composed of a large number of constituent networks, each of which differs in some or all of its transmission technologies, routing protocols, administrative models, security policies, QoS capabilities, pricing mechanisms, and other attributes. On top of this, a whole new structure of application-layer overlays and content distribution networks, equally diverse in the sorts of ways mentioned above, is rapidly evolving. A horizontal dissection of the current Internet structure reveals a loosely coupled federation of entities that are separately defined, operated, and managed. These entities are interconnected to varying degrees, and often differ drastically in their internal requirements and implementations. We can think of each entity as existing in a "region" of the network: each region has coherent internal technology and policies, and manages its own interactions with other regions of the network according to a predefined set of rules and policies.

## 1.1 Background

A region [1] is a partition of the network which models an externally defined or imposed structure. The region captures the concept of an area of consistent control, state, or

knowledge. There can be many sorts of regions at the same time - regions of shared trust, regions of physical proximity (the floor of a building or a community), regions of payment for service (payment zones for stratified cost structures), and administrative regions are examples. This general notion of a region is a powerful tool for managing the increasingly complex demands on the Internet and its successors, and thus should be made into an explicit, primary component of the network architecture. We propose, first, to separate the concept of "region" from any specific use of the concept and offer it as an independent, reusable abstraction because many uses of regions share a well-defined set of core ideas and operations. Second, we propose to implement the region abstraction as a concrete software entity, and hence to provide network protocol designers and implementers with a logically structured, scalable, high-performance means of access to an important class of functionality.

In this thesis, we develop the "region" abstraction and explore its utility as a general mechanism in networking in the future. We will explore sharing knowledge of being in one sort of region in order to improve performance with respect to another. We will focus on the use of regions defined at the network layer as a vehicle for optimizing and improving services defined at the application layer. Here the region defined at the lower level is unaffected by the higher level, and its information enables more informed decisions and operations at the higher level. A second order objective is to refine the definition and development of the region abstraction itself, focusing on requirements, choices of capabilities, abstract operations, and related issues.

## 1.2 Autonomous Systems

At present, there are already some entities we find it useful to conceive as "regions" defined at the routing level, known as routing domains or Autonomous Systems (ASs). An Autonomous System is a group of IP networks operated by one or more network operators which has a single and clearly defined external routing policy. These regions contain useful

routing and policy information. If this information can be propagated to the application-level systems, it will enable them to make more intelligent decisions, such as more efficient routing, or adherence to one AS or some trusted ASs, when deciding on a route through an overlay network.

## 1.3 Peer-to-Peer Systems

An area of interest in the research community today is building overlay networks. Overlay networks are typically sets of hosts from the perspective of the packet-level Internet that provide a network of infrastructural components for some applications. A peer-to-peer system (P2P), as a type of overlay network, is composed of a set of symmetrical end hosts or nodes: they act both as clients that utilize the services provided by other hosts, as well as servers that provide these services.

Now consider the problem of some application level requests in peer-to-peer networks. There may be a number of possible routes that such a request can take through the overlay network. Routing decisions for these overlay networks are often made at the application level. However, because no information about the underlying network is available to the application-level systems, these decisions cannot be made based on the actual underlying paths followed. For example, in structured peer-to-peer networks, querying is done by traversing a sequence of end hosts based on the application-level routing table. The traffic may go through a longer route than the optimal one, or even traverse the same links many times before actually reaching the desired location. Choices at the application level imply that when more than one option is possible, having lower level routing information can improve the routing performance at the application level.

Caching is a common technique in operating systems, and has spread to distributed systems. In a peer-to-peer system, caching mechanism can reduce lookup latency by temporarily storing previous lookup results along the lookup path. However, caching on every node along a lookup path requires significant storage. In this thesis, we use AS

information to reduce the storage of caching while maintaining comparable performance.

Availability is crucial to peer-to-peer systems. End hosts join and leave frequently, so replication is necessary to ensure resource availability. Some peer-to-peer systems replicate a node's data at its logical neighbors. However, such a scheme may not work well because the replica placement is random, with no knowledge of where the copies are. With information provided by the lower-level network, we hope that peer-to-peer networks can obtain explicit control and make flexible decisions on their replication schemes. Therefore, we can not only improve resource availability but also reduce the latency to retrieve data and balance the network traffic.

## 1.4 Contributions

In this thesis, we present the region concept and show how it applies to autonomous systems and peer-to-peer networks. We also design several schemes to improve peer-to-peer networks with the concept of regions. The contributions of this thesis follow three major themes:

First, we develop a framework of a generic structured peer-to-peer system, in which different routing optimization schemes are integrated, and the lower level region information is provided to the peer-to-peer system via simple interfaces. This framework provides flexibility for optimizations based on different kinds of information.

Second, we evaluate our idea by simulation. A Pastry/Tapestry-like peer-to-peer system acts as the upper region, and the Autonomous Systems act as the lower regions. We evaluate how AS regions improve the routing, caching, and replication in a peer-to-peer system.

Finally, as a part of the Region Project, we use the experience from the simulation to refine our abstraction of "regions" and extend this idea to a general case.

The rest of the thesis is organized as follows. In Chapter 2 we present related work on the Region Project, Autonomous Systems, and peer-to-peer networks. In Chapter 3, we present the concept of regions and how it is applied to two existing region examples: Autonomous Systems and peer-to-peer systems. Building on this, Chapter 4 describes the design of a lookup scheme, a caching scheme, and a replication scheme in peer-to-peer systems based on information from AS regions, and the additional functionalities gained by this approach. Chapter 5 presents the implementation and evaluates our schemes by simulation. Finally, Chapter 6 concludes the thesis and discusses some future research in the Region Project.

# Chapter 2
# Related Work

In this chapter, we provide the necessary background about the Region Project. We also present some related work on Autonomous Systems and peer-to-peer systems, especially some structured peer-to-peer systems and their routing algorithms.

## 2.1 The Region Project

The related work of this project falls into several major categories, partitioning of namespaces in order to handle scaling of name assignments and resolution including boundaries defined in order to reflect changes in some activities, cross protocol layer interaction, middleware infrastructure to support creation and execution of network-based applications, etc.

One set of problems is grouping of objects in order to address scaling problems. In each of these examples, the problem was to reduce the space to be searched in order to find something. The sole function of the Domain Name System [2][3] is to provide a single global hierarchy in which both name assignment and name resolution occur in order to find hosts. CORBA [4] provides a much richer set of middleware activities, but in conjunction with this provides a two level hierarchy for naming, by uniquely naming each ORB, and delegating unique assignment with the ORB to the ORB itself. Here the objective was to

find a specific CORBA object. The Intentional Naming System [5] was designed to route traffic to the named entity. It is an example of a different approach, in which names are composed of attribute value pairs, but these are organized hierarchical. An entity announces itself to any resolver, which in turns broadcasts the identity of the entity using a spanning tree to the universe of resolvers. A request to the entity is resolved and forwarded at each resolver between the requester and the entity itself. Although this work as it stands does not scale, it provides an interesting point within the space of naming alternatives, because it attempts to provide multi-layer functionality of both naming and routing.

Virtually any horizontal slice through the current Internet structure reveals a loosely coupled federation of separately defined, operated and managed entities, interconnected to varying degrees, and often differing drastically in internal requirements and implementation. Examples can be found in routing and QoS provision. BGP [6] boundary transitions reflect routing information exchanges between Autonomous Systems; we discuss this in detail in Chapter 3. DiffServ clouds [7] reflect boundary points at which per domain behavior may change. In each case the choices of what happens internally to a region or a scoping entity are made independently of what is happening outside.

In terms of middleware support for the creation and support of distributed applications there is an enormous collection of work, including CORBA [4], Microsoft's Universal Plug and Play [8], Sun's combination of Jini [9] and Rio [10], etc. It is an area where a great deal of work is occurring, so this is just a sampling of the activities.

## 2.2 Autonomous Systems

As mentioned in Chapter 1, there are Autonomous Systems in the Internet we find it useful to conceive as "regions" defined at the routing level. An Autonomous System is a group of IP networks operated by one or more network operators which has a single and clearly defined external routing policy. The Border Gateway Protocol (BGP) is used to exchange routing information between Autonomous Systems. There is an increasing interest in

Autonomous Systems research for various reasons, including troubleshooting operational problems, and facilitating more realistic simulation, and evaluation of network protocols and applications. However, today's Internet is increasingly shaped and controlled by commercial interests, and topology information is not directly accessible to researchers. To make it even worse, it is hard to deploy new services and the Internet is regarded as becoming ossified [11]. Under such circumstances, there have been several research areas emerging. One the one hand, Internet measurement and modeling emerge as an important area of network research [12] [13]. On the other hand, researchers are trying to build either an easily accessible network testbed in the current Internet, such as Emulab [14], or a new service-oriented network architecture, such as PlanetLab [11].

In this thesis, our key idea is to provide the underlying network information to the application. Specifically, Autonomous Systems provide coarse-grained information of network topology, network latency, etc. Although basing the distance estimations only on the number of AS boundaries the route crosses may not be sufficient since Autonomous Systems significantly vary in size, exploiting the AS-based map of the Internet is a way of dealing with the problem of distance measurement. It is also likely to be more accurate than the geographical distance [15].

## 2.3 Peer-to-Peer Systems

Peer-to-peer networks can be regarded as a form of regions in the application layer. There have been peer-to-peer systems for various purposes: for distributed computing, such as SETI@home [16], for collaboration, such as DOOM [17], used as platforms, such as JXTA [18], and for data sharing, such as Napster [19] and Gnutella [20]. In the data-sharing peer-to-peer systems there have been two generations of such peer-to-peer systems: unstructured peer-to-peer systems and structured peer-to-peer systems. Structured P2P systems are systems in which nodes organize themselves in an orderly fashion while unstructured P2P systems are ones in which nodes organize themselves randomly. The

unstructured peer-to-peer networks include Napster [19], Gnutella [20], and Freenet [21], etc. For the unstructured peer-to-peer systems, generally the discovery process is very localized. For example, Napster has a central server that stores the index of all the files within its community. To find a file, the user queries the central server and gets the IP address of a machine that stores the file. Meanwhile, Gnutella's lookup is localized to a node's locally discovered region, because it broadcasts a query within a certain number of forwarding steps.

The structured peer-to-peer networks include Chord [22], CAN [23], Tapestry [24], Pastry [25], Kademlia [26], Viceroy [27], and Expressway [28], etc. Most structured peer-to-peer systems support a distributed-hash-table functionality, so they are also called DHT (Distributed Hash Table) P2P systems. A key operation in these peer-to-peer systems is: given a key, the system will determine the node responsible for storing the key's value. This is called lookup or routing in peer-to-peer systems. We will use the two terms interchangeably when talking about peer-to-peer systems in this thesis. We will focus on improving structured P2P system performance in various aspects.

Pastry [25] uses a *128*-bit name space and each digit in an ID is in base *16*. Each node has a routing table with *32* rows and *16* columns. The entries in row *n* of the routing table refer to nodes whose IDs share the first *n* digits with the present node's ID. The $(n+1)^{th}$ digit of a node's ID in column *m* of row *n* equals *m*. The routing algorithm is based on address prefixes, which can be viewed as a generalization of hypercube routing. At each step in a lookup, a node forwards the message to a node whose ID shares with the destination ID a prefix that is at least one digit longer than the shared prefix between the destination ID and the present node's ID. The average number of lookup hops is $O(\log_{16} N)$, in which *N* is the total number of nodes. To optimize routing performance, Pastry uses proximity neighbor selection. Proximity neighbor selection constructs a topology-aware overlay network. The key idea is to fill routing table entries with topologically closest nodes among all node candidates. In Pastry, the upper rows of the routing table allow great flexibility in choosing

nodes as entries, while lower rows have exponentially fewer choices. This flexibility leads to the success of this technique. We also make use of this feature in the thesis. In our work, we use a generalized peer-to-peer structure similar to that of Pastry and Tapestry. But, instead of using fine-grained metrics, we use coarse-grained information provided by Autonomous Systems to improve the lookup performance and to reduce system overhead.

Tapestry [24] is similar to Pastry with minor differences. Tapestry has a similar routing structure to Pastry. It also uses proximity routing to achieve locality. Furthermore, Tapestry maintains two backup nodes for each entry in the routing table for sake of entry failure. It also maintains back pointers that point to the nodes from which it is referred as a neighbor.

Chord [22] uses a circular 160-bit name space. Each node has a number of fingers pointing to some other nodes. Chord forwards messages clockwise in the circular name space based on numerical difference with the destination address. To optimize the lookup performance, Chord uses proximity routing. Unlike proximity neighbor selection, the overlay is constructed without regard for the physical network topology. When routing a message, there are potentially multiple nodes in the routing table closer to the destination ID in the name space. Chord selects either the one that is closest in the physical network or one that represents a good compromise between progress in the name space and proximity among the set of possible next hops. However, choosing the lowest delay hop greedily may lead to an increase in the total number of hops taken in a lookup. Chord cannot use proximity neighbor selection because their routing table entries refer to specific IDs in the name space without the flexibility in Pastry and Tapestry. That is why we use a peer-to-peer model generalized from Pastry and Tapestry.

Chord, Pastry, and Tapestry all use one-dimensional name spaces and maintain some finger-like pointers to accelerate routing. CAN (Content-Addressable Network) [23] can be regarded as a multi-dimensional version of a Chord-like system with more restriction on fingers. In CAN, nodes are mapped into a d-dimensional coordinate space on top of zones

based on server density and load information. Each zone keeps information of its immediate neighbors only. Because addresses are points inside the coordinate space, each node simply routes to the neighbor that makes the most progress towards the destination coordinate. Therefore, CAN maintains an $O(d)$ routing table, smaller than all the three above. The drawback is the longer average routing path, $O(d \cdot n^{1/d})$. To improve the lookup performance, Ratnasamy et al. [29] introduces an idea of landmarks as the topology signals. Landmarks provide some coordination for a node when it joins the system. The distance from a node to those landmarks is represented as a distance vector. For example, when a node joins a peer-to-peer network, it first determines its distance vector and finds the ordering of bins with which it should be associated. Then it picks a random number within the bin area. Such topology-based ID assignments violate the uniform distribution of node IDs, which leads to load balancing problems, and neighboring nodes are likely to suffer correlated failures.

Brocade [30], which exploits knowledge of the underlying network characteristics, added a secondary overlay on top of the underlying peer-to-peer system. The secondary layer builds a location layer between super nodes (landmarks), which are members of the original peer-to-peer systems, and are situated near network access points with significant processing power, such as gateways. The super node covers an area that contains a set of local nodes and acts as their manager. In this way, Brocade introduces a hierarchy into peer-to-peer systems. The super node does not manage local communication, so the local communication could go out of the local network. Also, the super node serves as a centralized server, which may become the bottleneck and thus not scale.

Besides the ring structure and $d$-dimensional structure, Kademlia [26] introduces a binary-tree organization and *XOR* operation, Viceroy [27] introduces the butterfly structure. Both have desirable properties in the routing algorithms.

Caching is widely used in operating systems. In a peer-to-peer system, the goal of caching

is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. Caching reduces the lookup path length and therefore, the number of messages exchanged between peers. Reducing such transmissions is important because the communication latency between peers is a serious performance bottleneck facing P2P systems. There are two kinds of caching in P2P systems based on what is cached. One we call data caching. In data caching, data are copied in the requester and along the lookup path. The other is to address caching. In this scheme, the ID and its corresponding IP address are cached both at the requester and along the lookup path. Compared with data caching, address caching requires much less space. Most current systems use such both of them. In Freenet [21], when a file is found and propagated to the requesting node, the file is cached locally in all the nodes in the return path. CFS [31] is a peer-to-peer read-only storage system based on the Chord protocol. It uses a caching scheme similar to Freenet which leaves cached copies of data along the query path from client to where the data was found. Because CFS finds data in fewer hops than Freenet, and multiple structured lookup paths in CFS are more likely to overleap than those in Freenet, CFS can make better use of a given quantity of cache space. PAST [32] is a large-scale peer-to-peer persistent storage utility. In PAST, data that are routed through a node as part of a lookup are inserted into the local disk cache if its size is less than a fraction of the node's current cache size. Tapestry uses hotspot monitoring to detect query hotspots and place a copy of data on the source node of the most query traffic.

Replication is widely used in distributed systems. Replication puts multiple copies of data in the system, thus improving fault tolerance, and sometimes reducing the latency between the peers requesting and providing the data. Furthermore, the geographic distribution of replicas helps to reduce local congestion on both the peers and the network. Peers tend to be disconnected from the network at random, so replication also helps to cope with peers leaving. In [33], Lv et al. analyzed the number of replicas needed corresponding to the query rate in unstructured peer-to-peer networks, and find that for a fixed average number

of replicas per node, square-root replication distribution is theoretically optimal in terms of minimizing the overall search traffic. Most structured peer-to-peer systems use replication to cope with node failures [22][24][25][34]. They all place replicas in the nodes that follows the present node in the ID space. Since each ID is assigned randomly, they hope to distribute the replicas evenly in the peer-to-peer systems in this way. But none of those schemes has any explicit control over the placement of the replicas since they have no knowledge of the underlying networks.

