2012

Measuring Effectiveness of Address Schemes for AS-level Graphs

Yinfang Zhuang
University of Kentucky, yzhua3@uky.edu

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained and attached hereto needed written permission statements(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine).

I hereby grant to The University of Kentucky and its agents the non-exclusive license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless a preapproved embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's dissertation including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Yinfang Zhuang, Student

Dr. Kenneth Calvert, Major Professor

Dr. Raphael Finkel, Director of Graduate Studies

Measuring Effectiveness of Address Schemes for AS-level Graphs

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Yinfang Zhuang

Lexington, Kentucky

Director: Dr. Ken Calvert, Professor of Computer Science

Lexington, Kentucky

2012

ABSTRACT OF DISSERTATION

Measuring Effectiveness of Address Schemes for AS-level Graphs

This dissertation presents measures of efficiency and locality for Internet addressing schemes.

Historically speaking, many issues, faced by the Internet, have been solved just in time, to make the Internet just work [Han06]. Consensus, however, has been reached that today's Internet routing and addressing system is facing serious scaling problems: multi-homing which causes finer granularity of routing policies and finer control to realize various traffic engineering requirements, an increased demand for provider-independent prefix allocations which injects unaggregatable prefixes into the Default Free Zone (DFZ) routing table, and ever-increasing Internet user population and mobile edge devices. As a result, the DFZ routing table is again growing at an exponential rate.

Hierarchical, topology-based addressing has long been considered crucial to routing and forwarding scalability. Recently, however, a number of research efforts are considering alternatives to this traditional approach. With the goal of informing such research, we investigated the efficiency of address assignment in the existing (IPv4) Internet. In particular, we ask the question: "how can we measure the locality of an address scheme given an input AS-level graph?"

To do so, we first define a notion of efficiency or locality based on the average number of bit-hops required to advertize all prefixes in the Internet. In order to quantify how far from "optimal" the current Internet is, we assign prefixes to ASes "from scratch" in a manner that preserves observed semantics, using three increasingly strict definitions of equivalence.

Next we propose another metric that in some sense quantifies the "efficiency" of the labeling and is independent of forwarding/routing mechanisms. We validate the effectiveness of the metric by applying it to a series of address schemes with increasing randomness given

an input AS-level graph. After that we apply the metric to the current Internet address scheme across years and compare the results with those of compact routing schemes.

Yinfang Zhuang

Author's Signature
Dec. 2012

Date

Measuring Effectiveness of Address Schemes for AS-level Graphs

By

Yinfang Zhuang

Kenneth L Calvert

Director of Dissertation
Raphael A Finkel

Director of Graduate Studies
Dec. 2012

Date

To my parents, Huaheng Zhuang and Chengcan Zhou. Thank you for all your love and support to me and thanks for always being there no matter what.

ACKNOWLEDGEMENTS

My thanks first go to my supervisor, Dr. Ken Calvert, for his knowledgeableness and preciseness which have inspired me greatly in the networking area. I could not have achieved what I have achieved now without the professional guidence of him. I also gratefully acknowledge the support of the US National Science Foundation under grant numbers 0435272 and 0626918.

I also want to thank the departmental support staff for their help with machine maintenance and upgrade and allowing access to servers to execute my CPU and Memory intensive tasks.

Finally my thanks go to those who are in my laboratory and with whom I have had interesting technical discussions which are very informative and useful.

Table of Contents

List of Tables

List of Figures

1 | Introduction

This dissertation presents measures of efficiency and locality for Internet addressing schemes.

A (computer) network is "a collection of computers and other hardware components interconnected by communication channels that allow sharing of resources and information." [Wik12] An *address* specifies at which part of the network some resource or a set of resources is located; a *route* specifies the information needed to forward a piece of information across the network to the destination address; *routing* specifies how to maintain the connectivity information to the addresses of all the end hosts in the network. The Internet is one such example. Hierarchical addresses (conforming to the underlying topology) are used in the current Internet to both identify end hosts and indicate their locations. We will talk about hierarchical addresses in Section 2.1

Historically speaking, many issues faced by the Internet have been solved just in time to make the Internet just work [Han06]. Consensus, however, has been reached that today's Internet routing and addressing system is facing serious scaling problems: Multihoming, which helps realize protection from single-point failures, load balancing of incoming and outgoing traffic, and other policies; an increased demand for provider-independent prefix allocations that injects unaggregatable prefixes into the Default Free Zone (DFZ) routing table (also referred to as the DFZ RIB); and an ever-increasing population of Internet users as well as mobile edge devices—all drive the growth of the DFZ RIB size at an alarming rate [MZF07]. Some of the Internet Service Providers (ISPs) check this growth by refusing to propagate prefixes with lengths greater than 24, which has an adverse effect upon the reachability of those prefixes and is not widely used [Hus01b]. Having many destinations in a routing system, in combination with multiple paths per destination, increases the demand on routing processing for route selection and route filtering, and requires large memory space on routers. T. Li [MZF07] has pointed out that Moore's Law does not hold for high-end routers in terms of costs. The existence of many routes also increases the chances for the repeated advertizements and withdrawals (or flapping) of a destination to occur. Though route flap suppressing (or damping) [VCG98, MGVK02, LABJ01] has been widely deployed,

the routing convergence becomes a more significant problem than before. The advent of new Internet Protocol version 6 (IPv6) and its larger address space can worsen the current situation further, in the absence of any working scalable routing mechanism.

Y. Rekhter pointed out that addressing should be congruent with the underlying topology so that information aggregation (abstraction) can be effectively performed to achieve the scalability of routing systems. Inspired by this so-called "Rekhter's Law", this thesis explores the notion of *locality*—the idea that addresses in a network have some relation to location in the network topology. The ultimate goal is to come up with a rigorous definition that will allow us to quantify the efficiency of a given network instance (i.e., an assignment of addresses to nodes in a graph). The definition would admit discussions of *optimal* instances, so that, for example, we could quantify how far from optimal the current Internet is.

To achieve this goal we first study locality and optimality in the context of the current Internet. We want to address the following questions: How much locality is there in the assignment of prefixes to routing domains (we will talk about domains in Section 2.2), either equal to or part of administrative domains, in the current Internet? How far from optimal is the current assignment? In other words, how much is hierarchical addressing buying us (at the inter-domain level), and how much could be gained by a from-scratch assignment of prefixes (if it were possible to do so)? Answering these questions requires precise definitions of both locality and optimality of prefix assignment. In the context of a particular routing/forwarding protocol, some of the information aggregation is not allowed by the routing policies of that routing protocol (we will talk about the routing policies of one particular routing protocol and how they prevent certain information aggregation from occurring in Section 2.3). We stress that we are *not* proposing that addresses in the Internet actually be reassigned to improve locality. We are simply interested in the above questions as a way of assessing the importance of topology-based addressing for the scalability of routing systems at the interdomain level, and in particular in whether topology-based addressing is a necessary component of any future internetwork architecture. Here we are mainly focused on unicast addressing, as that is the only kind of addesses for which topology information is usually encoded.

Next we move on to study locality and optimality in a more general sense—that is, independent of any particular routing/forwarding mechanism. To be specific, given a finite undirected graph $G = (V, E)$, together with a labeling function $L : V \to \{g \mid g \in \Sigma^*\}$ where Σ is an alphabet, we want to compute a quantity $Q(G, L)$ that in some sense quantifies *efficiency* (or, conversely, *cost*) of the labeling. For simplicity, we assume $\Sigma = \{0, 1\}$.

Our main contributions can be summarized as follows:

1. We define a precise notion of cost of a prefix assignment (an inverse notion of locality) of the Internet. We focus on the savings due to abstraction in the control plane[1]. In particular, our measure is based on the bits exchanged by routing protocol (BGP) instances to advertize destinations. Using Route Views [UO], Réseaux IP Européens Network Coordination Centre's Routing Information Service (RIPE NCC's RIS) [RIP], and Internet Routing Registries (IRR) [IRR], we measure the cost of the Internet's actual address assignment in terms of our metric. We attempt to quantify how close to "optimal" the Internet operates in this respect. We do this by constructing alternative assignments of prefixes to autonomous systems from scratch, in a manner that preserves the semantics of addressing.

2. We propose another metric that in some sense quantifies the efficiency of the labeling and is independent of forwarding/routing mechanisms. We validate the effectiveness of the metric by applying it to a series of instances that we expect to have decreasing locality (according to our intuitive notion) for a given input interdomain-level graph. After that, we apply the metric to the current Internet address scheme to see how the locality is changing across years. Finally, we compare the locality of the address scheme of the Thorup and Zwick (TZ) stretch-3 compact routing mechanism [TZ01] with that of the Internet, which serves as evidence for the efficiency of the TZ stretch-3 compact routing mechanism.

[1] The control plane is responsible for building a routing information base (RIB) that defines what to do with incoming packets by using a list of destination addresses and the outgoing interface(s) associated with them. The data plane is responsible for building the forwarding information base (FIB) from the original RIB to facilitate destination address lookup and packet forwarding.

2 | Background

In this section, we explain several basic concepts related to our work: Internet Protocol version 4 (IPv4) and its address scheme, and Internet Protocol version 6 (IPv6) and its address scheme. Since our problem is defined at the inter-domain level, we will also explain the concepts of Autonomous System (AS) and Border Gateway Protocol (BGP). The concept of entropy in information theory is also to be reviewed, since our work studies optimal assignments of labels to nodes in a graph.

2.1 Internet Protocol

Internet Protocol version 4 (IPv4) [Pos81] is a connectionless protocol used in interconnected packet-switched computer networks for transmitting blocks of data called datagrams from source hosts to destination hosts. The two basic functions that the Internet protocol implements are addressing and fragmentation. Each host is identified by a fixed-length address (32-bit address in this case). Large blocks of data are fragmented in order to pass through networks that allow for only small packets and are reassembled at the destination. However, the Internet protocol only provides a best-effort communication facility. To be more specific, there are no acknowledgments either end-to-end or hop-by-hop, there is no assurance of in-order deliveries, and there is no avoidance of duplicate deliveries.

IPv4 addresses are 32 bits long. An IPv4 address consists of a network number and a local address of the specified network. For convenience, IPv4 addresses are usually written in a dot-decimal format: each of the four bytes of an IPv4 address is represented in decimal, and these four decimal integers (each in the range of 0 to 255) are separated by periods. Originally, there were three formats or classes of addresses. In class A, the most significant bit (msb) is 0, the next 7 bits represent a network number (or 128 possible networks), and the last 24 bits represent a local address. In class B, the msbs are 10, the next 14 bits represent a network number (or 2^{14} possible networks), and the last 16 bits represent a local address. In class C, the msbs are 110, the next 21 bits represent a network number (or 2^{21} possible networks), and the last 8 bits represent a local address. Due to the exhaustion of Class B

networks, accelerating growth of routing tables as well as the concerns of eventual exhaustion of the 32-bit address space, Classless Inter-Domain Routing (CIDR) was proposed as an interim solution. While formerly the network size was implicit (since the network component can be inferred from the msbs of an IPv4 address), classless IPv4 address blocks or prefixes require explicit notations of prefix lengths (in the form of a bitmask or number carried with the prefix) in order to determine the size of the network component. While formerly there were only three network sizes, prefixes allow us to define any network size that is equal to 2^x with x lying between 0 and 31 inclusively. We will discuss CIDR in greater detail in Section 2.3, after we introduce more concepts. Generally speaking, the splitting (or subnetting) of a prefix is the process of dividing the address space covered by this prefix into disjoint address spaces, the size of each of which is still power-of-two. For example, a prefix p of length l can be splitted into 2 prefixes q_1 and q_2 of length $l+1$. q_1 and q_2 share the same l msbs with p, and differ only in the $(l+1)$th msb from each other. The address space covered by p is exactly the same as the union of the address space covered by q_1 and that covered by q_2. The reverse process is called as aggregation (or supernetting).

In order to cope with the eventual exhaustion of the 32-bit address space, IP version 6 (IPv6) was proposed in [DH98] as a long-term solution. Though our work is mainly focused on IPv4 and its addresses (due to the limited deployment of IPv6 and its addresses), we briefly introduce IPv6 and its address mechanism because of their importance to the future Internet. Here we are mainly focused upon the primary changes from IPv4 to IPv6. IPv6 increases the IP address size from 32 bits to 128 bits and simpler autoconfiguration of addresses. A scope field is added to a multicast address to limit the scope of a multicast group. In addition, anycast address is defined for sending a packet to the nearest node in a group of nodes having that address. Some IPv4 header fields that are rarely used have been made optional to reduce the processing cost of packets by not allowing routers to do fragmentation and by dropping the checksum field. Options are moved to extension headers, which allows for simpler forwarding, greater flexibility in the length of options, and greater extensibility by allowing for new options added in the future. Internet Protocol Security (IPsec) [Ken05] is used to support authentication, integrity, and confidentiality.

As we mentioned before, an IPv6 address is 128 bits long. It is represented as follows: bits are divided into 8 groups; each group is 16 bits long, represented by 4 hexadecimal digits; groups are separated by colons. IPv6 addresses are classified into unicast, multicast and anycast addresses; IPv6 does not have the broadcast addresses like those implemented by IPv4. A unicast IPv6 address usually consists of a 64-bit network prefix and a 64-bit

interface identifier.

The Internet Corporation for Assigned Names and Numbers (ICANN) is responsible for full management of the coordination of the global Internet's systems of unique identifiers, including the IPv4 and IPv6 address spaces, autonomous system numbers, and the top-level domain name space (DNS root zone). The ICANN attempts to ensure the stability and the security of the Internet's daily operations. The IANA is currently a department operated by the ICANN. A RIR is a nonprofit organization that administers the allocation and registration of Internet number resources, for example IP addresses and AS numbers, for a particular geographic region of the world. Currently there are five RIRs:

- American Registry for Internet Numbers (ARIN) for countries mostly in North America;
- Asia-Pacific Network Information Centre (APNIC) for countries in the Asia Pacific region;
- Réseaux IP Européens Network Coordination Centre (RIPE NCC) for countries mostly in Europe;
- African Network Information Centre (AfriNIC) for countries in Africa;
- Latin America and Caribbean Network Information Centre (LACNIC) for countries mostly in South America.

Shifting the responsibility of the global address assignment from one single organization to several organizations—IANA, the RIRs, and the ISPs, the efficiency and response time for new assignments have been greatly improved and the single-point failures have been effectively solved. The current hierarchical allocation is described as follows. The IANA makes allocations from the unallocated pool of addresses to RIRs, as required. These allocations are made in /8 prefixes. The RIRs, in turn, allocate or assign smaller address blocks to Local Internet Registries (LIRs) or ISPs, based on the needs of each ISP or LIR. LIRs or ISPs may make direct use of these prefixes or may split these prefixes and make further allocations of the split prefixes to their customers.

2.2 Autonomous System

An AS is a set of routers that corresponds to a single administrative entity with a single coherent routing policy. Each AS uses (multiple) interior gateway protocol (IGP)(s) and (multiple) metric(s) to determine how to route packets within the AS and an inter-AS

routing protocol to determine how to route packets from and to other ASs; the only inter-AS routing protocol in use today is the Border Gateway Protocol, version 4. Each AS has an autonomous system number (ASN) that uniquely identifies it in the Internet and is used for inter-AS routing. Formerly, an ASN was a 16-bit integer; now Internet Assigned Numbers Authority (IANA) allocates 32-bit ASNs [VC07]. The 32-bit ASN is either simply written as a decimal integer or is divided into the higher 16-bit group and the lower 16-bit group separated by a dot, with each group represented as a decimal integer [HM08].

ASs can be classified into Tier-1, Tier-2 and Tier-3: Tier-3 ASs have a small number of localized customers; Tier-2 ASs have state-wide or region-wide customers; and Tier-1 ASs' routing tables have explicit default-free routes to all reachable Internet destinations (i.e., global coverage). There are 12 Tier-1 ASs in mid-2012 and they are connected with each other, forming almost a clique that we usually call the core of the Internet.

Faloutsos et al., in their seminal paper [FFF99], found that for both router-level graphs and AS-level graphs the distributions of node degrees, the degree ranks of the nodes, and the number of nodes within h hops of each other follow power-law distributions. We reiterate the facts about power-law graphs here [BC06]: "A power-law graph $G = (V, E)$ is an undirected, unweighted graph whose degree distribution approximates a power law, i.e., the number $y = |\{v \in V \mid \deg v = x\}|$ of vertices whose degree is x satisfies

- $$\begin{cases} y = \lfloor c \rfloor - r & \text{when } x = 1 \\ y = \lfloor \frac{c}{x^\gamma} \rfloor & \text{when } x = 2, 3, ..., \lfloor c^{\frac{1}{\gamma}} \rfloor \end{cases}$$
- $r = n - \sum_{x=1}^{\lfloor c^{\frac{1}{\gamma}} \rfloor} \lfloor \frac{c}{x^\gamma} \rfloor$
- c is a value minimizing $|n - r|$

for some constant $\gamma \in R^+$ called the power-law parameter of G."

The average hop distance between ASs (or the "diameter" of the Internet) stays relatively constant over time and even decreases in recent years due to the increase in the density of the interconnections [LKF07]. The average hop distance between any AS pair is observed to lie between 3 and 4, with most of the AS pairs 2 to 4 hops away from each other [MKF+06b, DD11]. This makes an AS-level graph a small-world graph. A small-world graph is typically defined to be a graph in which the hop distance between any two nodes grows logarithmically with the total number of nodes. Its defining property is a high clustering coefficient: for a given median hop distance, a small-world graph has a much larger percentage of 3-cycles among all connected node triples than a random graph [WS98].

2.3 Border Gateway Protocol

The contractual commercial agreements between ASs can be classified into three categories: in a provider-to-customer relationship, a customer AS pays a provider AS for sending/receiving traffic from/to the rest of the Internet on its behalf; in a peer-to-peer relationship, both ASs find it mutually beneficial to exchange the traffic of their customers; in a sibling-to-sibling relationship, both ASs normally belong to the same organization and each one exports all of its routes to the other.

Figure 2.1 shows a simplified description of the practical Internet with various types of ISPs interconnected via different business relationships among them. "($$)" denotes relationships involving financial settlements.

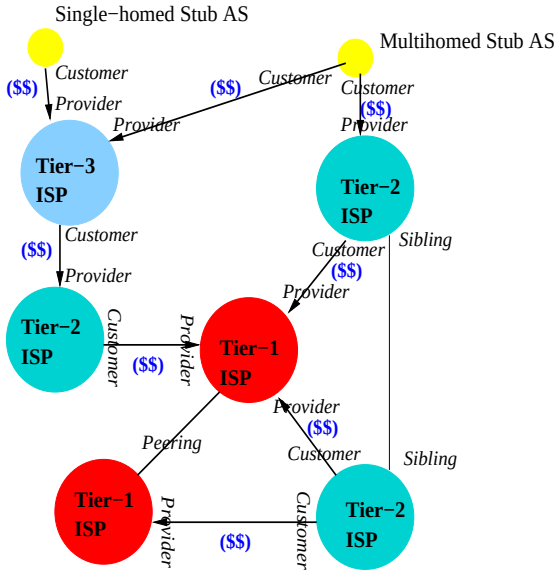


Figure 2.1: simplified picture of the wide-area Internet

ASs can also be classified into stub AS and transit AS. A stub AS is an AS that only carries traffic that either originates or terminates within that AS. A transit AS is an AS that carries traffic that does not originate or terminate within it.

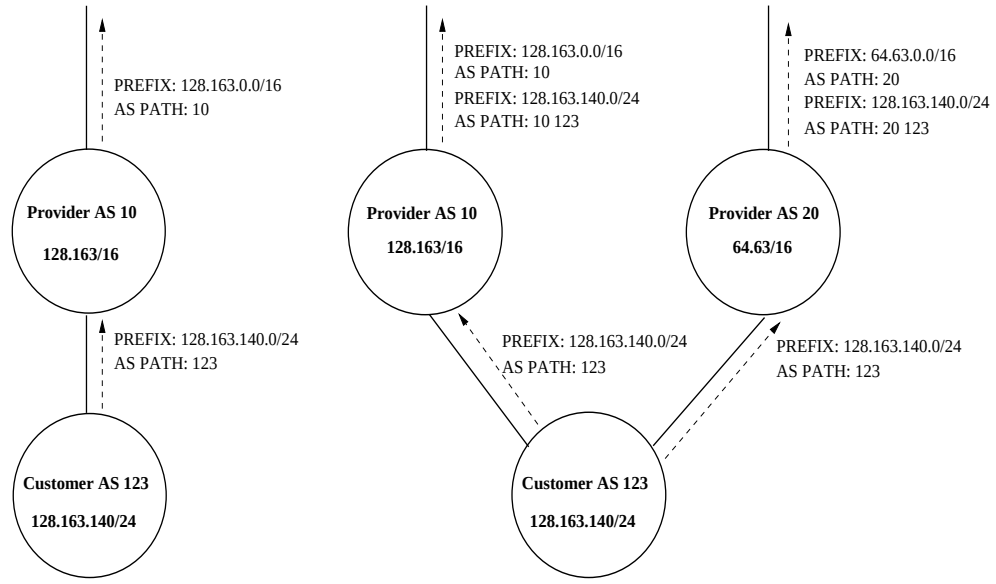
A Provider Aggregatable (PA) prefix is assigned by a transit provider, while a Provider Independent (PI) prefix is assigned directly by a Regional Internet Registry (RIR) instead. CIDR proposes the adoption of PA prefixes in order to make possible the aggregation of routing information along topological lines mainly defined by provider-to-customer relationships. A provider AS splits one of its prefixes and assigns one or more split prefixes to each of its customer ASs. If we assume that each customer AS is single-homed to this provider AS, no explicit route is needed for each customer AS; this provider AS only advertizes p.

This advertizement provides reachability and routeability for the split prefixes assigned to the customer ASs. As shown in Figure 2.2(a), AS123 extends 128.163.0.0/16 of its single provider—AS10—into 128.163.140.0/24, and, as a result, AS10 advertizes 128.163.0.0/16 without advertising any extensions of this address block. On the other hand, because a multihomed AS must be advertized into the Internet by each of its service providers, it is often infeasible to aggregate any of its address blocks into the address space of any one of those providers. Due to the deployment of CIDR-style addressing, forwarding in the Internet is done using the longest prefix match rule: each entry in a routing table corresponds to a prefix; one destination address may share some number of msbs with (or match) more than one routing table entry; the table entry, with the largest number of msbs matching those of the destination address, is called the longest prefix match; and it is the information contained in this longest prefix match that is used for forwarding.

A multihomed AS is an AS that has more than one transit provider. The driving forces for an AS to be multihomed include protection from single-point failures, load sharing across multiple transit providers, avoiding problematic paths, and routing traffic of a particular type to a particular ISP [ALD+05]. Figure 2.2(b) shows one example to illustrate why multihoming makes the number of routing entries in the routing tables closer to the number of stub ASs than to the number of transit ASs, making CIDR lose its original ability to keep the global routing table growth at a sustainable rate. As shown in Figure 2.2(b), though AS123 extends 128.163.0.0/16 of one of its two providers—AS10—into 128.163.140.0/24, AS10 advertizes 128.163.140.0/24 as well as 128.163.0.0/16. If AS10 were to advertize only 128.163.0.0/16 with AS20 advertising 128.163.140.0/24, then all the traffic destined to any address contained in 128.163.140.0/24 would traverse AS20, and none would traverse AS10, unless AS20 becomes unreachable.

Export policies of an AS are important since no ISP wants to transmit traffic that does not in some way generate revenue. An ISP usually provides full transit of incoming and outgoing packets for its own customers, and provides some transit for the customers of the other party involved in a peering relationship with it. When a router receives more than one route to a destination network, it prefers the route advertized by customers over the route advertized by peers, which are preferred over the route advertized by providers. This is because an AS tries to avoid paying its providers for transmitting traffic if possible.

Now that we have introduced the concept of AS in Section 2.2, we can introduce the most important inter-AS routing protocl—BGP [RLH06]. The current version is 4. A router that implements BGP is called a BGP speaker. A BGP speaker in a different AS is



(a) single-homed AS assigned a PA prefix

(b) multi-homed AS assigned a PA prefix

Figure 2.2: Multihoming and PI addressing prevent address aggregation

called an external peer while a BGP speaker in the local AS is called an internal peer. The main function of a BGP speaker is to exchange routes, each of which consists of the set of destinations and the path used to reach them, with other BGP speakers. Each AS chooses a single path to get to each destination prefix, even though it may learn many possible paths. The choice is based on policy considerations, which is based on business relationships that we just talked about at the beginning of this section. When a routing table changes, it only requires incremental updates rather than periodic updates.

Unlike interior gateway protocols (IGPs), BGP does not optimize metrics, such as the number of hops of paths; rather, it provides reachability information and implements various forms of routing policies. Therefore, the route announcement, due to either a route change or the appearance of a new route, does not contain metrics, such as the number of hops of paths or delay; it contains one or more BGP attributes. Here we enumerate the key BGP attributes: NEXT_HOP, ASPATH, and MULTI_EXIT_DISC. For each announced prefix the NEXT_HOP attribute indicates the address of the router in one neighboring AS the packet destined to any address contained in this prefix is forwarded to; the ASPATH attribute indicates the vector of ASs the packet destined to any address contained in this prefix traverses; the MULTI_EXIT_DISC is used to discriminate among multiple exit points to the same neighboring AS. Constrained by the routing policies based upon the contractual

commercial agreements between ASs, a BGP route used in forwarding packets by a BGP speaker may not be the shortest length. Since BGP only supports destination-based forwarding mechanisms, it can support only those policies that are consistent with this forwarding mechanism. From the BGP route announcements, we can construct an AS-level graph in the following way. An edge exists between ASi and ASj if and only if ASi advertizes some prefix to ASj or vice versa. The degree of an AS is defined to be the number of ASs adjacent to it in an AS-level graph.

BGP also supports CIDR [FL06] by aggregating a set of destination addresses into a single IP prefix and by aggregating a set of AS paths using unordered AS sets. The aggregation of a set of AS paths is performed as follows. Given two AS paths—123 and 124, the aggregate of these paths can be written as 12[34]; [34] is called an AS set.

The BGP that is used within an AS is called internal BGP (iBGP), and the BGP that is used to distribute information among ASs is called external BGP (eBGP). Formerly, to avoid routing loops, any externally learned route by one BGP speaker was redistributed over iBGP sessions to all other BGP speakers within the same AS, which then did not readvertize this route. This means all these iBGP speakers must be fully meshed, which does not scale well when the number of iBGP speakers is large. Therefore authors in [BCC06] proposed the use of route reflectors (RRs): in its simplest form one iBGP speaker of an AS is chosen to be a RR while all other iBGP speakers within this AS set up iBGP sessions with this RR. This RR advertizes BGP update messages received from one iBGP speaker to all the other iBGP speakers. Since this RR helps forward BGP update messages among iBGP speakers, iBGP speakers are no longer required to be directly connected to form a full mesh. Other benefits [POA+12] brought by the usage of RRs are: reduced operational costs (adding or removing a router only requires the reconfigurations of RRs this router connects to), reduced sizes of routers' RIB-in tables that contain unprocessed routing information received from routers' BGP neighbors, the reduced number of BGP updates, and the coexistence of RRs and conventional routers that do not understand RRs. Note that iBGP is not an IGP, such as Open Shortest Path First (OSPF), Routing Information Protocol (RIP), and Intermediate System To Intermediate System (IS-IS), since iBGP cannot be used to set up the routing states necessary to forward packets correctly among internal routers within an AS. Rather, iBGP is used when information about externally learned routes is exchanged among BGP routers in an AS, and this kind of information is routed among BGP routers via some IGP used by this AS. Figure 2.3 illustrates the relationship between eBGP and iBGP; eBGP sessions are set up between two routers that are usually directly connected while iBGP

sessions are set up between two routers that are usually not directly connected.

