Hitesh Ballani

4130 Upson Hall
Cornell University
Ithaca, NY-14853

hitesh@cs.cornell.edu
www.cs.cornell.edu/∼hitesh
Phone: 607-279-6780
Fax: 607-255-4428

Research Interests

Networked Systems: Routing, Protocols, Management, Security.

Education

Cornell University — Ithaca, NY.
Ph.D. candidate in Computer Science — Expected summer 2009
Advisor: Paul Francis; GPA: 4.19/4.0
Minor in Statistics

Indian Institute of Technology — Roorkee, India.
Bachelor of Technology, Computer Science — May 2003
GPA: 9.76/10.0

Research

ViAggre (Virtual Aggregation) — *HotNets'08, NSDI'09*
Cornell University — **Aug. 2007 – Present**
Invented and deployed ViAggre, a "configuration-only" technique to shrink the routing table on routers. ViAggre does not require any changes to router software and routing protocols and can be deployed by any ISP without the cooperation of other ISPs. It can reduce the routing table load on routers by more than an order of magnitude. ViAgree was deployed on a testbed of Cisco hardware routers. I also implemented an open-source configuration tool that can automatically reconfigure status-quo routers for operation according to ViAggre.

CONMan (Complexity Oblivious Network Management) — *INM'06, SIGCOMM'07, INFOCOM'09*
Cornell University — **Jan. 2006 – Present**
Conceived, designed and implemented CONMan, an architecture aimed to make IP networks amenable to management. CONMan restricts the operational complexity of protocols to their implementation by minimising the amount of protocol-specific information exposed in their management interface. I implemented a suite of CONMan protocols and management applications. The resulting testbed was shown to be configurable based on human specified high-level goals. Faults in the network were automatically detected and localised. Thus, in effect, the CONMan testbed is a largely self-managing network.

DNS DoS Mitigation — *HotNets'06, CCS'08*
Cornell University — **Jan. 2006 – Aug. 2008**
Designed "Stale Cache", a minor modification to the caching behavior of DNS resolvers that reduces the need for nameserver availability in the *existing* DNS framework. This can, in turn, mitigate the impact of DoS attacks on DNS. Conducted a long-term measurement study that shows that even a single Internet resolver can use a Stale Cache and acquire significant protection against DNS DoS attacks.

Prefix Interception and Hijacking — *SIGCOMM'07*
Cornell University — **May 2006 – Sep. 2007**

Investigated the possibility of *Traffic Interception* in the Internet wherein the attacker transparently intercepts traffic from a sender to a receiver. As a proof-of-concept, I designed an Interception attack, deployed it and used it to capture real Internet traffic (belonging to a prefix I own). Also conducted a trace-based study to quantify the possibility of Hijacking and Interception in the Internet and a measurement study to detect ongoing Internet Interception.

Denial-of-Service protection *HotNets'05*
Intel-Research, Berkeley **June 2005 – July 2005**
Designed the *DefaultOff* architecture wherein end-hosts are "off by default" and each host is allowed to explicitly declare to the routing infrastructure what traffic it wants routed to it. Using traces from real ISPs, I showed that such a network is technologically feasible and proposed a *reachability protocol* to allow for flexible expression of reachability by end-hosts and end-sites.

Proxy IP Anycast Service (PIAS) *Worlds'04, SIGCOMM'05, IMC'06*
Cornell University **Mar. 2004 – May 2006**
Designed, implemented and deployed PIAS, as IP-level anycast architecture that combines the advantages of native IP anycast and application level anycast. Besides obviating client modifications, PIAS offers a unique array of features that allow P2P and overlay applications to use the proposed anycast service. I deployed the PIAS service in the Internet (presently at 7 PIAS nodes in US, UK and Asia) and have been maintaining this testbed for the past four years.