# Chapter 3
# Regions

In this chapter, we introduce the definition of a region, its membership and operations, and the additional functionality of a region, such as boundary crossings and notifications. We also discuss how the region concept is applied to Autonomous Systems and peer-to-peer systems.

## 3.1 Region Definition

There are a number of issues related to the region definition. Two aspects we consider here are the invariants and the ability to distinguish a region from others.

One of the key aspects of a region is the set of invariants it represents or by which it is defined. One could argue that a simple definition with respect to invariants would be to declare that each region defines a single invariant. In conjunction with this, a key function will be operations for intersection among regions. But a single variant ignores the fact that the second central aspect of a region is a boundary, and, unless we provide some means of defining a merged boundary for the intersection, we have lost a significant aspect of the region abstraction. Hence, for the purposes of this work, we assume that a region can have more than one invariant.

The second issue we consider here is that of distinguishing regions from each other. There are many approaches taken to handling the question of distinguishing regions. An approach to defining sets of entities is to decide that all entities that share a pre-defined set of invariants are by definition in the same region. In this case, it is not difficult to specify different regions, simply by specifying the appropriate set of invariants, but if the scope of the universe is the whole Internet, then it is unlikely that one could ever have any significant level of confidence that all or most of a region's membership could be known at one time and place. Furthermore, this approach assumes a global definition space for invariants. This has two implications. First, there must be common agreement on invariant representation, so that two statements of invariants that are not intended to be the same as each other are distinctly represented. This implies global definition of invariants. Second, no two regions will have the same set of invariants assigned to them. This would also require global coordination. An alternative approach to this problem is that each region is assigned a globally unique name or identifier in order to distinguish one from another. There exist a number of global naming schemes at present with varying degrees of scalability, user-friendliness, etc.

Autonomous Systems can be regarded as a form of existing regions. These regions define their invariants as those destinations recognized by the routers as having the same BGP gateways into the region, generally based on the subnet masks of the IP address. The BGP configuration and the internal routing algorithm within an AS reflect the administrative policy of an AS, but the policy is not explicitly visible or accessible to end hosts or other ASs.

Each AS has a globally unique number, an Autonomous System number (ASN), associated with it; this number is used in both the exchange of exterior routing information (between neighboring Autonomous Systems), and as an identifier of the AS itself. ASNs are assigned by ICANN [35]. The global control over AS numbers guarantees their uniqueness. Exterior routing protocols such as the Border Gateway Protocol (BGP) defines how ASs

exchange routing information dynamically. Each AS has its own routing policy, reflected by its BGP protocol.

Peer-to-peer systems are other existing regions considered in this thesis. An end host joins a peer-to-peer system by cooperating with other members in the system to perform a critical function. There are many different types of P2P systems, and one type of P2P systems may have several instances running in the Internet at the same time Therefore, the invariants of a peer-to-peer system are the type and the instance. In other words, a node joins a P2P system by running a daemon for that system. If nodes join different types of peer-to-peer systems, the daemons will be different. If nodes join different instances of a peer-to-peer system, the daemons will be the same, but the ports will be different. Therefore, the invariants can also be regarded as a pair of the daemon and the port. The daemon contains a specific protocol of the peer-to-peer system, while the port number can distinguish peer-to-peer systems with the same protocol. IANA assigns well-known ports [36] while other ports are determined more locally as needed.

There is an important difference between Autonomous Systems and peer-to-peer networks: they are in different layers of networks. One the one hand, ASs exist in the network layer and P2P systems in the application layer. On the other hand, ASs are topologically separated. That is, there is no overlap between ASs and any end host only belongs to one AS. There are physical boundaries between ASs in the form of BGP routers. However, different peer-to-peer systems can overlap in the physical layer by having one node in multiple P2P systems at the same time as long as they do not conflict in resources.

## 3.2 Region Membership

The functions a region must support with respect to membership fall into two categories. First, there is an issue of insertion in and deletion from a region and, second, there is an issue of learning about and distinguishing among members. These two groups of functions will be discussed separately.

There are two key issues with respect to an entity joining a region, the invariants and introduction. The intention is that if the invariants were not true of the entity prior to joining a region, they become true. The difficulty here is that this may be impossible or unacceptable. So joining may be more than a simple decision to add an entity to a region.

Because invariants are not globally defined, membership in a region cannot be based on invariants of the entities. Therefore, an introduction function is needed for inserting entities into regions. In practice we may find two sorts of such introductions, those performed by a human and those performed by members of the region itself.

One can consider a similar distinction for removal or deletion from a region, but possibly with different conclusions. We assume that when an entity is assigned to a region, it inherits the invariants defining the region, and hence we could reasonably postulate that if the entity is explicitly and authoritatively assigned contradictory invariants, it is expelled from the region. It is worth noting that, in order to make this statement, we must assume that entities have a set of invariants, which can be modified. Further, if an entity was introduced into a region by another entity already in the region, one can ask whether the introduced entity will be expelled if its introducer is removed. Here, the intuition is that such a rule does not make sense. We can conclude that in addition to an explicit removal function, there may be a more implicit action.

As an existing form of regions, Autonomous Systems' members are end hosts and routers. An end host accepts the invariants of an AS automatically by joining the AS. Its introduction is done by either joining an organization within an AS, or joining the AS directly by choosing its Internet Service Provider (ISP). A host's identifier is the IP address either statically assigned or obtained dynamically with some protocol, such as DHCP. The router requires more complex operations to join an AS. It needs manual configuration of its routing table. An end host need not know the ASN of its AS, because ASN is only useful to the routing information exchanges between ASs. To delete a member, one can just

disconnect the host and change the routing tables. But usually no change is needed in the routing table because the routing is based on IP prefix, not complete IP addresses. ASs are simple regions in that merging operations seldom happen. To merge two ASs, three things must be made to be true. First, they must run the same routing protocol. Second, BGP routing policies must be changed to be consistent. Third, the routing tables must be merged. In a special case, if an AS is the provider of another AS, then the BGP routers of the provider AS may not need to change its routing table if the IP scope of the client AS is coincidentally included in that of the first one. This can be regarded as a kind of special merge. But current ASs do not provide much functionality to end hosts. End hosts have no idea of which AS it is in. The routers have much more AS information and functionality than end hosts. Most routing changes in an AS require human intervention.

As for the peer-to-peer systems, their members are end hosts in the systems since it is an overlay network. A host joins the network by obtaining an identifier through some hashing function and looking for some nodes already in the system so as to set up its own routing table and to introduce itself to the system. Hosts are equal to each other, and each host provides the same functionality such as routing, storage, etc, based on particular systems. To leave a peer-to-peer system, a node can gracefully leave by notifying other nodes or leave without any notification. Current peer-to-peer systems do not provide the deleting functions because nodes are equal to each other.

### 3.3 Additional Client Functions

There are two additional client functions: boundary crossing and notification. The explicit notion of the boundary of a region provides an opportunity to enable a rich functionality when activities touch or cross those boundaries. As stated earlier, a boundary is a logical concept, not bound to a particular topological or physical space. There are two aspects to the discussion about boundary management functions, the functions that take place when an activity within a region reaches a boundary and the activity models themselves.

ASs are connected to each other via BGP routers, so ASs have explicit boundaries. Their policies are embodied in the routing information exchanges in BGP routers. They decide the routes exported to the neighboring ASs. The policies are usually determined by the relationships between ASs.

Current Autonomous Systems do not have a notification mechanism. However, with the development of the Internet, we believe that there may be some cases in which notifications are useful. For example, some billing policies may require BGP routers to notify some controllers to count the number of packets.

At present there is no interaction between P2P systems. We believe that as P2P systems become more and more prevalent, they will find themselves competing for resources. A significant shared resource is network bandwidth. Therefore, we expect that multiple peer-to-peer systems need to exchange information in the future. In this case, they need boundary crossing, possibly modifications if data formats are different, and notifications. The boundaries between peer-to-peer systems are different from those of ASs: they are logical boundaries. Each member could be a boundary node as long as it is allowed to exchange information with other peer-to-peer systems. However, since boundary crossing may require some complex functionality, it is possible that only some designated nodes can act as boundary nodes. Such functionality needs further exploration in the future.

## 3.4 Summary

In this chapter we presented several features of the region concept, and discussed how these features apply to Autonomous Systems and peer-to-peer systems. This helps us to design schemes to improve P2P system performances in the next chapter.

# Chapter 4
# Design

## 4.1 Overview

The basic idea behind our design is to provide peer-to-peer (P2P) systems with the underlying Autonomous Systems information so that they can improve their routing performance and availability.

There are three layers in our system, as shown in Figure 4.1. The low layer is a physical layer, which represents the underlying Internet. We will discuss it in detail in Chapter 5. The middle layer is the AS Regions layer. An AS Region is a region that represents an underlying Autonomous System. We propose this layer so as to make Autonomous Systems information available to network applications. In real life, we can obtain routing information from BGP tables. In this thesis, we only use the information, as opposed to possibly affecting it or behavior at this level of abstraction. The high layer is a P2P region layer. The high layer retrieves information from the middle layer via the interfaces defined between them. We will discuss these two layer in detail in section 4.2 and 4.3, respectively.

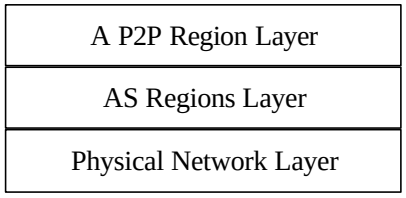



Figure 4.1 System Hierarchy

## 4.2 AS Regions

An AS Region is a region that represents an underlying routing domain and provides topology information to the upper regions. Each AS region corresponds to an Autonomous System. An Autonomous System is a group of networks and routers, subject to a common authority and using the same intra-domain routing protocol. Its boundary is instantiated in border gateways. The primary function of Autonomous Systems is to route packets through itself or to the destination if it is in this AS. Routing within an AS is managed by an internal routing protocol like OSPF or RIP, and routing between ASs is determined by the Border Gateway Protocol (BGP). Each AS uses its ASN to run BGP. The relationship between adjacent ASs are characterized as either provider-client or peering, depending on business relationships. Figure 4.2 shows an example.

We classify the information into two kinds. One is static and coarse-grained information, which is stable over a long time. Such information includes the ASN, the relationships between ASs, the shortest paths between ASs, etc. We call such information macro-information. The other is dynamic and fine-grained information, including latency between two end hosts, bandwidth, etc. Such information varies over time. We call it micro-information.

Macro-information can be provided to the upper layer with less overhead than micro-information, because such information is usually unchanged for a long time, and thus can be statically stored and easily retrieved. One objective of PlanetLab Project [37] is to implement a layer to provide such information to network services. Micro-information usually requires real-time measurements, which adds to the Internet traffic, but provides more accurate information than macro-information.

Some basic but important information that AS regions can provide are the mapping from an IP address to the ASN to which it belongs and the number of AS hops between two end hosts. In today's Internet, ASs are routing domains that implement different policies within

ISPs. Routing within an AS is usually efficient and the network is well managed. We believe that most congestion and failures occur on the links between ASs. Therefore, ASNs and the number of AS hops are a valuable criterion for network latency and other network properties [38]. Thus, two end hosts in the same AS, in other words, with the same ASN, usually experience small latency between them. Meanwhile, the latency between two end hosts in different ASs is more like to be large, which can be represented by the number of AS hops. With the length of AS paths, we can approximately compare latencies between two pairs of nodes in different ASs. The ASN and the AS path information can be retrieved by the cooperation between AS Regions.

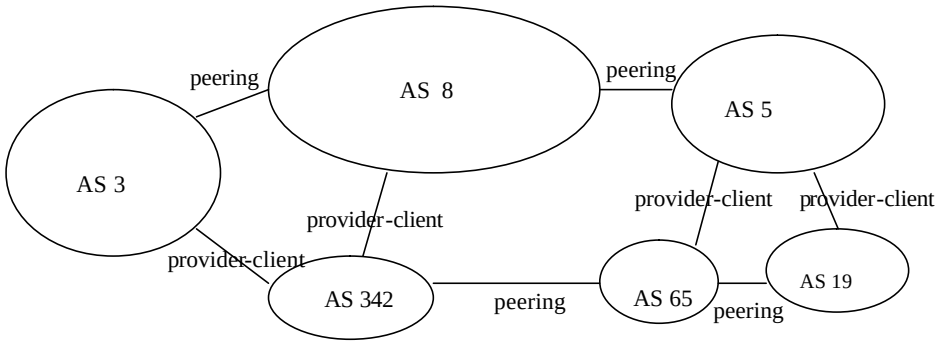



**Figure 4.2: AS Regions**

In our work, the AS regions aggregate the underlying Internet routing information, and provide this information to a P2P region. The AS regions cooperate with each other to provide the information. At present the AS regions provide the following functions:

getASN (IP)

Given an IP address, the underlying AS regions return the ASN of the AS to which this IP address belongs. When an AS region receives this query (from a node in this AS region), it first checks if the IP address in this query is in its own AS. If so, it returns its own ASN. Otherwise, it checks its routing information, for example, something similar to BGP tables, to find out the ASN. In some rare cases, if it still

does not find the ASN, it will forward the query to other AS regions, especially its provider AS regions.

getASPath (source_IP, destination_IP)

Given two IP addresses, source_IP and destination_IP, the underlying AS regions return an AS path between the source IP and the destination IP. Usually the node with source_IP should be in an AS region where this function is called. The AS region searches its routing information for the corresponding AS path. If it does not succeed, it will ask its neighboring AS regions, especially its provider AS regions. Other AS regions receiving this query will check their routing information and send back the AS path if they find it.

getDistance (source_IP, destination_IP, metric)

Given two IP addresses, the underlying AS regions return distance between source_IP and destination_IP based on the metrics specified in the third parameter. If the metric is AS hops, it will return the number of AS hops between them. There may be other metrics in the future. The AS region to which the source_IP belongs implements this function by cooperating with other AS regions.

There are several advantages of using coarse-grained information. First, such information is stable and static, and thus can be collected once and used many times later. Second, such information collection can be done in an aggregate way, thus reducing the network traffic.

### 4.3 A Peer-to-Peer Region

Our model of a P2P system is a generalization of several structured P2P systems. First, in all those systems, files are associated with a key (usually generated by hashing the file name), and each node in the system is responsible for storing data corresponding to a certain range of keys. Second, a primary operation is to look up a key and figure out the

node that stores the corresponding data. Third, each node is assigned an ID in a fixed-size ID space. Fourth, nodes form an overlay network, and each node has the knowledge of a number of other nodes for the purpose of lookup. To accelerate the lookup procedure, each node usually knows more nodes among its logical neighbors, and fewer nodes that are far away. Each node usually also maintains some other data structures like its logical neighbors and physical neighbors. Fifth, the lookup is routed through the overlay network to the destination by a series of query forwardings. Finally, each system has varying degrees of additional mechanisms to provide increased or improved performance in some way or other. Usually such mechanism is about which nodes should be known in one's routing table when there are multiple choices.

Data structures maintained by each node include a routing table and a leaf set. The routing table is the most important data structure, and largely determines the lookup performance. A leaf set is a set of nodes that are logical neighbors of the present node. By logical neighbors, we refer to nodes that are logically closest to the present node in the name space. With a leaf set for each node, we can guarantee the lookup correctness, because in the worst case we can move one step closer to the destination in each step with leaf sets only.

As mentioned before, lookup is a primary function of P2P systems. The lookup function is defined as: given an ID, the P2P system will find the IP address of this ID. A lookup procedure is carried out in the following way: when a node receives a lookup query for an ID, it checks a certain entry in its routing table, and forwards the query to a node in its routing table which is closer to the destination than the present node. The node receiving the query again checks its routing table and forwards the query to a node closer to the destination. In each step the query gets closer to the destination. This procedure continues until the destination node receives the query or the destination IP address is found in a node's routing table. Lookup is very important to the performance of a P2P system because it is a primary operation. Furthermore, in some systems, data transmission follows the lookup path, so the system performance is largely determined by the lookup efficiency. The

underlying Internet topology can help reduce the lookup latency. We will present how AS regions improve the lookup performance in section 4.4.

Second, the idea of caching derives from operating systems, and now has spread to many areas, such as content addressing networks. In P2P systems, we can cache previous lookups for future use to reduce lookup latency. Our key concern in this area is how to reduce the cost of caching while maintaining comparable performance. We propose to improve caching efficiency based on AS information. This issue will be discussed in section 4.5.

Finally, P2P systems are very dynamic, in that nodes are expected to join and leave at will. Therefore, replication is necessary in all peer-to-peer systems to make the data reasonably available. The performance of data transmission can also be improved if we can find a good replica placement scheme. Again, we propose to use AS information to design an improved replication scheme. It is presented in section 4.6.