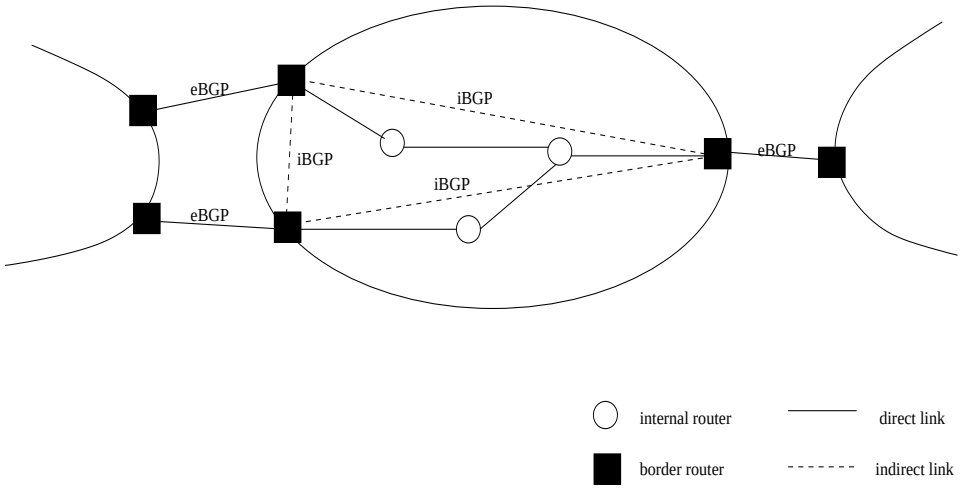


Figure 2.3: eBGP and iBGP

The drawbacks of BGP are slow convergence due to not propagating policy information, error-proneness due to the requirement of explicit configuration, and insecurity due to the lack of the security enhancements.

There are several projects that collect real-time BGP routing information about the Internet from different locations around the world and make public these Internet routing data. One of such projects is the RIS project done by RIPE NCC [RIP]. Currently it collects and stores Internet routing data from more than 542 peers around the globe. Three of the Remote Route Collectors (RRC)[1] do not expose their BGP peering lists. The Route Views project [UO] of the University of Oregon is also intended to obtain and store Internet routing data from the perspectives of 350 peers. In order to significantly facilitates the study and analysis of the BGP routing protocol, the authors in [BKL11] proposed Multi-Threaded Routing Toolkit (MRT) as a standardized data representation to encapsulate, export, and archive routing protocol behaviors and RIB snapshots. Both the Route Views Project and the RIPE NCC's RIS archive their collected BGP feeds in MRT format. ISPs record some or all of their policies in Internet Routing Registries (IRRs) that aim to improve the consistency and integrity of routing policies among different ISPs. Routing Policy Specification Language (RPSL) is used by an ISP to publish routing policies for IPv4 unicast [AVG+99, MSO+99], IPv6 unicast, and multicast address families [BDPR05], which facilitates the analysis of the policy information from each ISP.

[1] rrc02.ripe.net, rrc08.ripe.net and rrc09.ripe.net

2.4 Entropy

Entropy (also called Shannon Entropy) of a discrete random variable X (denoted by $H(X)$), with possible values in $\{x_1, ..., x_n\}$ and probability mass function $p(X)$—$p(x_i) = p_i$, is defined as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

where b is the base of the logarithm. In particular, the unit of entropy is bit if $b = 2$. Generally speaking, an entropy is used to measure the uncertainty of a random variable or the average information content conveyed by the value of a random variable. The $H(X)$ reaches the maximum when the probability mass function $p(X)$ is a uniform distribution. Shannon's source coding theorem tells us that no lossless compression scheme can compress a message, treated as a sequence of independent random variables with identical distributions (or i.i.d. random variables), in a way such that there is more than one bit of information contained in each bit of message. The total amount of information contained in a message is then equal to the entropy of the message times the length of the message.

The two most famous entropy encoding algorithms include Huffman coding and Algorithm coding. The main idea common to these two encoding algorithms is that more frequently used symbols (or values of a random variable) are encoded with fewer bits, through which fewer bits are used in total (close to those required by the corresponding entropy). The main difference between these two lies in that Huffman coding replaces each symbol of a message with a code while Algorithm coding encodes the whole message into a number in $[0, 1]$.

3 | Related Work

In this section we will enumerate several important research directions that are related to our work. There are research works studying the representative characteristics of an AS-level graph, and works proposing different ways to improve the accuracy of the inferred AS relationships. There are also research works studying the issues faced by the Border Gateway Protocol, especially the exponential Default Free Zone table growth, and their possible solutions in order to achieve routing scalability.

3.1 AS-level Graph

Researchers observed that in the Internet there is a hierarchy imposed on nodes (distinguishing stub domains from transit domains) rather than random structure. After that, Faloutsos et al., in their seminal paper [FFF99], found that in both router-level graphs and AS-level graphs the distributions of node degrees, the degree ranks of the nodes and the number of nodes within h hops of each other follow power-law distributions. Later on, the authors in [MKF+06a] pointed out that the joint degree distribution (JDD) appears to fundamentally characterize Internet AS-level graphs and narrowly define the values of other widely considered metrics. The JDD is defined to be the probability that a randomly selected edge connects i- and j-degree nodes.

The authors in [CGJ+04] tried to quantify the completeness of the AS-level graph obtained from the Route Views, and found that a significant number of inter-AS connections are invisible from the BGP routing tables. In addition, they tried to capture a more representative AS-level graph by supplementing the Route Views data with information from Internet Looking Glass sites as well as IRR databases. However, the authors in [MKF+06a] compared AS-level topologies obtained from the three most commonly used BGP data sources—traceroutes, BGP, and WHOIS—in terms of a range of topology metrics and analyzed the interplay between the collection mechanisms of these three data sources and the resulting AS-level topologies. They pointed out the open question as to what data source contains reliable information about what type of links, the answer to which can help guide us in

combining the right information from the right data source in order to get the most representative AS-level topology. The authors in [DCDkc12] studied the connectivity of a small group of ASs that feed BGP RIBs to Route Views and RIPE NCC's RIS collectors, since the authors in [DD08, OPW+08] indicated that many AS links, especially peering links, are missing in Route Views and RIPE NCC's RIS data unless either one or both endpoints of the link are BGP route monitors. They first used Route Views and RIPE NCC's RIS data to identify such usable monitors, and found that the low visibility of the connectivity of Content Providers (CPs)—ASs that make money by providing content instead of Internet transit, e.g., a network that supports e-commerce—mostly contribute to the incompleteness of the AS topology. They also proposed the CMON algorithm to classify links of each usable monitor, and found that customers increasingly choose Large Transit Providers (LTPs) as their primary transit providers and only use Small Transit providers (STPs) as backup transit providers and that, in order to reduce upstream transit costs, CPs are peering aggressively.

3.2 AS Relationship

L. Gao in [Gao01] showed that the selective export rule set up according to an AS's relationship with its neighboring ASs made the AS path of a BGP routing table entry "valley-free". That is, a provider-to-customer edge or a peer-to-peer edge can be followed by only provider-to-customer or sibling-to-sibling edges. This property sets up the pattern of an AS path. That is, one or zero uphill paths (a sequence of edges that are either customer-to-provider or sibling-to-sibling), followed by one or zero peer-to-peer edges, followed by one or zero downhill paths (a sequence of edges that are either provider-to-customer or sibling-to-sibling). Gao then proposed an algorithm for inferring AS relationships based upon the above AS path patterns and AS degrees (i.e., a provider is typically larger than its customers and two peers are generally of comparable size). The authors in [SARK02a] assigned a rank to each AS according to the directed AS-level graph generated from each vantage point and inferred the relationship between any two ASs by comparing their vectors, each of which was composed of ranks assigned to an AS from multiple vantage points. Based on these relationships they constructed the AS-level hierarchy of the Internet. X. Dimitropoulos et al. in [DKF+07] introduced new heuristics for inferring sibling-to-sibling and peer-to-peer relationships and improving the integrity of provider-to-customer (or customer-to-provider) relationships. They also validated the inferred AS relationships with organizations' network administrators operating the ASs.

3.3 Peer-to-peer Overlay Networks

There are several structured peer-to-peer overlay networks implementing object location based on keys. They find a node that is responsible for the key contained in a message, and route the message, by using the existing Internet infrastructure, to this node. They put no constraints on the structure of the keys used, and thus no location-based aggregation can be applied. Pastry [RD01] and Tapestry [ZHS+04] take into account network topology in order to reduce the routing stretch. However, Chord [SMK+01] sacrifices network latency for the simplicity in handling concurrent node arrivals and departures. Though the routing state maintained at each Pastry/Tapestry/Chord node in order to find the node that is responsible for a key is scalable, and the routing stretch incurred by Pastry/Tapestry/Chord can be made small, they still rely on the existing Internet to get to the node found.

3.4 BGP Table Growth

In [Hus,Hus01a], Huston pointed out several practical operations that may have contributed to the growth of the tables. Bu et al [BGT04] attributed the BGP table growth to four major factors: any prefix of a multihomed AS cannot be aggregated by any one of its providers (refer to Figure 2.2(b)); prefixes, with the same set of policies applied to them, cover discontinuous address space; a prefix of an AS is split and these split prefixes are advertized along different AS paths; and prefixes, which use the same set of policies and cover continuous address space, fail to aggregate. The above four factors are called multihoming, address fragmentation, load balancing, and failure-to-aggregate respectively. They found that the main contributor among these is address fragmentation; however, both load balancing and multihoming contributions grow faster than routing tables, and load balancing is identified as the fastest growing contributor. They based the above conclusions on the BGP routing data collected from year 1997 to year 2002. Here we extend their work by measuring how these four factors affect the BGP table size over more recent years in Figure 3.1. Each line in Figure 3.1 shows the change in the difference between the number of prefixes in total and the number of prefixes after the corresponding contributor were removed over time. Refer to [BGT04] for the details in how to calculate the number of prefixes after each of the contributors were removed. From Figure 3.1, we can see that load balancing and multihoming contributions keep growing faster than the other two contributions, as pointed out in [BGT04]. However, we identify multihoming, instead of load balancing, as the fastest growing contributor from year 2005 to year 2008. The steep fall from year 2008 for the

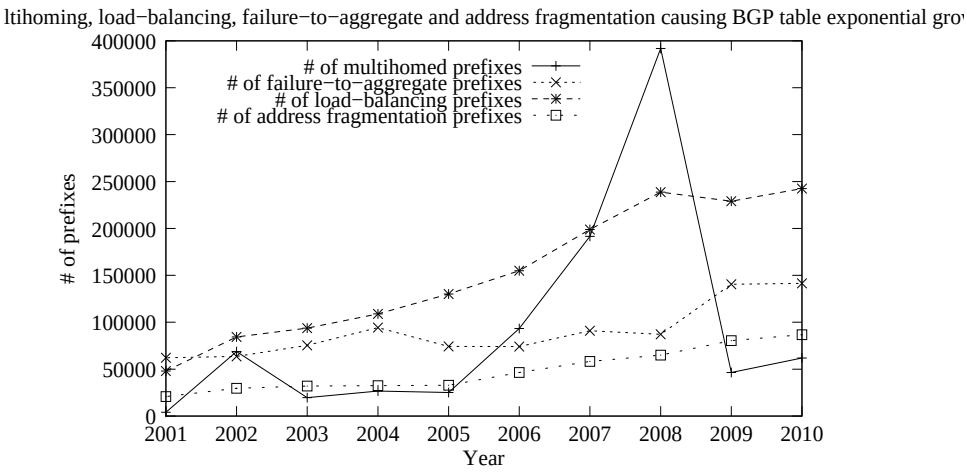


Figure 3.1: BGP table growth caused by four factors in [BGT04]

multihoming contributor can, in some extent, measure how far the PI prefixes are assigned to multihomed ASs. Bu et al [BGT04] identified the multihomed prefixes based on the assumption that the prefix of a multihomed AS was a split prefix of some prefix of one of its provider ASs. When more PI prefixes are assigned to multihomed ASs, the number of the multihomed prefixes computed in Bu et al's way decreases.

More recently, Xu et al. [MXZ+04] showed that among all the address blocks allocated and announced into the routing tables, 45% were split into smaller fragments by ISPs or stub ASs for load sharing or for the geographic reason, more than doubling the size of the routing table. Furthermore, they found that the evolution of routing tables consists of not only the appearance of new prefixes but also the withdrawal of old prefixes, and the dynamics of each of the two processes are much higher than that of the size of the routing table. Finally, they distinguished covering prefixes from covered prefixes and identified practical motives behind covered prefixes.

Narayan et al. [NGV03] proposed a causal model called ARAM. It captures the causes, rather than the effects, of table growth that govern the structure of the routing table. It is validated by using abstract shape measures (prefix length distribu- tion, prefix depth, and number of tree nodes), and they showed that the model matches the shape of the existing routing tables. They then used this new model to evaluate the storage requirement of various IP lookup schemes as a function of the table size. They concluded that the algorithms based upon multibit tries provide more density (more prefixes per chip) than TCAMs unless TCAMs can be engineered to use 8 or fewer transistors per cell.

3.5 BGP Convergence

The authors in [OZPZ09] made the first attempt to quantify the pervasiveness of BGP slow convergence based on the entire operational Internet rather than controlled experiments using a small number of prefixes [LABJ01, LAWS01]. They used a timer-based approach to cluster routing updates into events, classified these routing events into different categories, and computed the duration as well as the number of paths explored for each category of events. As a by-product of event classifications, the authors developed a new usage-time-per-path based algorithm to emulate a router's policy in ranking available AS paths to the same destination. They justified the argument that path exploration and BGP slow convergence do widely exist in the Internet, but they also showed that the severity of the convergence problem varies depending on both the origin of the prefix associated with the routing event, and the point in the routing hierarchy at which the event was observed. They also found an order for the durations of various route convergence events.

3.6 Solutions to BGP Scalability Issue

The concept of subnetting [MP85] was first proposed in the late 1980s for moderately large organizations with more than one Local-Area Network (LAN). If a distinct network number had been assigned to each LAN, then an explosion in the size of Internet routing tables would have been incurred.

Later on, CIDR [FL06] was proposed in the mid-90s and has been served as a mid-term solution to the routing scalability issue faced by Internet. CIDR introduces a new allocation architecture based on the actual and short-term needs of individual organizations, uses supernets to aggregate multiple contiguous prefixes to reduce the number of entries in each global routing table, and assigns prefixes following the underlying Internet topology so that aggregation can be applied to further reduce the routing state. By using route aggregation, CIDR alleviates route flapping since any components of an aggregated route would not cause that aggregated route to flap.

The original IP design stipulated that an IP address be globally unique and reachable, as well as being the identifier of a network attachment point to the Internet. Network Address Translation (NAT) [EF94] has changed the original definition of an IP address and was proposed as another short-term solution to address space depletion and routing scalability issues (for example, renumbering, multihoming). A NAT is placed between a stub domain and the backbone and translates the local IP addresses (i.e., IP private addresses) used by

this stub domain and reused by other stub domains to the globally unique IP addresses. This approach is feasible when only a subset of the local IP addresses within a stub domain are required to be translated into the globally unique IP addresses (only incurring small translation tables). NATs are accepted as a reality in today's IP architecture, and various NAT traversal solutions [ea] have been developed to restore the end-to-end reachable model required in the original IP architecture. A review of NAT can be found here [Zha08].

In [BFCW08] and [BFCW09], the authors proposed Virtual Aggregation (ViAggre) which no longer requires FIBs to include routes to all prefixes visible in today's Internet. Instead, it uses a set of virtual prefixes, which, topologically speaking, are not valid aggregates. This set of virtual prefixes covers the whole global address space. An Aggregation Point Router (APR) is responsible for one or more virtual prefixes and installs entries for all the longer prefixes covered by these virtual prefixes into its FIB table. A non-APR, on the other hand, only installs the routes for virtual prefixes into its FIB table and suppresses any routes for longer prefixes covered by virtual prefixes. ViAggre reduces the size of FIB without requiring any architectural change or any change in router software and protocols. However, it will stretch internal paths within an ISP as well as demand configuration changes for internal and external routers of the ISP.

In [VBc+04] a group of prefixes originated at an AS and deemed to be equivalent by this originating AS is declared as an atom; other ASs must announce each atom as one routed object. An atomized routing architecture was also put forward, which results in the reduction in the size of the DFZ routing table and improvement over convergence behavior.

The authors in [NB09] proposed a layer 3 shim protocol for IPv6 that achieves scalable multihoming by making each multihomed host in a site have multiple provider-dependent IPv6 address prefixes. When something fails or one communicating end host finds that one or more of its locators becomes unreachable or has changed locator preference settings, another working IPv6 address pair is chosen to preserve established communications, causing minimal impact on upper-layer protocols. In addition, load-spreading can be achieved by using different IPv6 address prefixes of a host for different communications to the host. This protocol does not separate the identifying and the network-layer routing and forwarding functions overloaded in an IPv6 address.

The author in [Chi99] pointed out that due to the failure in distinguishing the host object class from the interface object class, the name, i.e., an IP address, is used to identify conceptually different things—what we seek, where it is, and how to get there. The multiplicity of roles for IP addresses makes host mobility and host multihoming more difficult to

solve.

In [MWZZ07] the authors put forward the general idea that customer networks be separated from the backbone routing system consisting of various provider networks. By performing this separation, the size of the global routing table and the amount of routing churn is dramatically reduced, the roadblocks for end sites to use multihoming are removed, there is no need for renumbering when changing providers, and the barrier against security attacks from compromised end hosts is raised. However, it is difficult to detect border link or border router failures. In addition, the critical mapping function incurs delays in packet forwarding, a target for security attacks, and a system cost for implementation and deployment. Similar work can be found in [FFML12], which follows the ENCAPS scheme [Hin96]. The authors outlined a detailed Locator/ID Separation Protocol (LISP) to implement the division of the global address space (existing IPv4 or IPv6 address space) into Endpoint Identifiers (EIDs) and Routing Locators (RLOCs). RLOCs are togologically aggregatable and globally routable, while EIDs are only routable within sites without regard to topology. LISP routers intercept EID addressed packets and help them route through the core of the Internet by mapping source and destination EIDs within packets to corresponding RLOCs. In [ZFWY06] the authors used inter-provider tunneling and again a mapping service that maps a customer site's prefix to a tunnel endpoint (i.e., an address of a provider's edge router) so that the routers in the provider infrastructure only need to compute routes to tunnel endpoint prefixes. To shrink the FIB table size, virtual prefixes are introduced to decouple network topology from addressing in order to scale global IP routing in the Internet. However, by doing this, AS paths can be stretched.

The author in [O'D97] proposed an alternate addressing architecture for IPv6 in order to solve the scaling issues such as rehoming, multihoming, and global route computation. The 16-byte IPv6 address now consists of Routing Goop (RG), Site Topology Partition (STP), and End-system Designator (ESD). The architecture separates public topology (allowing for aggressive hierarchical topology aggregation) from site-local topology, separate site identity (using the globally unique ESD element of an address) from where the site is attached to the public topology (using the RG element of an address), insulate the hosts in the site from the global Internet (rewriting the RGs of the addresses at site border routers), and circumvents a hierarchical forwarding path by a cut-through within the optimization region (an engineering choice made by two parties of the cut-through).

Without requiring the majority of the ASs to deploy the solution in order to fully achieve its benefit, the authors [KJZ+10] proposed an evolutionary path towards scaling the global

routing system: at each step the ASs making the changes should interoperate with those that have not changed so far, and should gain immediate benefits to amortize the cost for the deployment of those changes. The main idea of this approach, which can be termed as aggregation with increasing scopes, involves FIB aggregation within a router, virtual aggregation [BFCW08, BFCW09] within an AS as well as across the ASs, and mapping exchange (mapping a destination prefix to its provider's egress router) through which the flaps of customer prefixes can be isolated from the core of the Internet.

The authors in [DD10] reviewed the functional requirements for an inter-domain routing system in order to meet not only the needs of the current Internet routing system but also those of the future Internet [Lit89]. It also reviewed other work on requirements for domain-based routing [Tsu87,HD90,ISO94,Chi91]. After that, they pointed out several issues related to the current domain-based routing architecture in order to gain some insights into the requirements for the future domain-based routing architecture.

3.7 Compact Routing

The main idea of hierarchical routing based on address aggregation used in today's Internet comes from the work [KK77]. However, due to the small-world phenomenon of the AS-level graph (most of the node pairs are within 2-4 hops away from each other), we cannot truly realize the effectiveness induced by abstracting out the topological details about the remote parts of the Internet. In addition, various business relationships and operational requirements have induced various forms of address deaggregation. Another direction to deal with the routing scalability (one that would require a radical redesign of the Internet) is compact routing. Here compact means the sizes of address, header and routing table are logarithmic in the network size, and the path stretch is bounded by some small constant. Under typical assumptions, shortest path routing cannot guarantee sublinear routing table sizes [GP96], except for special graphs (for example, grids and trees). In order to make the table size sublinear to the network size, some of the topological information is sacrificed, and the path hop stretch significantly larger than 1 is thus incurred. Compared with geographic routing (refer to next subsection), compact routing provides a guarantee for the worst-case path hop stretch. More specifically, it has been shown that, to achieve a worst case path stretch of 3, the size of a route table should be of order $O(\sqrt{n \log n})$ [TZ01], and, to achieve a worst case path stretch less than 3, the size of a route table has to be of order $O(n)$ [GG01, GP96]. The best performing name-dependent (topology-aware) and name-

independent (topology-unaware) general schemes are [TZ01, AGM+08] [1]. The multiplicative stretch achieved by any of these two is 3 [GG01]. On the other hand, for static scale-free graphs, the logarithmic routing table size and the close-to-one stretch can be achieved [BC06, KFY04]. In particular, the authors [BC06] built a small-size set of spanning trees, and the metadata stored in a message header as well as at a node's routing table for each tree in the tree cover are computed using the compact routing algorithm for trees in [TZ01]. An exact distance labeling scheme for trees in [Pel99] is also applied for picking which tree in the cover to use for forwarding.

However, for Internet-like graphs, there is some bad news about compact routing [KkccFB07]: (1) using flat identifiers can only result in polynomial rather than logarithmic scaling, which means the locator-identifier split cannot really alleviate the routing scalability concerns; (2) in the face of a topological change, the communication costs incurred by achieving coherent full views of the topology cannot scale better than linear, which, though better than the exponential communication costs per topological change of any currently deployed routing scheme, does not allow for "infinite scaling" [DDK06].

3.8 Survey on Addressing Schemes Using Geometric Information

As an alternative to the traditional routing table approach, each router in a large internetwork could be assigned either a physical location from some Global Positioning System (GPS) device or virtual coordinates in a metric space (X, d) in which the initial graph topology is embedded. Each packet is forwarded to routing elements that make the greatest progress toward the destination in terms of geometric distances. This kind of greedy forwarding requires little routing state to be maintained at each node, and the size of each routing table is proportional to the degree of each node.

Actually, nodes in many networks can efficiently find the targets they want to communicate with, even though they do not have any global knowledge of the topologies of the networks. The authors in [TM01] conducted the following experiment in 1960s: an arbitrary target person and a group of starting persons are selected. Each starting person is provided with a document and the information related to the target person. The sender has to choose one recipient that can advance the progress of the document toward the intended target. Their experiment results showed that 29% of the documents sent out reached the intended target. In addition, the completed acquaintance chain from some starting person to the

[1] the authors of compact routing work use "name" instead of "address" in order to avoid certain connotations of the latter term—viz., that addresses are tied to topology

target has only 5.2 links on average.

J. Kleinberg [Kle00] proposed a network model to explain the small-world phenomenon. Specifically, in a two-dimensional $n \times n$ lattice, each node u has a short-range connection to each of the nodes within $p \geq 1$ Manhattan distance, and has $q \geq 1$ long-range connection chosen independently from a distribution with a clustering exponent α. Each node knows its own coordinates, the coordinates of its neighbors, and the coordinates of the target node in the Euclidean plane. The message is then forwarded using the connection that advances it closest to the target, in terms of the lattice distance. He found that the efficient navigability can be achieved when $\alpha = 2$.

3.8.1 Using Physical location information

Geo-based routing has been considered mostly in the context of wireless and ad-hoc networks. Geographical locations from GPS devices were used and the routing based upon these physical locations was performed as follows: a packet is forwarded to the node that minimizes the distance to the destination. This process is known as greedy routing. However, a packet may get stuck at some node that is closer to the destination than any of its neighbors. This “local minimum issue" has been solved by using the face routing method [KK00]: each node is assumed to be equipped with a radio with the same circular radio range r, and we say there is an edge between two nodes if the geographical distance between them is smaller than r; when a packet gets stuck at a node, it is forwarded on progressively closer faces of the planar subgraph of the network, and each of these faces is traversed by the right-hand rule. The procedure continues until the packet reaches a node that is closer to the destination than the one where the packet entered this face routing phase. The forwarding of the packet then returns to the greedy phase. Here the right-hand rule states that when node x is reached from node y, the next edge to be visited is the first link after sweeping counter-clockwise about x from edge(x,y). A planar graph is a graph that can be embedded in a plane; it can be drawn on a plane in such a way that its edges intersect only at their endpoints. Here the planar subgraph is constructed to make the above solution to the local minimum issue work. If the graph is not planar, face routing may fail. Two well-known planar graphs are relative neighborhood graph (RNG) and Gabriel graph (GG) [GS69, Tou80]. The planarization technique proposed in [KK00] reduces to removing edges from the original graph, until either RNG or GG is obtained without disconnecting the graph. The drawback of this graph planarizing method is the idealized assumption of unit disk graphs as connectivity graphs. In such graphs, a node is always linked to all nodes within its nominal radio

range, and never connected to nodes outside this range. Another problem with this method is that, without using any effective bounding, it may need to explore a considerable part of the entire network [KWZ03].

There are many variants of early fallback heuristics based upon the observation that the greedy forwarding is on average more efficient than the face routing. The "first closer" heuristic, used by the "face routing" [KK00] mentioned earlier, makes the algorithm resume the greedy routing as soon as a packet arrives at a node closer to the destination than the starting point of the current face routing phase. The algorithm using this heuristic is proved to be not worst-case optimal. It is believed that the complete boundary of the current face has to be explored for the algorithm to achieve asymptotic optimality, but a very large face may have to be explored, which is prohibitively expensive compared with an optimal path from the source to the destination in Greedy Other Adaptive Face Routing (GOAFR) [KWZ03]. GOAFR [KWZ03], on the other hand, provided worst-case guarantees as well as average-case efficiency. F. Kuhn et al. [KWZZ03] extended GOAFR to GOAFR$^+$ which provided theoretically asymptotically worst-case optimality as well as average-case efficiency on practical networks by dropping the assumption that the distance between nodes may not be smaller than some constant value.

Young-Jin Kim et al., in [KGKS05], showed that this idealized assumption of unit disk graphs is grossly violated by real radios, which leads to permanent failures in geographic routing. They instead proposed the Cross-Link Detection Protocol (CLDP) that makes geographic routing always succeed on any kind of connectivity graphs. It mainly differs from [KK00] in its planarizing method—dropping the unit disk graph assumption.