Delay-tolerant Bulk Transfers
Telefonica Research **June 2008 – present**
Ongoing work that focusses on the problem posed by transfer of bulk traffic across the Internet. Devising mechanisms to take advantage of the delay-tolerance of bulk traffic to make it amenable to the network. Implementing an overlay that uses storage as one such mechanism to time-shift bulk traffic away from the peak hours for Internet ISPs. Also conducting a measurement study to determine bounds for the benefits that such time-shifting can entail.

Teaching

Computer Networks *Teaching Assistant*
CS519, Cornell University **2004**
Designed and implemented the course project for CS519 - a master's level networks course. The project comprised of basic sockets programming and a user-level IP stack implementation. Other tasks included holding office hours, lecturing on certain occasions, and grading.

Java Practicum *Teaching Assistant*
CS212, Cornell University **2003**
Designed problem sets, taught discussion sections, graded and conducted office hours for CS212, an undergraduate course that introduces students to the ways of software engineering using the Java programming language.

Publications

[1]. **Hitesh Ballani**, Paul Francis, Tuan Cao and Jia Wang. Making Routers Last Longer with ViAggre. In *Proc. of USENIX Networked Systems Design and Implementation*

(NSDI) (April 2009).

[2]. **Hitesh Ballani** and Paul Francis. Fault Management Using the CONMan Abstraction. In *Proc. of IEEE INFOCOM* (April 2009).

[3]. **Hitesh Ballani** and Paul Francis. Mitigating DNS DoS Attacks. In *Proc. of ACM Conference on Communications and Computer Security (CCS)* (October 2008).

[4]. **Hitesh Ballani**, Paul Francis, Tuan Cao and Jia Wang. ViAggre: Making Routers Last Longer! In *Proc. of workshop on Hot Topics in Networks (Hotnets-VII)* (October 2008).

[5]. **Hitesh Ballani** and Paul Francis. CONMan: A Step towards Network Manageability. In *Proc. of ACM SIGCOMM* (August 2007).

[6]. **Hitesh Ballani**, Paul Francis and Xinyang Zhang. A Study of Prefix Hijacking and Interception in the Internet. In *Proc. of ACM SIGCOMM* (August 2007).

[7]. **Hitesh Ballani** and Paul Francis. A Simple Approach to DNS DoS Defense. In *Proc. of workshop on Hot Topics in Networks (HotNets-V)* (November 2006).

[8]. **Hitesh Ballani**, Paul Francis and Sylvia Ratnasamy. A Measurement-based Deployment Proposal for IP Anycast. In *Proc. of Internet Measurement Conference (IMC)* (October 2006).

[9]. **Hitesh Ballani** and Paul Francis. CONMan - Taking the Complexity out of Network Management. In *Proc. of Sigcomm Workshop on Internet Network Management (INM)* (September 2006).

[10]. **Hitesh Ballani**, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe and Scott Shenker. Off by default! In *Proc. of workshop on Hot Topics in Networks (Hotnets-IV)* (November 2005).

[11]. **Hitesh Ballani** and Paul Francis. Towards a global IP Anycast service. In *Proc. of ACM SIGCOMM* (August 2005).

[12]. **Hitesh Ballani** and Paul Francis. Towards a Deployable IP Anycast Service. In *Proc. of First Workshop on Real, Large Distributed Systems (WORLDS)* (December 2004).

Talks

ViAggre: Making Routers Last Longer!
– *HotNets*, 2008

Mitigating DNS DoS Attacks
– *ACM CCS*, 2008

CONMan: A Step Towards Network Manageability
– *ACM SIGCOMM*, 2007
– *Cisco Tech Talk*, 2007
– *Ph.D Candidacy Exam*, 2007
– *AFOSR Funding Meeting* 2007

A Study of Prefix Hijacking and Interception in the Internet
– *ACM SIGCOMM*, 2007

CONMan - Taking the Complexity out of Network Management
– *ACM SIGCOMM INM*, 2006
– *Microsoft Edgenet Summit (Poster)*, 2006