## 4.4 A Lookup Scheme

In this section, we will present our data structures of routing tables based on ASs. Then we present a lookup scheme based on the data structures and analyze its performance.

### 4.4.1 Introduction

Structured P2P systems are the second generation of P2P systems. They are structured in terms of the overlay organization. In this thesis, we focus on the routing structures of Pastry and Tapestry, because they provide more flexibility, which is crucial to our optimization.

We first describe the general lookup idea in P2P systems. In the current Internet, end hosts are connected to hubs or routers, routers are interconnected to some other routers in an AS, and ASs are interconnected to some other ASs via BGP routers. CIDR [39] aggregates IP addresses to reduce the number of routing entries. A routing procedure is composed of a series of message forwarding operations, from one router to another, from one AS to

another (via BGP routers). In each hop, the message moves closer to the destination.

Conceptually, structured P2P lookup is similar to the Internet routing. However, a P2P network is an overlay network without any dedicated router, and members are peering to each other: each end host acts as both an end host and a router. Each host/node is assigned a unique identifier in a distributed manner when joining the system. The lookup operations are based on node identifiers. As mentioned before, each node knows the IDs and the corresponding IP addresses of a number of nodes, which compose its routing table. The distribution of nodes in a routing table is inversely proportional to their logical distance to this node. That is, a node knows more about its logical neighbors, and less about nodes logically faraway. Similar to the Internet routing, a P2P lookup is also composed of a series of message forwardings. In each hop the message is forwarded to a node that is logically closer to the destination. In many P2P systems it takes only a logarithmic number of steps to reach the destination. Note that a significant difference between the Internet routing and the P2P routing is that the latter is completely based on logical overlay topology, which has nothing to do with the real Internet topology.

Figure 4.3 shows an example of a routing table, which is similar to that in Pastry and Tapestry. In this example, a one-dimensional name space is drawn as a ring, from *00000* to *33333*. Each node identifier consists of five digits, and each digit is base-4. Node *02311*'s routing table should contain a number of nodes satisfying the following property. First, the name space is evenly divided into four parts, shown by two orthogonal bold dashed lines in the figure. Thus, node IDs in the four quarters begin with *0, 1, 2*, and *3*, respectively. We term the four quarters as quarter *0, 1, 2* and *3* based on common ID prefix in each quarter. Node *02311* needs to know one existing node in each quarter. In this case, node 02311 finds *1*****, *2*****, and *3***** in quarter *1, 2*, and *3*, respectively, shown in this figure by three arrows starting from *02311* and pointing to three nodes in quarter *1, 2*, and *3*. Node *1***** refers to any node whose ID begins with *1*. In its own quarter, node *02311* itself satisfies the requirement. Second, quarter *0* (node *02311*'s own quarter) is evenly divided into four parts.

We term the four sub-quarters as *00*, *01*, *02*, and *03*, based on common ID prefix, shown by three thin dashed lines starting from the center point of the ring. Again, node *02311* needs to know one node in each sub-quarter, if each sub-quarter contains at least one existing node. The three nodes are pointed by three dashed arrows starting from node *02311*, as shown in the figure. Then sub-quarter *02* is evenly divided into four parts, and such division continues until each division contains one node only. We can see that in this case, the final division is to divide *02310*, *02311*, *02312*, and *02313* into four separate parts, each containing only one node.

Table 4.1 shows node *02311*'s routing table. The four entries in the first row contain one node in each of the four quarters after the first division. The second row contains four nodes in each sub-quarter of quarter *0*. "*X*" means that node *02311* itself can be put in those entries. To generalize, suppose a name space is *m* bits, and each digit has *b* bits, so each ID consists of *m/b* digits. Under such a name space, a routing table has $m/b \cdot 2^b$ entries. For example, the Tapestry name space is *160* bits, and each digit is base-*16* (*4* bits), so its routing table has *640* entries. A node in an entry *(i, j)* shares the first *i* digits with the present node, and its $(i+1)^{th}$ digit is equal to *j*.

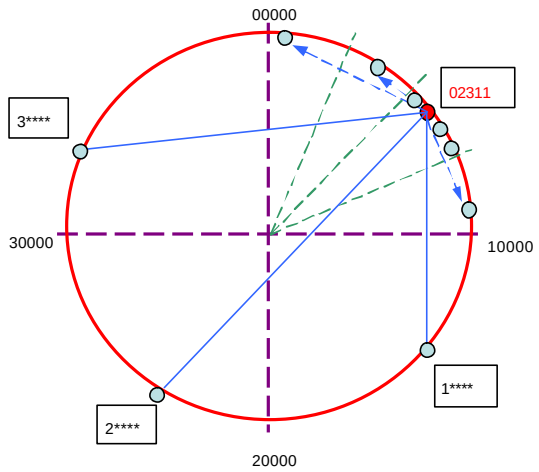



**Figure 4.3: Nodes in *02311*'s routing table.**

**Table 4.1: Node *02311*'s routing table.**

| X | 1**** | 2**** | 3**** |
|---|---|---|---|
| 00*** | 01*** | X | 03*** |
| 020** | 021** | 022** | X |
| 0230* | X | 0232* | 0233* |
| 02310 | X | 02312 | 02313 |

After talking about the routing table, we describe how a lookup is performed with such a routing table. The P2P lookup algorithm can be viewed as a generalization of hypercube routing. A general routing algorithm is shown in Table 42. Figure 4.4 shows how node *02311* looks up the IP address of an ID *23211*. First, node *02311* checks its routing table, finds node *2*****, which shares the longest prefix with the destination in the name space. Node *02311* then forwards the lookup message to node *2*****. Second, when receiving with a lookup message with *23111*, node *2***** checks its routing table and finds a node with *23***, and forwards the message to node *23***. This procedure continues, as shown in the figure. After 5 hops, the message finally reaches node *23211*. Finally, node *23211* returns its IP address to node *02311*, which completes the lookup. We can see that in each hop the message is getting exponentially closer to the destination in the name space. Thus, we can usually finish a lookup within an $O(\log_b N)$ number of forwarding steps.

In real life, because P2P systems are constructed in a decentralized manner, a node does not have complete knowledge of the whole P2P network, and a node may not construct an idea routing table. In this case, next hop will be chosen from nodes that are closer to the destination than the present node. This is shown in Lines 16-19 in Table 4.2.

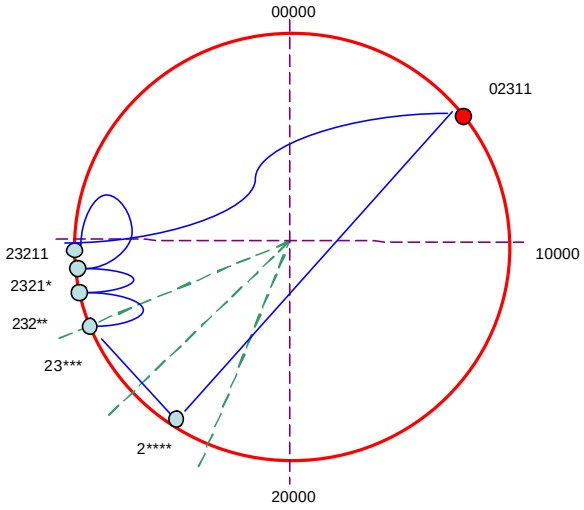



**Figure 4.4: A P2P lookup procedure.**
Node 023111 looks up for the IP address of ID 23211.

**Table 4.2: A Classical Lookup algorithm**

1) $A$: a node that receives a lookup message.
2) $R$: a routing table.
3) $R_r^c$: an entry in a routing table $R$ at row $r$ and column $c$.
4) $L_i$: the i-th closest node ID in the leaf set $L$ of node $A$.
5) $D_l$: the value of the l's digit in a node ID $D$.
6) *SP(A, B)*: the length of the longest shared prefix between ID $A$ and $B$, in digits.
7) *DIS(A, B)*: the logical distance between ID $A$ and $B$.
8) If ($D \in L$) {
9) Look for a node $T \in L$, such that for any node $T' \in L$, *DIS(D, T) = DIS(D, T')*;
10) Forward $D$ to node $T$;
11) Return;
12) }
13) $r = SP(A, D)$;
14) if ($R_r^{Dr} \neq null$) {
15) Forward the message to the node in $R_r^{Dr}$;
16) } else {
17) Look for a node $T \in L \bigcup R$ such that for any node $N$ in $L \in R$, *DIS(T, D) = DIS(N, D)*;
18) Forward to $T$;
19) }

### 4.4.2 An AS-based Lookup Scheme

Our goal is to provide a fast lookup with low overhead. Some P2P systems have tried to match the overlay topology with the underlying topology, as in [28] and [40]. However, the logical structure of P2P systems we are concerned with is one-dimension name space and the lookup is completely based on IDs. The Internet topology is obviously not one-dimensional, so it is hard to match such logical topology with the Internet. Fortunately, we can still improve their performance in an ad hoc way. As shown in Table 4.1, 1**** refers to any node whose ID begins with *1*, so there may be several candidates. If we fill each entry with a candidate satisfying a certain criterion, the whole lookup can then be optimized for this criterion. For example, to achieve a low latency, we should choose nodes

with low latencies for each entry; to avoid passing an AS, we can choose nodes in other ASs for each entry. Latency is the most common criterion for performance. To attain low lookup latencies, an intuitive scheme is to keep all entries its a routing table local to the present node so that each forwarding hop is short. The locality has two levels. The first level is fine-grained locality, which includes several criteria such as latency, bandwidth, etc. These criteria change over time, so they require real-time measurements, adding to the Internet traffic. The second level is coarse-grained locality. Two nodes are considered local to each other if they are in the same AS. For two nodes in different ASs, the shortest AS path between them can be viewed as an indication of latency. Such AS-based information can be obtained from AS regions with low overhead because each AS region maintains such information, and such information is usually static.

In this thesis, we use AS information to reduce lookup latency. To find out whether two nodes are in the same AS, a simple solution is to compare their ASNs. If two nodes have the same ASN, then they are in the same AS and will probably have a small latency between them. Otherwise, they will probably have a large latency, and we refine the latency by the length of their AS path. Therefore, the ASN and the AS path become our criteria of latency. Furthermore, we believe that it is valuable to distinguish between the local AS and other ASs, because such distinction may be useful for security, policy, etc. We can also choose nodes to maximize the number of different ASs covered by the interAS table. This method can improve system robustness, because network failures often happen at BGP links between ASs, and network partitions usually happens in units of AS. Other more sophisticated methods can be used too, but we begin with the simplest. In the following sections, we will use the following algorithm: when receiving a query, a node always first checks its intraAS routing table; if no proper entry is available, then check its interAS routing table instead. Pseudo code for the lookup algorithm is shown in Table 4.4.

In our scheme, each node has two routing tables. Both tables have the same structure as that in Pastry and Tapestry. One routing table contains only nodes in the owner's AS, namely an

**Table 4.4: Our Lookup algorithm**

```
1)  A: a node that receives a lookup message.
2)  R_r^c: an entry in a routing table R at row r and column c.
3)  intraR: an intraAS routing table.
4)  interR: an interAS routing table.
5)  L_i: the i-th closest node ID in the leaf set L of node A.
6)  D_l: the value of the l's digit in a node ID D.
7)  SP(A, B): the length of the longest shared prefix between ID A and B, in digits.
8)  DIS(A, B): the logical distance between ID A and B.
9)  If (D∈ L) {
10)     Look for a node T∈ L, such that for any node T'∈ L, DIS(D, T) = DIS(D, T');
11)     Forward D to node T;
12)     Return;
13) }
14) r = SP(A, D);
15) if (intraR_r^Dr ≠ null) {
16)     Forward the message to the node in intraR_r^Dr;
17) } if (interR_r^Dr ? null) {
18)     Forward the message to the node in interR_r^Dr;
19) } else {
20)     Look for a node T∈ L ⋃ intraR ⋃ interR, such that for any node N in L∈
        intraR ⋃ interR, DIS(T, D) = DIS(N, D);
21)     Forward to T;
22) }
```

intraAS routing table; the other contains only nodes in other ASs, namely an interAS routing table. Such structures can not only achieve low-latency lookup, but also make possible certain desirable policies. A routing table's size is determined by the name space, as mentioned before. Table 4.3 shows a node *02311*'s two routing tables. In Table 4.3, the name space is *10* bits, and each digit is *2* bits, so each table has *5* rows and *4* columns.

### 4.4.3 Performance Analysis

The lookup algorithm always moves closer to the destination in each step, so it is guaranteed to end in finite time. As to the time complexity, it is easy to see that it is the

same as that of Pastry and Tapestry, *O(logN)*, where *N* is the number of nodes in a P2P region. The space complexity is about twice that of Pastry and Tapestry, $O(M \cdot 2^K)$, where *M* is the number of digits in an ID, and *K* is the base of each digit (The name space is $M \cdot K$ bits.).

**Table 4.3: Node *02311* with two routing tables.**

**(a) The intraAS routing table (AS 3)**

| | | | |
|---|---|---|---|
| X | 1**** | 2**** | 3**** |
| 00*** | 01*** | X | 03*** |
| 020** | | 022** | X |
| | X | | 0233* |
| | X | 02312 | |

**(b) The interAS routing table**

| | | | |
|---|---|---|---|
| X | 1**** (2) | 2**** (437) | 3**** (84) |
| 00*** (63) | 01*** (981) | X | 03*** (195) |
| 020** (17) | | 022** (86) | X |
| 0230* (4) | X | 0232* (5) | |
| | X | | 02313 (63) |

*1*** in (a) means that a node whose ID begins with *1* can fill that entry as long as it is in AS *3*. *020** *(17)* in (b) means that a node beginning with *020* in an AS different from AS *3* can be filled here; *(17)* means that the node in this entry is in AS *17*. *X* means that this entry can be filled by the owner's ID, so it is not necessary to find a node to fill in it. For two entries with the same position in the last line of both tables there can be at most one entry filled because each node can only be in one AS. Blank entries means that the node has not found a proper node to fill in.

### 4.4.4 Routing Table Setup

With such routing table structures, a joining procedure will be different from that in Pastry and Tapestry. In a P2P system, assume a new node, say *n*, knows a node already in the system, say *m*, and *n* wants to join the system. The joining procedure is similar a lookup procedure: node *n* sends *m* a joining message containing its ID and its ASN. Node *m* then performs a lookup query for node *n*'s ID as a way of helping *n* to join. Node *m* first checks its intraAS routing table, and if a valid entry is found in the corresponding position, it will forward the query to the node in that entry. Otherwise, it will check its interAS routing table to find a node and forward the query to it. The procedure is described in Table 4.4. This forwarding operation continues until the query reaches a node that is logically closest

to *n*.

Node *n* can then set up its routing tables with the routing tables from nodes along the lookup path. Its interAS routing table is comparatively easy to set up. If a node in the lookup path is in the same AS as node *n*, certain rows in its interAS routing table can be directly copied to node *n*'s interAS routing table (to be explained below). If a node is in a different AS, certain rows in its interAS routing table can be copied to node *n*, except that those entries in the same AS as node *n* in the interAS routing table are copied into node *n*'s intraAS routing table.

The intraAS routing table setup is more complicated. If most nodes in the lookup path are in the same AS as node *n*, *n* can set up its intraAS routing table by copying rows in those nodes' intraAS routing tables. In some cases, node *n* may not find enough valid nodes for its intraAS routing table. We solve this problem by doing extra search: if node *n* finds some nodes in its AS during the first few lookup steps, it will ask one of the nodes to do another lookup, so as to find more nodes in its own AS, because nodes already in the system probably knows more about nodes in their own AS. This additional search result can be used to fill *n*'s interAS routing table. It is possible that sometimes node *n* may not find any nodes in its own AS during its joining procedure, either due to the small number of nodes in its AS or because nodes in the joining lookup path do not know any nodes in that AS. In this case, it can just join with an empty intraAS routing table, but can ask nodes in its interAS routing table to store its ID in their interAS routing tables so that other nodes or new nodes in that AS may find it later. It may also search exhaustively for nodes in its AS, but this is generally too costly and thus not practical. After setting up its two routing tables, node *n* will send a message to all nodes in its routing tables, and those nodes will add node *n* to their routing tables in an appropriate position if applicable.