The main drawbacks of the methods in [KK00, KGKS05, KWZZ03, KWZ03] are:

- significant amounts of network congestion along the boundaries of the network holes.
- a planar subgraph maintained at each node, introducing more states.

To cope with the first drawback, S. Subramanian et al. [SSG07] proposed a randomized multi-path routing algorithm called the RANDOMWAY algorithm that can achieve near-optimal throughput, with delays near the optimal throughput-delay trade-off curve, even in networks with holes. They also put forward a constructive scheme called RANDOM-SPREAD to uniformly distribute traffic flows over the region, and thus can support any traffic demands, even those with wide variations. The main problem with their methods is that they only used simplified traffic models, a small class of holes, and planar input graphs.

As for the second problem, Q. Fang et al. in [FGG04] tried to specify what underlying

geometric properties cause the local minimum phenomenon. They defined "stuck nodes," where packets can possibly get stuck in the greedy forwarding process. They then put forward a local rule—namely, TENT rule—for each node to test whether it is a stuck node, and a distributed algorithm—BOUNDHOLE—to find the so-called "holes" associated with stuck nodes. Finding the boundary of each hole gives rise to a route around the hole, which is used to help get packets out of the local minimum. Unlike the approaches based on planar subgraphs, computing and storing the information of the routes around holes are only required at the parts of the network where there are indeed communication voids.

3.8.2 Embedding into Geometric Space

The main problem with geographic routing based on physical locations is that this location information may be unavailable. If the location of a destination is unknown, one solution was provided in [LJC+00]. A distributed location service called Grid's Location Service (GLS) was proposed: each node maintains its current location in a number of location servers, and each node acts as a location server on behalf of some other nodes. However, if some node is not equipped with a GPS and thus has no location information available, inferring this missing physical location for this node has been proven to be NP-hard [BK98, KMW04].

Instead of using the actual location of each node, building virtual coordinates on top of the graph were proposed, and greedy routing was performed based on these coordinates instead of physical locations. The main idea of [RPSS03] is as follows: a few vertices were chosen as anchor nodes, and each vertex computed the length of the shortest path from itself to each of these anchor nodes and combined those distances to serve as its virtual coordinates. The problem with this scheme is that in order to ensure that a distance-decreasing path exists for any pair of nodes, $\Theta(n)$ anchors had to be used in the worst case.

To overcome the local minimum issue and to guarantee 100% packet delivery rate, the concept of "greedy embedding" was defined in [PR05], and different embedding methods ensuring greedy property have been proposed in recent years. We will examine these methods in the following sections. First we will illustrate the definition of a greedy embedding: a greedy embedding is a mapping $f : V \to X$, such that $\forall u, w \in V : u \neq w : (\exists v \in V \setminus \{u, w\} : v \in N_u : d(f(v), f(w)) < d(f(u), f(w)))$, where N_u denotes the neighboring nodes of u. R. Kleinberg [Kle07] proved that every finite graph with n nodes has a greedy embedding in a d-dimensional normed vector space where $d = \Omega(\log n)$, and that every finite graph with n nodes has a greedy embedding in the hyperbolic plane.

3.8.2.1 Embedding into Euclidean Space

C. Papadimitriou et al. [PR05] put forward a famous conjecture: any 3-connected planar graph can be greedily embedded in an Euclidean plane, such that there is an Euclidean-distance-decreasing path between any pair of nodes. T. Leighton et al. [ML08] resolved this conjecture and constructed a greedy embedding into the Euclidean plane for all circuit graphs (generalized forms of 3-connected planar graphs). However, restrictions have to be put upon the underlying graphs in order to greedily embed into the Euclidean plane.

It is known [May06] that the dimension of an Euclidean space needs to be at least $\log n$ for a greedy embedding to exist for any n-node graph.

C. Westphal et al. [WP09] put forward a method to embed a graph into the Euclidean space of dimension $O(\log n)$. Their algorithm resulted in the size of each entry in a route table of order $\log^2 n$, and the number of entries of a route table the degree of the node, achieving the scalable routing. In order to embed a n-node graph into this low dimensional space, it makes use of the refined version [Ach03] of Johnson-Lindenstrauss Lemma [JL84].

The Johnson-Lindenstrauss (JL) Lemma [JL84] is stated as: for $0 < \epsilon < 1$, $u, v \in l_2^n$, and $k > k_0$ with $k_0 = O(log(n)/\epsilon^2)$,

$$(1-\epsilon) \leq \frac{\|f(u) - f(v)\|}{\|u - v\|} \leq (1+\epsilon) \tag{3.1}$$

Here l_2^n stands for using l_2-norm ($\|.\|$) to compute the distance in an n-dimension Euclidean space. f is a random projection of l_2^n onto l_2^k with $k < n$, which realizes dimension reduction. To be specific,

$$f(x) = \frac{1}{\sqrt{k}}(\langle x, r^1\rangle, \langle x, r^2\rangle, ..., \langle x, r^k\rangle)$$

Each coordinate r_i^j of r^j in l_2^n, where $1 \leq j \leq k$ and $1 \leq i \leq n$, is an independent and identically distributed (i.i.d.) random variable satisfying the standard normal distribution. The lemma shows that one can reduce the dimension of an Euclidean space from n to k, while the distance distortion is only within a factor of ϵ.

The refinement [Ach03] uses very simple probability distributions to generate the projection matrix, reducing the computation of the projection to summations and subtractions. Specifically, it uses the Rademacher distribution, equal to 1 or -1 with equal probability. The resulting distortion is no worse than the previous JL implementation. Here k_0 should be chosen as follows [Ach03]: for $\beta > 0$, $k_0 = (4 + 2\beta) \times \log n/(\epsilon^2/2 - \epsilon^3/3)$, and (3.1) is satisfied with probability $1 - n^{-\beta}$. By using this refinement, the number of bits used for each coordinate is at most $\log(n)$.

Even in the static case where the greedy property is satisfied, the greedy path found by the above method may be longer than the shortest path in terms of number of hops. The maximum stretch can be quite large, though the average stretch is never more than 1.5. This is mainly due to the fact that they tried to build a tree structure out of the original graph for the assignment of coordinates. Some connectivity information is lost; two nodes, which are close in the original graph, may be far away on the tree and thus be assigned coordinates that differ greatly in terms of the distance function of this metric space. See Figure 3.2.

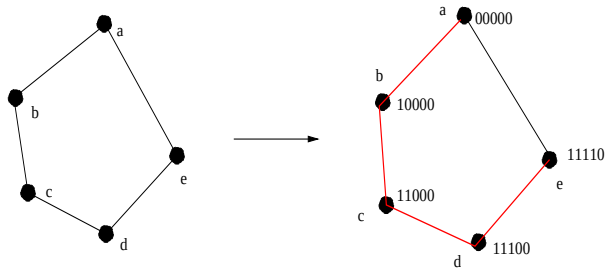


Figure 3.2: A spanning tree built with coordinates assigned

Coordinates are assigned as in [WP09]. For a n-node graph, first build a spanning tree from it. Next, get the canonical basis $\xi = \{e_1, ..., e_n\}$, where e_i is the vector with 1 in the i-th position and 0 otherwise. Assign a n-dimensional zero vector to the root of the tree. Define g as a mapping from a node in the tree to an n-dimensional vector. For each node v at depth $i+1$, $g(v) = g(p(v)) + e_l$, where $p(v)$ is the parent node of v and $e_l \in \xi$ has not been assigned to any edge yet. Put another way, associate a distinct $e_l \in \xi$ with each edge in the tree. We can see that for the spanning tree built out of the original graph, we would use $a \to b \to c \to e \to d$ as a greedy path, even though the shortest path is $a \to d$. This resulted in the path stretch equal to 4. The authors [KkccFB07] pointed out that any truly scalable Internet routing scheme should have stretch very close to 1.

C. Westphal et al. [WP09] proposed one optimization: multiple embeddings. m different trees are constructed and m sets of coordinates are assigned. The problem with this optimization is that these m trees may differ in a very trivial way, resulting in little gain in connectivity information while m times coordinates have to be dealt with at each hop along each path.

R. Flury et al. [FPW09] uses a small-size small-stretch tree cover instead of a single tree. This is the *first* greedy embedding scheme with stretch guarantee. Given a graph $G = (V, E)$, a tree cover of size k and stretch ρ is a family $T = \{T_1, T_2, ..., T_m\}$ of acyclic

connected subgraphs of G such that, for every $u, v \in V$, there is a tree T_i satisfying $d_G(u,v) \leq \rho d_{T_i}(u,v)$. Here $m \geq k$, and $d_G(u,v)$ and $d_{T_i}(u,v)$ denote the number of hops of the shortest path between u and v in G and T_i. The size of a tree cover is defined as the maximum number of acyclic connected subgraphs or trees in the tree cover a node appears. Each tree in the tree cover constructed in [FPW09] is required to be spanning, and the size of the tree cover is thus equal to the number of trees in the tree cover. By extending the celebrated Lipton-Tarjan separator theorem for planar graphs [LT79], and the approach to construct tree covers [GKR04] for planar graphs, they constructed a constant-stretch tree cover of size $O(\log n)$ for an n-vertex combinatorial unit disk graph (CUDG which is a UDG without any geometric information). A CUDG allows for edge crossings and has the property that if edge (u,v) of G crosses edge (x,y) of G then at least one of the (u,x), (u,y), (v,x) and (v,y) is also an edge of G.

The authors [AP90] showed that every graph has a $O(\log n)$-stretch tree cover with the size equal to $O(\log^2 n)$, which can be obtained in polynomial time, and the authors [FPW09] claimed that their approach can be used for any graph as long as a small-size, small-stretch tree cover can be constructed. Therefore, the algorithm proposed in [FPW09] works for any arbitrary n-vertex graph by greedily embedding it into $(R^d, \text{min-max})$, where $d = O(\log^3 n)$, the stretch $\rho = O(\log n)$, using $O(\log n)$ bits for each coordinate of each vertex, and defining the min-max function as follows: let c be a factor of d. Let $s = (s_1, s_2, ..., s_d)$ and $t = (t_1, t_2, ..., t_d)$ be points in R^d. Divide the d-dimensional Euclidean space into $\frac{d}{c}$ c-dimensional Euclidean spaces. Project s and t onto each of the $\frac{d}{c}$ c-dimensional Euclidean spaces, compute the corresponding L_∞ norm, and take the minimum of these L_∞ norms as the result of the min-max function. Here $\frac{d}{c}$ is set to the size of the tree cover. The key part of the algorithm is how to construct a small-stretch, small-size tree cover for any arbitrary graph.

In this dissertation, we complemented this work by extending algorithm 1, which is taken from [FPW09], to the graphs we are most interested in—AS-level graphs. We revise step 1 by using the proof of [AKP91, Lemma 3.1], combined with algorithm *MAX_COVER* and algorithm *cover*($\mathcal{R}$) in [AP90]. Each tree in the tree cover is not required to be spanning, and the size of a tree cover is usually far less than the number of trees in the tree cover. For step 2, set $c = \frac{1}{\log_2 3 - 1}$ and $k = (\log_2 n)^2$. Each T is isometrically embedded in $l_\infty^{c \log_2 n}$ [LLR95, Theorem 3.3]. The 400-node graph and 2000-node graph are generated using Georgia Tech Internetwork Topology Models (GT-ITMs) [CDZ97]. We can see that by building the tree cover our way, both the maximum stretch and the average stretch are bounded and remain

Table 3.1: Avg/Max stretch by embedding into $(R^{O(\log^3 n)}, \text{min-max})$

graph size	tree cover size	maximum stretch	average stretch
400	8	2.00	1.02
2000	10	2.33	1.03

small. However, according to [AKP91], any topological change will unavoidably incur the cost of the recomputation of the whole tree cover. More research efforts are required to make use of the resemblance between the topology before and after the given change in order to make the tree cover update more efficiently.

3.8.2.2 Embedding into Hyperbolic Space

In this section, we recall some facts about hyperbolic geometry, especially the geometry of a hyperbolic plane. The facts listed below are most widely used when embedding a graph into a hyperbolic plane (refer to [Thu97] and [And07] for more details). The reason we are interested in a hyperbolic space is that when scale-free networks are embedded in a hyperbolic plane, the efficiency of greedy forwarding is maximized even in the face of network dynamics, realizing a convergence-free routing scheme. However, the authors [KkccFB07] pointed out that compact routing schemes, whether universal or not, do not scale well in the face of network dynamics.

There are many possible models for the hyperbolic plane H, each useful in its context. Two standard models of the hyperbolic plane H are the upper half-plane model and the Poincaré disk model. In the upper half-plane model, H is represented by the set of points $(x, y) \in R^2$ satisfying $y > 0$. In the Poincaré disk model, H is represented by the set of points satisfying $x^2 + y^2 < 1$. The hyperbolic plane has a boundary circle ∂H "at infinity." In the Pointcaré disk model, ∂H is identified with the circle $\|z\| = 1$. where $z = x + yi$, $(x, y) \in R^2$. In the upper half-plane model, ∂H is identified with the one-point compactification of the real line. The points on the boundary circle ∂H are called "points at infinity" or "ideal points." A hyperbolic line or geodesic is represented by an arc of a circle that is perpendicular to ∂H and meets ∂H at two ideal points. In the upper half-plane model, a vertical line is treated as a geodesic that intersects ∂H at the point at infinity and the point on the real line. Every orientation-preserving isometry of H is represented by a Möbius transformation [And07, p. 27]

$$z \mapsto \frac{az + b}{cz + d}$$

for some complex coefficients a,b,c,d such that $ad \neq bc$. A Möbius transformation can also be represented by a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

It turns out that the composition of two Möbius transformations is represented by the product of the corresponding matrices. This defines a mapping from the group of invertible 2-by-2 complex matrices to the group of Möbius transformations denoted as $M\ddot{o}b^+$. $\mu : GL_2(C) \mapsto M\ddot{o}b^+$ is defined as follows [And07, p. 47]

$$\mu(M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}) = (m(z) = \frac{az+b}{cz+d})$$

$$GL_2(C) = \{\begin{bmatrix} a & b \\ c & d \end{bmatrix} | a, b, c, d \in C, ad \neq bc\}$$

In particular, a Möbius transformation is an isometry of the upper half-plane model of H if and only if it preserves the upper half-plane as a point set. It is easy to check that this kind of Möbius transformation has the form [And07, Theorem 2.26]

$$m(z) = \frac{az+b}{cz+d}$$

where a,b,c,$d \in R$ such that $ad - bc = 1$ and its corresponding matrix form is

$$SL_2(C) = \{\begin{bmatrix} a & b \\ c & d \end{bmatrix} | a, b, c, d \in R, ad - bc = 1\}$$

The standard hyperbolic distance ρ for the Pointcaré disk model is computed using the following formula: $\forall z_1, z_2 \in H$

$$\cosh \rho(z_1, z_2) = \frac{2\|z_1 - z_2\|^2}{(1 - \|z_1\|^2)(1 - \|z_2\|^2)} + 1$$

This formula shows that the hyperbolic distance can be obtained from the Euclidean distance.

The main property of hyperbolic geometry is the exponential expansion of space. For example, in the hyperbolic plane, the area of a r-radius disc grows as $\sim e^r$. Consequently, if nodes are uniformly or quasi-uniformly distributed over a two-dimensional hyperbolic disc, their density grows exponentially with the Euclidean distance from the disc center in terms of Euclidean geometry.

R. Kleinberg [Kle07] proposed a method to embed an n-node graph in the hyperbolic plane to get the virtual coordinate for each node. The main property of hyperbolic geometry, as we mentioned, is the exponential expansion of space: expanding much faster than Euclidean spaces. Hyperbolic plane is thus metrically equivalent to an e-ary tree, i.e., a

tree with the average branching factor e. In fact, it can be thought of as the continuous version of a tree. Kleinberg used this fact to embed a spanning tree T of the original graph G in the hyperbolic plane. The algorithm proposed in [Kle07] found a greedy embedding of an infinite d_0-regular tree, where d_0 is the maximum degree of G. The nodes of T were then identified with the embedded nodes of the infinite d_0-regular tree built. By doing this, the greedy embedding of T is obtained. This embedding is also a greedy embedding of G. Through the usage of greedy embedding, greedy routing is guaranteed to be successful in finding a route to the destination as long as such a route exists.

However, the coordinates obtained are not scalable since the coordinates in the hyperbolic plane require n bits to describe one node. This implies that even though the number of entries in the route table is scalable (equal to the degree of the node), the size of each entry is not. In addition, the maximum degree d_0 has to be fixed; otherwise, the coordinates of all nodes in the network have to be changed. It is therefore not suitable for dynamic networks.

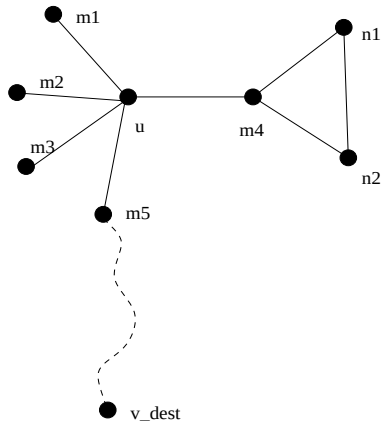
A. Cvetkovski et al. [CC12] optimized the procedure of [Kle07] by making use of topological and geometric properties of greedy embeddings to decrease the hop stretch of a greedy path. They found that the average node degree of an input graph does affect the hop stretch of the embedding. In addition, though the mapping of the spanning tree of an input graph to the embedded nodes of a d-ary tree in the hyperbolic plane has little impact upon the stretch, the choice of this particular spanning tree does influence the hop stretch. They proposed two heuristics: maximum-weight spanning tree (MWST) and minimum-diameter spanning tree (mDST).

A. Cvetkovski et al. [CC09] presented an algorithm for incremental greedy embedding of network nodes as they join the network by allocating disjoint subspaces of the hyperbolic plane in an online fashion. To cope with the disturbance of greedy property by node or link failures, Andrej Cvetkovski et al. also proposed a generalized greedy routing called the gravity pressure (GP) routing. The GP routing normally forwards packets to the neighbor that provides most positive progress towards the destination. When a packet reaches a local minimum, the GP routing forwards the packet to the neighbor that provides least negative progress towards the destination.

This GP routing does not require precomputation or maintenance of special subgraphs (for example, planar subgraphs). In addition, the GP routing is independent of the choice of the underlying metric space as well as the choice of whether to use physical or virtual coordinates.

However, it has the following issues:

Figure 3.3: The problem with GP routing: unbounded worst-case stretch



- each node iterates over each of the neighbors in the order first specified by the distance to the destination and then specified by the number of visits so far if the distance values are equal;
- backtracking has to be performed if in the wrong direction;
- a packet may traverse a very long path before reaching its destination. One of the causes is shown in the Figure 3.3.

In Figure 3.3, we assume that $d(u, v_{dest}) < d(m_1, v_{dest}) < d(m_2, v_{dest}) < d(m_3, v_{dest}) < d(m_4, v_{dest}) < d(m_5, v_{dest})$, and assume that only the neighbor m_5 has a path to v_{dest} and all other neighbors do not have paths to v_{dest}. Since $Visits[m_1] = Visits[m_2] = Visits[m_3] = Visits[m_4] = 0$, we set $Visits[u] = 1$ and forward to m_1. Since m_1 does not have a path to v_{dest}, we set $Visits[m_1] = 1$ and backtrack to u. Now u has to choose another neighbor. Since $Visits[m_2] = Visits[m_3] = Visits[m_4] = 0$, we set $Visits[u] = 2$ and forward to m_2. Since m_2 does not have a path to v_{dest}, we backtrack to u and set $Visits[m_2] = 1$. We do the same thing to m_3. Now $Visits[u] = 3$. Since $Visits[m_4] = 0$, we set $Visits[u] = 4$ and forward to m_4. After one loop, $Visits[u] = 4$ and $Visits[n_1] = Visits[n_2] = 1$ and m_4 has to forward to n_1 and loop once again. It takes 4 loops before $Visits[u] = Visits[n_1] = Visits[n_2] = 4$. Suppose $d(u, v_{dest}) < d(m_4, v_{dest}) < d(n_1, v_{dest}) < d(n_2, v_{dest})$ and only at that time can we backtrack to u. These 3 extra loops are totally a waste of time.

In order to help reduce the amount of traffic incurred by the backtracking when the greedy property is partially destroyed, we use a blacklist to record nodes that are not in the right direction from the source to the destination node. This list of nodes should be checked to decide upon the next-hop node. If the next-hop node chosen by the current node

is included in this list, the packet will not be passed on to that node. The current node then chooses another node. This procedure continues until the next-hop node chosen is not in the list. We will find the node not in the list, as long as there exists a path between the source and destination. For example, in figure 3.3, the first packet from u to v_{dest} is first forwarded to m_1. When it is returned to u, we record m_1 in the blacklist. We do the same to m_2, m_3, and m_4. Subsequent packets from u to v_{dest} will not enter m_1, m_2, m_3, and m_4. By using the blacklist, unnecessary detours are largely avoided by later packets. According to our evaluation, the maximum stretch is improved by 21 times, while the average stretch is improved by 3 times on average. There are drawbacks with this approach: more computation time is incurred at each node to access and update the blacklist before packets can be forwarded, and extra bits in a packet header are added to record the current content of the blacklist.

One way to avoid these drawbacks is to resort to flooding when a packet reaches a local minimum. The one-hop lookahead heuristic can be used to reduce the amount of flooding. Moreover, the nodes of degree one, which is not the final destination, need not be sent the packet during the flooding phase.

Algorithm 1 Forward packet at node u

Input: packet P and node u

1: **if** $u == pkt.dest$ **then**
2: done
3: **else**
4: $m_1 = \underset{n \in N(u)}{\operatorname{argmin}}\, d(n, pkt.dest)$
5: $d^1_{\min} = d(m_1, pkt.dest)$
6: $(p, m_2) = \underset{n \in N(u), n' \in N(n)}{\operatorname{argmin}}\, d(n', pkt.dest)$
7: $d^2_{\min} = d(m_2, pkt.dest)$
8: $\delta = (pkt.mode == \text{GRAVITY}\ ?\ d(u, pkt.dest)\ :\ pkt.dv)$
9: **if** $d^1_{\min} < \delta$ **then**
10: $pkt.mode = \text{GRAVITY}$
11: forward to m_1
12: **else if** $d^2_{\min} < \delta$ **then**
13: $pkt.mode = \text{GRAVITY}$
14: forward to p
15: **else**
16: $pkt.dv = d(u, v_{dest})$, $pkt.mode = \text{FLOOD}$
17: **for** $m \in N(u)$ with $deg(m) > 1$ **do**
18: make a copy of packet P and forward it to m
19: **end for**
20: **end if**
21: **end if**

This restricted flooding, equipped with the above optimizations, can still result in a huge number of packet copies in a network when a lot of nodes are removed, for example, 25% of the total number of nodes. However, the occurrence of the simultaneous failure of the $\geq 10\%$ of links in any large network is rather rare. Although greedy forwarding is subject to the failure of nodes with large degree values, these nodes usually have thousands of routers within them, and the occurrence of the simultaneous failure of all the routers within one of these nodes is also rather rare [PKBV10].

The authors in [BKC09, BK09, KPBV09, PKBV10] showed that the efficiency of greedy forwarding is maximized in the Internet-like synthetic network generated using the Einsteinian model [KPVB09, KPK$^+$10]. In [BPK10] the authors tried to find the coordinates in a hyperbolic space for each AS of the Internet, so that the likehood that the observed Internet topology has been produced by the Einsteinian model is maximized. This is achieved by computing the parameters of the Einsteinian model using the observed Internet topology. Greedy forwarding based on the inferred coordinates thus achieves the high success ratio. Low greedy path stretch, indicating the congruence between hyperbolic geodesics and topologically shortest paths [PKBV10, KPVB09, KPK$^+$10], and robustness in the face of network dynamics are also achieved. The amount of routing information maintained at each AS as well as the amount of routing communication overheads exchanged upon any change of the network topology are minimized, which means this hyperbolic mapping of the Internet achieves routing efficiency close to theoretically optimal.

Though the network growth model has been proposed for the Internet-like synthetic network generated using the Einsteinian model, there is currently no feasible network growth model for the real AS-level Internet. The question of "how do newly added ASs compute their coordinates in the hyperbolic plane without resorting to global knowledge of the network topology, so that the resulting embedding is still congruent with the hyperbolic plane to guarantee the efficiency of the greedy forwarding" has not been answered yet.

4 | Metric Based on BGP

Hierarchical, topology-based addressing is a fundamental component of the internetwork today, and is widely regarded as necessary for routing scalability. The purpose of addressing hierarchy is *abstraction*. Hierarchically structured addresses make it possible to treat groups of destinations as a single unit and identify them by a single label, typically a prefix common to all the addresses in the group, which helps abstract out topological details of remote parts of a given network. This improves scalability in both the control plane and the data plane. The savings in the control plane come from aggregating destination advertizements that share a common prefix, thus reducing communication costs. The savings in the forwarding plane result from having more destinations represented in the forwarding table by a single entry (corresponding to the aggregated route advertizement). The primary *cost* of this abstraction is that the paths followed by packets may in some cases be longer than is strictly necessary [PU89].

Achieving these benefits depends on the way addresses (prefixes) are assigned to destinations in the network. To maximize benefit, the following *locality property* should hold: addresses of destinations that are closer to each other in the network should share a longer common prefix. In the Internet today, the routing hierarchy has two levels. At the bottom (intradomain) level, prefixes (i.e., blocks of addresses) are assigned to networks, and individual hosts on the same network always have addresses with the same prefix, and thus locality generally holds at that level. At the top (interdomain) level, however, addressing is much less tied to the topology. That is, in the graph whose nodes correspond to ASs and whose edges denote paths used to forward packets, prefixes may not be assigned to nodes in a manner that allows aggregation. Thanks to CIDR, most customer ASs obtain their address blocks from their provider ASs—their neighbors in the AS graph—so some locality is present. However, for various reasons, including prefix assignments that predate CIDR, it would be surprising if the current configuration were optimal.

All of which brings up the questions we want to address in this chapter: How much locality is there in the assignment of prefixes to ASs in the current Internet? How far from *optimal*

is the current assignment? In other words, how much is hierarchical addressing buying us (at the interdomain level), and how much could be gained by a from-scratch reassignment of prefixes? Answering these questions requires precise definitions of both locality and optimality of prefix assignment, as well as a precise notion of semantic equivalence between prefix assignments. These questions are critical to next-generation networking because a number of proposals have been put forward regarding the use of non-topology-based addressing for global routing and forwarding [YCB07, PCG04, CCK+06, KCR08, CGP07, CCS96]. Quantifying the degree to which the current Internet benefits from hierarchical, topology-based addressing may help in designing next-generation routing and forwarding mechanisms. We stress that we are *not* proposing that addresses in the Internet actually be reassigned to improve locality. We are simply interested in the above questions as a way of assessing the importance of topology-based addressing at the interdomain level.

4.1 Defining BGP-based Locality Metric

Here we use the cost of advertising the prefixes associated with an AS to other ASs as a proxy. Intuitively, we want to measure something like the "entropy" of the addressing assignment. We take a simple approach: we calculate how much information must be conveyed across the edges of the AS graph to ensure that all prefixes are reachable. We can measure this directly from the BGP routing data. The following definitions are all relative to a collection of BGP route advertizements (like the Route Views data sets).