A Measurement-based Deployment Proposal for IP Anycast
– *ACM IMC*, 2006

A Simple Approach to DNS DoS Mitigation
– *HotNets*, 2006

Towards a Global IP Anycast Service
– *ACM SIGCOMM*, 2005
– *DNS OARC Meeting*, 2005
– *ICSI Seminar*, 2005

Off by Default!
– *HotNets*, 2005

Towards a Deployable IP Anycast Service
– *WORLDS*, 2004

Honors and Awards

Best Paper Award
ACM IMC, 2006

President's Gold Medal, 2003
For obtaining highest CGPA amongst graduating students at the Indian Institute of Technology, Roorkee

IIT Roorkee, Gold Medals, 2000-2003
Best undergraduate academic performance in each of the four years of study at Indian Institute of Technology, Roorkee

Cornell University, Outstanding TA award, 2004
Awarded in recognition of excellent TA'ship for Computer Networks (CS519), 2004.

IISc Bangalore, Summer Research Fellowship, 2002
Awarded a fellowship for summer research at the Indian Institute of Science, Bangalore

Professional Activities

- Reviewer for ACM CCR, IJCNS, IEEE Communication Letters
- External Reviewer for NSDI'05, ICNP'05, DSN'07
- Member of IEEE and ACM

References

Prof. Paul Francis
Assistant Professor
Department of Computer Science
Cornell University
Ithaca, NY 14853
francis@cs.cornell.edu

Sylvia Ratnasamy
Researcher
Intel-Research, Berkeley
2150 Shattuck Avenue, Suite 1300
Berkeley, CA 94704
sylvia.p.ratnasamy@intel.com

Pablo Rodriguez
Scientific Director
Telefonica Research
Via Augusta, 177
08021, Barcelona. Spain
pablo.rodriguez@tid.es

Jia Wang
Researcher
AT&T Labs – Research
180 Park Avenue
Building 103, Room A165
Florham Park, NJ 07932
jiawang@research.att.com

Statement of Research Interests

Hitesh Ballani

I enjoy research. My research goal is to tackle problems afflicting the Internet. The explosive growth in the size of the Internet over the past couple of decades has meant that many of the assumptions that the Internet design is based on no longer hold true. This has led to a plethora of problems and has made it imperative that we rethink such assumptions and the concomitant design decisions. However, the tremendous success of the Internet has also been a bane for Internet research. It is difficult, if not impossible, to expect a wholesale change in Internet infrastructure. Throughout my graduate career, I have tried to stay cognizant of this ground truth and have strived to strike a balance between two competing urges regarding my research.

On one hand is the freedom of doing blue sky research that openly questions the fundamentals underlying the Internet architecture in the face of new needs and challenges. For instance, as part of my dissertation I argue that the fact that the Internet and its precursors started off as simple research networks meant that "manageability" was never a first-class design goal. Instead, humans operating the network were expected to delve into low-level network details in order to make it work. However, as the Internet becomes bigger and more complex, such an approach becomes intractable. Frustrated by the lamentable state-of-art in network management, I proposed and implemented *CONMan* [2,5,9], a cohesive architecture that leads to easy-to-manage and even largely self-managing networks. Similarly, my work on *DefaultOff* [10], a DoS-resilient Internet architecture recognises the mismatch between the security needs of the original and today's Internet and shows how this can be addressed by flipping default Internet reachability from "on" to "off".

On the other hand, I want my research to have practical impact through solutions that are immediately deployable. While it is well accepted that it is difficult to address the Internet's problems without changing the protocols involved, I have found that in many cases, simply by focussing on a subset of the given problem space, it is possible to devise a solution that does not require architectural change. In other words, it is possible to use existing protocols in novel ways to address some of the problems. I call this style of research *dirty-slate*. Such incremental solutions have a couple of important benefits. First, they offer a better alignment of cost vs benefits and hence, have a good chance of real-world adoption. Second, if the subset of problems solved happen to be the most pressing of the lot, such solutions buy the time needed for architectural proposals to mature and get deployed.