Figure 4.5 shows an example of a joining procedure. Assume that a node with ID *0231* wants to join a P2P system and it already knows node *3210*, a node already in the system

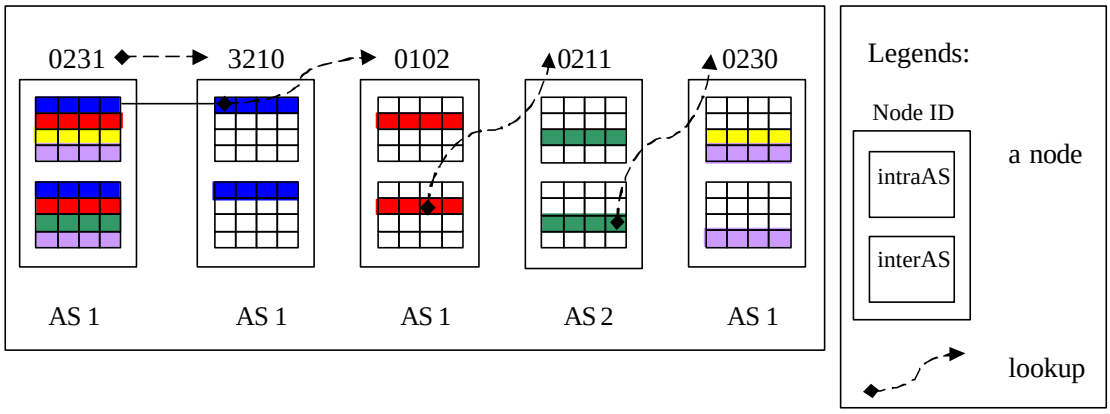



**Figure 4.5: A node joining procedure.**

Dashed arrows show a lookup path in the joining procedure. Legends are shown on the right side. New node *0231* sets up its two routing tables by copying different rows from nodes in the lookup path. The bars with different degree of gray in node *0231* show from which nodes the rows are copied. Note that *0211* is in a different AS from *0231*, so its third row in its intraAS table cannot be used by *0231*'s intraAS routing table. Instead, it is copied from *0230*. (In this case, *0230* happens to be in the same AS as 0231.)

and in the same AS. First, 0231 sends a joining message to node *3210*. The joining message is similar to a lookup query for *0231*, except that the message type is different, so receivers will do more than just message forwarding. *3210* receives this message and checks its intraAS routing table. In this example, it finds node *0102* and forwards this message to it. Meanwhile, *3210* sends back its first rows of both intraAS and interAS routing tables to *0231*. Suppose node *0102* does not find a valid entry in the corresponding position of its intraAS routing table; it checks its interAS routing table, finds *0211*, and forwards the message to node *0211*. Meanwhile, *0102* also sends back its second rows of both intraAS and interAS routing tables to *0231*. Node *0211* receives this message, and checks its interAS routing table first and then the intraAS routing table, because it is in a different AS and cannot find a node in its intraAS routing table that is in the same AS as *0231*. Luckily, in this example it finds *0230* in its interAS routing table, which is in the same AS as *0231*. It then forwards the message to 0230. The third rows of *0211*'s two routing tables cannot be

used by *0231*'s intraAS routing table, since they are in different ASs. However, if *0211* finds some nodes in its interAS routing table are in *0231*'s AS, it will send such entries to *0231*, too. Node *0230* will find that the corresponding entry, the fourth row and the second column in this case, is empty, because *0231* has not joined the P2P system. *0230* knows it is the end of the path because *0231* is within the scope of its leaf set and no one is closer to *0231* than itself. The previous node *0211* is in a different AS, so *0230* sends both the third and fourth rows of its intraAS routing table to *0231*.

On the other hand, in each hop node *0231* can be used to update entries in those nodes' routing tables. If, for example, the corresponding entry in *0102* is empty, node *0231* can then be put into that entry. Furthermore, *0231* can be propagated to all nodes in the row that this entry is in and in the rows below. However, such propagation may be costly because it is an $O(N \times M \times R)$ operation in which $N$ is the base of the ID digit, $M$ is the number of digits in each ID, and $R$ is the depth of the propagation. There are several methods for reducing the overhead: (1) 0231 is only propagated to nodes that are in its own AS in the first-level routing table; (2) $R$ is limited to 1 or 2; (3) loop detection can be used to prevent duplicated updates.

### 4.4.5 Routing Table Update

In a P2P system, nodes join and leave the P2P systems at random. Meanwhile, a node may not find enough nodes to meet its needs within a reasonable overhead due to its partial knowledge of the system. Therefore, routing table updating becomes a crucial component of a P2P system.

Dedicated updating is expensive, and our idea is to update routing tables by piggybacking during lookups. During a lookup, besides a query message, two nodes of a hop can exchange part of their routing table entries. They decide whether to exchange information or not based on their conditions. If the two nodes are in the same AS, their intraAS and interAS routing tables (respectively) can exchange information. If they are in different ASs,

then such exchange will be done between their interAS routing tables, and be less useful to their intraAS routing tables, although they may find some nodes in their own ASs from the other's interAS routing table.

Figure 4.6 shows an example. We look up the IP address of ID *0231*. Node *0231* is searched in the path *3210, 0102, 0211*, and *0230*. Node *3210, 0102,* and *0211* are in the same AS. *0231* is in a different AS from *0211*, and *0231* is in a different AS from *0230*. Node *3210* and *0102* can exchange the first row in both intraAS and interAS routing tables. Node *0102* and *0211* can exchange the first two rows in both routing tables. Node *0211* and *0230* can exchange the first three rows of their interAS routing tables, etc. Thus, a lookup procedure acts as a way of exchanging information.

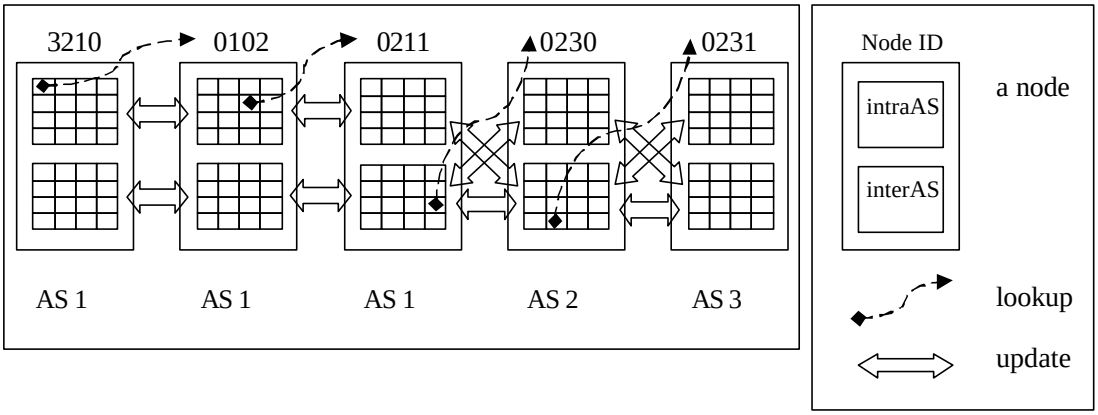



**Figure 4.6: A routing table update during a lookup.**

The possible updates are shown by the solid left-right block arrows. The legend is shown on the right side. The lookup procedure is shown by dashed single arrows. The block arrows show the potential updating.

### 4.4.6 Additional Functionality Enabled by Regions

Besides lookup performance, AS Regions enable more functionality, especially on security and policies. For example, one possible scheme is to keep traffic within an AS as much as possible. There are a variety of reasons for this scheme, such as cost, performance, or

privacy. Since an AS is often a definition of a corporate boundary, a corporation may prefer that the traffic of its applications remains within its own AS. An enhanced region reflects not only the IP level boundaries of the corporation, but also information about how the application overlay can stay within those boundaries. The power of the region abstraction is that alternative regions can be defined by the use of invariants other than those arising from ASs, in order to define new routing regimes.

Furthermore, two possible policies are enabled as follows. One is to avoid passing some unsafe ASs. To implement this policy, a node just avoids using any node in its interAS routing table which is in the unsafe ASs. This simple implementation has a defect: Although in the P2P Region the lookup does not pass any unsafe AS, it cannot guarantee that a lookup does not pass any unsafe ASs in the underlying AS regions. A possible remedy is that a node always queries the AS region for an AS path between the current node and the chosen node to see if the AS path between them passes by any unsafe AS when looking up for the next hop. If so, choose another path.

The other is a more complex, in which again each AS reflects a corporate boundary, but now rather than requiring that traffic stay within one boundary, a corporation may have a set of priorities. The preferred option is to stay within itself. But if that is not possible, there may be an ordered preferential list of alternatives. Such a set of regions may include the elements of a variety of application overlay networks, for different applications or application suites. A single set of region definitions may serve many applications, helping to make routing choices for each using the same set of corporate criteria such as cost, efficiency or privacy policies. Within each such AS there may be an application network overlay router.

### 4.5 A Caching Scheme

Caching derives from operating systems. In operating systems, caching reduces memory access latency based on two localities: temporal locality and spatial locality. Similarly, in

P2P systems caching mechanism serves to reduce lookup latency. Previous lookup results can be used to reduce future lookup latencies. Temporal locality in P2P systems is that if an ID is queried by a node, it may be queried again, either by this node or other nodes, in the near future. The spatial locality has two meanings. One is that if an ID is queried by a node, other IDs close to it in the name space may be queried either by this node or other nodes. The second is that if a node queries an ID, other nearby nodes may query the same ID in the near future. The cost of caching is additional operations and space requirements.

As mentioned before, there are data caching and address caching. Here we only talk about address caching. Our idea is to cache the lookup result addresses only once in each AS along a lookup path. The key is to reduce the caching space. Because the destination IP is found at the last hop in a lookup, caching can only be done by reversing the lookup path. To be more specific, we introduce two terms: *an up node* and *a down node*. We define these terms by an example. Figure 4.7 shows two hops of a lookup path, in which node *A* forwards a lookup message to node *B*, and then *B* forwards it to node *C*. In Figure 4.7, the up node of node *B* is the node that sends a query to *B*. In this example, the up node of *B* is node *A*. A down node of node *B* is the node to which *B* sends a query. In this example, the down node of node *B* is node *C*. Each node in the middle of a lookup path has one up node and one down node. The node at the beginning of a lookup has only a down node, while the end node has only an up node.

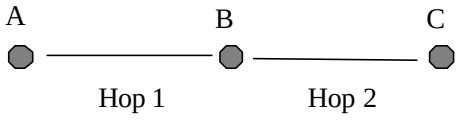



Figure 4.7: An example of an up node and a down node

During a lookup procedure, each node temporarily stores the lookup information including its up node and the lookup ID. After the destination IP address for the lookup ID is found, the node at the end of the lookup path sends a caching request to its up node. The cache

request includes the ID range on the node with the IP address. This is to make use of spatial locality. When its up node receives the cache request, it will check if it is in the same AS as its down node. If so, it will ignore this information. Otherwise, it will store this information. In both cases, it will forward the caching request to its up node. The caching procedure stops when the caching request reaches the beginning node in the lookup path. The beginning node always caches the information. This caching path is to make use of temporal locality. This procedure is shown in Figure 4.8.

With such a scheme, we treat all ASs, big or small, equally, by caching only once in each AS in a lookup path. We hope that we can still get good performance with fewer cached nodes. This scheme can also applies to data caching with minor modifications.

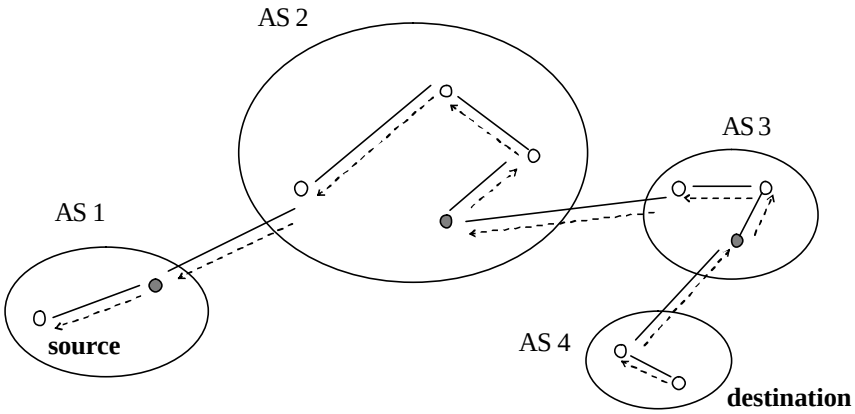



**Figure 4.8: The caching algorithm**

The solid line shows the lookup path, and the dashed line shows the caching path. The ovals represent the ASs. The small circles represent nodes and those that are black represent nodes with caches of the lookup result.

## 4.6 A Replication Scheme

Peer-to-peer systems are dynamic. Nodes frequently join and leave, sometimes without notification. To guarantee resource availability and to improve robustness, replication is necessary. P2P system performance depends not only on the lookup efficiency, but also on the replication efficiency. Furthermore, an operation in a P2P system usually consists of a

lookup and a data accessing. The second part, communication between a source and a destination, is usually a long procedure with large traffic and overhead. That performance can be greatly affected by the choice of a replica. For example, in a P2P system for data sharing, the main overhead is the download procedure after a lookup. There has been much work on server selection [41][42], and some of these approaches can be applied to peer-to-peer systems [22][25]. Current structured P2P systems usually replicate data on nodes that are logical neighbors of the present node. They try to achieve even replica distribution in this way because IDs are randomly generated in the name space. This may not be effective or enough because they rely solely on the randomness and have no control over where the replicas go. In this section, we present a scheme with explicit replica placement with the help of AS information.

### 4.6.1 Static Strategy

Our basic idea is to provide AS region information to P2P regions so as to make better decisions on data transmission. When a piece of data is inserted into a P2P network, it should be replicated somewhere, so the first issue is where replicas should be placed. AS Regions represent the Internet topology, and their boundaries often become boundaries of congestion and network partitions. Thus, it is reasonable to replicate in different ASs to improve data availability. Since there are many ASs of different sizes, it is not practical or necessary to place one replica in every AS. We replicate data only in some "important" ASs. The definition of "important" ASs varies in different applications and policies. For example, replicas can be placed in nearby ASs, such as peering ASs, parent ASs and customer ASs. Replica placement also depends on how much information a node knows about the underlying AS regions. The initial number of replicas can be a fixed number, and the number of replicas can evolve based on popularity and dynamic replication algorithms discussed in the next section.

The second issue is how to locate replicas. An intuitive idea is to make a lookup

automatically return the nearest replica. One shortcoming is that such automation excludes the source's freedom to choose replicas based on its own interests. In many P2P systems, data are replicated on the owner's logical neighbors. This method produces a natural and automatic transfer when the owner fails, because subsequent lookups will automatically find those neighbors. With AS Regions, we can have explicit control over where data are replicated. We propose a scheme which deploys replicas on nodes in different ASs chosen from a node's interAS routing table. To choose replication nodes from the original node's interAS routing table, there are many alternatives. A simple scheme is to maximize the number of different ASs where replicas are placed. Also, the relationship between the ASs may be used. In this way, logical neighbors of the present node now store only a list of replica pointers, instead of real data. In this way, a lookup will automatically return a replica list, and it is the source node that decides which replica to access based on its own preference. The preference criteria may be latency, bandwidth, security, etc. Figure 4.9 shows an example of our replication scheme.

### 4.6.2 Dynamic Strategy

In a dynamic scheme, replicas propagate or are removed based on frequency of access. If many nodes in an AS frequently access some data, then the data can be replicated several times in this AS. Meanwhile, with AS region information, we can improve this scheme by restricting the number of replicas an AS can hold, because it is not necessary to have many copies within one AS. Also, for some frequently accessed cached data, the P2P system can change it into replicated data, etc. In addition, more sophisticated schemes can be designed as needed for more complex policy requirements.

## 4.7 Summary

In this chapter, we described a generic P2P lookup algorithm, and proposed an improved lookup algorithm based on AS regions. The key is to improve lookup efficiency with coarse-grained information, which incurs less overhead than previous schemes.

We also designed a caching scheme to reduce the storage requirements of caching, and a static replication scheme to achieve even data distribution. Both schemes try to achieve informed control over caching and replication.

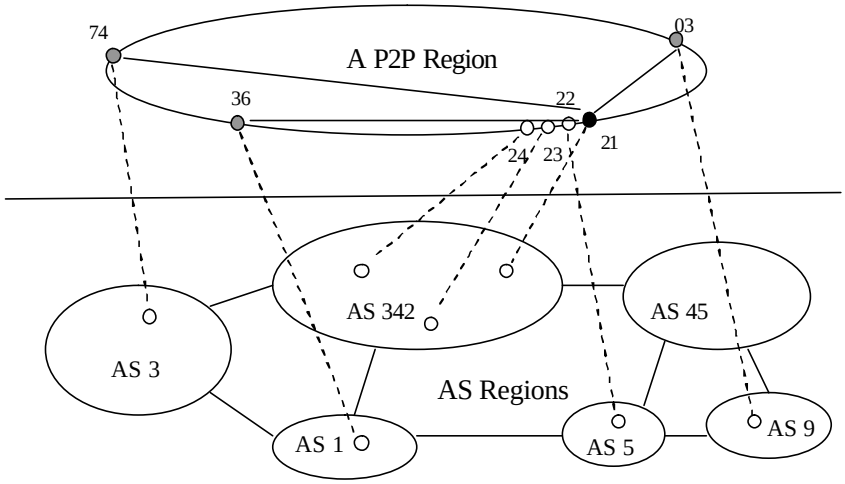



**Figure 4.9: The replication scheme**

The upper part is a P2P region, one-dimensional name space. The lower part is AS regions. Dashed lines show the mapping between logical nodes in the P2P region and physical nodes in AS regions. The black node (21) is an original node. Grey nodes *03*, *36*, and *74* store real data replicas. Node *22*, *23*, and *24* are node *21's* logical neighbors, storing a replica list like ($IP_{21}$, $IP_{03}$, $IP_{36}$, $IP_{74}$).