In what follows, i and j denote AS numbers. We denote links (directed edges) in the AS graph as pairs (i, j);

As we mentioned before in Section 2.3, the primary function of a BGP speaker is to exchange network reachability information with other BGP speakers. This network reachability information includes the set of destinations and the list of ASs that are traversed to reach these destinations. We say an edge (i, j) exists if and only if the sequence j, i occurs in the AS path of some BGP advertizement of network reachability information.[1] The letters p and q denote prefixes, while x and y denote AS paths appearing in some advertizement. *AllPrefs* denotes the set of all prefixes advertized. For path x, $(i, j) \in x$ means that i and j occur in that order in x. For any prefix $p \in$ *AllPrefs*, *paths*(p) denotes the set of AS paths over which p is advertized, i.e., the set of AS paths that occur in any announcement.

For each link (i, j) in the AS graph, we associate a set of prefixes, namely all those that

[1] Note that, for this part of the discussion, we consider edges to be directed.

are advertized across that link. This set is denoted by $destprfs(j, i)$. That is, for any p:

$$p \in destprfs(i, j) \equiv \exists x : x \in paths(p) \wedge (j, i) \in x$$

Now we define $codeLen(p)$ to be the number of bits needed to encode the prefix p for transmission. A simple coding scheme suffices for our purposes. For a prefix q of length L, $1 \leq L \leq 32$, we define $codeLen(q)$ to be $5 + L$ (that is, 5 bits for the length plus the prefix itself).

We define $advbits(i, j)$ to be the total number of bits needed to encode the prefixes in $destprfs(i, j)$:

$$advbits(i, j) = \sum_{p \in destprfs(i,j)} codeLen(p)$$

Finally, we define $TCost$ to be the number of bit-hops, summed over all links:

$$TCost = \sum_{i,j} advbits(i, j)$$

This is a cost measure, so smaller is better.

We would like to consider the trajectory of this cost measure over time, using different samples of BGP data. To do so, however, we need a way to normalize across graphs; the AS graph changes over time. In order to support at least rough comparison across graphs, we normalize $TCost$ by dividing it by the number of links (edges) in the AS graph.

This metric is useful for measuring the extent to which efficiency changes over time. However, it does not address the question that originally motivated our work: how much does hierarchical addressing buy us at the interdomain level? Answering that question requires a notion of an optimal (or minimum-cost) address assignment. The next section details some ways to construct such an assignment.

4.2 Defining Optimality for BGP-based Locality Metric

We would like to construct a minimal-cost address assignment for a given AS graph and total address demand. However it is (probably) too hard to be feasible due to the following reasons: AS-level graphs have more complex structure than trees; business relationships between nodes are confidential and complex (varying with space, time and prefix), and inferring them is very hard; the address space is limited. Therefore we have to resort to heuristics. We assign prefixes to ASs from scratch in a manner that preserves the semantics of a given RIB, while lowering cost by attempting to maximize aggregation (as it would occur in BGP). Specifically, we assign prefixes so that:

- The address demand of each AS is satisfied by the prefix(es) assigned to that AS, i.e., the new prefix assignment covers at least as many addresses as the original prefix assignment.
- The policies of each AS with respect to route advertizements are respected. That is, we preserve distinctions among prefixes reflected in the source data.
- *TCost* is minimized.

Our approach is heuristic. We do not attempt to find a true minimum-cost assignment, as that would be prohibitively expensive. Rather, we compare several different approaches that preserve semantics with varying levels of strictness.

The basic approach is as follows. We define an equivalence relation on prefixes, so that two prefixes are considered equivalent if and only if they have the same semantics with respect to the BGP data. This allows us to identify candidates for aggregation. We then assign prefixes so that the aggregate demand represented by an equivalence class of prefixes is satisfied. There is more than one way to do this, however, and we consider several alternatives.

4.2.1 Prefix Equivalence

We consider the conditions under which prefixes p and q can be considered semantically equivalent—that is, for the given source data, the two prefixes are indistinguishable with respect to policy. These conditions are based on the criteria provided by BGP for selecting prefixes.

We define the following conditions for a given set of BGP advertizements. p and q:

Path equivalence. $paths(p) = paths(q)$

Next Hop equivalence. For every BGP announcement a of p with AS path x and every announcement b of q with AS path x, the next hop attribute of a equals the next hop attribute of b. (The “next hop" attribute of a BGP announcement is the IP address of the next hop on the path.)

MED equivalence. For every BGP announcement a of p with AS path x and every announcement b of q with AS path x, the Multi-exit Discriminator attribute of a equals the Multi-exit Discriminator attribute of b. (The “Multi-exit Discriminator” attribute is used for so-called hot/cold potato routing when there are multiple connections between ASs.)

In our experiments we use three variants of this definition, based on the above conditions (beginning with the strictest):

Definition 1. Prefixes p and q are equivalent (belong to the same equivalence class) iff they satisfy all three equivalence conditions.

Definition 2. Prefixes p and q are equivalent if and only if they satisfy both Path equivalence and Next Hop equivalence.

Definition 3. Prefixes p and q are equivalent if and only if they satisfy Path equivalence.

Our next step is to assign prefixes to each AS to cover the total address demand of that AS. In the next subsection we show two ways to do that, given a set of prefix equivalence classes associated with each AS; the two vary in their aggressiveness about aggregation within and across ASs. Then we must define the set of prefixes that will be advertized over each link (i, j) in the AS graph, i.e., $destprfs(j, i)$.

In what follows, the address demand of a prefix equivalence class is the amount of address space covered by all the prefixes in the class. For example, an equivalence class $\{4.228.64.0/20, 192.168.3.0/24\}$ has a total demand of $2^{12} + 2^8 = 4352$.

4.2.2 Assigning Prefixes

We present two ways to assign prefixes to ASs so as to maximize aggregation. We dub them One Prefix Per Class (OPPC) and Provider-Based (with and without aggregation across ASs).

The **OPPC** Scheme simply assigns a single prefix of minimum adequate length to each equivalence class. The number of addresses contained in this single prefix may be larger than the actual address demand of the prefix equivalence class when the address demand of this class is not a power of 2. No attempt is made to make prefixes assigned to the same AS contiguous/aggregatable, and customer-provider relationships among ASs are not considered. This is the least-aggressive reasonable prefix assignment strategy. In general, an AS will still have multiple prefixes assigned using this strategy.

The **Provider-Based** scheme is more aggressive. It is based on the provider-customer relation on ASs. The idea is to let each provider assign prefixes to its customer ASs, making sure that the provider's prefix is large enough to cover the address demand of both the provider and its customers. This provides maximum opportunities for aggregation across ASs. The procedure is as follows

Step 1 Build trees. The root of each tree is a tier-1 AS. (We use connectivity information to identify tier-1 ASs, and verify using outside information.) Each link in a tree represents a provider-to-customer relationship. We do not include any peer-to-peer information into the trees. If a customer x has more than one provider, we pick the provider with the largest degree assuming the degree of an AS is closely related to the size of an AS [TDG+01]. We infer AS relationships using the method of Dimitropoulos et al. [DKF+07].) Add an imaginary dummy root, which is the parent of all Tier-1s. Note as mentioned in [DSK08], there may exist AS relationship cycles in the Internet.

Step 2 Calculate the address demand for each AS in each tree from the bottom up. The demand for a leaf node is the sum of the demands of its associated equivalence classes. The demand for a provider (nonleaf) node is the sum of its childrens' demands plus the sum of the demands of its associated equivalence classes. If the prefixes advertized by the provider belong to more than one class, we pick the class to whose address demand we add the sum of the childrens' demands so that the binary form of the addition result contains the fewest number of 1s.

Step 3 Assign prefixes from top down. The dummy root is assigned 0.0.0.0/1 and 128.0.0.0/1. We assign to the direct children of the root (Tier-1 ASs), keeping any remaining prefixes that are left unassigned for the next step. Then we assign prefixes from parent to child, as follows. First determine the child's minimum acceptable prefix length; if the nth msb of the binary representation of the node's address demand is set, a prefix of length $32 - n$ is needed. If there exist such blocks in the prefix set of the parent, assign any one of them to the child. If there exists a shorter prefix in the parent's set, break it into the minimum number of longer prefixes necessary to cover the child demand and allocate those to the child, leaving the remaining pieces in the parent's set. Otherwise we have to assign more than one prefix from the parent prefix set in order to cover the child demand, and we want this number to be as small as possible.

Step 4 Assign prefixes that have not been assigned to Tier-1 ASs (and thus have not been assigned to any node in trees) from Step 3 to any ASs that do not appear in trees, using the same assignment method as in the previous step. The reason why there exist ASs not appearing in the trees is that those ASs have peer-peer relationships but do not have customer-provider relationships with other ASs in our tree built in Step 1.

4.2.3 Computing *destprfs*(·)

In order to compute *TCost* once we have assigned fresh prefixes using the above methods, we must compute $destprfs(j, i)$ for each link (i, j) in the AS graph. However, this requires that we determine exactly which set of prefixes will be advertized across each link. While it is straightforward to compute this from actual BGP data, now we are dealing with artificial prefix assignments and we need to define what *cross-AS aggregation* will occur.

For the **Provider-Based** assignment scheme, we consider two possibilities: one where aggregation occurs across ASs (**Aggregate-Across-ASs**, PBAAA), and one where it does not (**Not-Aggregate-Across-ASs**, PBNAAA).

For the **PBAAA**: intuitively, we want to use the same tree that was used for address assignment, and aggregate prefixes upward (i.e., across customer-provider links) in that tree, adjusting $destprfs(i, j)$ as we go.

For each distinct equivalence class denoted by $[p]$, where p represents any prefix belonging to this class and for each unique path $x \in paths(p)$, we do the following. Suppose there are n ASs in x; reverse the path and denote the result by $k_n, k_{n-1}, ..., k_2, k_1$. The corresponding link sequence is $(k_n, k_{n-1}), (k_{n-1}, k_{n-2}), \ldots, (k_2, k_1)$. Let the set of prefixes assigned to k_i be $\mathcal{P}_{k_i}$ $(1 \leq i \leq n)$. We will work bottom-up in the tree used when addresses were allocated, that is, from customer to provider. Let the set of prefixes to be advertized across the current link (k_i, k_{i-1}) be denoted by $PSET(k_i)$ $(2 \leq i \leq n)$. Set i to n, and initialize $PSET(k_n) = [p]$.

- update $destprfs(k_i, k_{i-1})$ as follows:

$$destprfs(k_i, k_{i-1}) = destprfs(k_i, k_{i-1}) \bigcup PSET(k_i) \tag{4.1}$$

- update $PSET(k_{i-1})$ as follows: if k_{i-1} is the parent of k_i, then for any prefix q in $PSET(k_i)$ if all k_{i-1}'s children each of which has the prefix set containing some prefix(es) assigned from the same prefix q' which was assigned to k_{i-1} in our top-down prefix reassignment before q' is further broken down to be assigned to k_{i-1}'s children (for example, assigning q to k_i) are single-homed, replace q with q' in $PSET(k_{i-1})$.

- Decrement i.

We iterate these steps repeatedly until k_{i-1} is a customer or peer of k_i (i.e., the path starts downward in the tree), or i reaches 2. If $i > 2$, we only perform the first step repeatedly until $i = 2$.

For the **PBNAAA** and the **OPPC** scheme no cross-AS aggregation occurs; each prefix is advertized across all links in the set of AS paths of that equivalence class.

The detailed algorithm for calculating address demand from bottom up, assigning prefixes from scratch based on customer-provider relationships and computing $destprfs(j,k)$ for PBAAA scheme is outlined in algorithm 2 to algorithm 5.

Here we use the method proposed in [SARK02a] to find the core of Internet and the Tier-1 ASs constituting this core. We infer AS relationships using the method of Dimitropoulos et al. [DKF+07]. With the above two methods we can build trees rooted at each of the Tier-1 ASs. Each tree node corresponds to a distinct AS and has the following fields: *asNo*, *children*, *layerNo*, *parent*, *prefsPerClassArr*, *ASPathsPerClassArr*, *addrDemPerClassArr*, *newPrefsBefAssgnToChild*, and *newPrefsAftAssgnToChild*. Among them, *prefsPerClassArr* and *ASPathsPerClassArr* of each AS are initialized according to different definitions of equivalence class. *newPrefsBefAssgnToChild* is obtained from the parent node and before prefixes are assigned to any of its children, and *newPrefsAftAssgnToChild* is equal to what is left in *newPrefsBefAssgnToChild* after prefixes are assigned to all of its children according to their address demands. Each entry in *newPrefsBefAssgnToChild* and *newPrefsAftAssgnToChild* has two fields: *pref* and *classIdx* (the class this prefix belongs to). The reason we keep prefixes before being fragmented to be assigned to child nodes in the prefix field of the entry in *newPrefsBefAssgnToChild* is for the convenience of computing $destprfs(x,y)$ for the PBAAA scheme. We denote the binary representation of a number i by $(i)_2$. We set the root node as: $r.newPrefsBefAssgnToChild = \{\langle 0.0.0.0/1, -1\rangle, \langle 128.0.0.0/1, -1\rangle\}$; $r.asNo = -1$; $r.children$ equal to the set of Tier-1 ASs; $r.layerNo = 1$; $r.parent = -1$; $r.prefsPerClassArr$ equal to null; $r.ASPathsPerClassArr$ equal to null; $r.addrDemPerClassArr[0] = 0$. Suppose p is a prefix. Then $getPrefLength(p)$ is a function to return the length of p and we omit the implementation details for this function.

We give a detailed example shown in Figure 4.1 assuming the prefixes associated with each AS belongs to one single equivalence class. ASx is the provider of both ASy and ASz. Both ASy and ASz are single-homed to ASx. The resulted relabeled graph after OPPC, PBAAA and PBNAAA are applied respectively are shown in Figure 4.2 to Figure 4.4. The address demand of each AS as well as the routing policies are well preserved. The only difference between Figure 4.3 and Figure 4.4 lies in the value of $destprfs(x,w)$. Note: AAA stands for Aggregate-Across-ASs.

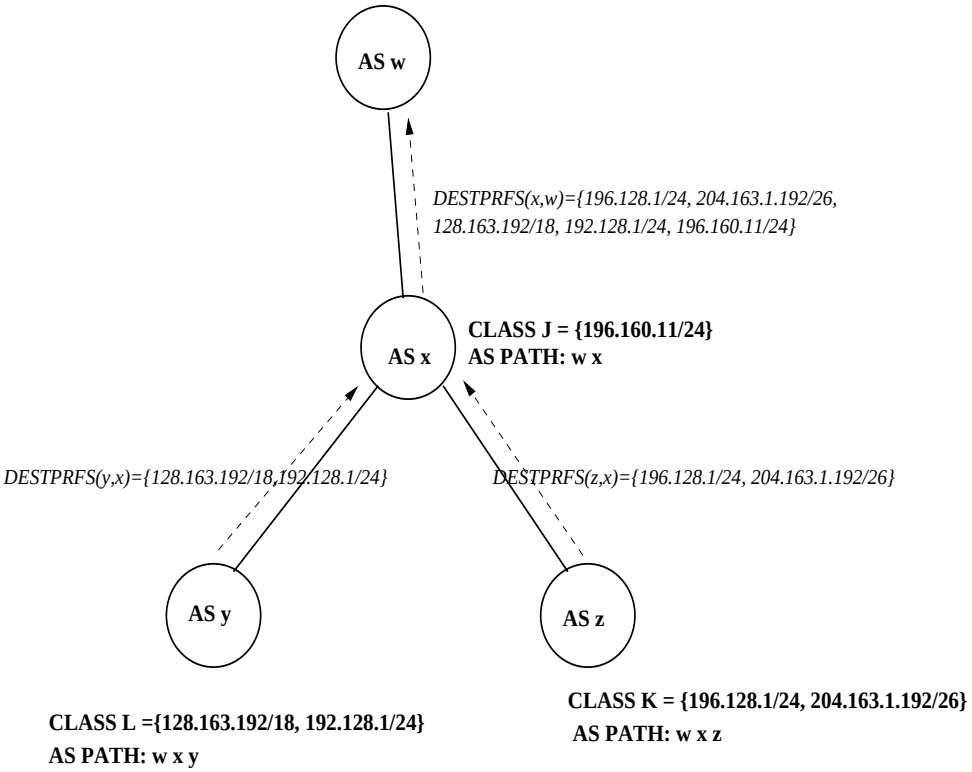


Figure 4.1: Partial AS-level graph

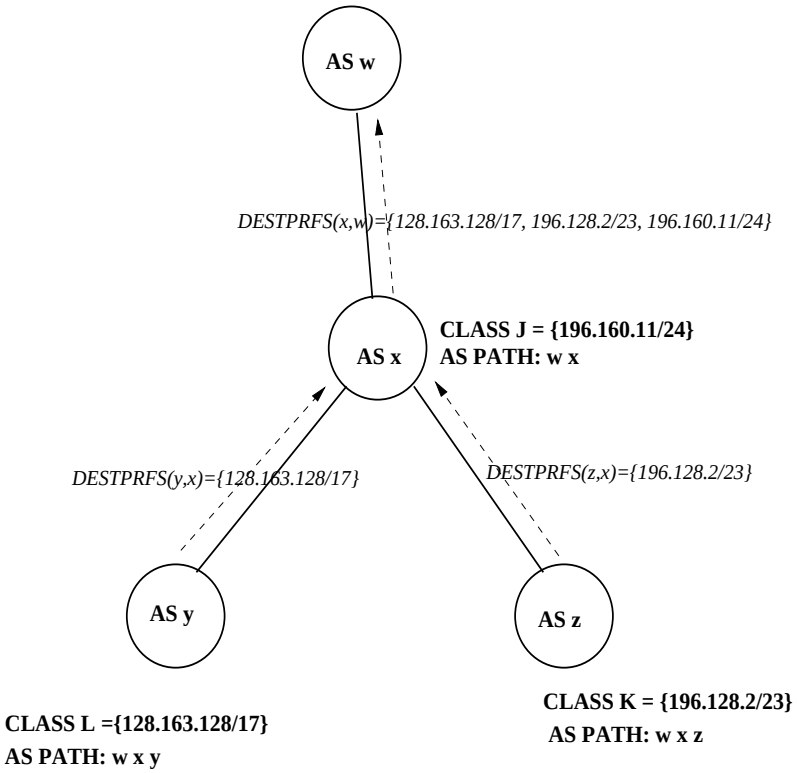


Figure 4.2: Reassign prefixes using OPPC. No AAA during advertizement.

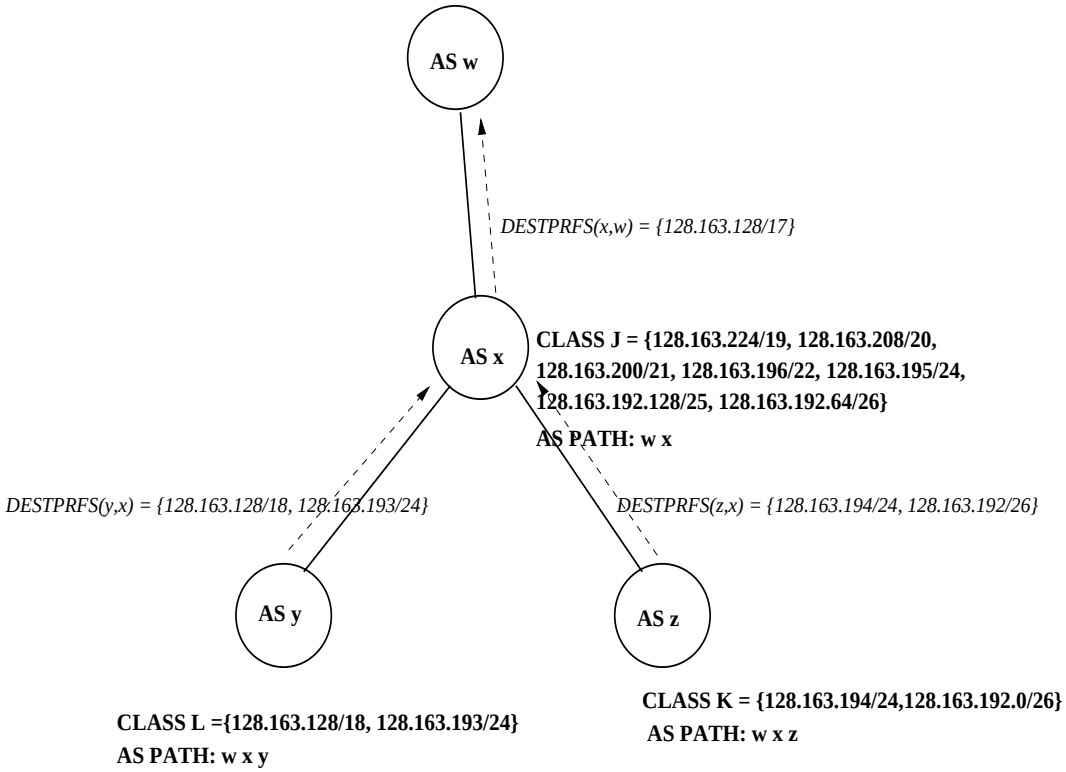


Figure 4.3: Reassign and advertize prefixes using PBAAA

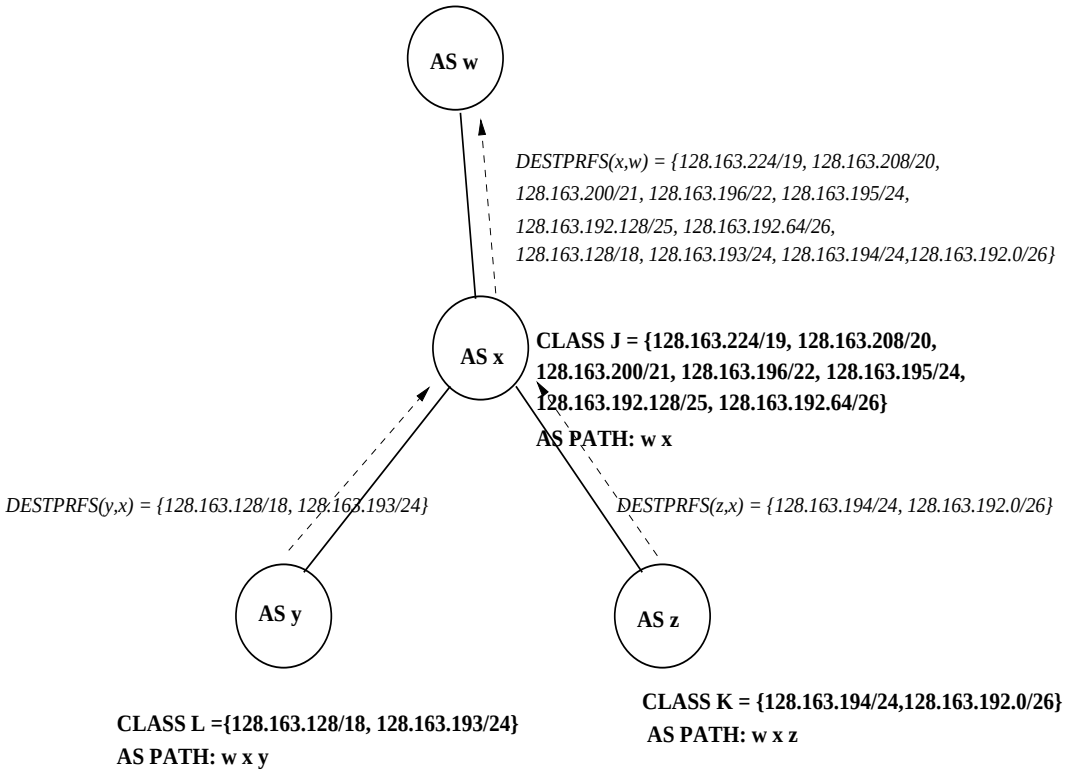


Figure 4.4: Reassign and advertize prefixes using PBNAAA

Algorithm 2 *setOwnAddrDemPerClass*(T): initialize *addrDemPerClassArr* field of each node of the tree T to be its own address demand

Input: the tree T built according to p2c or c2p AS relationships
Output: each node's own address demand

```
1: find the maximal layerNo denoted by maxLayerNo.
2: currLayerNo = 2
3: while currLayerNo ≤ maxLayerNo do
4:    find nodes at the currLayerNo layer, denoted by nodesAtCurrLayer
5:    for each node u in nodesAtCurrLayer do
6:       for i = 0 to u.prefsPerClassArr.size − 1 do
7:          ownAddrDemOfOneClass = 0
8:          for each p in u.prefsPerClassArr[i] do
9:             ownAddrDemOfOneClass+ = (long)1 << (32 − getPrefLength(p))
10:         end for
11:         u.addrDemPerClassArr[i] = ownAddrDemOfOneClass
12:      end for
13:   end for
14:   currLayerNo+ = 1
15: end while
```

Algorithm 3 *calcSubTreeAddrDemPerClass*(T): given the tree T built, calculate the address demand of each node to be the sum of its demand plus those of its descendants, and update its *addrDemPerClassArr* field. When the address demand of its descendants are added, try to pick one of its classes so that the minimal number of prefixes is required to cover the sum

Input: the tree T built according to p2c or c2p AS relationships

Output: the address demand of each node of each tree to be the sum of its demand plus those of its descendants

1: *setOwnAddrDemPerClass*(T)
2: find the maximal *layerNo* denoted by *maxLayerNo*.
3: $currLayerNo = maxLayerNo$
4: **while** $currLayerNo > 1$ **do**
5: find nodes at the *currLayerNo* layer, denoted by *nodesAtCurrLayer*
6: divide *nodesAtCurrLayer* into groups with each group of nodes having the same parent and different groups having different parents
7: **for** each of the above groups denoted by *nodesWithSameParent* **do**
8: $totalChildAddrDem = 0$
9: $prtOfGrp = nodesWithSameParent[0].parent$
10: **for** each node u in *nodesWithSameParent* **do**
11: **for** $i = 0$ to $u.addrDemPerClassArr.size - 1$ **do**
12: $totalChildAddrDem+ = u.addrDemPerClassArr[i]$
13: **end for**
14: **end for**
15: $minNumOfOnes = ((\text{long})1 << 32) - 1$
16: **for** $i = 0$ to $prtOfGrp.addrDemPerClassArr.size - 1$ **do**
17: $ownAddrDemOfOneClass =$
18: $prtOfGrp.addrDemPerClassArr[i] + totalChildAddrDem$
19: **if** # of 1's contained in the $(ownAddrDemOfOneClass)_2 < minNumOfOnes$ **then**
20: $minNumOfOnes =$ # of 1's contained in the $(ownAddrDemOfOneClass)_2$
21: $indx = i$
22: **end if**
23: **end for**
24: $prtOfGrp.addrDemPerClassArr[indx]+ = totalChildAddrDem$
25: **end for**
26: $currLayerNo- = 1$
27: **end while**

Algorithm 4 *assignPrfsToChildren*(x): assign prefixes from $x.newPrefsBefAssgnToChild$ to cover the demand of each of the child nodes y, and set its $y.newPrefsBefAssgnToChild$ accordingly. After that, we set $x.newPrefsAftAssgnToChild$ to be what is left in $x.newPrefsBefAssgnToChild$. For leaf node x, $x.newPrefsAftAssgnToChild = x.newPrefsBefAssgnToChild$. Note: $x.newPrefsBefAssgnToChild$ stays the same.