Over the past few years, I have developed a knack for recognising problems areas that can be alleviated in an incremental fashion and actually devising solutions that do not require change to deployed protocols and devices. For instance, I have invented *ViAggre* [1,4], a "configuration-only" approach to shrinking the routing table on Internet routers. While there are many other problems that afflict the Internet's routing system and *ViAggre* is not a cure-all, it does solve an important part of the routing problem area without requiring changes to both routing protocols and the routers themselves and without requiring global agreement. The same theme of tackling problems without protocol changes and global deployment pervades my work on mitigating DoS attacks on DNS nameservers [3,7] and making IP Anycast deployments practically feasible [8,11,12].

Overall, I feel that I have been successful at striking a good balance between clean-slate and incremental research. I have made a conscious effort to explore different facets of systems research in the context of my interest in networking. This appears in the fact that my work has included measurement studies, analytical studies, architectural proposals, implementation and even wide-area deployment.

1 Previous Work

Clean-slate research.

– **Network Management**. IP networks are hard to manage (install, configure, provision, monitor, test, debug). I have been able to boil down a large fraction of our management troubles to one specific shortcoming:

"Today, protocols and devices expose their internal details leading to a deluge of complexity that burdens the management plane." Consequently, a basic requirement for an Internet architecture conducive to management is that the management interface of protocols contain as little protocol-specific information as possible. This concept is at the core of my dissertation proposal called *CONMan* or Complexity Oblivious Network Management. Among other things, I have developed a generic abstraction that is used by CONMan-compliant protocols and devices to express their functionality to management applications. I have shown that this abstraction can serve as the narrow waist for the Internet's management plane, not much different from the way IP has served as the narrow waist for the Internet's data plane.

While the CONMan idea can be applied to all aspects of network management, my thesis focusses on the use of CONMan for configuration [5,9] and fault management [2]. To this effect, I implemented a suite of CONMan protocols and management applications. The resulting testbed can thus be configured by humans simply by specifying desired high-level goals. Further, any faults occurring in the network are automatically detected and localised. In effect, the CONMan testbed is a largely self-managing network. This work was one of the top-rated papers at ACM Sigcomm'07 and has evoked interest and received funding support from Cisco and NSF FIND.

– **Denial-of-Service attacks**. The original Internet architecture was designed to provide universal reachability; any host can send any amount of traffic to any destination. Unfortunately, today's less trustworthy Internet environment has revealed the downside of such openness—*every* host is vulnerable to attack by *any* other host(s). I argue that the simplest and the most direct approach to this problem is to flip Internet reachability on its head – end hosts should be "off by default" and should explicitly declare what traffic they want to be routed to them. This was the basis of the *DefaultOff* architecture [10] that, despite the seemingly intractable burden imposed on the Internet infrastructure, proved to be technologically feasible.

Dirty-slate research.

– **Routing Scalability**. The Internet routing table has been growing at a rapid rate for the past few years. Till now, it was almost universally accepted that maintaining a large routing table is just a fact of life in the existing architecture and there is no way to reduce the burden on Internet routers other than through a significant redesign. Consequently, most recent routing research has focussed on new architectures. While I agree that maintaining the entire routing table on routers is the de-facto mode of operation for Internet routing, I don't think that any part of the Internet architecture necessitates it. Instead, the load imposed by the routing table can be reduced simply by dividing the task of maintaining it amongst the routers.

This insight led me to invent *ViAggre* (Virtual Aggregation) [1,4], a scalability technique that shrinks the routing table on routers. ViAggre's most exciting feature is that it is a "configuration-only" approach. ViAggre does not require any changes to routers or routing protocols and can de deployed independently by any ISP on the Internet. It is mostly due to these reasons ViAggre has attracted a lot of positive attention in the IETF community and has been put on the standards track by IETF. As a matter of fact, a major router vendor (Huawei) is experimenting with implementing ViAggre natively into its routers.