# Chapter 5

# Implementation and Evaluation

## 5.1 Overview

The goal of this thesis is to improve the performance of a peer-to-peer region with AS regions. In this chapter, we present an implementation and an evaluation of how the region architecture can meet this goal. Results show that AS regions provide coarse-grained information to achieve similar performance to methods with fine-grained information, and also make other peer-to-peer functions more efficient.

## 5.2 Implementation

### 5.2.1 System Architecture

As described in Chapter 4, our system is divided into three layers: a physical networks layer, an AS Regions layer, and a P2P layer. The physical networks layer represents the underlying Internet, composed of end hosts and routers interconnected to each other. The AS Regions layer is composed of multiple AS regions. Each AS region consists of a number of nodes and routers. The AS regions cooperate to retrieve the underlying Internet information from the physical layer and provide such information to the P2P layer. The P2P

layer is a P2P region composed of several components: a P2P control module, a lookup module, a caching module, and a replication module. The routing, caching, and replication modules are implemented independently of the P2P control module so that different schemes can be easily added. The system architecture is shown in Figure 5.1.

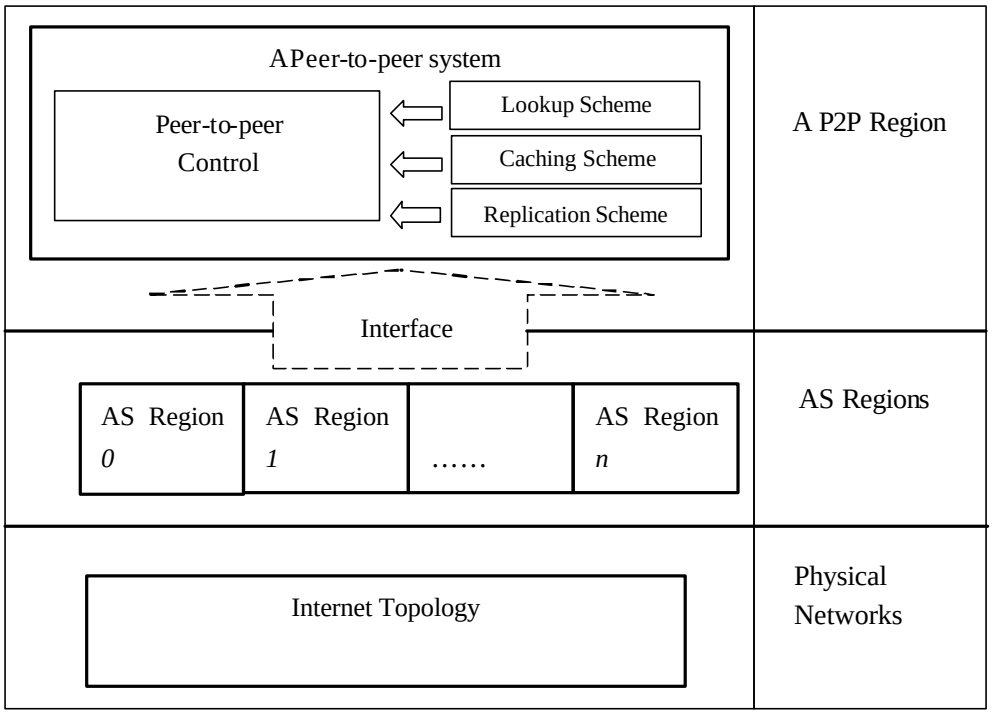



Figure 5.1: System Architecture

### 5.2.2 The Physical Networks Layer

The physical networks layer represents the underlying physical network. In the current Internet, there are more than *10,000* ASs and about *35,000* connections between the ASs. For example, by 2002, Skitter [43] consists of *11,122* ASs and *35,752* peering sessions. In our simulation, the underlying network is generated by the BRITE Topology Generator [44]. The topologies we generate are two-level networks. The first level is the AS level, and the second is the router level (In the simulation we use routers as P2P nodes, since conceptually there is no difference.). BRITE also assigns latency and bandwidth to each

link. In this way, we generate the underlying networks.

In the following sections, we use four topologies: *td40w100w, td40w100g, td100w40w, td100w40g*. The four topology names specify their topologies. In "*td40w100g*", "td" means that the topology is generated in a top-down manner. That is, BRITE first generates an AS-level topology, with each node representing an AS, and then it generates topologies within each AS separately. "*40w*" means there are a total of 40 ASs, interconnected using the Waxman model [45]. The Waxman model refers to a generation model for a random topology using Waxman's probability model for interconnecting the nodes of the topology, which is given by:

$$P(u, v) = a \cdot e^{-d/(ß L)}$$

where $0 < a, ß < 1$, $d$ is the Euclidean distance from node $u$ to node $v$, and $L$ is the maximum distance between any two nodes. "100g" means that each AS contains 100 nodes interconnected in a power law model. In the power law model, nodes are interconnected according to the incremental growth approach. When a node $i$ joins the network, the probability that it connects to a node $j$ already belonging to the network is given by:

$$P(i, j) = d_j / \sum_{k \in V} d_k$$

where $d_j$ is the degree of the target node, $V$ is the set of nodes that have joined the network and $\sum_{k \in V} d_k$ is the sum of out-degrees of all nodes that previously joined the network. Basically in a power law model, nodes are interconnected into a tree. Table 5.1 summarizes all the four topologies we use in the simulation.

### 5.2.3 The AS Regions Layer

The AS regions aggregate the underlying Internet routing information, and provide it to the P2P region. The AS regions cooperate with each other to provide the information. As discussed in Chapter 4, our AS regions provide the following functions: getASN (IP);

getASPath (source_IP, destination_IP); getDistance (source_IP, destination_IP, metrics).

In the simulation, we make two assumptions on the AS regions. First, if two ASs are connected, then they are peering. Second, packet routing follows the shortest AS path. We can set up BGP tables for each AS under the two assumptions. Each AS region also knows all nodes in its own domain.

Table 5.1 Network topologies in the simulation

| Topologies | Between ASs | | Within an AS | |
|---|---|---|---|---|
| | # ASs | Model | # Nodes | Model |
| Td40w100w | 40 | Waxman | 100 | Waxman |
| Td40w100g | 40 | Waxman | 100 | Power Law |
| Td100w40w | 100 | Waxman | 40 | Waxman |
| Td100w40g | 100 | Waxman | 40 | Power Law |

### 5.2.4 A P2P Layer

A P2P region consists of a number of P2P nodes. Each node has some core information including its IP address, its ID, its ASN, etc. It also maintains a number of data structures discussed in Chapter 4, including two routing tables and a leaf set. It can retrieve information from interfaces provided by AS regions. Different routing schemes, caching schemes and replication schemes can be easily implemented in a P2P region.

In our examples, we use a name space of 18 bits, and each node ID consists of six base-8 digits. Our name space, compared with those in Chord (160 bits), Tapestry (160 bits), and Pastry (128 bits), is very small. But this does not affect our simulation because all schemes are implemented in the same name space. Such name space also provides us a small routing table. Based on the analysis in section 4.4.1, a routing table in our simulation has only 48

entries, while a Tapestry routing table has 640 entries. Note that with such a small routing table, we still have the similar number of lookup hops to that of a Pastry system with a 128-bit name space and base-16 digits. We give an analysis in section 5.3.2. In our simulation, we assign node IDs in a centralized way, thus avoiding ID conflicts even with this small name space.

To evaluate our work, we implement four lookup schemes namely *Raw*, *AS*, *Latency* and *LatAS*. A *Raw* scheme is a P2P system without any optimization. A node in a *Raw* scheme has a simple routing table, and sets up its routing table by randomly choosing nodes without any criterion. This is the basis of comparison.

An *AS* scheme is the scheme designed in the Chapter 4. A node has two routing tables. An intraAS routing table is filled with nodes within the owner's AS, and an interAS routing table is filled with nodes from other ASs, based on the lengths of AS paths between nodes and the owner. The lookup/routing is implemented by first checking the intraAS routing table. If valid entry cannot be found, it then checks the interAS routing table.

A *Latency* scheme is a scheme similar to Pastry and Tapestry. For this scheme, each node has only one routing table filled with nodes based on latency.

A *LatAS* scheme is a hybrid scheme that combines the *AS* and *Latency* schemes. Similarly to the *AS* scheme, a node in this scheme has two routing tables, but nodes in an intraAS routing table are chosen based on latency, instead of being randomly chosen.

## 5.3 Evaluations

### 5.3.1 Overview

The goals of our work are to improve application-level performance and to generalize the region concept. In this section we present a performance evaluation of how well our schemes meet these goals, and show how regions can be useful to applications.

Our evaluation includes several parts. First, we analyze the occupancy of routing tables, which helps us to understand how P2P systems work and provides a way to estimate the average number of hops. Second, we study how underlying AS regions help to improve P2P lookup efficiency. Third, we design a caching scheme and test how it further reduces lookup latency. Finally, we make use of region information to achieve informed replica placement in P2P systems. Each scheme is tested on four topologies: td40w100w, td40w100g, td100w40w, td100w40g. We implemented our scheme in about 10,000 lines of Java codes, including about 2,000 codes for AS regions, 6000 lines of code for a P2P region, and 2,000 lines of evaluation code.

### 5.3.2 Peer-to-Peer Routing Table Analysis

Based on previous analyses in Pastry, and Tapestry, the average number of lookup hops is largely determined by the digit base and the number of nodes. On the other hand, a typical lookup begins with a node, say *n*, in the first row of a node's routing table, then a node in the second row of node *n*'s routing table, and then the third, …, until the very node is found, usually in the last row with valid entries. Therefore, routing table occupancy is closely related to the average number of lookup hops. In light of the estimation of the routing table occupancy, we can estimate the average number of lookup hops.

Because of the large name space, nodes usually have very large routing tables. For example, a Tapestry node has a routing table with 40 rows and 16 columns. That is, there are 640 entries per table. Furthermore, each entry usually contains multiple nodes. In Tapestry, each entry contains three nodes. Therefore, each routing table can contain up to a maximum of 1920 nodes. Keeping those nodes' state up to date requires a large amount of maintenance cost. In actual network, the network size is much smaller than the name space, so many entries are empty. In this section we will discuss the relationship between network size and the fraction of valid entries.

In our simulation, we use a relatively small name space, compared with those in Pastry and

Tapestry. As mentioned before, in our system each ID consists of six base-8 digits, which means our name space supports at most 256K nodes. In each routing table, either an intraAS table or an interAS table, the number of rows is equal to the number of digits for an ID, and the number of columns is equal to the base of each digit. Therefore, our routing table has 6 rows and 8 columns, for a total of 48 entries. It is a relatively small routing table. Because the name space is small, we can achieve high node density in the name space with a relatively small number of nodes. With the maximum network size of 4000 nodes in the following evaluation, we occupy 1.56% of the name space. With the minimum network size of 500 nodes in the following evaluation, we use 0.195% of the name space. Meanwhile, Pastry uses 128 bits, so its name space is $2^{128}$ nodes, and to the best of my knowledge, the biggest simulation size attempted now was 100K nodes [25]. Hence the occupancy is only about $2.94\times 10^{-34}$% of the name space. A Pastry P2P system will need $5.3\times 10^{36}$ nodes to reach an occupancy rate of 1.5625%. It is easy to see that due to the sparsity in the name space, many routing table entries are empty, especially entries near the bottom rows.

Table 5.2, Figure 5.2, and Figure 5.4 show an average occupancy of routing tables in a 500-node P2P system. This is a small network. 500 nodes are randomly chosen from td40w100w. Figure 5.4 shows the node distribution in each AS. It is obvious that the number of nodes for an interAS routing table is more than that for an intraAS routing table, so the interAS routing table has more node candidates for its entries. Figure 5.2 (a) and (b) show the occupancy of each entry in the two routing tables, respectively. We can see that the corresponding occupancy for the interAS routing table is much higher than that for the intraAS routing table because of the difference in the number of nodes that feeds the two tables. Entries in the same row have similar occupancies because they have the same restriction on the prefix length of node IDs. Table 5.2 summarizes the average occupancy of each row, which shows the trend of occupancy between each row more clearly. Note that even the smallest occupancy, in the bottom row of the intraAS table, is 12.5%, which is

quite high considering the small number of nodes. Actually the reason for this is because a special empty entry is always counted in each row. The special entry is an entry that matches the owner's ID in each row. Each row has one such special entry. For example, a node 023112 can be put into positions (0, 0), (1, 2), (2, 3), (3, 1), (4, 1), and (5, 2) in its routing table because its ID matches these positions. Our routing table has 8 columns, so it is guaranteed that the minimum occupancy is 12.5%, which means all entries in this row are empty except a special one. If the special entry is not counted, the maximum occupancy will be 87.5%, instead of 100%.

Table 5.3, Figure 5.3, and Figure 5.5 show the routing table occupancy of a 4000-node P2P system. We can see that the first three rows in interAS routing tables are all full. But the last three rows of the intraAS routing tables and the last two rows of the interAS routing tables are almost empty. Not surprisingly, routing table occupancy of the 4000-node P2P system is much higher than that of the 500-node P2P system.

The occupancy of routing tables can be estimated using the number of nodes and the ID structure. For example, in our system, each ID consists of six base-8 digits, and the maximum network size is 4000. An entry in the first row has the least restriction: only the first digit is fixed. Therefore, supposing that nodes are evenly distributed in the name space, there will be about 500 (4000/8) node candidates for each entry in the first row. (Here we count the intraAS and interAS routing tables together.) Similarly, each entry in the second row has about 62 (500/8) candidate nodes, each entry in the third row has about 8 (62/8) nodes, each entry in the fourth row has about 1 (8/8) node, and so on. When there are at least 8 candidate nodes for an entry, the entry has a good chance of finding a node to fill in (in an ideal case). This explains why the first three rows are all 100% filled (shown in Figure 5.3 (b)). But when there is only 1 candidate node, an entry may not be able to find a node to fill in either because nodes are not absolutely evenly distributed in the name space or because nodes only have partial knowledge of the whole region. This is why the fourth row is only about half occupied and the last two rows are almost empty. The following

formula shows how to calculate the sum of approximate occupancy of the intraAS table and the interAS table:

$$O_i = N/c^{i+1}/2 + 12.5\%$$

where $O_i$ is the average occupancy for row $i$, $c$ is the number of columns, and $N$ is the total number of nodes. It is verified by Table 5.3.

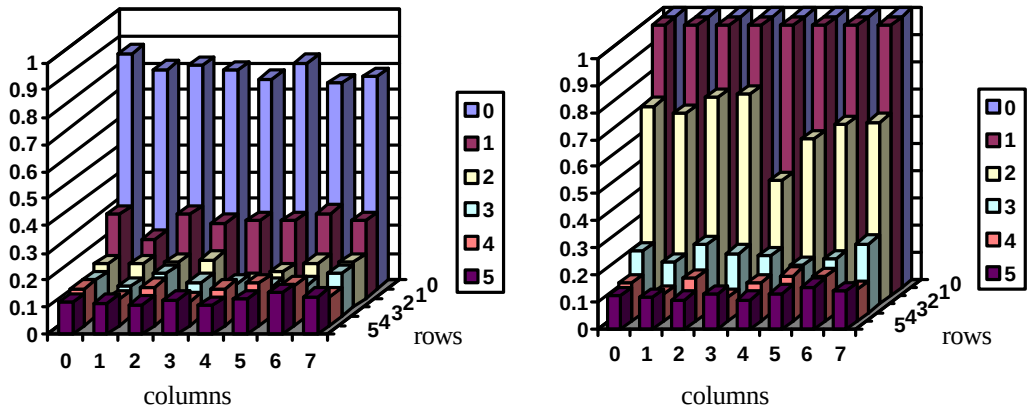
The number of lookup hops is approximately equal to the average number of valid routing rows. For example, the first three rows in Table 5.3 are full, the fourth row is half empty, and the last two rows are almost empty, so the average number of lookup hops for a 4000-node P2P network will be about 3.5, according to Tables 5.1 and 5.2. This value is similar to the number of lookup hops in a 128-bit Pastry with 60,000 nodes [46]. Therefore, our small name space with 4000 nodes will not affect our simulation just because of an atypical number of lookup hops.

### 5.3.3 Lookup Evaluation

In this evaluation, we compare the lookup efficiencies of different schemes. As we mentioned before, there are four lookup schemes: a *Raw* scheme, a *Latency*-based scheme, an *AS*-based scheme, and a hybrid *LatAS* scheme. We test lookup latencies in P2P regions with different sizes under four topologies in the four schemes. Figure 5.6 to Figure 5.9 show the lookup performances of the four topologies.