Input: $x.newPrefsBefAssgnToChild$ and the set of x's child nodes

Output: $y.newPrefsBefAssgnToChild$ of each child node y and $x.newPrefsAftAssgnToChild$ after x assigns prefixes to all of its children

1: **if** the child node set is empty **then**
2: $x.newPrefsAftAssgnToChild = x.newPrefsBefAssgnToChild$
3: **else**
4: sort $x.newPrefsBefAssgnToChild$ according to the ascending order of the length of each entry's prefix field
5: deep copy the sorted $x.newPrefsBefAssgnToChild$ to temporary variable $sortedPrefs$
6: $mask = 0x80000000$
7: **for** $i = 1$ to 32 **do**
8: **for** each u in the child node set **do**
9: **for** $k = 0$ to $u.addrDemPerClassArr.size - 1$ **do**
10: **if** $mask \& u.addrDemPerClassArr[k] \neq 0$ **then**
11: **if** $\exists j : getPrefLength(sortedPrefs[j].pref) = i$ **then**
12: $u.newPrefsBefAssgnToChild \leftarrow sortedPrefs[j]$
13: $sortedPrefs = sortedPrefs \setminus \{sortedPrefs[j]\}$
14: **else if** $\exists j : i \in [getPrefLength(sortedPrefs[j].pref),$
15: $getPrefLength(sortedPrefs[j+1].pref)]$ **then**
16: break up $sortedPrefs[j].pref$ into prefixes of length i denoted by $\{p_1, ..., p_n\}$,
17: where $n = (\text{long})1 << (i - getPrefLength(sortedPrefs[j].pref))$
18: $u.newPrefsBefAssgnToChild \leftarrow \langle p_m, k \rangle$, where $1 \leq m \leq n$
19: $sortedPrefs =$
20: $sortedPrefs \cup \{\langle p_1, k \rangle, ..., \langle p_{m-1}, k \rangle, \langle p_{m+1}, k \rangle, ..., \langle p_n, k \rangle\} \setminus \{sortedPrefs[j]\}$
21: resort $sortedPrefs$ according to the ascending order of the length of each
22: entry's prefix field

```
23:                 else
24:                     assignedAddrDem = 0
25:                     for j = 0 to sortedPrefs.size − 1 do
26:                         u.newPrefsBefAssgnToChild ← sortedPrefs[j]
27:                         assignedAddrDem+ = (long)1 << (32 − getPrefLength(sortedPrefs[j].pref))
28:                         if assignedAddrDem == (long)1 << (32 − i) then
29:                             sortedPrefs = sortedPrefs \ {sortedPrefs[0], ..., sortedPrefs[j]}
30:                             break
31:                         end if
32:                     end for
33:                 end if
34:             end if
35:         end for
36:     end for
37:     mask >>= 1
38:   end for
39:   x.newPrefsAftAssgnToChild = sortedPrefs
40: end if
```

Algorithm 5 *calcAvgBitsPerLnk*(): given the tree T, the average number of bits advertized per link using the PBAAA scheme to emulate what BGP does. Normalization is performed just before the RETURN statement such that the value of our BGP-based locality metric is not affected by the graph size.

Input: $u.newPrefsBefAssgnToChild$ and $u.newPrefsAftAssgnToChild$ of each AS u
Output: the average number of bits advertized per link using BGP and PBAAA scheme

find the maximal *layerNo* denoted by *maxLayerNo*.
$currLayerNo = 2$
while $currLayerNo \leq maxLayerNo$ **do**
  find nodes at the *currLayerNo* layer, denoted by *nodesAtCurrLayer*
  for each node $u \in nodesAtCurrLayer$ **do**
    for $i = 0$ to $u.newPrefsAftAssgnToChild.size - 1$ **do**
      $p = u.newPrefsAftAssgnToChild[i].pref$
      $classIdx = u.newPrefsAftAssgnToChild[i].classIdx$
      for each x in $u.ASPathsPerClassArr[classIdx]$ **do**
        reverse the path x denoted by x'
        for each link (j, k) in x' **do**
          $destprfs(j, k) = destprfs(j, k) \cup \{p\}$
          if (j, k) is not the last link and
              k is j's parent and k's children are single-homed and
              there exists a $q \in k.newPrefsBefAssgnToChild$ s.t. $q \prec p$ **then**
            $p = q$
          end if
        end for
      end for
    end for
  end for
  $currLayerNo += 1$
end while
$totalbits = 0$
for $(i, j) \in E(G)$ **do**
  $bits = 0$
  for $p \in destprfs(i, j)$ **do**
    $bits = bits + (getPrefLength(p) + 5)$
  end for
  $totalbits = totalbits + bits$
end for
return $totalbits/|E(G)|$

Algorithm 6 Given the tree T built according to p2c or c2p AS relationships and *prefsPerClassArr* of each node of T, first calculate the address demand of each AS; then assign prefixes from scratch while preserving the address demand of each AS and the semantics of routing policies; finally calculate the average number of bits advertized per link using BGP and PBAAA scheme

Input: T

Output: the average number of bits advertized per link

1: *setOwnAddrDemPerClass*(T)
2: *calcSubTreeAddrDemPerClass*(T)
3: $r.newPrefsBefAssgnToChild = \{\langle 0.0.0.0/1, -1\rangle, \langle 128.0.0.0/1, -1\rangle\}$
4: $r.asNo = -1$
5: $r.children$ equal to the set of Tier-1 ASs
6: $r.layerNo = 1$
7: $r.parent = -1$
8: $r.prefsPerClassArr$ equal to null
9: $r.ASPathsPerClassArr$ equal to null
10: $r.addrDemPerClassArr[0] = 0$
11: find the maximal *layerNo* denoted by *maxLayerNo*.
12: $currLayerNo = 1$
13: **while** $currLayerNo \leq maxLayerNo$ **do**
14: find nodes at the *currLayerNo* layer, denoted by *nodesAtCurrLayer*
15: **for** each node $u \in nodesAtCurrLayer$ **do**
16: *assignPrfsToChildren*(u)
17: **end for**
18: $currLayerNo+ = 1$
19: **end while**
20: print *calcAvgBitsPerLnk*()

5 | Results for BGP-based Locality Metric

Here we present the results of our experiments using three (increasingly strong) definitions of prefix semantic equivalence, two methods of assigning fresh prefixes to the AS graph and three methods of advertising them.

R. Cohen et al. in [CR06] argued that a small number of BGP routing tables from different sources (less than 10) reveals a significant number of links and incrementing the number of tables used only discovers limited amount of hidden links. Therefore, we only use 120 routing tables for each year with 10 tables per month (5 from Route Views archives and 5 from RIPE NCC's RIS database). P. Mahadevan et al. [MKF+06b] argued that there are only minor differences in the aspects of node and link sets of the AS-level graphs derived from BGP routing tables and BGP updates, respectively. Therefore, we do not use BGP updates here.

As in [HB96], we remove private ASes [MKF+06b] ranging from 64512 to 65535 because they are not unique and normally they should not be leaked to a global BGP table. To reduce the impact of BGP misconfigurations [MWA02], which can affect 200-1200 BGP table entries each day, we only keep the paths that appear in all of the routing tables used in our experiments as in [DKF+07].

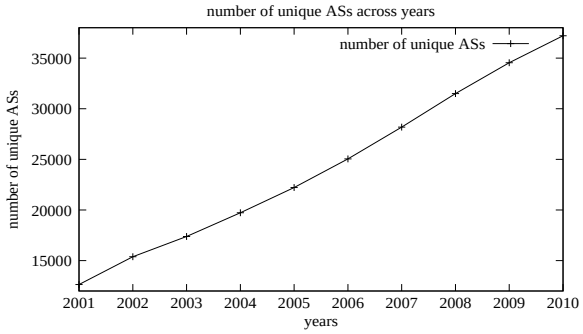
R. Cohen et al. in [CR06] argued that when routing policy is considered, around 80% to 90% of the customer-provider links are discovered by Route Views data and peer-to-peer links are mostly not revealed. X. Dimitropoulos et al. in [DKF+07] analyzed the reason why peer-to-peer links are not generally revealed by the existant BGP collectors. Since prefixes learned from a peer AS are not exported to any of its providers, a peer-to-peer link can only be observed by the customers or siblings of the two p2p ASs. Therefore, in order to make the peripheral peer-to-peer links visible, we need far more BGP collectors peering with these peripheral ASs. DIMES [SS05] indicates a shift from a few dedicated nodes with the sole objective of collecting routing tables, to a large number of hosts running background measurement agents, which helps reveal peripheral peer-to-peer links. However, DIMES lacks control plane information that we make use of and, since mapping router IP

address to AS number using longest prefix matching is very error-prone [ZOW+11], we do not use any traceroute-derived AS topologies from the DIMES project or from the Cooperative Association for Internet Data Analysis (CAIDA)'s Archipelago (Ark) Measurement Infrastructure [ARK], to solve the problem of hidden peer edges. In addition, we do not use any public route servers [RS] or looking glasses [LG] since they lack the history information of BGP data.

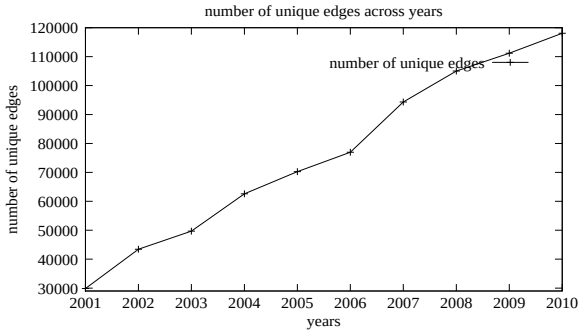
Instead, we resort to the IRR database to deal with the visibility of peer-to-peer links. It still remains an open question about what data source (for example, Route Views + RIPE NCC's RIS or IRR) contains the most reliable information about what type of link since Route Views and RIPE NCC's RIS reflect the topology seen from the control plane, while IRR reveals the topology seen from the management plane [MKF+06a]. The limitations of IRR database include: (1) the information in IRR database can be stale and inaccurate; and (2) not all ASs are willing to share their peering relationships; and (3) some providers tend to over-report their peering relationships to make them look more important than they actually are in the current Internet. We mainly use the RIPE NCC's IRR database in which almost all the 6800 registered ASs publish their peering relationships. According to [SF04] the RIPE NCC's IRR database is the most accurate registry. Since the IRR database, unlike RV database, is not affected by the policy, it reveals more peer-to-peer links. We analyze the policies of ASs in the IRR specified using RPSL and infer AS relationships as in [CR06] and [SF04]. When there exists the inconsistency of an edge type between using Route Views + RIPE NCC's RIS and using RIPE NCC's IRR, we label the edge the type inferred by using Route Views + RIPE NCC's RIS. For those newly discovered links (i.e., links that exist in IRR but not in Route Views + RIPE NCC's RIS) we have to guess the prefixes that traverse each of them in either direction by the following rules: we export local routes and the routes from the customers to providers and peers and we export everything from all the neighbors to siblings and customers.

The basic statistic information in terms of the number of nodes (ASs) and the number of edges (eBGP sessions) of our input AS-level graph for each year is shown in Figure 5.1.

We say prefix p and q are *matched* if p is advertized by AS x and q is advertized by one of the x's neighboring ASs y and either $p \prec q$ or $q \prec p$ is satisfied and we say x and y are a *matched* AS pair. Figure 5.2(a) and Figure 5.2(b) show the trends in the number of matched AS pairs and the number of matched prefix pairs across years respectively. From Figure 5.3(a) we can see that the number of matched AS pairs keeps decreasing. This can be explained as follows: given a matched AS pair one AS is usually the other AS's provider



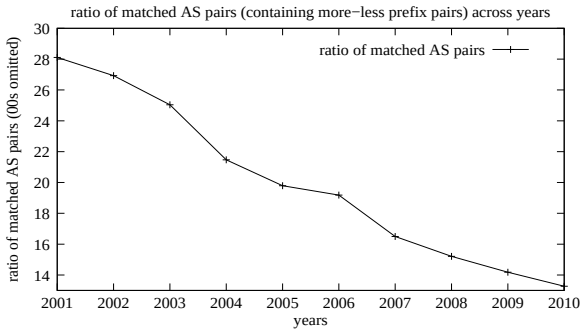
(a) number of nodes (ASs) across years



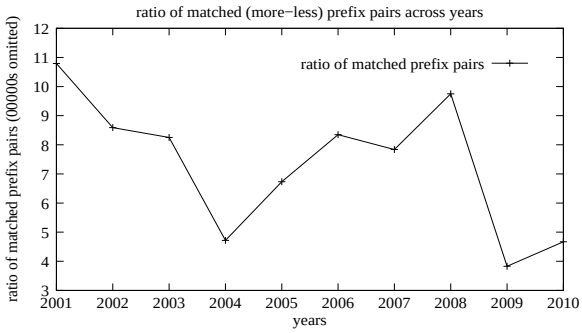
(b) number of edges (eBGP sessions) across years

Figure 5.1: Statistic information of input AS-level graphs across years

or customer [FL06] and the number of peering links grows more significantly over recent years than the number of provider-to-customer links [DD11], which means that the number of matched AS pairs does not grow as quickly as the total number of AS pairs.



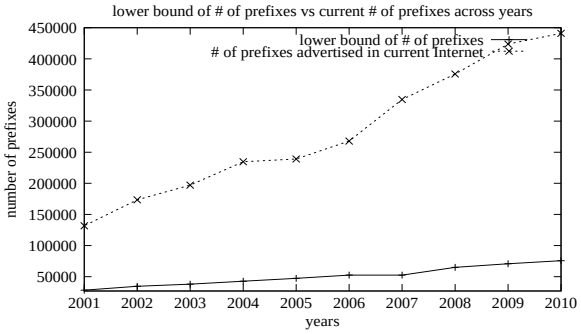
(a) number of matched AS pairs across years



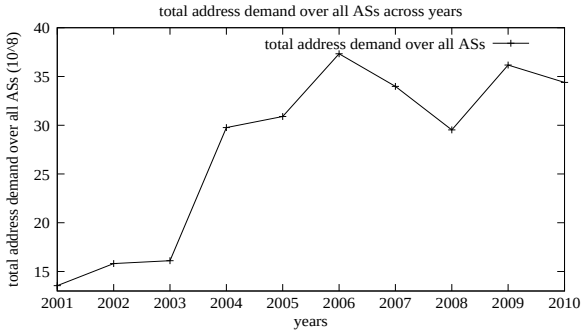
(b) number of matched prefix pairs across years

Figure 5.2: number of matched AS/prefix pairs across years

Figure 5.3(a) shows the minimal number of prefixes required to exactly cover the address demand of each AS by assuming all the prefixes associated with each AS follow the same routing policies. Figure 5.3(b) shows the actual number of prefixes that are advertized in the Internet. From Figure 5.3(a), we can see that the gap between the minimal number of prefixes required to exactly cover the address demand of each AS by assuming all the prefixes associated with each AS follow the same routing policies and the total number of prefixes advertized keeps increasing. This can be explained as follows. Each AS peers more ASs and adopts more complex policies for its prefixes; the number of policies used by each AS on average is far from one. Figure 5.3(b) indicates the total address demand from ASs keep increasing.



(a) lower bound of # of prefixes vs current # of prefixes advertized in Internet across years

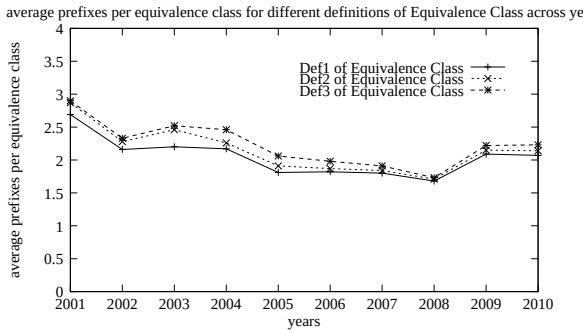


(b) total address demand over all ASs across years

Figure 5.3: current and lower bound of # of prefixes and total address demand

In the following sections, we first present some statistics about the equivalence classes according to the various definitions; this allows calibration of the differences in strictness. Then, we present $TCost$/link using various combinations of the methods. Finally, we compare our results with those from the CIDR report [GH], using routing data for September 15th, 2012.

5.1 Equivalence Statistics



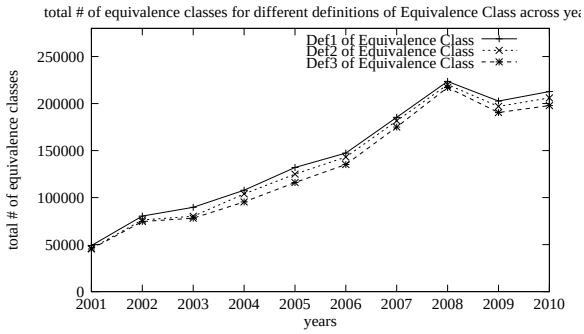


Figure 5.4: statistics for equivalence classes of increasingly more strict definitions

From Figure 5.4 we can see that with the increase in the strictness of the definitions of equivalence class we obtain more classes with fewer prefixes per class, which means finer granularity for routing policy, though in terms of both average number of prefixes per class and total number of classes the differences across these three definitions are not significant. In addition, the granularity of the BGP routing policy continues to get finer until year 2008 (indicated by more equivalence classes with the smaller number of prefixes per class) and

has been a little bit coarser since 2008.

One of the reasons for there being more than one prefix in an equivalence class is that these prefixes belong to different geographical parts (for example, different countries or different cities). By using GeoLite [Max]—a weekly updated database that maps IPv4 addresses to geolocations, we get the following results in Table 5.1. We cannot find the

Table 5.1: geographic reason for there being >1 prefix in an equivalence class

# of classes in >1 countries/with no country info		
2001	# of classes in >1 countries	4236
	# of classes with no country info	300
	total # of classes	45389
2002	# of classes in >1 countries	3982
	# of classes with no country info	213
	total # of classes	74602
2003	# of classes in >1 countries	4373
	# of classes with no country info	1957
	total # of classes	78148
2004	# of classes in >1 countries	4293
	# of classes with no country info	17202
	total # of classes	95439
2005	# of classes in >1 countries	4278
	# of classes with no country info	21904
	total # of classes	116024
2006	# of classes in >1 countries	5007
	# of classes with no country info	10085
	total # of classes	135211
2007	# of classes in >1 countries	4908
	# of classes with no country info	11234
	total # of classes	175070
2008	# of classes in >1 countries	4322
	# of classes with no country info	27030
	total # of classes	217124
2009	# of classes in >1 countries	6226
	# of classes with no country info	38467
	total # of classes	202714
2010	# of classes in >1 countries	9416
	# of classes with no country info	10354
	total # of classes	212762

geographic information from the GeoLite database for all the prefixes advertized. For those prefixes for which we find country information, we observe that there are only around 4% to 9% of classes, each having prefixes belonging to more than one country. Thus, in most cases, the reason for a class to have more than one prefix is due to previous topology-unaware

prefix assignment.

5.2 Optimal Assignments: Comparison

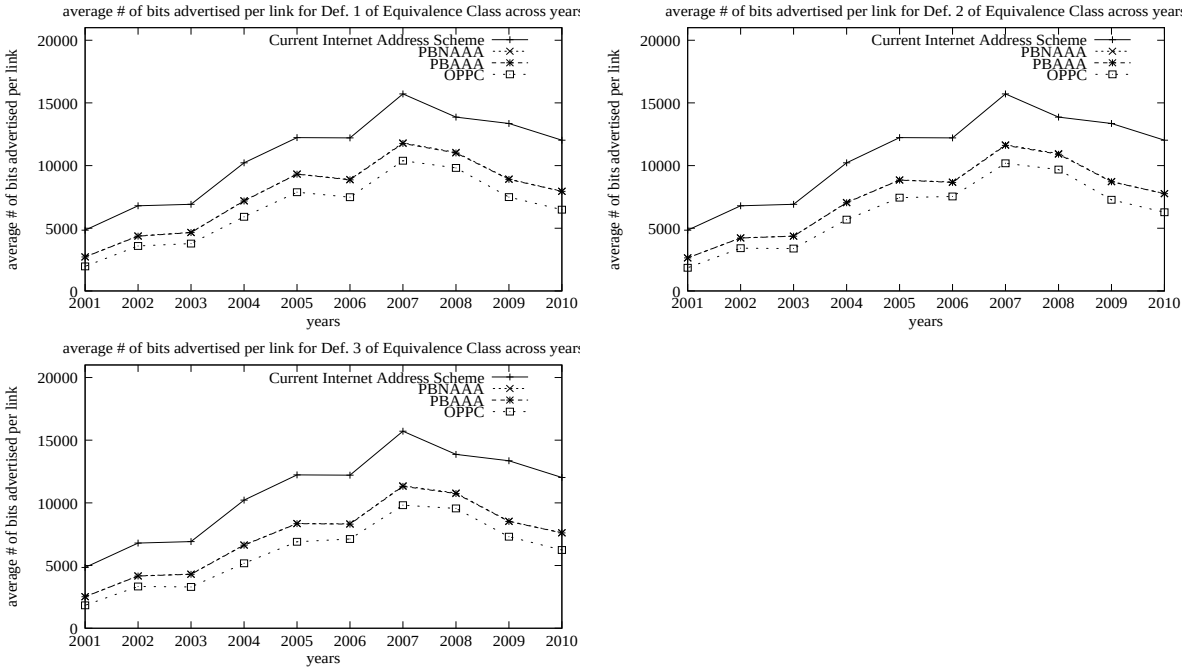


Figure 5.5: average $TCost$/link across years

Our initial experiment results show we cannot use OPPC to reassign address blocks from scratch without running out of the 32-bit address space for year 2005 no matter what definition of equivalence class is used and for year 2006 using definition 2 and definition 3 of equivalence class and for year 2009 and year 2010 using definition 3 of equivalence class. This is because for a given class of prefixes, the number of addresses covered may not be equal to a power of two, and, in order to assign a single prefix that covers at least the same number of addresses, we end up assigning a prefix that covers more addresses than it really needs to. For each of the above cases we expand the 32-bit address length limit to the 33-bit, to get a rough idea of what the average $TCost$/link will be across years.

By using OPPC, PBAAA, and PBNAAA methods to reassign prefixes from scratch with the routing policies staying unchanged we can save 25% at the minimum and 209% at the maximum, which serves as an evidence to the argument that the current address assignment is not close to optimal, and still retains a good deal of its original “flat" characteristics.

The difference between PBAAA and PBNAAA is trivial, as indicated by those two lines almost merging into one in Figure 5.4. This is because: the multihoming degree has been

increasing for both transit ASs and stub ASs with the increase rate for transit ASs much larger than that for stub ASs [DD11]; and links between transit ASs carry much larger traffic than those links at the edge of the Internet [ASS03]; and we do not aggregate children's prefixes with the prefix in their parent's prefix set into one new prefix if there exists among these children some child that is multihomed.

One interesting observation is that "the Current Internet Address Scheme" line keeps increasing until the year 2007 and then begins to decrease slightly. This is consistent with the observations in [Hus11] that the granularity of the BGP routing policy continues to get finer, though at a smaller rate in recent years, while the number of edges between ASs continue to increase at a linear rate and is consistent with the conclusion in [Hus11] that no signs of BGP's dire fate are visible now. In addition, in the face of an IPv4 address shortage, a number of ISPs aggregated some of their IPv4 prefix advertizements [Hus11], as we observe "/7" blocks in the BGP routing data in recent years.

5.3 Another Metric: Huston's CIDR Report

Here we use the provider-based reassignment of prefixes to ASs instead of OPPC prefix reassignment since if we use OPPC we have to expand the available address space from 2^{32} to 2^{33} in order to meet the address demand of each AS and preserve the routing policies of each AS no matter what definition of Equivalence Class is used.

For fair comparison we use Definition 3 (least strict) of Equivalence Class, which coincides with what is used in Huston's CIDR Report [GH]. That is, aggregation of prefixes may occur when AS paths used to advertize these prefixes are identical. For prefixes advertized using the same set of AS paths, CIDR aggregates either numerically consecutive and aggregatable prefixes or prefixes that are not numerically consecutive and aggregatable but can be made so by introducing non-routable space in between, which enlarges the actual address demand of an AS.

However, in our provider-based prefix reassignment, we treat the original prefixes advertized using the same set of AS paths as one equivalence class and reassign the minimal number of prefixes that covers the same amount of address space.

The results for the day September 15th, 2012 are shown in Table 5.2. "% gain" is computed by the difference between the number of prefixes before aggregation and the number of prefixes after aggregation divided by the number of prefixes before aggregation. The "CIDR Aggregation" column shows the results given in [GH]. Given the same routing policy of each AS, our provider-based reassignment saves a larger number of prefixes required

per AS than the Huston's method does. This is because we reassigned prefixes from scratch while Huston only did local prefix aggregation whenever possible. Local prefix aggregation can only solve the issues, such as the failure-to-aggregate. On the other hand, global prefix reassignment can also solve issues such as the address fragmentation. Nevertheless, these results show that the vast majority of the inefficiencies in today's addressing scheme arise from provider behavior, and could be easily fixed, without requiring renumbering.

Table 5.2: % Gain of CIDR vs Our Provider-based Reassignment on Sept.05,2012

AS#	CIDR Aggregation	Our Provider-based Reassignment
AS6389	94.4%	99.94%
AS28573	97.3%	99.89%
AS4766	77.1%	98.34%
AS17974	92.3%	99.53%
AS22773	49.7%	99.39%

6 | Topology-only Metric

First, we introduce notation necessary for discussing topological matters. Given is a connected labeled undirected graph $G = (V, E)$. Any subset of V is denoted by U. Lower-case letters u and v (and subscripted variants) range over nodes in V. Each node in V has ≥ 1 labels, and the intersection of any two nodes' label sets is empty. Labels are assumed (for now) to be binary strings of varying length, drawn from a set $\mathcal{L}$; the label set of node u is denoted by $lab.u$. Lower-case letters g and h (and subscripted variants) range over labels; ϵ denotes the empty label. Upper-case italic letter L (and subscripted variants) ranges over sets of labels. If we want to emphasize the node set U that a particular label set L is associated with, we use the notation $L(U)$.

Abusing notation slightly, let $lab.V$ denote the set of labels used in the graph G: $lab.V = \{g \mid \exists v \in V : g \in lab.v\}$. The length of a label g is denoted by $len.g$.

We denote the common prefix of g and h by $g \downarrow h$, and denote the common prefix of the label set L by $\Downarrow L$: $\Downarrow L = g \downarrow (\Downarrow L \setminus \{g\})$ if $|L| > 2$; $\Downarrow L = g \downarrow h$ if $L = \{g, h\}$; $\Downarrow L = g$ if $L = \{g\}$.

In this section, we explore the notion of "locality"—the idea that addresses in a network have some relation to location in the network topology, or put it another way, the idea that the distance between labels corresponds to the distance between nodes. We aim to come up with a measurement that would allow us to quantify the efficiency of a given network instance (i.e., an assignment of addresses to nodes in a graph). This measurement would also indicate what "optimal" instances should look like, which helps us quantify the degree to which the current Internet is non-optimal. Though the distance between nodes is clear enough, the distance between labels is not that obvious. We postpone making concrete the distance between labels.

It will be useful to first explore some basic cases in order to illustrate our intuition of locality. Let us consider the most basic nontrivial case: two nodes connected by one edge and each labeled a binary string. We describe our intuition about "good" and "bad" locality for this case:

- At the “bad" locality end of the spectrum, the two labels must be very dissimilar. This implies the length of at least one of the labels is more than the minimum, according to our intuition of the distance between labels.

- In the case of “good" locality, the difference between the two labels is minimized: only 1-bit difference is required to distinguish the labels of two nodes. The ideal labeling has one node labeled “0" and the other labeled “1". Any other labelings introduce redundancy—bits that are not necessary to distinguish these two nodes.

The general problem is this: given a finite undirected graph $G = (V, E)$, together with a labeling function $L : V \to \{g \mid g \in \Sigma^*\}$, where Σ is an alphabet, we want to compute a quantity $Q(G, L)$ that in some sense quantifies the “efficiency” of the labeling. The labeling function should be injective: $L(x) = L(y) \Rightarrow x = y$ for any nodes x and y.

Some high-level requirements on the solution:

- The measurement must be abstract—independent of any particular network system or routing protocol. It should be informed by the requirements of scalable network routing, since we know good locality leads to routing scalability.

- For the measurement to be of any practical usage, it should be solvable in polynomial time (instead of exponential time) for any graph as large as the current Internet.

- The measurement should be consistent with our intuition of locality.

We can think of an optimal network instance as follows: a network instance that requires the minimal number of bits for encoding by a sender, so that the receiving end can reconstruct the network instance—the graph structure and the label of each node.

We are inspired by Shannon's derivation of the properties of a quantitative notion of “choice" or uncertainty, which he called “entropy". Entropy, as we mentioned before, is defined to measure the amount of “choices" or uncertainty present in a situation where there are a finite set of individual outcomes in a sample space, each of the outcomes occurs with a given probability, and the sum of the probabilities of all the outcomes is equal to 1. Two basic components for Shannon's entropy are a set of choices and their probabilities. We can think of entropy as the length (the number of bits) needed to encode an outcome from a sample space. Shannon found that the minimal average taken over the number of bits to encode, in a uniquely decodable way, all the outcomes from a sample space is almost the same as the entropy of a sample space.

Unfortunately, our problem is more complex. We do not find an analog of an "experiment" and, as a result, we do not have a probability distribution. Instead we have a finite set of labels and the structure a finite graph.

However Shannon's third requirement seems helpful to our problem. It was a form of compositionality; it stated that the total measure should not change when a single choice is broken into a series of sequential choices without changing the probabilities. Therefore a number of transformations on graphs, which correspond to the reverse of the breaking down of choices, conserve the locality of the original network instance.

What transformation could be carried out on an arbitrary graph that would preserve locality? We consider a transformation in which we "merge" nodes whose labels are in some sense "close". Moreover, any transformation we might make on the graph should preserve uniqueness of labels.

To make this concrete, consider the following merging transformation assuming each node has only one label for simplicity and assuming the similarity between labels (the inverse of the distance between labels) is the common prefix of labels. When a node labeled α contains the label γ, and another node is labeled β, we say there is a misleading match, because longest-prefix matching stops working as expected as a routing technique. Combine two neighboring nodes into a single node whose label is their common prefix; all edges connecting either of these two neighboring nodes to the rest of the graph are attached to the replacement node. Before this merging transformation can happen, these two neighboring nodes have to satisfy the following conditions: their labels have a common nonempty prefix differing from the label of every node in $V(G) \setminus \{u, v\}$, and no misleading match is incurred: $\nexists w : w \in V(G) \setminus \{u, v\} : lab.u \downarrow lab.v \preceq lab.w \preceq lab.u \vee lab.u \downarrow lab.v \preceq lab.w \preceq lab.u$.

This above transformation preserves uniqueness. It also, in a sense, captures what we mean by locality: the ability to represent a subgraph by a single node. If we iterate the process of the above combining, it results, ultimately, in one of three cases:

- A single node, with a nonempty label. In this case the labeling in some sense correlates with graph topology, but contains redundancy.
- A pair of adjacent nodes with no common prefix. In the best case, one node is labeled 0 and the other is 1. Any other case represents a less than optimal labeling.
- A line or star or loop of nodes (or a set of interconnected loops and lines and stars), such that adjacent nodes share no common prefix.

The last case represents a non-optimal labeling; the larger the graph of this shape, the

worse the locality. We therefore want to define $Q(G, L)$ in terms of the size of the graph remaining when the process terminates, assuming that each node has only one label.

6.1 Greedy Contraction Process

The process mentioned previously is nondeterministic. That is: at any step, there may be more than one set of nodes, with distinct common prefixes, and depending on which set is combined, the final graph may be different. We are trying to solve the problem of deciding "Is there a sequence of choices that results in a final configuration having the number of labels/the number of bits on the graph when the process terminates at most k?", by introducing some greedy heuristics to make our hierarchical clustering tractable. Moreover, it requires only a minimal level of "nearness" in labels, such as a nonempty common prefix. For example, nodes with labels 0110101001 and 0010100000 could be merged, even though their common prefix is only 10% of their total length.

6.1.1 Heuristics

For simplicity of discussion we assume each node has only one label.[1] For each node u, $N(u)$ denotes the set of u's neighbors; For each set of nodes U, $G[U]$ denotes the subgraph induced by U: $V(G[U]) = V(U)$, and $e = (u, v) \in E(G[U])$ if and only if $u \in U$, $v \in U$, and $(u, v) \in E(G)$. Suppose, for a label denoted by cp, there exist more than one *maximal* node set denoted by U_1, U_2, ..., U_m: $\Downarrow L(U_1)) = \Downarrow L(U_2) = ... = \Downarrow L(U_m) = cp$; each of $G[U_1], G[U_2], ..., G[U_m]$ forms a connected subgraph of the graph G; $\forall u, v : u \in U_i \wedge v \in N(u) \setminus U_i : cp \downarrow lab.v \prec cp$. We will call each of them a *cluster* with the common prefix equal to cp. At most one of them can be merged into a new node and labeled with cp, without violating the uniqueness of each label in the updated graph. We apply the following heuristics, so as to make our metric as precise as possible:

- We say a cluster is *valid* for merging if no misleading match is incurred: for any node u not in the node set of a given cluster, if u's label is the extension of cp, then it is not a prefix of any label of any node in the cluster.
- In addition to considering how much structural similarity is shared by labels in $L(U)$, we also consider how much structural dissimilarity exists among these labels. The larger the difference among these labels, the more redundancy contained: more bits than necessary is used to distinguish the nodes in U. If there is still more than one U_i

[1] In this case we use $lab.v$ and the label in $lab.v$ interchangeably for the following discussion.

left after all the invalid clusters have been removed, we use another metric which we call *difference of the label set of a node set* and define it to be:

$$\mathit{diff}(U) = \sum_{u \in U} len.(lab.u) - |U| \times len.cp$$

We choose the cluster(s) with the minimal $\mathit{diff}(U_i)$.

- When we merge one U_i, other U_js $(i \neq j)$ cannot subsequently be merged without violating the uniqueness of labels. At later steps, any $u \in U_j$ $(i \neq j)$ cannot be merged with any $v \in N(u)$ without violating the validity condition, unless this v is (in)directly contracted from the node set containing U_i. Therefore $N(U_j)$ gives a rough estimation about how many future mergings can happen if U_j is merged. Let's illustrate this with a concrete example: Figure 6.1 shows the initial labeled graph:

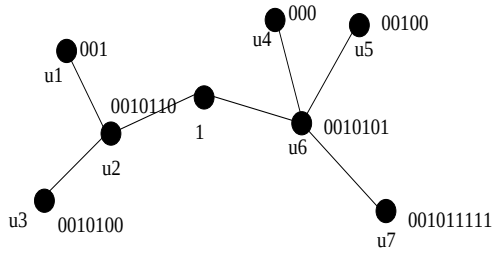


Figure 6.1: Initial labeled graph

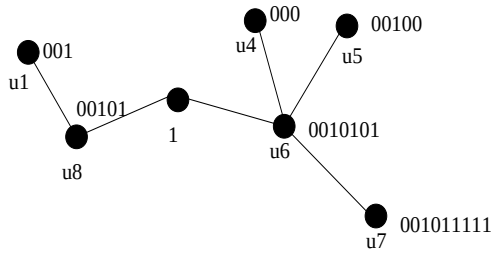


Figure 6.2: Contraction process: step 1,2

  - at step 1 both $\{u_2, u_3\}$ and $\{u_6, u_7\}$ are valid for merging, but we choose $\{u_2, u_3\}$ because $\mathit{diff}(\{u_2, u_3\}) < \mathit{diff}(\{u_6, u_7\})$;
  - at step 2, $\{u_5, u_6, u_7\}$ is not valid for merging—misleading matches would be incurred if we were to merge $\{u_5, u_6, u_7\}$. The result after step 1 and step 2 is shown in Figure 6.2;
  - at step 3, $\{u_1, u_8\}$ is valid for merging into node u_9;

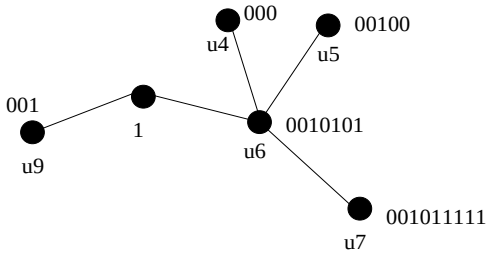


Figure 6.3: Contraction process: step 3,4

– at step 4, $\{u_4, u_5, u_6, u_7\}$ is not valid for merging—misleading matches would be incurred if we were to merge $\{u_4, u_5, u_6, u_7\}$. The result after step 3, 4 is shown in Figure 6.3.

We argue that, after we have processed clusters U_is at step $len.(\Downarrow L(U_i))$ (either merge or not) and got an intermediate labeled graph, there is no need to recheck any clusters U_js with $len.(\Downarrow L(U_j)) > len.(\Downarrow L(U_i))$ in this intermediate graph. Refer to Section 6.1.2.

Based on the above observations, we define $mrgablesWithin1Hop(U_i)$ as

$$mrgablesWithin1Hop(U_i) = |\{v \mid v \in N(U_i) \setminus U_i \wedge lab.v \downarrow (\Downarrow L(U_i)) \neq \epsilon\}|$$

If there are still more than one U_i left, we choose the cluster(s) with the maximal $mrgablesWithin1Hop(U_i)$.

- If there are still more than one U_i left, we sort labels in each U_i according to the lexicographical order, concatenate them, and pick the cluster with the smallest concatenated label according to the lexicographical order.

When each node has more than one label, the definition of *cluster* has to be generalized. A cluster associated with the common prefix cp is now a node-label pair set, with $nodeSet(cluster) = \bigcup_{\langle u,g\rangle \in cluster} u$ and $lblSet(cluster) = \bigcup_{\langle u,g\rangle \in cluster} g$: $G[nodeSet(cluster)]$ forming a connected subgraph; $\Downarrow lblSet(cluster) = cp$; $\forall u, v : u \in nodeSet(cluster) \wedge v \in N(u) \setminus nodeSet(cluster) : (\forall g : g \in lab.v : cp \downarrow g \prec cp)$; $\forall u : u \in nodeSet(cluster) : (\forall g : g \in lab.u \setminus lblSet(cluster) : cp \downarrow g \prec cp)$.

The above process is called *contraction process*. A *step* of the contraction process corresponds to a distinct structural similarity, for example, a distinct common prefix length l—where l lies between 1 and the largest length of the common prefix between any two

labels either belonging to one node or belonging to two adjacent nodes. We start from the maximum of l, decrease 1 at a time, and terminate when $l = 0$.

6.1.2 Proof of Stability of Non-mergeability

For simplicity of proof, we assume each node has only one label. Let's first explain more notations that we are going to use. For each node u, we introduce a new set of strings (labels), denoted by $u.contained$; we abuse terminology slightly, and say "u contains α" if $\alpha \in u.contained$. Initially $u.contained = \{lab.u\}$ for each node u. The following expression:

$$\alpha \preceq \beta \preceq \gamma$$

is shorthand for $\alpha \preceq \beta \wedge \beta \preceq \gamma$.

We only perform the following kind of transformations on the graph: each transformation merges two neighboring nodes into a single node, and removes the edge between them from the graph; if two neighbors u and v (we denote this situation by u-v) are merged, we denote the resulting merged node by $\langle uv \rangle$, to indicate that the graph structure apart from the edge u-v is unchanged; in particular, other edges terminating on u or v are moved to $\langle uv \rangle$.

More specifically, the above transformation—merging two neighboring nodes u and v—consists of the following steps:

- adding a new node denoted by $\langle uv \rangle$ with $lab.\langle uv \rangle = lab.u \downarrow lab.v$ and $\langle uv \rangle.contained = u.contained \cup v.contained$;
- adding edges between $\langle uv \rangle$ and each of the neighbors of u and v that is not v or u;
- removing edges terminating on u;
- removing edges terminating on v.

There are three invariants that the above transformation must obey in order to preserve locality:

I0 $\forall \alpha \in u.contained \rightarrow lab.u \preceq \alpha$. (A node's label is a prefix of every label it contains.)

I1 $\exists \alpha : \alpha \in u.contained : lab.u \preceq \alpha$. A node contains at least one label that matches its label.

I2 $lab.x \preceq lab.y \preceq \alpha \rightarrow x = y \vee \alpha \notin x.contained$. This is the *locality property*; it states that there are no misleading matches in the graph.

First we show that Invariant I1 is guaranteed to be satisfied. Since, for each node u, we initialize $u.contained$ to be $\{lab.u\}$, we can see that Invariant I1 is guaranteed to be satisfied.

Next we show that Invariant I0 is also guaranteed to be satisfied. Initially $u.contained = \{lab.u\}$ for each node u. Thus Invariant I0 holds. Assume, after performing the above transformation $k-1$ times, Invariant I0 holds. We perform the above transformation the kth time—say merging two neighboring nodes u and v into the new node $\langle uv\rangle$—and have the following:

0	$lab.\langle uv\rangle \preceq lab.u$	Hypothesis
1	$lab.\langle uv\rangle \preceq lab.v$	Hypothesis
2	$\langle uv\rangle.contained = u.contained \cup v.contained$	Hypothesis
3	$\forall \alpha \in u.contained \rightarrow lab.u \preceq \alpha$	Hypothesis
4	$\forall \beta \in v.contained \rightarrow lab.v \preceq \beta$	Hypothesis
5	$\forall \alpha \in \langle uv\rangle.contained \rightarrow lab.\langle uv\rangle \preceq \alpha$	1, 2, 3, 4, 5

By using mathematical induction, we show that Invariant I0 holds no matter how many the above transformations are performed.

Contrary to Invariant I0 and Invariant I1, Invariant I2 does not always hold, and special care must be taken to ensure the locality property is satisfied after the above transformation is performed. There are two ways to violate Invariant I2, when we try to merge two neighboring nodes u and v.

The first way to violate Invariant I2—which we call "Type 1" violation—occurs when there exists a third node $w \neq u, v$ and a label $\alpha \in u.contained \cup v.contained$, such that $lab.u \downarrow lab.v \preceq lab.w \preceq \alpha$. We call such w and α the witnesses to "Type 1" violation.

The second way to violate Invariant I2—which we call "Type 2" violation—occurs when there exists a third node $x \neq u, v$ and a label $\beta \in x.contained$ with $lab.u \not\preceq \beta$ and $lab.v \not\preceq \beta$, such that $lab.x \preceq lab.u \downarrow lab.v \preceq \beta$. We call such x and β the witnesses to "Type 2" violation.

We say an edge is *mergeable* if the above merging transformation does not violate any invariant; we say an edge is *unmergeable* otherwise.

Theorem 6.1.1. (stability of non-mergeability) *An edge that is unmergeable does not become mergeable as a result of the merging of some other edge.*

Proof. We first consider two neighboring nodes u and v cannot be merged due to "Type 1" violation. Then we know that there exists a third node $w \neq u, v$ and a label $\alpha \in u.contained \cup v.contained$, such that $lab.u \downarrow lab.v \preceq lab.w \preceq \alpha$. If there exists another node $x \neq u, v, w$ that is adjacent to w and satisfies $lab.w \downarrow lab.x \prec lab.u \downarrow lab.v$, then, after merging

w and x (assuming no violations are incurred by this merging), $lab.w$ is replace with $lab.\langle wx\rangle$ and $lab.u \downarrow lab.v \preceq lab.\langle wx\rangle \preceq \alpha$ does not hold any more. However we show that by doing so, a "Type 2" violation is incurred.

0	$\exists\sigma : \sigma \in w.contained : lab.w \preceq \sigma$	I1
1	$lab.w \downarrow lab.x \prec lab.u \downarrow lab.v$	Hypothesis
2	$lab.u \downarrow lab.v \preceq lab.w$	Hypothesis
3	$lab.u \downarrow lab.v \preceq \sigma$	0, 2 Transitivity of $\preceq$
4	$lab.w \downarrow lab.x \prec lab.u \downarrow lab.v \preceq \sigma$	1, 3

Thus, after merging w-x, merging u-v is still unmergeable.

We move on to consider two neighboring nodes u and v cannot be merged due to "Type 2" violation. Then we know that there exists a third node $x \neq u, v$ and a label $\beta \in x.contained$ with $lab.u \not\preceq \beta$ and $lab.v \not\preceq \beta$, such that $lab.x \preceq lab.u \downarrow lab.v \preceq \beta$. If there exists another node $y \neq u, v, x$ that is adjacent u (or v) and satisfies $lab.u \downarrow lab.y \prec lab.x$ (or $lab.v \downarrow lab.y \prec lab.x$), then, after merging u and y (or merging v and y), $lab.u$ is replace with $lab.\langle uy\rangle$ (or $lab.v$ is replace with $lab.\langle vy\rangle$) and $lab.x \preceq lab.\langle uy\rangle \downarrow lab.v \preceq \beta$ (or $lab.x \preceq lab.u \downarrow lab.\langle vy\rangle \preceq \beta$) does not hold any more. However we show that by doing so, a "Type 1" violation is incurred. For simplicity of discussions, we assume y is adjacent to u.

0	$lab.u \not\preceq \beta$	Hypothesis
1	$lab.v \not\preceq \beta$	Hypothesis
2	$lab.x \preceq lab.u \downarrow lab.v \preceq \beta$	Hypothesis
3	$\exists\delta : \delta \in u.contained : lab.u \preceq \delta$	I1
4	$lab.u \downarrow lab.y \preceq \delta$	$lab.u \downarrow lab.y \preceq lab.u$, 3, Transitivity
5	$lab.x \preceq \delta$	2, 4, Transitivity
6	$lab.\langle uy\rangle \downarrow lab.v \prec lab.x \preceq \delta$	$\delta \in \langle uy\rangle.contained$, Hypothesis, 5

Thus, after merging u-y, merging u-v is still unmergeable.

Before we end this section, we show that the item I1 and item I2 together imply the uniqueness of labels. To see this, suppose $x \neq y$ and $lab.x = lab.y$. By I1, $\exists\alpha : \alpha \in x.contained : lab.x \preceq \alpha$. Since $lab.x = lab.y$, we have $lab.y \preceq \alpha$. Combined above, $lab.x = lab.y \preceq \alpha$, which violates I2. Thus preservation of the invariants implies each node's label is unique.

6.1.3 Justification of Our Choice of Valid Candidate Clusters

Suppose cp is the common prefix of all the labels belonging to a given cluster. By removing the assumption that each node has only one label, we generalize the concept of a *valid* cluster as: if, for any node u with none of u's labels belonging to a given cluster, each of u's labels does not simultaneously satisfy (1) it is the extension of cp; and (2) it is the prefix of some label belonging to the cluster. To be specific a cluster is valid if $\forall u : u \in V(G) \wedge (\forall \langle v, g\rangle : \langle v, g\rangle \in cluster : u \neq v) : (\forall h : h \in lab.u : (\nexists \langle v, g\rangle : \langle v, g\rangle \in cluster : cp \preceq h \preceq g))$ [2]. Or put it another way, a valid cluster is a cluster that, after being merged into one node, does not incur any misleading match or does not violate Invariant I2.

Let's illustrate it with a concrete example. In Figure 6.4, a straight line represents an edge in a graph, while a curved line means there may be more than one node in between the endpoints of the curve. Suppose we have found a candidate cluster consisting of u and v

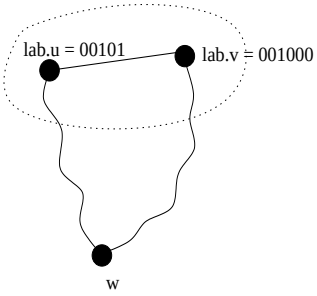


Figure 6.4: Example for Justification of Our Choice of Valid Candidate Clusters

that can be labeled with 0010. Suppose there exists a node $w \neq u, v$ with $lab.w = \{00100\}$. If we were to merge u and v into a new node u' with its label 0010, we would conclude that all the labels that are extensions of 00100 are clustered around w and all the labels that are extensions of 0010 but not extensions of 00100 are clustered around u'. Locality, however, is distorted because of the existence of $lab.v = \{001000\}$. Therefore we say that this candidate cluster consisting of u and v is not valid for merging.

Suppose there exists a node $w \neq u, v$ with $lab.w = \{00111\}$. If we were to merge u and v into a new node u' with its label 0010, we would conclude that all the labels that are extensions of 00111 are clustered around w and all the labels that are extensions of 0010 are clustered around u'. We see that locality is preserved after this merging. We still need to check each node in $V \setminus \{u, v, w\}$ before we can safely say that this candidate cluster consisting of u and v is valid for merging.

[2] For any node u with some of u's labels belonging to a given cluster, the remaining labels of u cannot satisfy the these two conditions; otherwise they would be included in the cluster.

6.1.4 Algorithm for Our Contraction Process

Given a common prefix, we list the algorithms to compute a cluster by using the above heuristics. Here we assume there is ≥ 1 label per node. *getNumOfLbls*(U) denotes a function to get the number of the labels of the nodes in U, assuming there are no redundant labels in $L(U)$. *getSumOfLensOfLbls*(U) denotes a function to get the sum of the lengths of the labels of the nodes in U, assuming there are no redundant labels in $L(U)$.

Algorithm 7 *getMaxCpLenOfAdjNodes*(G): get the maximum among the lengths of the common prefixes of labels within one node or from adjacent nodes in G

Input: $G = (V, E)$
Output: the maximal length of the common prefixes of labels within one node or of adjacent nodes in G

```
1: maxCpLenOfAdjNodes = 0
2: for u ∈ V(G) do
3:    for g ∈ lab.u do
4:       for h ∈ lab.u \ {g} do
5:          if len.(g ↓ h) > maxCpLenOfAdjNodes then
6:             maxCpLenOfAdjNodes = len.(g ↓ h)
7:          end if
8:       end for
9:    end for
10: end for
11: for e = (u, v) ∈ E(G) do
12:    for g ∈ lab.u do
13:       for h ∈ lab.v do
14:          if len.(g ↓ h) > maxCpLenOfAdjNodes then
15:             maxCpLenOfAdjNodes = len.(g ↓ h)
16:          end if
17:       end for
18:    end for
19: end for
20: return maxCpLenOfAdjNodes
```

Algorithm 8 *getCpsOfAdjNodesOfGivenLen*(G, l): get all the common prefixes of labels within one node or from adjacent nodes in G of the specified length l

Input: $G = (V, E)$, the label length l

Output: the set of the common prefixes of labels within one node or of adjacent nodes in G of the specified length l

```
1: initialize the set cpsOfAdjNodesOfLen = {}
2: for u ∈ V(G) do
3:    for g ∈ lab.u do
4:       for h ∈ lab.u \ {g} do
5:          if len.(g ↓ h) = l then
6:             cpsOfAdjNodesOfLen ← g ↓ h
7:          end if
8:       end for
9:    end for
10: end for
11: for e = (u, v) ∈ E(G) do
12:    for g ∈ lab.u do
13:       for h ∈ lab.v do
14:          if len.(g ↓ h) = l then
15:             cpsOfAdjNodesOfLen ← g ↓ h
16:          end if
17:       end for
18:    end for
19: end for
20: return cpsOfAdjNodesOfLen
```

Algorithm 9 *getMaxClstr*$(u, g, candCp, procedVtxs, G)$: get a cluster *cluster*— its nodes forms a connected subgraph; the common prefix of its labels is equal to *candCp*; $\forall u, v : u \in nodeSet(cluster) \wedge v \in N(u) \setminus nodeSet(cluster) : (\forall g : g \in lab.v : cp \downarrow g \prec cp)$; $\forall u : u \in nodeSet(cluster) : (\forall g : g \in lab.u \setminus lblSet(cluster) : cp \downarrow g \prec cp)$.

Input: node u, its label g starting with *candCp*, the node set *procedVtxs* each of which has been searched for labels starting with *candCp* and $G = (V, E)$ for finding neighbors

Output: the maximal set of node-label pairs with its nodes forming a connected subgraph and with the common prefix of its labels equal to *candCp*

1: $cluster \leftarrow \langle u, g \rangle$, $q \leftarrow \langle u, g \rangle$
2: **while** $q.isEmpty()$ =false **do**
3: $p = q.removeFirst()$
4: $v = p.getFirst()$
5: $h = p.getSecond()$
6: **if** $u = v$ and $g = h$ **then**
7: **for** $h \in lab.u$ **do**
8: **if** $h = g$ **then**
9: continue
10: **else if** $h.startWith(candCp) =$ **then**
11: $cluster \leftarrow \langle u, h \rangle$, $q \leftarrow \langle u, h \rangle$
12: **end if**
13: **end for**
14: $procedVtxs \leftarrow u$
15: **end if**
16: **for** $w \in N(v)$ **do**
17: **if** $procedVtxs.contains(w)$ =false **then**
18: **for** $h \in lab.w$ **do**
19: **if** $h.startWith(candCp)$ =true **then**
20: $cluster \leftarrow \langle w, h \rangle$, $q \leftarrow \langle w, h \rangle$
21: **end if**
22: **end for**
23: $procedVtxs \leftarrow w$
24: **end if**
25: **end for**
26: **end while**
27: $lblSetOfClstr = \{\}$
28: **for** $\langle v, g \rangle \in cluster$ **do**
29: $lblSetOfClstr \leftarrow g$
30: **end for**
31: **if** $|lblSetOfClstr| == 1$ **then**
32: return null
33: **end if**
34: return *cluster*

Algorithm 10 *isValidClstr*$(cluster, candCp, U)$: we say the cluster is valid for merging if it satisfies $\forall u : u \in U \wedge (\forall \langle v, g\rangle : \langle v, g\rangle \in cluster : u \neq v) : (\forall h : h \in lab.u : (\nexists \langle v, g\rangle : \langle v, g\rangle \in cluster : candCp \preceq h \preceq g))$

Input: the candidate cluster *cluster*, the label *candCp* and the node set U

Output: return true if the cluster is valid for merging; return false otherwise.

```
1:  nodeSetOfClstr = ∅
2:  for ⟨u, g⟩ ∈ cluster do
3:     nodeSetOfClstr ← u
4:  end for
5:  for v ∈ U \ nodeSetOfClstr do
6:     for h ∈ lab.v do
7:        if h = candCp then
8:           return false
9:        else if h.startWith(candCp) then
10:          for ⟨w, g⟩ ∈ cluster do
11:             if g.startWith(h) then
12:                return false
13:             end if
14:          end for
15:       end if
16:    end for
17: end for
18: return true
```

Algorithm 11 *findClstrOfMinBits*(*candClstrs*, *candCp*): find the cluster among *candClstrs* with the minimal sum of lengths of labels of it

Input: the set of clusters *candClstrs* and the common prefix of each cluster *candCp*
Output: find the cluster among *candClstrs* with the minimal sum of lengths of labels of it

```
1:  minLenSumOfClstr = 0
2:  for cluster ∈ candClstrs do
3:      lblSetOfClstr = ∅
4:      for ⟨u, g⟩ ∈ cluster do
5:          lblSetOfClstr ← g
6:      end for
7:      lenSumOfClstr = 0
8:      for g ∈ lblSetOfClstr do
9:          lenSumOfClstr = lenSumOfClstr + len.g
10:     end for
11:     if lenSumOfClstr < minLenSumOfClstr then
12:         minLenSumOfClstr = lenSumOfClstr
13:     end if
14: end for
15: clstrsOfMinLenSum = ∅
16: for cluster ∈ candClstrs do
17:     lblSetOfClstr = ∅
18:     for ⟨u, g⟩ ∈ cluster do
19:         lblSetOfClstr ← g
20:     end for
21:     lenSumOfClstr = 0
22:     for g ∈ lblSetOfClstr do
23:         lenSumOfClstr = lenSumOfClstr + len.g
24:     end for
25:     if lenSumOfClstr == minLenSumOfClstr then
26:         clstrsOfMinLenSum ← cluster
27:     end if
28: end for
29: return clstrsOfMinLenSum
```

Algorithm 12 *findMaxNumOfMatchedLbls*$(candClstrs, G)$: find the maximal number of distinct labels; these labels share nonempty common prefix with a cluster's label set, and belong to the label sets of the nodes, either adjacent to or is equal to, some node in the cluster's node set.