#### 5.3.3.1 Lookup Latency Evaluation

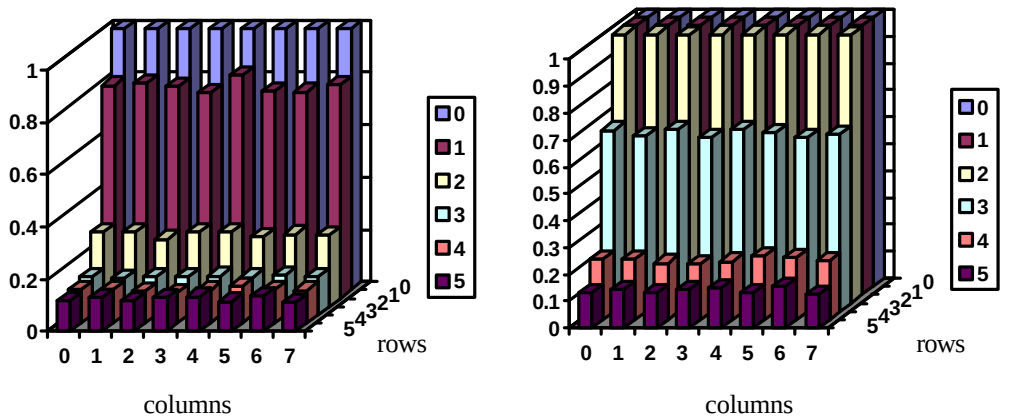
Figure 5.6 (a), Figure 5.7 (a), Figure 5.8 (a), and Figure 5.9 (a) show the lookup latencies of four schemes on four topologies. The X-axis shows the P2P network size. The size increases from 500 nodes to 4000 nodes. Nodes are randomly chosen from underlying network topologies. The Y-axis shows the lookup latency in seconds. The most significant result is that there is not much difference in lookup latencies between *Latency* scheme and




**(a) intraAS** **(b) interAS**

**Figure 5.2: Routing Table Occupancy of a 500-node AS-based p2p network.**

A network topology used here is td40w100w. There are 40 ASs, and each AS has 100 nodes. ASs are connected in Waxman method, and nodes within each AS are connected in this way too. (a) shows the occupancy of intraAS routing tables. (b) shows the occupancy of interAS routing tables.




**(a) intraAS** **(b) interAS**

**Figure 5.3: Routing Table Occupancy of a 4000-node AS-based P2P network.**

A network topology used here is td40w100w. There are 40 ASs, and each AS has 100 nodes. ASs are connected in Waxman method, and nodes within each AS are connected in this way too. (a) shows the occupancy of intraAS routing tables. (b) shows the occupancy of interAS routing tables.

**Table 5.2:**
**Average occupancy in Figure 5.2**

| Row | intraAS | interAS |
|---|---|---|
| 0 | 80.6% | 100% |
| 1 | 28.5% | 100% |
| 2 | 15.2% | 67.2% |
| 3 | 12.8% | 21.5% |
| 4 | 12.6% | 13.4% |
| 5 | 12.5% | 12.7% |

**Table 5.3:**
**Average occupancy in Figure 5.3**

| Row | intraAS | interAS |
|---|---|---|
| 0 | 100% | 100% |
| 1 | 81.0% | 100% |
| 2 | 27.8% | 100% |
| 3 | 14.7% | 66.3% |
| 4 | 12.7% | 22.2% |
| 5 | 12.5% | 13.9% |

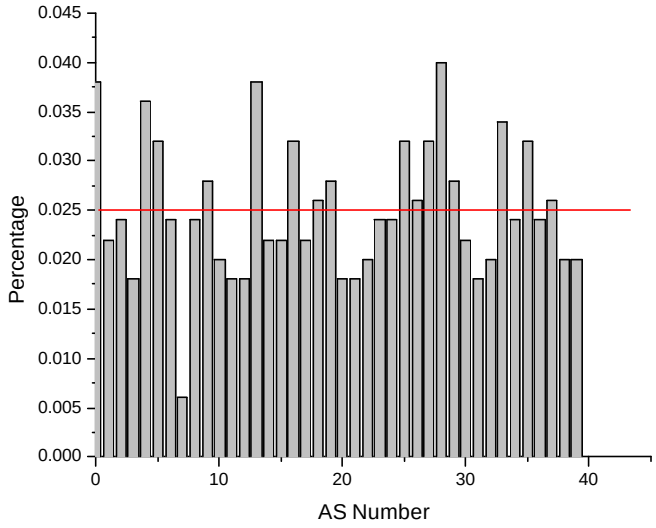



**Figure 5.4: Node distribution in the 500-node p2p system.**

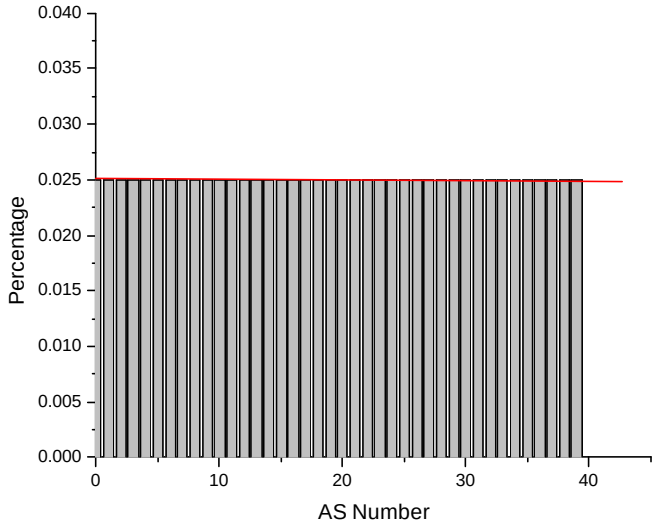



**Figure 5.5: Node distribution in the 4000-node p2p system.** Because the underlying topology contains 40 100-node ASs, each AS must count for 2.5%

*AS* scheme for all four topologies. *Optimal* is latency of direct communication between source and destination, which is represented by the length of the shortest path between source and destination. The *Raw* scheme performs the worst in all four schemes. Its latency is about three times the *Optimal* latency, and increases with the size of P2P systems. The *Latency*, *AS*, and *LatAS* schemes achieve similar lookup latencies, which are about twice the *Optimal* latency. More accurately, the *Latency* scheme works a little better than the *AS* scheme in three topologies: td100w40g, td40w100g, and td40w100w. The *AS* scheme works better than the *Latency* scheme in td100w40w. From the results, we can see that the fine-grained, more-overheaded *Latency* scheme works almost no better than our coarse-grained, less-overheaded *AS* scheme. The hybrid *LatAS* scheme does not gain extra advantages from the combination of a latency-based intraAS routing table and an AS-hop-based interAS routing table. All the three schemes scale well with P2P network size, maintaining a constant average latency as long as new nodes come from the same network scope. As to topologies, td100w40g achieves the largest lookup latency, td40w100g the second largest, td100w40w the third, and td40w100w achieves the smallest lookup latency. This is due to two facts. First, latency between ASs is usually larger than latencies between two nodes that are directly interconnected in the same AS. Therefore, the more ASs a network has, the larger latency two nodes will probably have. Second, topologies with a power law model have larger latency than those with a Waxman model, because in a power law model of BRITE nodes are interconnected to form a tree in which there is only one path between any pair of nodes.

#### 5.3.3.2 Lookup Node Hops Evaluation

Figure 5.6 (b), Figure 5.7 (b), Figure 5.8 (b), and Figure 5.9 (b) show the average number of node hops in a lookup. The X-axis shows the P2P network size. The Y-axis shows the average number of lookup hops. All the schemes perform similarly, with the number of lookup hops increasing from 3 to 4 gradually as network size increases from 500 nodes from 4000 nodes. The number of lookup hops is entirely determined by P2P logical

structure, and has nothing to do with either the way to choose an entry or the physical network topologies. This causes the similarity between all schemes. Ideally, each lookup should go through the number of digits to find out the destination, but because a P2P network size is usually very small compared with the name space, a lookup spends fewer hops in real life.

#### 5.3.3.3 Lookup AS Hops Evaluation

Figure 5.6 (c), Figure 5.7 (c), Figure 5.8 (c), and Figure 5.9 (c) show the average number of AS hops in a lookup. The X-axis shows the P2P network size. The Y-axis shows the average number of lookup hops. The *Optimal* in the figure refers to the length of AS path between source and destination. Our *AS* scheme and the *LatAS* scheme achieve better results than the *Latency* scheme. The average number of AS hops in a lookup of the *AS* scheme and the *LatAS* scheme is less than twice the *Optimal* value. The *Latency* scheme is about 15%-30% worse than the *AS* scheme. The average number of AS hops for networks formed from td100w40g and td100w40w is about 6, and that of td40w100g and td40w100g is about 4, which is not surprising because the first two topologies contain more ASs than the other two.

In a nutshell, the *AS* scheme achieves similar performances in lookup latency and lookup hops to the *Latency* scheme, but better performance on the number of AS hops. Therefore, we conclude that AS regions can provide coarse-grained information to improve P2P regions with less overhead and network traffic without performance degradation.

### 5.3.4 Caching Evaluation

Our caching scheme is aimed to reduce the number of cached nodes while maintaining comparable lookup performance. In this scheme, only the lookup results, i.e., IP addresses, are cached. We term it *IP caching.* The real data is not cached. In this simulation, each node is assigned a 1000-entry buffer. Each entry is a (*ID range*, *IP*) pair. *ID range* is the range of

ID covered in a node, and *IP* is the IP address of the node. A cache of 1000 entries requires about 12K bytes, which is a small storage. Figure 510 shows how different *IP caching* schemes work with caching as the number of queries increases. The underlying topology is td100w40g. There are four schemes. The *Raw* scheme here is the *Raw* scheme mentioned in the last section plus caching on all nodes in a lookup path. The *Latency1* scheme here is the *Latency* scheme mentioned in the last section plus caching on all nodes in a lookup path, i.e. the classic caching scheme. The *AS* scheme here is the *AS* scheme mentioned in the last section plus caching only once in each AS in a lookup path. The *Latency2* scheme here is the *Latency* scheme mentioned in the last section plus caching on the same percentage of nodes in a lookup path as that in the *AS* scheme. The *LatAS* scheme here is the *LatAS* scheme mentioned in the last section plus caching only once in each AS in a lookup path. The performance evaluation is done by calculating the average value of each metrics every 1000 queries.

Figure 510 (a) shows lookup latencies under different *IP caching* schemes. The X-axis shows the number of queries executed. Each query is randomly generated. The Y-axis shows the average lookup latency decreasing with the number of queries. *Optimal* is latency of direct communication between source and destination without caching. From the figure we can see that the *Raw* scheme benefits most from caching. After 16,000 queries, latency in the *Raw* scheme decreases from about 100 seconds at the beginning to about 27.8 seconds, almost the same as the *Optimal* latency, and then enters a stable state in which latency does not decrease greatly. Meanwhile, latency in all the other schemes decreases much less quickly. The *Latency1* scheme, the *AS* scheme, and the *LatAS* scheme achieve similar latencies to that of the *Raw* scheme after about 70,000 queries, and continue to decrease, while latency of the *Raw* scheme remains stable. The *Latency2* scheme performs the worst. Its latency is always about 20% more than that of the *Latency1*, *AS* and *LatAS* schemes. After 100,000 queries, the *Latency1, AS*, and *LatAS* schemes achieve the smallest latencies, and that of the *Raw* scheme is a little larger than theirs, but

still smaller than the *Optimal* latency. The *Latency2* scheme shows the largest latency. Compared with latency without caching, the *Latency1*, *AS*, and *LatAS* schemes have their latencies decreased by more than 50%, the *Latency2* scheme by 40%, and the *Raw* scheme by more than 70%.

Figure 5.10 (b) shows how the number of lookup hops decreases with caching. The X-axis shows the number of queries executed, and the Y-axis shows how the average number of lookup hops decreases with the number of queries. A significant phenomenon is that the number of lookup hops in the *Raw* scheme decreases from 3.75 to about 1 after 16,000 queries. After 16,000 queries, the number of hops of the *Raw* scheme remains almost constant at 1 hop, which means that each node finds the IP address of a lookup ID by asking only one other node. On the other hand, the number of lookup hops for all the other schemes decreases slowly. After 16,000 queries, the number of hops of the *AS*, *Latency1*, and *LatAS* schemes decreases only from 3.75 to about 3.1, and that of *Latency2* scheme from 3.75 to 3.4. Even after 100,000 queries, the number of hops of the first three schemes is still 2.3, decreased by 39%, and that for the *Latency2* scheme is about 2.6, decreased by 30% only.

Figure 5.10 (c) shows how the number of AS hops in a lookup decreases with *IP caching*. The X-axis shows the number of queries executed, and the Y-axis shows how the average number of AS hops in a lookup decreases with the number of queries. Similar to Figure 5.10 (b), the number of AS hops of the *Raw* scheme drops dramatically within the first 16,000 queries, reducing from 12 AS hops to 3.2 AS hops. After almost the same number of queries, the AS hops for the *AS* scheme and the *LatAS* scheme drop from 5.6 to about 3.2 hops too, but that for the *Latency1* scheme drops from 6.7 to 4.5 only and that for the *Latency2* scheme drops from 6.7 to 5.4 only. After 16,000 queries, the *Raw* value begins to decrease very slowly, almost constantly, a little below the *Optimal* value. The values of the *AS* and *LatAS* schemes continue to decrease quickly, although not as fast as before. The *Latency1* scheme and the *Latency2* schemes behave similarly to the *AS* scheme and the

*LatAS* scheme. After 45,000 queries, *Latency1* attains 3.2 AS hops, similar to the *Optimal* value, while the *Latency2* value is about 4.3. After 100,000 queries, the *AS* scheme attains 1.5 AS hops, the *LatAS* scheme 1.6 AS hops, the *Latency1* scheme 2.4, the *Raw* scheme 3.1, and the *Latency2* scheme 3.4, while the *Optimal* value is 3.2.

Finally, Figure 5.10 (d) shows the percentage of nodes that have IP addresses cached in them. The *Raw* scheme and the *Latency1* cache results on all nodes in a lookup path, so their caching percentage is constantly 1. From this figure, we can see that at the beginning, the *AS* scheme and the *LatAS* scheme have about 65% (the *Latency2* scheme is the same since we force it to have the same percentage of caching nodes). The percentage decreases smoothly with the number of queries executed. After 100,000 queries, the *AS* scheme goes down to 40% and the *LatAS* scheme goes down to 45%. On average the *AS* scheme caches about 50% of nodes in a lookup path.

From the above results, we draw several conclusions. First, latency, the number of lookup hops, and the number of lookup AS hops decrease the fastest in the *Raw* scheme, which has randomly distributed routing table entries. This implies that in a stable P2P system, lookup optimization may not be crucial to the system performance because IP caching can greatly reduce lookup latency. Second, the AS-based scheme attains performance in terms of latency and lookup hops similar to the latency-based scheme, and better performance on AS hops. Third, the AS-based scheme requires only half the storage of latency-based scheme. If the latency-based scheme caches lookup results on the same percentage of nodes as AS-based scheme, its performance is worse (in our simulation, 20% worse).

### 5.3.5 Replication Evaluation

A main advantage of the *AS* scheme is that it provides topology information for applications to make decisions. Our static scheme, as described in Chapter 4, distributes data replicas evenly in ASs. This scheme does not reduce lookup latency because it can not automatically route a lookup query to the best replica. Instead, a lookup procedure will

return a list of replicas to the source, and it is the source who decides which replica to access based on its own preferred criterion. We compare lookup latency between an AS scheme without replication (the *Raw* scheme in this figure), an *AS* scheme with replication, a *LatAS* scheme with replication, and a *Latency* scheme with replication. Figure 5.11 shows data access latency of the four schemes. In Figure 5.11 (a), lookup latency is reduced by about 32% in all the three schemes, from about 28 seconds to about 17 seconds. Figures (b), (c) and (d) are similar to (a). With 4 replicas, lookup latency is reduced by 39% in *td100w40g*, 41% in *td100w40w*, 41% in *td40w100g*, 40% in *td40w100w*, so the effect of replication is almost the same on different topologies. However, our efforts to achieve explicit control over replica placement is counteracted by our AS-hops-based node selection for interAS routing tables, because data are mostly replicated in nearby ASs in this way, instead of distributing evenly in the underlying networks.

An improvement is to have each node maintain a node set which contains nodes in ASs with different numbers of AS hops, or just to maximize the number of different ASNs. Then we can use these nodes to replicate data. However, this improvement incurs additional maintenance cost.

## 5.4 Summary

From the simulation, we find that AS regions can improve P2P performance in several aspects. The coarse-grained information, including ASNs and AS hops, can help a P2P region to attain performance similar to schemes with fine-grained metrics. There are several advantages of using coarse-grained information. First, such information is stable and static, thus can be collected once and used many times later. Second, such information collection can be done in an aggregate way, thus reducing network traffic.