Input: a set of clusters *candClstrs* and input graph G

Output: the maximal number of labels sharing nonempty common prefix with some cluster's label set and those labels not belonging to the cluster's label set but belonging to the nodes either adjacent to or within the cluster's node set

```
1:  maxNumOfMatchedLbls = 0
2:  for cluster ∈ candClstrs do
3:    nodeSetOfClstr = ∅, lblSetOfClstr = ∅,
4:    locProcedVtxs = ∅, matchedLblSet = ∅
5:    for ⟨v, g⟩ ∈ cluster do
6:      nodeSetOfClstr ← v, lblSetOfClstr ← g
7:    end for
8:    for v ∈ nodeSetOfClstr do
9:      for w ∈ N(v) ∪ {v} do
10:       if locProcedVtxs.contains(w) then
11:         continue
12:       end if
13:       for g ∈ lab.w do
14:         if nodeSetOfClstr.contains(w) ∧ lblSetOfClstr.contains(g) then
15:           continue
16:         else
17:           if matchedLblSet.contains(g) then
18:             continue
19:           end if
20:           pick h in lblSetOfClstr
21:           if len.(g ↓ h) > 0 then
22:             matchedLblSet ← g
23:           end if
24:         end if
25:       end for
26:     end for
27:   end for
28:   if maxNumOfMatchedLbls < |matchedLblSet| then
29:     maxNumOfMatchedLbls = |matchedLblSet|
30:   end if
31: end for
32: return maxNumOfMatchedLbls
```

Algorithm 13 *findClstrsOfMaxMatchedLbls*$(candClstrs, G)$: find the set of clusters— each of which has the maximal number of labels; these labels share nonempty common prefix with a cluster's label set, and belong to the label sets of the nodes, either adjacent to or is equal to, some node in the cluster's node set.

Input: a set of clusters *candClstrs* and input graph G

Output: find the set of clusters, each of which has the maximal number of labels sharing nonempty common prefix with its label set and those labels not belonging to its label set but belonging to the nodes either adjacent to or within its node set

```
1: maxNumOfMatchedLbls = findMaxNumOfMatchedLbls(candClstrs, G)
2: for cluster ∈ candClstrs do
3:    nodeSetOfClstr = ∅, lblSetOfClstr = ∅,
4:    locProcedVtxs = ∅, matchedLblSet = ∅
5:    for ⟨v, g⟩ ∈ cluster do
6:       nodeSetOfClstr ← v, lblSetOfClstr ← g
7:    end for
8:    for v ∈ nodeSetOfClstr do
9:       for w ∈ N(v) ∪ {v} do
10:         if locProcedVtxs.contains(w) then
11:            continue
12:         end if
13:         for g ∈ lab.w do
14:            if nodeSetOfClstr.contains(w) ∧ lblSetOfClstr.contains(g) then
15:               continue
16:            else
17:               if matchedLblSet.contains(g) then
18:                  continue
19:               end if
20:               pick h in lblSetOfClstr
21:               if len.(g ↓ h) > 0 then
22:                  matchedLblSet ← g
23:               end if
24:            end if
25:         end for
26:      end for
27:   end for
28:   if maxNumOfMatchedLbls == |matchedLblSet| then
29:      clstrsOfMaxMatchedLbls ← cluster
30:   end if
31: end for
32: return clstrsOfMaxMatchedLbls
```

Algorithm 14 *getClstrOfSmallestConcatLbl*(*candClstrs*): first sort labels in each cluster and concatenate them. Then we sort these concatenated labels and return the cluster corresponding to the first concatenated label.

Input: a set of clusters *candClstrs*

Output: first sort labels in each cluster and concatenate them. Then we sort these concatenated labels and return the cluster corresponding to the first concatenated label

1: $clstrConcatLblToClstrIdx = \emptyset$, $concatLblOfClstrs = \emptyset$
2: **for** $i = 0$ to $|candClstrs| - 1$ **do**
3: $cluster = candClstrs[i]$
4: $lblSetOfClstr = \emptyset$
5: **for** $\langle v, g \rangle \in cluster$ **do**
6: $lblSetOfClstr \leftarrow g$
7: **end for**
8: sort labels in *lblSetOfClstr* ascendingly
9: concatenate the sorted labels one by one into *concatLbl*
10: $concatLblOfClstrs \leftarrow concatLbl$
11: $clstrConcatLblToClstrIdx \leftarrow \langle concatLbl, i \rangle$
12: **end for**
13: sort labels in *concatLblOfClstrs*
14: $j = clstrConcatLblToClstrIdx.get(concatLblOfClstrs[0])$
15: return $candClstrs[j]$

Algorithm 15 *findFinalClstrsOfCps*($candCpSet, G$): for each *candCp* in *candCpSet*, find one maximal set of node-label pairs satisfying (1) its nodes form a connected subgraph; (2) the common prefix of its labels is equal to *candCp*; (3) it is valid; (4) the sum of the lengths of its labels is minimal; (5) it has the maximal number of labels sharing nonempty common prefix with some cluster's label set and belonging to the nodes either adjacent to or is equal to some node in the cluster's node set; (6) it has the smallest concatenated label. Each *candCp* in *candCpSet* is distinct and of the same length.

Input: the set of labels *candCpSet* and $G = (V, E)$

Output: the set of the maximal sets of node-label pairs each of which satisfies that (1) its nodes form a connected subgraph; (2) the common prefix of its labels is equal to *candCp*; (3) it is valid; (4) the sum of the lengths of its labels is minimal; (5) it has the maximal number of labels sharing nonempty common prefix with some cluster's label set and belonging to the nodes either adjacent to or is equal to some node in the cluster's node set; (6) it has the smallest concatenated label.

```
1: for candCp ∈ candCpSet do
2:   candClstrs = {}, procedVtxs = {}
3:   for u ∈ V(G) do
4:     if procedVtxs.contains(u) then
5:       continue
6:     end if
7:     for g ∈ lab.u do
8:       if g.startWith(candCp) then
9:         cluster = getMaxClstr(u, g, candCp, procedVtxs, G)
10:        if |cluster| > 1 then
11:          valid = isValidClstr(cluster, candCp, V(G))
12:          if valid then
13:            candClstrs ← cluster
14:          end if
15:        end if
16:        break
17:      end if
18:    end for
19:  end for
20:  tmpClstrs = findClstrOfMinBits(candClstrs, candCp)
21:  if |tmpClstrs| > 1 then
22:    oldTmpClstrs = tmpClstrs
23:    tmpClstrs = findClstrsOfMaxMatchedLbls(candClstrs, candCp)
24:    if |tmpClstrs| == 0 then
25:      tmpClstrs = oldTmpClstrs
26:    end if
27:  end if
28:  finalClstrs ← getClstrOfSmallestConcatLbl(tmpClstrs)
29: end for
30: return finalClstrs
```

6.2 Using Last k Steps of Contraction Process

Our first metric to capture the hierarchical clustering we have built using the above heuristics is to use the last k steps of our contraction process, or put it another way, the top k levels of our clustering structure.

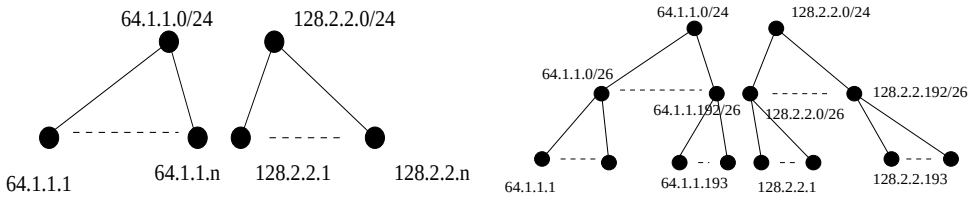
6.2.1 Justification of Using History Information

In this subsection, we try to argue that purely using the number of labels and the number of bits at the end of the contraction process is not sufficient to measure locality; introducing some *history* information related to the contraction process (i.e. the number of labels and the number of bits at each of the last k steps of the contraction process where $k > 1$) make our metric more accurate in capturing locality.

We will give one example to support the above argument. Consider two connected networks A and B, each of which has n hosts. Network A is assigned a class C IPv4 address block 64.1.1.0/24, and network B is assigned a class C IPv4 address block 128.2.2.0/24. We can see that the address space covered by 64.1.1.0/24 is disjoint from the address space covered by 128.2.2.0/24. In the first labeling scheme, no subnetting is used in network A nor in network B; IPv4 addresses from 64.1.1.0/24 are assigned randomly to each host in network A; IPv4 addresses from 128.2.2.0/24 are assigned randomly to each host in network B. In the second labeling scheme, hosts are grouped into 4 subnets in network A; hosts are grouped into 4 subnets in network B; each subnet in network A is assigned a distinct split prefix from 64.1.1.0/24; each subnet in network B is assigned a distinct split prefix from 128.2.2.0/24; hosts in each subnet obtain IPv4 addresses from the subnet's prefix.

To measure the locality of these two address schemes using our abstract metric, a hierarchical clustering structure is built using Algorithm 15 for each of the two labelings. The hierarchical clustering structure for the labeling with no subnetting is shown in Figure 6.5(a), and the hierarchical clustering structure for the labeling with subnetting is shown in Figure 6.5(b).

As mentioned in Section 3.6, the introduction of subnetting has greatly reduced the routing state kept at each host. However if we were to use the number of labels and the number of bits at the end of our contraction process, we would make a wrong conclusion that these two address schemes have the same locality. Only by introducing some history information, can our metric give the correct measurement of the locality of these two different address schemes: if the number of labels and the number of bits at the last two steps of our contraction process is used instead, we make the correct conclusion that the address scheme



(a) no subnetting is used (b) subnetting is used

Figure 6.5: Hierarchical clustering structure built using Algorithm 15 for each of the labelings

for network B is better than that for network A.

6.2.2 Normalization across Graphs of Different Sizes

The minimal number of labels left at each step of our contraction process is 2 (with one label starting with 0 and with the other label starting with 1[3]), and the maximal number of labels left at each step is $\sum_{u \in V(G)} |lab.u|$; the minimal number of bits left at each step of our contraction process is 2, and the maximal number of bits left at each step is $\sum_{u \in V(G)} \sum_{g \in lab.u} len.g$. Given two distinct labeled graphs, the minimal number of labels/bits at each step of our contraction is the same, while the maximal number of labels/bits at each step of our contraction is not. The main idea of our normalization method is to find a bijective mapping f such that, given two distinct integer ranges $[l_1, r_1]$ and $[l_2, r_2]$, $f(l_1) = l_2$ and $f(r_1) = r_2$.

Let $numOfLbl(G, 1, k)$ denote the sum of the number of labels left from the last step to the kth step from the end; let $numOfBits(G, 1, k)$ denote the sum of the number of bits left from the last step to the kth step from the end. In particular, $numOfLbl(G, i, i)$ denotes the number of labels left at the ith step from the end, and $numOfBits(G, i, i)$ denotes the number of bits left at the ith step from the end. Our normalization formulas are shown in Equation 6.1.

To explain Equation 6.1 works, consider two distinct labeled graphs—say G and G'. We can either normalize $numOfLbl(G, 1, k)$ according to Equation 6.1, or normalize $numOfLbl(G', 1, k)$ after we exchange G and G' in Equation 6.1. Suppose we normalize $numOfLbl(G, 1, k)$, and, after that, we simply compare it with $numOfLbl(G', 1, k)$: the smaller the better the locality.

If they are equal, we move on to either normalize $numOfBits(G, 1, k)$ according to Equation 6.2, or normalize $numOfBits(G', 1, k)$ after we exchange G and G' in Equation 6.2. Suppose we normalize $numOfBits(G, 1, k)$, and, after that, we simply compare it with

[3]If all the labels in $L(V(G))$ share some nonempty prefix, we remove this prefix from each label as a preprocessing step.

$$numOfLbl(G,1,k) = \sum_{1\le i\le k}((numOfLbl(G,i,i)-2)\times\frac{|\sum_{u\in V(G')}|lab.u|-2|}{|\sum_{u\in V(G)}|lab.u|-2|}+2)$$
$$\approx \sum_{1\le i\le k} numOfLbl(G,i,i)\times\frac{|\sum_{u\in V(G')}|lab.u||}{|\sum_{u\in V(G)}|lab.u||} \quad (6.1)$$

$$numOfBits(G,1,k) = \sum_{1\le i\le k}((numOfBits(G,i,i)-2)\times\frac{|\sum_{u\in V(G')}\sum_{g\in lab.u}|len.g|-2|}{|\sum_{u\in V(G)}\sum_{g\in lab.u}|len.g|-2|}+2)$$
$$\approx \sum_{1\le i\le k} numOfBits(G,i,i)\times\frac{|\sum_{u\in V(G')}\sum_{g\in lab.u}|len.g||}{|\sum_{u\in V(G)}\sum_{g\in lab.u}|len.g||} \quad (6.2)$$

$numOfBits(G',1,k)$: the smaller the less the redundancy.

If they are equal, we conclude the two labelings have the same locality. How precise this conclusion is depends, to some extent, on how many history information we uses.

6.2.3 Algorithm

Since not all labels of a node belongs to the same cluster for a given *cp* we cannot simply merge *nodeSet*(*cluster*) into a new node and label it with *cp*. Instead, as shown in Line 23 of Algorithm 16, we replace the label g of the node u with the label *cp*, and add this new label to the label set of u. Special care, however, must be taken when calculating *sumOfLensOfLbls* and *numOfLbls*, to make sure that we do not count each distinct label more than once.

Algorithm 16 Using Last k Steps of Contraction Process as one abstract locality metric

Input: a labeled AS-level graph $G = (V, E)$

Output: # of labels and # of bits for each step

$maxCpLenOfAdjNodes = getMaxCpLenOfAdjNodes(G)$
$sumOfLensOfLbls = getSumOfLensOfLbls(V(G))$
$numOfLbls = getNumOfLbls(V(G))$
while $maxCpLenOfAdjNodes > 0$ **do**
  $cpsSet =$
  $getCpsOfAdjNodesOfGivenLen(G, maxCpLenOfAdjNodes)$
  if $cpsSet \neq \emptyset$ **then**
    $finalClstrs = findFinalClstrsOfCps(cpsSet, G)$
    for $i = 0$ to $|cpsSet| - 1$ **do**
      $cluster = finalClstrs.get(i)$
      if $cluster =$null **then**
        continue
      end if
      $candCp = cpsSet.get(i)$
      $lblSetOfClstr = \emptyset$
      for $\langle u, g \rangle \in cluster$ **do**
        $lblSetOfClstr \leftarrow g$
      end for
      $sumOfLensOfLbls = sumOfLensOfLbls - \sum_{g \in lblSetOfClstr} len.g$
      $sumOfLensOfLbls = sumOfLensOfLbls + len.candCp$
      $numOfLbls = numOfLbls + |lblSetOfClstr| - 1$
      for $\langle u, g \rangle \in cluster$ **do**
        $lab.u = (lab.u \setminus \{g\}) \cup \{candCp\}$ (instead of merging)
      end for
    end for
  end if
  print $maxCpLenOfAdjNodes$, $sumOfLensOfLbls$ and $numOfLbls$
  $maxCpLenOfAdjNodes - -$
end while

7 | Results for Abstract Locality Metric

The input AS-level graphs used in this section is the same as those used for the BGP-based locality metric. The basic statistic information in terms of the number of nodes (ASs) and the number of edges (eBGP sessions) of our input AS-level graph for each year is the same as shown in Figure 5.1. Here structural similarity between two labels is measured by the length of the common prefix between two labels.

7.1 Validation of the Metric

In this section, we validate the effectiveness of this metric by applying it to a series of locality-decreasing address schemes given an input AS-level graph.

The general idea for creating our locality-decreasing (in high probability) address schemes is as follows. Since the metric is independent of any forwarding/routing mechanism, only one label is assigned to each node in an input AS-level graph. In addition, we put no limit upon the length of a label and labels can be of variable lengths. We still assign to either a single-homed or a multihomed site a label out of a service provider's address space. By assigning a site in this way, topologically closer nodes have structurally more similar addresses (in high probability). We then create other addressing schemes by randomly picking a subset of nodes U of different sizes and randomly reassigning $L(U)$ to each node in U, thus making address assignments partially or completely not service-provider oriented. We expect our locality metric to be smaller if we assign each (single-homed or multihomed) customer's address to be the extension of its direct primary provider's address than after we randomly switch the addresses of some subset of nodes.

We will now discuss the details for constructing these locality-decreasing (in high probability) address schemes. We first create the addressing scheme with each (single-homed or multihomed) customer's address to be the extension of its direct primary provider's address as follows: Suppose the input AS-level graph is connected. If not, we simply use the largest connected component as the input graph instead. Given the input AS-level graph:

- We find the clique (or the core), achieved by using the method proposed in [SARK02b].

- For each node contained in the clique we build a tree rooted at that node. For each customer-provider link the customer AS is the child node and the provider AS is the parent node. If an AS is multihomed to more than one provider, we pick the provider with the largest degree as mentioned before.

- For each node contained in the node set of the clique denoted by clq, we assign a distinct label from the set $\{\underbrace{0...0}_{\lceil \log_2 |V(clq)| \rceil}, ..., \underbrace{1...1}_{\lceil \log_2 |V(clq)| \rceil}\}$.

- For each tree we assign labels from top down. Suppose the current node u is assigned γ. If it does not have a child we do nothing; if it has $x \geq 1$ children, we first change its label to $\gamma 1$ and then assign to each child a distinct label from the set $\{\gamma 0 \underbrace{0...0}_{\lceil \log_2 x \rceil}, ..., \gamma 0 \underbrace{1...1}_{\lceil \log_2 x \rceil}\}$.

In this way we divide the infinite address space matching $(0|1)*$ into disjoint infinite subspaces with each infinite subspace assigned to one node. The (sub)space is unbounded because, as we mentioned before, there is no limit upon the length of a label, and since the (sub)space is unbounded we can meet any amount of address demand of a node. Suppose the address demand of u is 2^8 and $lab.u$ =001. We then assign $\{001\underbrace{0...00}_{8}, 001\underbrace{0...01}_{8}, ..., 001\underbrace{1...11}_{8}\}$ to u.

We create other addressing schemes by randomly picking a subset of nodes U and randomly reassigning a distinct label in $L(U)$ to each node in U, thus making address assignments partially or completely not service-provider oriented. More precisely, for the 10% case we randomly pick 10% of nodes denoted by U, get their labels denoted by $L(U)$, and randomly reassign a unique label in $L(U)$ to each node in U. By doing so we get a labeled graph denoted by $G_{10\%}$. For the 20% case we use $G_{10\%}$ as the input graph instead of the original graph or $G_{0\%}$. Then we randomly pick another 10% of nodes out of those nodes whose labels have not been switched yet instead of $V(G_{0\%})$. We denote this node set by U, get their labels denoted by $L(U)$, and randomly reassign a unique label in $L(U)$ to each node in U. By doing so we get a labeled graph denoted by $G_{20\%}$. We can proceed in this way until we get $G_{50\%}$. After that, for the 60%, 70%, 80%, 90% and 100% case we use $G_{50\%}$ as the input graph. Then we randomly pick another 10%, 20%, 30%, 40%, and 50% of nodes out of those nodes whose labels have not been switched yet instead of $V(G_{0\%})$. We denote the node set again by U, get their labels denoted by $L(U)$, and randomly reassign a unique label in $L(U)$ to each node in U. By doing so we get labeled graphs denoted by $G_{60\%}$, $G_{70\%}$, $G_{80\%}$, $G_{90\%}$ and $G_{100\%}$ (a totally random labeling of the input AS-level graph).

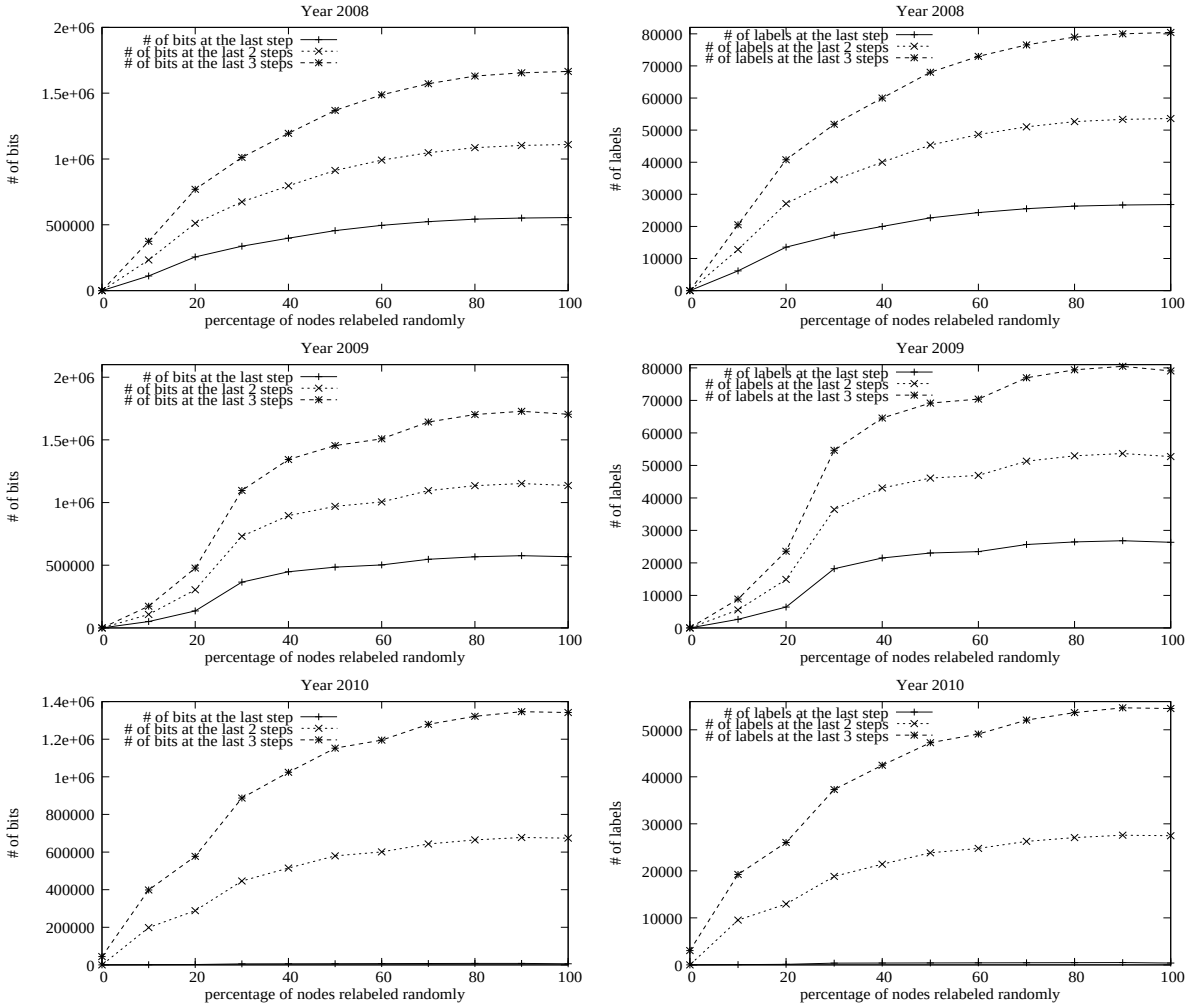


Figure 7.1: # of bits/# of labels of the last 3th steps of the contraction

The experiment results are averaged over 10 runs, each of which corresponds to a distinct random seed for picking a subset of nodes and switching their labels.

As we have expected, the abstract locality metric keeps increasing when more randomness is introduced by different address schemes applied to the same input graph. We argued earlier that purely using the number of labels/number of bits at the end of the contraction process is not sufficient to measure locality, and introducing some history information related to the contraction process (i.e., the number of labels/the number of bits at each of the last k steps where $k > 1$) seems necessary in capturing the hierarchical structure of the labeling. Though, from Figure 7.1, we observe that all lines keep strictly increasing, we still use the sum of the number of labels/the number of bits for the last 3 steps of our contraction process in the following discussions.

7.2 Study on Internet Case

We will now show how efficient the current Internet's addressing scheme is by applying the abstract locality metric. The main results are shown in Figure 7.2 after applying Equation 6.1 and Equation 6.2. In Figure 7.2, for each year, we also calculate the upper bound and lower bound for our abstract locality metric. The upper bound is calculated in the following way: if for the given common prefix there exist only one cluster then we merge it; otherwise, we do nothing. The lower bound is calculated in the following way: if for the given common prefix there exist only one cluster then we merge it; otherwise, we merge all of them. By doing so some nodes may have the same labels in the intermediate or final graphs. Note, in order to visualize the lower bound line, the y-axis is the logarithm of the number of labels and the logarithm of the number of bits respectively.

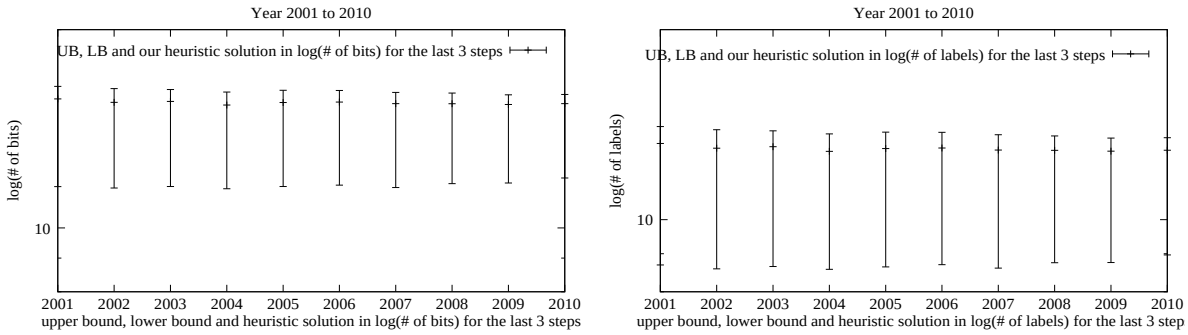


Figure 7.2: # of bits/# of labels of the last 3 steps of the contraction with UB/LB

We see from Figure 7.2 that current Internet address scheme is far from optimal in terms of locality and redundancy (in best cases, our metric is equal to $1 \times 2^1 + 2 \times 2^2 + 3 \times 2^3 = 34$ bits and $2^1 + 2^2 + 2^3 = 14$ labels of the last 3 steps of our contraction process).