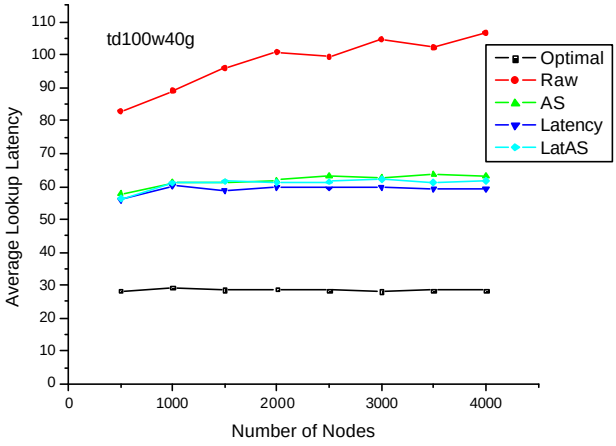
In our simulation, node leaving and failures are not considered yet. Node leaving and failures worsen the performance. In the lookup, node failures lead to a longer lookup path. In the caching, node failures lead to outdated caching content, which requires

modifications of lookup protocols to deal with such cases. In any case, node failures incur additional maintenance cost. However, node failures will not fundamentally change our results.

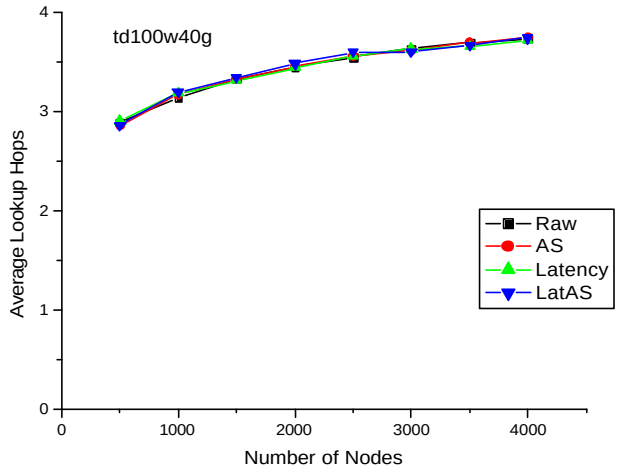
In the caching simulation, we only consider address caching, instead of data caching. Both address caching and data caching have a outdated problem in case of node failures and leaving, and data caching has a consistency problem if data are writeable. A solution is to compare version numbers between the cached data and the data in the original node. AS regions can reduce the overhead by maintaining only one data copy in each AS.

The current replication scheme is static. A more sophisticated one is to consider the data query frequency, and convert cached data into permanent replicas when the query frequency of a cached copy reaches a predefined threshold.

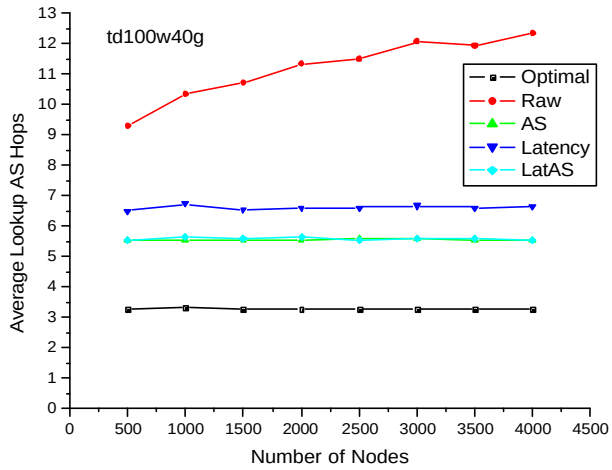
In a nutshell, there are still several aspects that we have not considered in the current evaluation. We plan to explore those schemes in the future.




(a) Average Lookup Latency

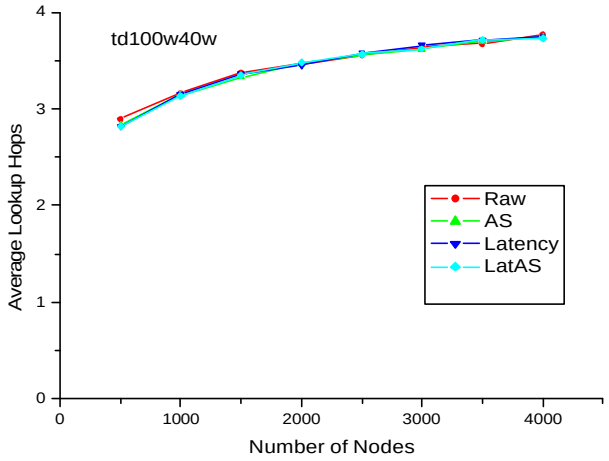



(b)Average Lookup Node Hops

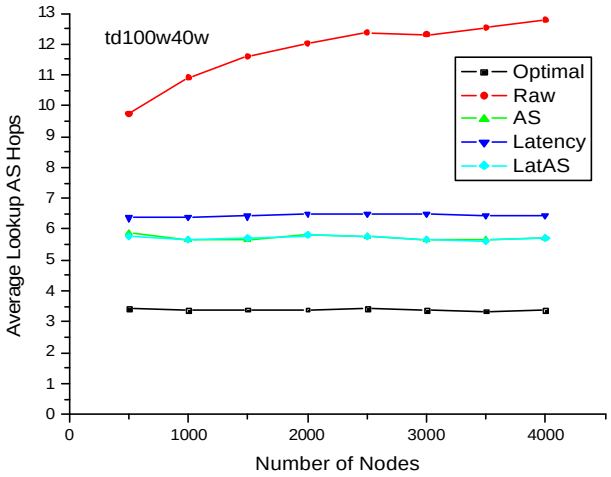



(c) Average Lookup AS Hops

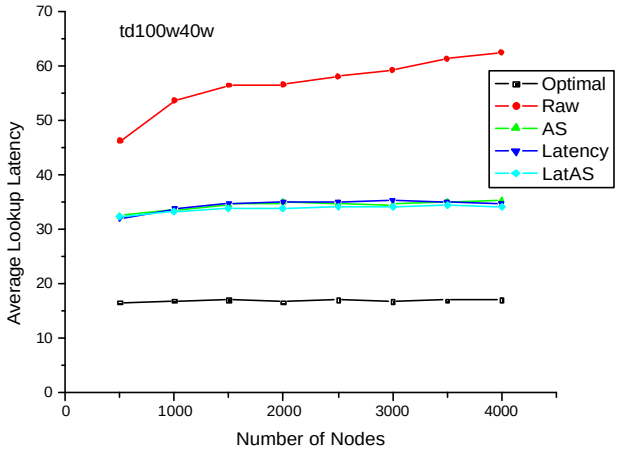
**Figure 5.6: Average Lookup Performance on Topology td100w40g**




(a) Average Lookup Latency

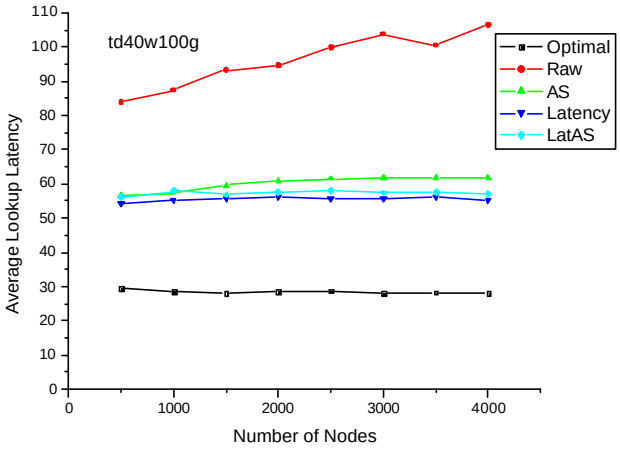



(b)Average Lookup Node Hops

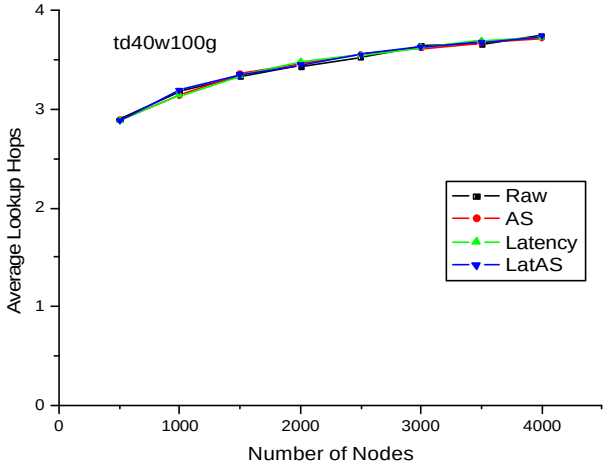



(c) Average Lookup AS Hops

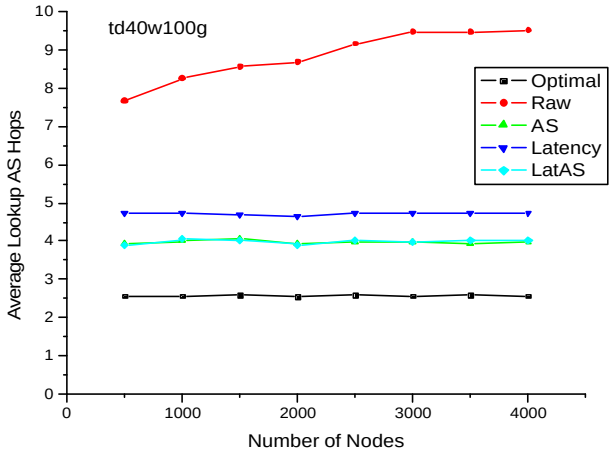
**Figure 5.7: Average Lookup Performance on Topology td100w40w**




(a) Average Lookup Latency

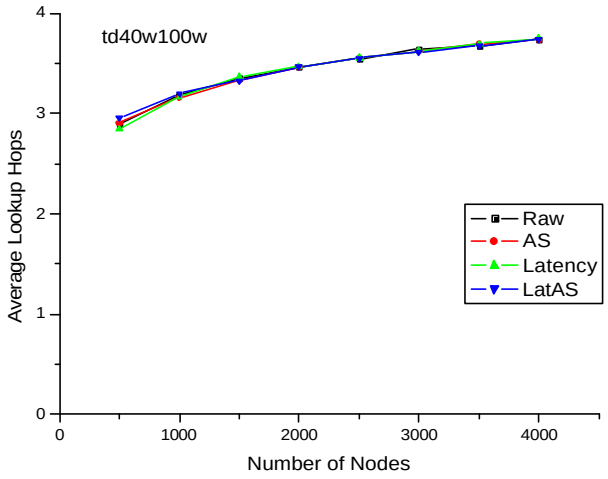



(b)Average Lookup Node Hops

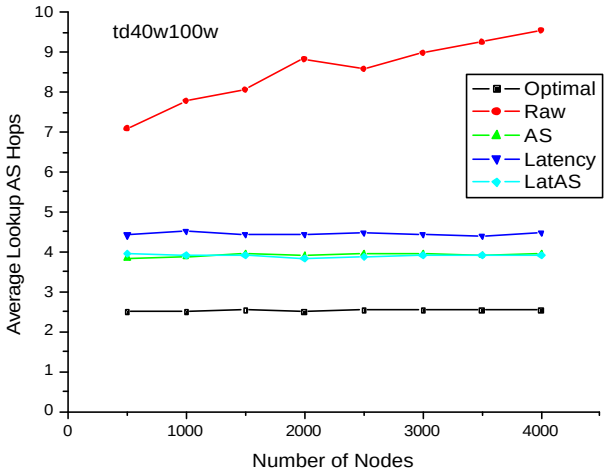



(c) Average Lookup AS Hops

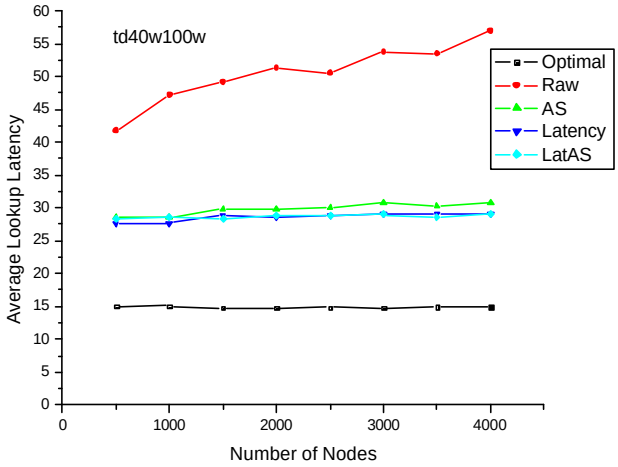
**Figure 5.8: Average Lookup Performance on Topology td40w100g**