The locality of current address scheme gets a little bit better, though with small fluctuations. This is due to the wider deployment of CIDR and the gradual withdrawals of pre-CIDR prefixes. In addition, the practice of IPv4 address allocation and assignment does not change much over time, though there has been a slight change in the allocation policy of both ARIN and RIPE NCC by shrinking the "window" of demonstrated need from 12 months to 3 months. Moreover, with more edges introduced over time, nodes with structurally similar labels become closer to each other.

However, our abstract locality metric is not negatively affected by the increase in multihoming practice as long as a multihomed AS is assigned out of one of its service providers' address space. Other engineering practices, such as load-balancing by fragmenting one single

prefix, also have no negative impact upon our abstract locality metric, though redundancy contained in labels is increased.

7.3 Study on the Compact Routing Case

As mentioned in Section 3.7, name-dependent compact routing scheme assign the “name" of each node from scratch in a topology-aware way, so as to make the routing state (mainly packet header and routing table) logarithmic in the network size and the path stretch bounded by some small constant. The main goal of this section is to study the efficiency of the “naming" mechanisms of name-dependent compact routing schemes in terms of our abstract metric. Since the name-dependent compact routing mechanism proposed by Thorup and Zwick [TZ01]—TZ Stretch-3 CR for short—is the basis of other name-dependent compact routing mechanisms, we are only focused on the study of the “naming" mechanism for the TZ Stretch-3 CR scheme.

Let’s reiterate how the “name" of each node is assigned in the TZ Stretch-3 CR scheme: a set of landmark nodes denoted by A of size at most $2s \log n$ (where s is parameter) is randomly selected in an input graph $G = (V, E)$; for each $w \in V$, define $cent(w)$ to be the node in A that is closest to w than to any other node in A; for each $w \in V$, define $C_A(w)$ to be the set of nodes each of which is closer to w than to its center in A, and $|C_A(w)| \leq \frac{4|V|}{s}$; each node $w \in V$ is assigned a label $label(w) = (w, cent(w), port(cent(w), w))$, where $port(cent(w), w)$ gives the port number used by $cent(w)$ to route the packet towards w.

We change the “names" of each node w into $label(w) = (cent(w), port(cent(w), w), w)$ in order to keep using the length of the common prefix as the measurement of the similarity between any two “names". We let $s = (n \log n)^{\frac{1}{2}}$ and according to Corollary 3.3 in [TZ01] we have $|A| \leq 2(n \log n)^{\frac{1}{2}}$ and $|C_A(w)| \leq 4(n \log n)^{\frac{1}{2}}$ for each $w \in V$. After we label each node in this manner, we apply our abstract locality metric to this labeled graph. We then compare the computed value of our metric with that of the Internet address scheme. Here we are only focused upon year 2010. The results are shown in Table 7.1.

Table 7.1: Internet addressing vs addressing of TZ Stretch-3 CR

Address Scheme	For TZ Stretch-3 CR	For the current Internet
# of labels for the last 3 steps	1.17×10^4	2.24×10^5
# of bits the last 3 steps	4.20×10^4	4.72×10^6

As expected, the locality of the address scheme of the TZ stretch-3 CR is much better

than that of the Internet address scheme in terms of our abstract locality metric. It is not surprising since “names" used by the TZ stretch-3 CR are assigned in the topology-aware manner. They do not have the issues like legacy class C networks that make the Internet less efficient.

7.4 Comparison with BGP-based Locality Metric

Our BGP-based locality metric tells us that the locality of the Internet address scheme keeps worsening till year 2007, and then becomes a little bit better. Our abstract locality metric also tells us that the locality of the Internet address scheme gets a little bit better over the years.

8 | Conclusion and Future Work

This thesis is a first step toward quantitative assessment of the locality of address assignment schemes applied to AS-level graphs of different sizes. We defined a cost measure, *TCost*, in terms of the control-plane overhead required to advertize prefixes using BGP. Our metric is designed as a proxy for the benefits of locality of addressing in both the control and data plane. To investigate and quantify the degree of optimality of current prefix assignments, we developed several methods of reassigning prefixes while preserving semantics. We are able to reduce *TCost*/link by 25% to 210%, depending on the strictness of the definition of semantic equivalence. Though by our prefix reassignment we can save some bits, we still cannot achieve logarithmic scaling that allows for "infinite scaling." Thus hierarchical addressing only helps us in a limited way. Finally, we propose another metrics that in some sense quantifies the efficiency of the labeling and is independent of forwarding/routing mechanisms. We validate the effectiveness of the metric by applying it to a series of locality-decreasing address schemes with variable-length labels assigned from scratch given the same input AS-level graph. After that we apply the metric to the current Internet address scheme across years. We find that the current Internet address scheme is far from optimal in terms of locality and redundancy. Moreover, we find that the normalized number of labels and the normalized number of bits for the last 3 steps of our contraction process get a little bit better over the years. We also study the locality of the address scheme for the TZ stretch-3 compact routing mechanism, and find that, as we expected, its locality is far better than that of the Internet.

In future work, we hope to extend our study to examine the locality of other address schemes (for example, the IPv6 address scheme). Though currently less than 1% of the Internet users prefer IPv6, a four-fold increase in the deployment of IPv6 in the Internet is witnessed from August 2011 to August 2012 [Hus12]. Some work [kc11] has been done to survey available data that are capable of limited tracking of IPv6 deployment and additional candidates of data that would support better tracking. We are also interested in how IPv4 address transfer markets will affect the locality of the current Internet address scheme.

Bibliography

[Ach03] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[AGM+08] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. *ACM Transactions on Algorithms*, 4(3), 2008.

[AKP91] Baruch Awerbuch, Shay Kutten, and David Peleg. On buffer-economical store-and-forward deadlock prevention. In *INFOCOM*, pages 410–414, 1991.

[ALD+05] J. Abley, K. Lindqvist, E. Davies, B. Black, and V. Gill. IPv4 Multihoming Practices and Limitations. RFC 4116 (Informational), July 2005.

[And07] James W. Anderson. *Hyperbolic Geometry*. Springer Undergraduate Mathematics Series. Springer, second edition, 2007.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *In IEEE Symposium on Foundations of Computer Science*, pages 503–513. IEEE, 1990.

[ARK] ARK. Archipelago measurement infrastructure. Available from: `http://www.caida.org/projects/ark/`.

[ASS03] Aditya Akella, Srinivasan Seshan, and Anees Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *Internet Measurement Comference*, pages 101–114. ACM, 2003.

[AVG+99] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). RFC 2622 (Proposed Standard), June 1999. Updated by RFC 4012.

[BC06] Arthur Brady and Lenore Cowen. Compact routing on power-law graphs with additive stretch. In *ALENEX*, 2006.

[BCC06] T. Bates, E. Chen, and R. Chandra. BGP Route Reflection - An Alternative to Full Mesh IBGP. RFC 4456, Apr. 2006. Obsoletes: 2796, 1966.

[BDPR05] L. Blunk, J. Damas, F. Parent, and A. Robachevsky. Routing Policy Specification Language next generation (RPSLng). RFC 4012 (Proposed Standard), March 2005.

[BFCW08] Hitesh Ballani, Paul Francis, Tuan Cao, and Jia Wang. Viaggre: Making routers last longer! In *7th ACM Workshop on Hot Topics in Networks HotNets*, 2008.

[BFCW09] Hitesh Ballani, Paul Francis, Tuan Cao, and Jia Wang. Making routers last longer with viaggre. In *USENIX NSDI*, 2009.

[BGT04] Tian Bu, Lixin Gao, and Donald F. Towsley. On characterizing bgp routing table growth. *Computer Networks*, 45(1):45–54, 2004.

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is np-hard. *Comput. Geom.*, 9(1-2):3–24, 1998.

[BK09] Marián Boguñá and D. Krioukov. Navigating ultrasmall worlds in ultrashort time. *Physical Review Letters*, 102(058701), 2009.

[BKC09] Marián Boguñá, D. Krioukov, and K. C. Claffy. Navigability of complex networks. *Nature Physics*, 5(1):74–80, 2009.

[BKL11] Larry Blunk, Manish Karir, and Craig Labovitz. Multi-Threaded Routing Toolkit (MRT) Routing Information Export Format. RFC 6396 (Proposed Standard), October 2011.

[BPK10] Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri V. Krioukov. Sustaining the internet with hyperbolic mapping. *CoRR*, abs/1009.0267, 2010.

[CC09] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM*, pages 1647–1655. IEEE, 2009.

[CC12] Andrej Cvetkovski and Mark Crovella. Low-stretch greedy embedding heuristics. In *INFOCOM Workshops*, pages 232–237. IEEE, 2012.

[CCK+06] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, and Ion Stoica. Rofl: routing on flat labels. *SIGCOMM' 06*, 2006.

[CCS96] I. Castineyra, N. Chiappa, and M. Steenstrup. The nimrod routing architecture. RFC 1992, August 1996.

[CDZ97] Kenneth Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[CGJ+04] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott Shenker, and Walter Willinger. Towards capturing representative as-level internet topologies. *Computer Networks*, 44(6):737–755, 2004.

[CGP07] K. L. Calvert, J. Griffioen, and L. Poutievski. Separating Routing and Forwarding: A Clean-Slate Network Layer Design. In *In proc. of the Broadnets Conf.*, 2007.

[Chi91] J.Noel Chiappa. A new ip routing and addressing architecture, 1991. Internet Draft.

[Chi99] J. Noel Chiappa. Endpoints and endpoint names: A proposal enhancement to the internet architecture, 1999. Internet Draft.

[CR06] Rami Cohen and Danny Raz. The internet dark matter - on the missing links in the as connectivity map. In *INFOCOM*. IEEE, 2006.

[DCDkc12] Amogh Dhamdhere, Himalatha Cherukuru, Constantine Dovrolis, and kc claffy. Measuring the evolution of internet peering agreements. In Robert Bestak, Lukas Kencl, Li Erran Li, Joerg Widmer, and Hao Yin, editors, *Networking (2)*, volume 7290 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 2012.

[DD08] Amogh Dhamdhere and Constantine Dovrolis. Ten years in the evolution of the internet ecosystem. In Konstantina Papagiannaki and Zhi-Li Zhang, editors, *Internet Measurement Comference*, pages 183–196. ACM, 2008.

[DD10] Elwyn Davies and Avri Doria. Analysis of Inter-Domain Routing Requirements and History. RFC 5773, Feb. 2010.

[DD11] Amogh Dhamdhere and Constantine Dovrolis. Twelve years in the evolution of the internet ecosystem. *IEEE/ACM Trans. Netw.*, 19(5):1420–1433, 2011.

[DDK06] Avri Doria, Elwyn B. Davies, and Frank Kastenholz. Requirements for inter-domain routin, 2006. Internet Draft.

[DH98] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification, 1998. RFC 2460. Available from `http://www.ietf.org/rfc/rfc2460.txt`.

[DKF+07] Xenofontas A. Dimitropoulos, Dmitri V. Krioukov, Marina Fomenkov, Bradley Huffaker, Young Hyun, Kimberly C. Claffy, and George F. Riley. As relationships: inference and validation. *Computer Communication Review*, 37(1):29–40, 2007.

[DSK08] Xenofontas A. Dimitropoulos, M. ÃĄngeles Serrano, and Dmitri V. Krioukov. On cycles in as relationships. *Computer Communication Review*, 38(3):102–104, 2008.

[ea] E.Osterweil et al. Nat traversal through tunneling (nattt). Available from: `http://www.cs.arizona.edu/~bzhang/nat/`.

[EF94] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *Computer Communication Review*, 29(4):251–262, 1999.

[FFML12] Dino Farinacci, Vince Fuller, Dave Meyer, and Darrel Lewis. Locator/id separation protocol(lisp), 2012. Internet Draft.

[FGG04] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing rouing holes in sensor networks. In *INFOCOM. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, number 4, pages 2458–2468, 2004.

[FL06] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), August 2006.

[FPW09] Roland Flury, Sriram V. Pemmaraju, and Roger Wattenhofer. Greedy routing with bounded stretch. In *INFOCOM*, pages 1737–1745. IEEE, 2009.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, 2001.

[GG01] Cyril Gavoille and Marc Gengler. Space-efficiency of routing schemes of stretch factor three. *Journal of Parallel and Distributed Computing*, 61(5):679–687, 2001.

[GH] GH. Cidr report. Available from: `http://www.cidr-report.org/as2.0/`.

[GKR04] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Traveling with a pez dispenser (or, routing issues in mpls). *SIAM J. Comput.*, 34(2):453–474, 2004.

[GP96] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *PODC '96: Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 125–133, New York, NY, USA, 1996. ACM.

[GS69] K.Ruben Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 3(18):259–278, 1969.

[Han06] M. Handley. Why the internet only just works. *BT Technology Journal*, 24(3):119–129, 2006.

[HB96] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930 (Best Current Practice), March 1996.

[HD90] Christian Huitema and Walid Dabbous. Routing protocols development in the osi architecture. In *Proceedings of ISCIS V*, 1990.

[Hin96] R. Hinden. New Scheme for Internet Routing and Addressing (ENCAPS) for IPNG. RFC 1955 (Informational), June 1996.

[HM08] G. Huston and G. Michaelson. Textual Representation of Autonomous System (AS) Numbers. RFC 5396 (Proposed Standard), December 2008.

[Hus] Geoff Huston. Bgp routing table statistics. Available: http://www.telstra.net/ops/bgp/.

[Hus01a] Geoff Huston. Analyzing the internet's bgp routing table. *The Internet Protocol Journal*, 4(1), 2001.

[Hus01b] Geoff Huston. Commentary on Inter-Domain Routing in the Internet. RFC 3221 (Informational), December 2001. Obsoletes: 2796, 1966.

[Hus11] Geoff Huston. Bgp growth revisited, 2011. The ISP Column.

[Hus12] Geoff Huston. The end of ipv4, part 2, 2012. The ISP Column.

[IRR] IRR. Internet routing registry. Available: www.irr.net/docs/list.html.

[ISO94] ISO. Information technology-telecommunications and information exchange between systems-protocol for exchange of inter-domain routeing information among intermediate systems to support forwarding of iso 8473 pdus. ISO/IEC 10747, 1994.

[JL84] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[kc11] kc claffy. Tracking ipv6 evolution: Data we have and data we need. *Computer Communication Review*, 43(3):43–48, 2011.

[KCR08] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In Victor Bahl, David Wetherall, Stefan Savage, and Ion Stoica, editors, *SIGCOMM*, pages 3–14. ACM, 2008.

[Ken05] Stephen Kent. IP Encapsulating Security Payloa (ESP). RFC 4303, December 2005. Obsoletes RFC 2406.

[KFY04] Dmitri V. Krioukov, Kevin R. Fall, and Xiaowei Yang. Compact routing on internet-like graphs. In *INFOCOM*, 2004.

[KGKS05] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *NSDI*. USENIX, 2005.

[KJZ+10] Varun Khare, Dan Jen, Xin Zhao, Yaoqing Liu, Daniel Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. Evolution towards global routing scalability. *IEEE Journal on Selected Areas in Communications*, 28(8):1363–1375, 2010.

[KK77] Leonard Kleinrock and Farouk Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1(3):155–174, 1977.

[KK00] Brad Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In Raymond L. Pickholtz, Sajal K. Das, RamÃşn CÃąceres, and J. J. Garcia-Luna-Aceves, editors, *MOBICOM*, pages 243–254. ACM, 2000.

[KkccFB07] Dmitri Krioukov, k c claffy, Kevin Fall, and Arthur Brady. On compact routing for the internet. *ACM SIGCOMM Computer Communications Review*, 37(3):41–52, 2007.

[Kle00] Jon Kleinberg. Navigation in a small world. *Nature*, 406:845, August 2000.

[Kle07] Robert Kleinberg. Geographic routing using hyperbolic space. In *INFOCOM*, pages 1902–1909. IEEE, 2007.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Unit disk graph approximation. In Stefano Basagni and Cynthia A. Phillips, editors, *DIALM-POMC*, pages 17–23. ACM, 2004.

[KPBV09] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Marián Boguñá, and Amin Vahdat. Greedy forwarding in scale-free networks embedded in hyperbolic metric spaces. *SIGMETRICS Performance Evaluation Review*, 37(2):15–17, 2009.

[KPK+10] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *CoRR*, abs/1006.5169, 2010.

[KPVB09] Dmitri V. Krioukov, Fragkiskos Papadopoulos, Amin Vahdat, and Marián Boguñá. On curvature and temperature of complex networks. *CoRR*, abs/0903.2584, 2009.

[KWZ03] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *MobiHoc*, pages 267–278. ACM, 2003.

[KWZZ03] Fabian Kuhn, Rogert Wattenhofer, Yan Zhang, and Aaron Zollinger. Geometric ad-hoc routing: Of theory and practice. In *PODC*, pages 63–72. ACM, 2003.

[LABJ01] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. *IEEE/ACM Trans. Netw.*, 9(3):293–306, 2001.

[LAWS01] Craig Labovitz, Abha Ahuja, Roger Wattenhofer, and Venkatachary Srinivasan. The impact of internet policy and topology on delayed routing convergence. In *INFOCOM*, pages 537–546, 2001.

[LG] LG. Ipv4 looking glass sites. Available from: `http://www.bgp4.net/wiki/doku.php?id=tools:ipv4_looking_glasses`.

[Lit89] M. Little. Goals and functional requirements for inter-autonomous system routing. RFC 1126, October 1989.

[LJC+00] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris. A scalable location service for geographic ad hoc routing. In Raymond L. Pickholtz, Sajal K. Das, RamÃşn CÃąceres, and J. J. Garcia-Luna-Aceves, editors, *MOBICOM*, pages 120–130. ACM, 2000.

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1):2, 2007.

[LLR95] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[LT79] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graph. *SIAM J.Comput.*, 2(36):177–189, 1979.

[Max] MaxMind. Geolite databases. Available from: `https://www.maxmind.com/app/geolite`.

[May06] Petar Maymounkov. Greedy embeddings, trees and euclidian vs. lobachevsky geometry, 2006. Technical Report. Available from `http://pdos.csail.mit.edu/petar/pubs.html`.

[MGVK02] Zhuoqing Morley Mao, Ramesh Govindan, George Varghese, and Randy H. Katz. Route flap damping exacerbates internet routing convergence. In *SIGCOMM*, pages 221–233. ACM, 2002.

[MKF+06a] Priya Mahadevan, Dmitri V. Krioukov, Marina Fomenkov, Xenofontas A. Dimitropoulos, Kimberly C. Claffy, and Amin Vahdat. The internet as-level topology: three data sources and one definitive metric. *Computer Communication Review*, 36(1):17–26, 2006.

[MKF+06b] Priya Mahadevan, Dmitri V. Krioukov, Marina Fomenkov, Bradley Huffaker, Xenofontas A. Dimitropoulos, Kimberly C. Claffy, and Amin Vahdat. The internet as-level topology: three data sources and one definitive metric. *Computer Communications Review*, 36(1):17–26, 2006.

[ML08] Ankur Moitra and Tom Leighton. Some results on greedy embeddings in metric spaces. In *FOCS*, pages 337–346. IEEE Computer Society, 2008.

[MP85] J.C. Mogul and J. Postel. Internet Standard Subnetting Procedure. RFC 950 (Standard), August 1985.

[MSO+99] D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoglu. Using RPSL in Practice. RFC 2650 (Informational), August 1999.

[MWA02] Ratul Mahajan, David Wetherall, and Thomas E. Anderson. Understanding bgp misconfiguration. In *SIGCOMM*, pages 3–16. ACM, 2002.

[MWZZ07] Dan Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. A scalable routing system design for future internet. *ACM SIGCOMM Workshop on IPv6*, August 2007.

[MXZ+04] Xiaoqiao Meng, Zhiguo Xu, Beichuan Zhang, Geoff Huston, Songwu Lu, and Lixia Zhang. Ipv4 address allocation and the evolution of the bgp routing table. *Computer Communications Review*, 35(1):71–80, 2004.

[MZF07] D. Meyer, L. Zhang, and K. Fall. Report from the IAB Workshop on Routing and Addressing. RFC 4984 (Informational), September 2007.

[NB09] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[NGV03] Harsha Narayan, Ramesh Govindan, and George Varghese. The impact of address allocation and routing on the structure and implementation of routing tables. In Anja Feldmann, Martina Zitterbart, Jon Crowcroft, and David Wetherall, editors, *SIGCOMM*, pages 125–136. ACM, 2003.

[O'D97] Mike O'Dell. Gse - an alternate addressing architecture for ipv6, 1997. Internet Draft.

[OPW+08] Ricardo V. Oliveira, Dan Pei, Walter Willinger, Beichuan Zhang, and Lixia Zhang. In search of the elusive ground truth: the internet's as-level connectivity structure. In Zhen Liu, Vishal Misra, and Prashant J. Shenoy, editors, *SIGMETRICS*, pages 217–228. ACM, 2008.

[OZPZ09] Ricardo V. Oliveira, Beichuan Zhang, Dan Pei, and Lixia Zhang. Quantifying path exploration in the internet. *IEEE/ACM Trans. Netw.*, 17(2):445–458, 2009.

[PCG04] Leon Poutievski, Kenneth L. Calvert, and Jim Griffioen. Speccast. In *INFOCOM*, 2004.

[Pel99] David Peleg. Proximity-preserving labeling schemes and their applications. In Peter Widmayer, Gabriele Neyer, and Stephan Eidenbenz, editors, *WG*, volume 1665 of *Lecture Notes in Computer Science*, pages 30–41. Springer, 1999.

[PKBV10] Fragkiskos Papadopoulos, Dmitri V. Krioukov, Marián Boguñá, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *INFOCOM*, pages 2973–2981. IEEE, 2010.

[POA+12] Jong Han Park, Ricardo V. Oliveira, Shane Amante, Danny McPherson, and Lixia Zhang. Bgp route reflection revisited. *IEEE Communications Magazine*, 50(7):70–75, 2012.

[Pos81] J. Postel. Internet Protocol. RFC 791 (Proposed Standard), September 1981.

[PR05] Christos H. Papadimitriou and David Ratajczak. On a conjecture related to geometric routing. *Theor. Comput. Sci.*, 344(1):3–14, 2005.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *Journal of the ACM*, 36(3):510–530, 1989.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329—350. Springer, 2001.

[RIP] RIPE. Routing information services. Available: http://www.ris.ripe.net.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (bgp-4). RFC 4271, January 2006.

[RPSS03] Ananth Rao, Christos H. Papadimitriou, Scott Shenker, and Ion Stoica. Geographic routing without location information. In David B. Johnson, Anthony D. Joseph, and Nitin H. Vaidya, editors, *MOBICOM*, pages 96–108. ACM, 2003.

[RS] RS. Ipv4 route servers. Available from: `http://www.bgp4.net/wiki/doku.php?id=tools:ipv4_route_servers`.

[SARK02a] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM*, 2002.

[SARK02b] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the internet hierarchy from multiple vantage points. In *INFOCOM*, 2002.

[SF04] Georgos Siganos and Michalis Faloutsos. Analyzing bgp policies: Methodology and tool. In *INFOCOM*, 2004.

[SMK+01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. *Computer Communication Review*, 31(4):149—160, October 2001.

[SS05] Yuval Shavitt and Eran Shir. Dimes: let the internet measure itself. *Computer Communications Review*, 35(5):71–74, 2005.

[SSG07] Sundar Subramanian, Sanjay Shakkottai, and Piyush Gupta. On optimal geographic routing in wireless networks with holes and non-uniform traffic. In *INFOCOM*, pages 1019–1027. IEEE, 2007.

[TDG+01] Hongsuda Tangmunarunkit, John Doyle, Ramesh Govindan, Walter Willinger, Sugih Jamin, and Scott Shenker. Does as size determine degree in as topology? *Computer Communication Review*, 31(5):7–8, 2001.

[Thu97] William P. Thurston. *Three-dimensional geometry and topology. Vol. 1*, volume 35 of *Princeton Mathematical Series*. Princeton University Press, 1997. Edited by Silvio Levy.

[TM01] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):p425 – 443, 19691201.

[Tou80] Godfried T. Toussaint. The relative neighborhood graph of a finite planar set. *Pattern Recognition*, 4(12):261–268, 1980.

[Tsu87] P.F. Tsuchiya. An architecture for network-layer routing in osi. *Computer Communication Review*, 17(5):185–190, 1987.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001.

[UO] UO. Routeviews project. Available: www.routeviews.org.

[VBc+04] P. Verkaik, A. Broido, k. claffy, R. Gao, Y. Hyun, and R. van der Pol. Beyond cidr aggregation. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), Feb 2004.

[VC07] Q. Vohra and E. Chen. BGP Support for Four-octet AS Number Space. RFC 4893 (Proposed Standard), May 2007.

[VCG98] C. Villamizar, R. Chandra, and R. Govindan. BGP Route Flap Damping. RFC 2439 (Proposed Standard), November 1998.

[Wik12] Wikipedia. Computer network—Wikipedia, the free encyclopedia, 2012. [Online; accessed 11-12-2012].

[WP09] CÃľdric Westphal and Guanhong Pei. Scalable routing via greedy embedding. In *INFOCOM*, pages 2826–2830. IEEE, 2009.

[WS98] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of “small-world" networks. *Nature*, 393:440–442, 1998.

[YCB07] Xiaowei Yang, David Clark, and Arthur W. Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 15(4):775–788, 2007.

[ZFWY06] Xinyang Zhang, Paul Francis, Jia Wang, and Kaoru Yoshida. Scaling ip routing with the core router-integrated overlay. In *ICNP*, pages 147–156. IEEE Computer Society, 2006.

[Zha08] Lixia Zhang. A retrospective view of network address translation. *IEEE Network*, 22(5):8–12, 2008.

[ZHS+04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

[ZOW+11] Yu Zhang, Ricardo V. Oliveira, Yangyang Wang, Shen Su, Baobao Zhang, Jun Bi, Hongli Zhang, and Lixia Zhang. A framework to quantify the pitfalls of using traceroute in as-level topology measurement. *IEEE Journal on Selected Areas in Communications*, 29(9):1822–1836, 2011.

Vita

Name

Yinfang Zhuang

Place of Birth

Shanghai, China

Education

2006–now	**PhD in Computer Science, University of Kentucky, USA.**
2003–2006	MSc in Computer Science, Tongji University, China.
1998–2002	BSc in Computer Science, Tongji University, China.

Professional Experience

2012–2012	Work at Human Development Institute of University of Kentucky Work with HDI's Area of Early Childhood projects technology team to develop interactive database driven websites.
2011–2012	Grader at Computer Science Department of University of Kentucky CS405 Introduction to Database and CS375 Discrete Structures, Logic and Computability.
2007–2011	Research Assistant at Computer Science Department of University of Kentucky Work on Internet topology (esp. AS-level graph) and measurement of Internet efficiency (esp. the efficiency of different addressing schemes). Proposed metrics to measure the locality of current Internet address scheme and theoretically proposed more generalized metrics to measure the locality of different addressing schemes for different kinds of graphs (i.e. graphs with different structures).
2006–2006	Teaching Assistant at Computer Science Department of University of Kentucky Lab Supervisor and Grader for CS101.

Honors

2011-2012	Thaddeus Curtz Memorial Graduate Fellowship
2011	Student Travel Support from Graduate School Fellowship of University of Kentucky
2010-2011	USEC Inc. Graduate Fellowship
2004	Guanghua Graduate Scholarship
2003-2005	Tuition-free Scholarship
1998-2002	Tongji Excellent Student Scholarship

Publications

now	Yinfang Zhuang, Ken Calvert, *Measuring the Locality of Different Addressing Schemes on Different Graphs*, to be submitted
2010	Yinfang Zhuang, Ken Calvert, *Measuring the Efficiency of Hierarchical Address Assignment*, IEEE Globecom 2010