(a) Average Lookup Latency

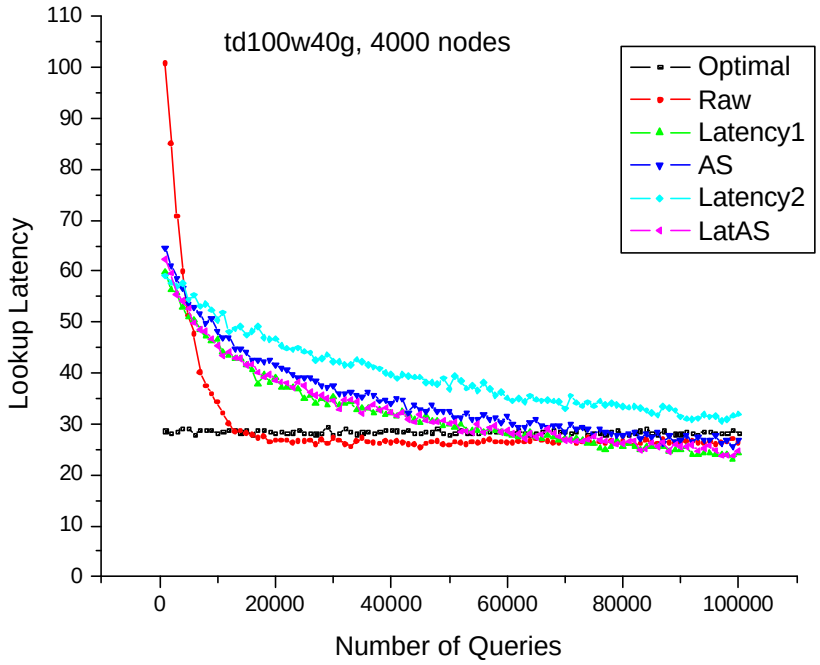



(b)Average Lookup Hops

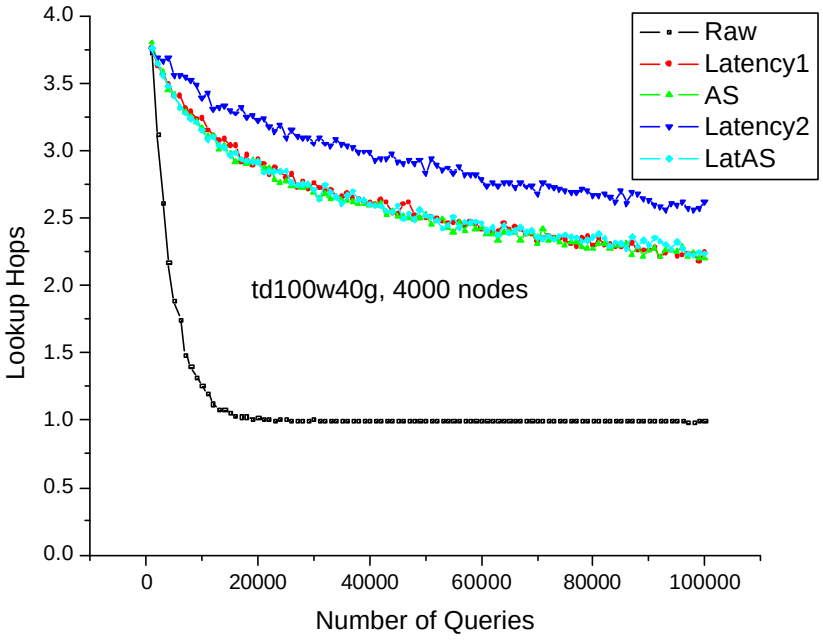



(c) Average Lookup AS Hops

**Figure 5.9: Average Lookup Performance on Topology td40w100w**

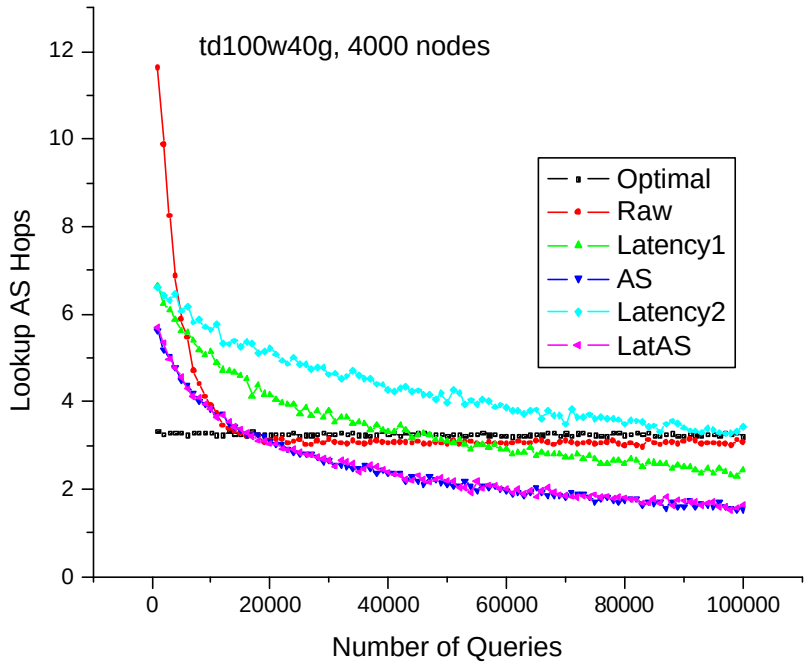



(a) Lookup Latency

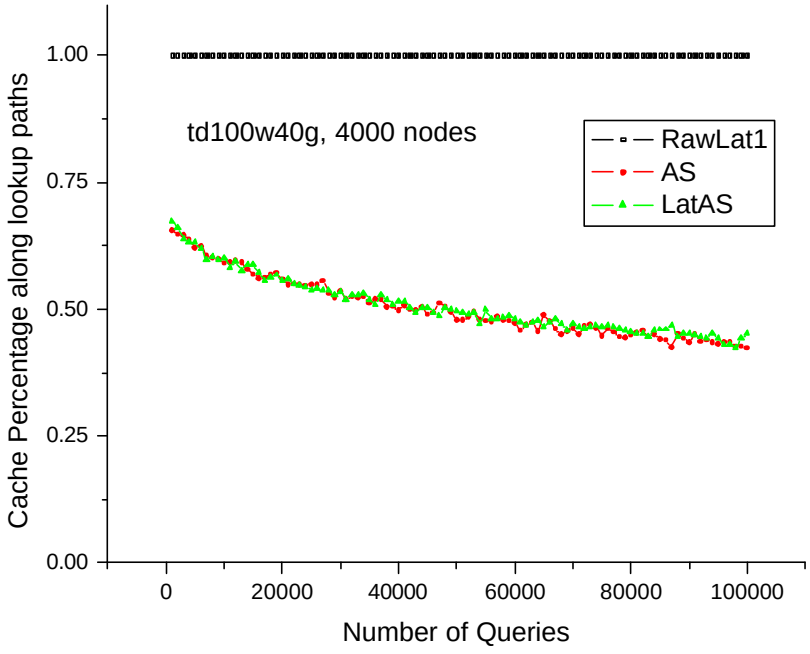



(b) Lookup Hops

**Figure 5.10: Caching performance**

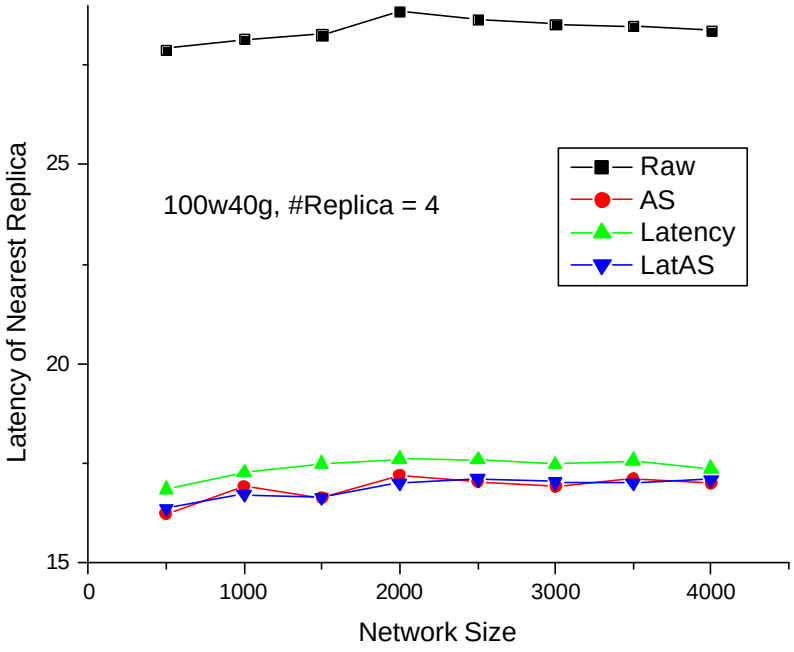



(c) Lookup AS Hops

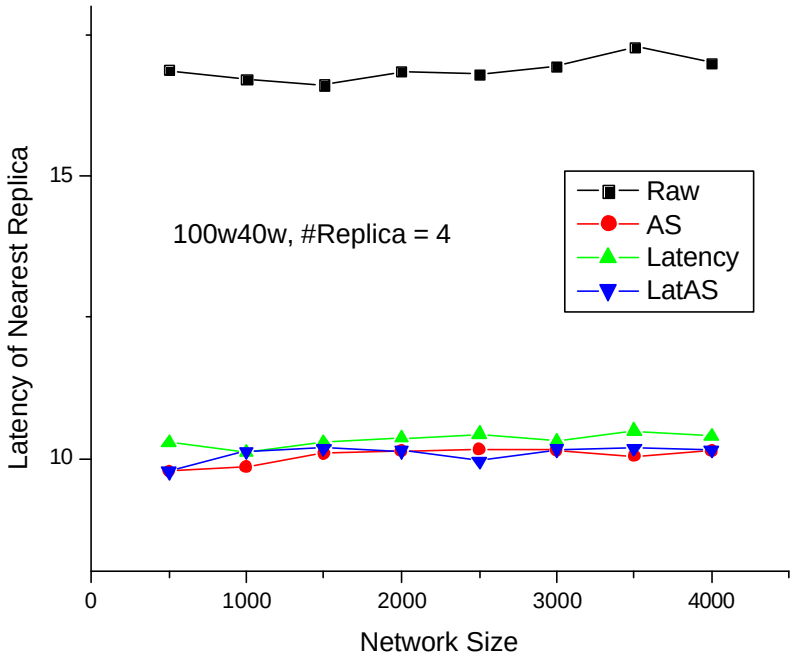



(d) Caching Percentage

**Figure 5.10: Caching performance**

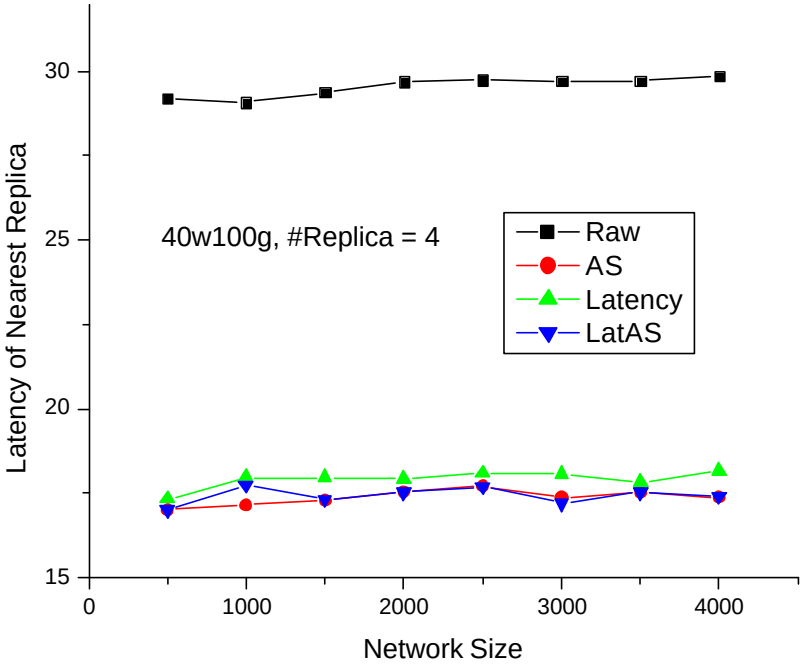



(a) td100w40g

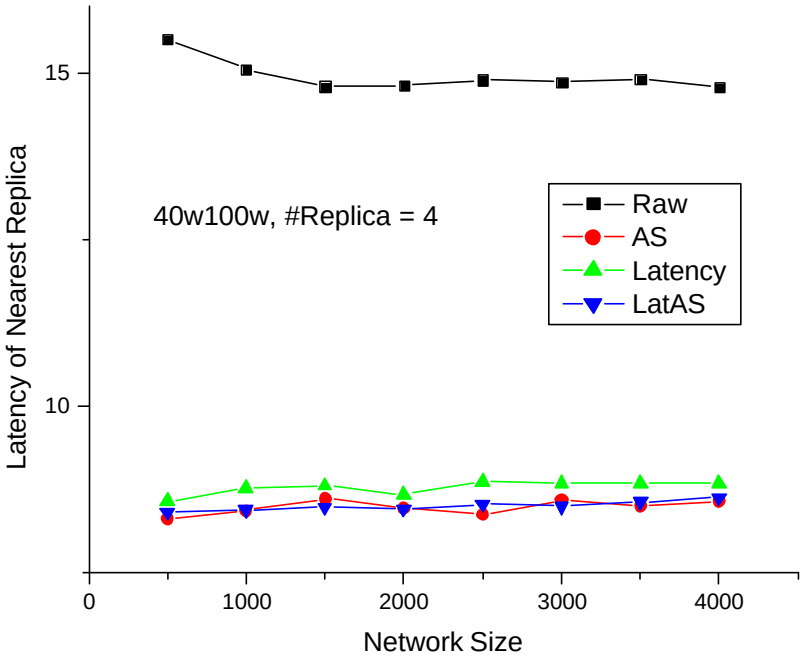



(b) td100w40w

**Figure 5.11: Replication performance with 4 replicas.**




(c) td40w100g




(d) td40w100w

**Figure 5.11: Replication performance with 4 replicas.**

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this thesis, we have shown that applying the region concept can improve application-level services with less overhead than doing the optimizations at the application level as in latency-based schemes. We choose two instances of existing entities, peer-to-peer networks and Autonomous Systems, as distinct sorts of regions. Using information about Autonomous System regions, we design several schemes to improve peer-to-peer performance in the lookup, caching, and replication. By applying these ideas to the application-level network services, we not only explore the utility of having regions but also further refine the definition of regions.

In terms of the Region Project more broadly, we learn several things from this work. First, we need new capabilities so that the upper-layer regions can easily learn about invariants of the underlying regions. The current Internet is largely shaped by commercial interests and Autonomous Systems information is not directly or easily accessible to researchers. Therefore, we realize that two features are crucial to regions. One the one hand, a region should be open and extensible to entities in itself so that not only current applications can

retrieve necessary information from the underlying regions, but also new applications can easily retrieve new information. On the other hand, cooperation between regions should be simple and scalable so that regions can provide global, stable, and coarse-grained information of the underlying regions to the upper-layer regions with less overhead.

Second, we need a good paradigm for boundary crossing. In our work, we use boundary crossing in the caching scheme directly and in the lookup and replication schemes indirectly. In all these schemes, boundary crossing is detected by comparing two ASNs in the application layer. Such method is not flexible and restricted in functionality, and not well integrated with the underlying AS regions. We believe that a good boundary crossing behavior should be simple, efficient, non-intrusive but accessible.

## 6.2 Future Work

Although our simulation has shown good performance, the real internet topology is much more complicated than our generated topologies. To evaluate our schemes more accurately and comprehensively, we can either do the simulation on topologies generated from real Internet data, or deploy a real system on the Internet. There are several issues related to the second method. One is that we need to modify the Internet infrastructure so that Autonomous Systems information is accessible to application-level services. The other issue is that although there are network testbeds like PlanetLab and Emulab, would they improve the quality of the evaluation results? One the one hand, the requirement of a large number of nodes in a peer-to-peer system is hard to satisfy in a testbed. On the other hand, it is still questionable whether such a testbed is a really representative topology.

We hope that we can answer the above questions in the future and further advance the research in the Region Project.

# Bibliography

[1] Karen Sollins. Regions: A new architectural capability in networking. White paper, August 2001.

[2] P. V. Mockapetris. Domain Names – concepts and facilities, RFC 1034, Internet Engineering Task Force, November, 1987.

[3] P. V. Mockapetris. Domain Names – implementation and specification, RFC 1035, Internet Engineering Task Force, November, 1987.

[4] Object Management Group. The Common Object Request Broker: Architecture and Specification, Rev. 2.4.2, Doc. Num. 01-02-33, February, 2001.

[5] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H. and Lilley, J., The design and implementation of an intentional naming system, 17th ACM Symposion on Operating Systems Principles (SOSP '99), Operating Systems Review, 34(5), December, 1999, pp. 186-201.

[6] Y. Rekhter, T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1654, Internet Engineering Task Force, July. 1994.

[7] Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475, IETF, December 1998.

[8] Microsoft Corp., Universal Plug and Play Device Architecture, Version 1.0, June, 2000. Available as http://www.upnp/org/download/UPnPDA10_20000613.htm.

[9] Sun Microsystems, Jini ™ Technology Core Platform Specification, v. 1.1, Sun Microsystems, October, 2000. Available through http://www.sun.com/jini/specs/.

[10] Sun Microsystems, Rio Architecture Overview, White paper from Sun Microsystems, March, 2001. Available as http://www.sun.com/jini/whitepapers/rio_architecture_overview.pdf.

[11] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. Appears in the Proceedings of ACM HotNets-I Workshop, Princeton, New Jersey, USA, October 2002.

[12] http://www.caida.org/

[13] L. Gao. On Inferring Automonous System Relationships in the Internet. IEEE Global Internet, Nov 2000.

[14] Emulab. http://www.emulab.net/

[15] L. Subrmanian, S. Agarwal, J. Rexford, and R. H. Katz, Characterizing the Internet Hierarchy from Multiple Vantage Points, To appear in INFOCOM 2002, June 2002.

[16] SETI@HOME 2001. setiathome.ssl.berkeley.edu

[17] DOOM. http://games.activision.com/games/doom/

[18] JXTA. http://www.jxta.org.

[19] Napster. http://www.napster.com.

[20] Gnutella website. http://gnutella.wego.com.

[21] Ian Clarke, Oskar Sandberg, Brandon Wiley, Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Proceedings of the Workshop on Design Issues in Anonymity and Unobservability (Berkeley, California, June 2000). http://freenet.sourceforge.net.

[22] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In Proceedings of SIGCOMM'2001.

[24] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 2001. 5.

[25] A. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. 18th IFIP/ACM Conference on Distributed Systems Platforms, Heidelberg (D), Nov. 2001.

[26] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02), March 7-8, 2002, MIT.

[27] Dalia Malkhi, Moni Naor and David Ratajczak, Viceroy: A Scalable and Dynamic Emulation of the Butterfly, Proc. PODC, 2002.

[28] Zhichen Xu and Zheng Zhang. Building Low-maintenance Expressways for peer-to-peer Systems. HP Lab Palo Alto, Mar. 2002.

[29] Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In Proceedings of Infocom 2002, New York.

[30] Ben Y. Zhao, Yitao Duan, Ling Huangl. Brocade: Landmark Routing on Overlay Networks. . In First International Workshop on Peer-to-Peer Systems (IPTPS) 2002, Cambridge, MA.

[31] F. Dabek. A Cooperative File System. Master's thesis, Massachusetts Institute of

Technology, September 2001.

[32] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-toPeer Storage Utility. SOSP, Oct 2001.

[33] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In ICS'02, New York, USA, June 2002.

[34] J. Kangasharju, J. Roberts, and K. Ross. Object Replication Strategies in Content Distribution Networks. In Proceedings of Web Caching and Content Distribution Workshop (WCW'01), Boston, MA, June 2001.

[35] http://www.icann.org/

[36] http://www.iana.org/

[37] http://www.planet-lab.org/

[38] K. Obraczka and F. Silvia. Network Latency Metrics for Server Proximity. In Proceedings of the IEEE Globecom, November 2000.

[39] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless interdomain routing (CIDR): An address assignment and aggregation strategy. RFC 1519, September 1993.

[40] M. Waldvogel and R. Rinaldi. Efficient Topology-Aware Overlay Network. HorNets 2002.

[41] P. Radoslavov, R. Govindan, and D. Estrin. Topology-Informed Internet Replica Placement. In Proceedings of the Web Caching and Content Distribution Workshop (WCW'01), Boston, MA, June 2001.

[42] Tina Tyan. A Case Study of Server Selection. Master's thesis, Massachusetts Institute of Technology, September 2001.

[43] K.C. Claffy and D. McRobb. Measurement and Visualization of Internet Connectivity and Performance. URL = http://www.caida.org/Tools/Skitter.

[44] BRITE. http://cs-www.bu.edu/brite/

[45] B. Waxman. Routing of Multipoint Connections. IEEE J. Select. Areas Commun., December 1988.

[46] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks, 2002. Technical report MSR-TR-2002-82.