Measurement results show that ViAggre can shrink routing tables by more than an order of magnitude. I also implemented a configuration tool that allows network operators to adopt ViAggre without manual router reconfiguration. Finally, I deployed ViAggre on a testbed of Cisco hardware routers. This exercise provided me with useful hands-on experience with routers used by commercial ISPs and am sure will help me in future routing research endeavors.

– **DNS Denial-of-Service attacks**. Recent years have seen many instances of Denial-of-Service (DoS) attacks on DNS, the Internet's naming system. I realised that such attacks are essentially targeting the skewed division of name resolution functionality between DNS servers and clients. Specifically, the attacks aim to make DNS servers unavailable and hence, can be tackled by doing away with the need for 100% server availability. Guided by this observation, I proposed a minor modification in DNS caching behavior that mitigates the impact of such attacks [3,7]. A long-term measurement study showed that even a single DNS resolver in the Internet can adopt this modification and acquire significant protection against DNS DoS

attacks. I am especially proud of this work; while certainly not the highlight of my research credentials, it does underscore my belief in dirty-slate research and shows how a very trivial hack can be used to address daunting problems in the existing setup.

– **Practical IP Anycast**. IP anycast, with its innate ability to find nearby resources in a robust and efficient fashion, has long been considered an important means of service discovery. However, it suffers from severe scalability problems. I argue that IP Anycast entails a tight coupling of the the anycast functionality to Internet routing mechanisms and this is the root-cause of its scalability problems. Hence, I designed, implemented and deployed *PIAS* (Proxy IP Anycast Service) [11,12], an anycast service that decouples anycast functionality from Internet routing.

As part of the PIAS deployment, I deployed a small testbed comprising of seven nodes spread across the Internet with each node advertising an address prefix into Internet routing. This, apart from the technical know-how involved, equipped me with an insight into the workings of commercial routing agreements. I have been maintaining this testbed for the past four years. Apart from the anycast service, the testbed provides a unique and very useful tool for researchers to perform active routing experiments. For instance, the testbed has been used by me [6,8] and other researchers [13,14] for wide-area BGP studies.

2 Future Work

From the beginning, I have focussed my research on problems in networking. This has allowed me to really delve deeply into one research area so as to have the most impact. This approach has worked well for me, and has led to a number of new ideas that I plan to pursue.

What to solve? A spate of routing research over the past few years has led to an agreement that severe problems afflict the routing system. For instance, scalability and manageability are commonly cited as the two biggest challenges facing Internet routing. However, there is still no agreement on what specific factors represent a scalability bottleneck and in what order: Is it memory or processing power or the ability to power the routers and dissipate the generated heat. Further, the answer depends not only on the characteristics of the routing system (such as the routing table size), but also on the characteristics of the routers themselves (such as the number of peers). My thesis is that only after understanding the key pain points can we come up with a practical alleviative for our routing pains. I am working on such benchmarking of the control plane on Internet routers. The ultimate goal of this exercise is to guide the design of the next generation routing system.

The Power of Tunnels. To a large extent, multihoming and traffic engineering have been the biggest contributors to the growth in the Internet routing table. This, in turn, is the result of the unholy marriage between Routing and Traffic Engineering. Specifically, Internet operators lack the appropriate primitives to control traffic flowing in and out of their networks. Instead, they try to achieve goals regarding network traffic through archaic knobs of routing protocols. The key problem with this marriage is the mismatch between the granularity provided by the routing system (prefix-level) and the granularity required for traffic engineering (flows or aggregates of flows).

On the other hand, recent years have seen an increasing use of tunnels in both enterprise networks and wide-area Internet. The adoption of MPLS by ISPs and the surge in the use of VPN technologies has promoted tunneling to a first-class Internet mechanism. In spite of this very public evolution, I am surprised at how little attention tunnels have received in the research community. My work on ViAggre exploits tunnels to shrink the routing table size on routers. Extending this, I believe that tunneling is a very powerful primitive and represents an opportunity to have a genuine impact on many Internet problems. Hence, my vision is to move the Internet towards an inter-domain tunneling system with the goal of tackling the complementary problems of routing scalability and traffic engineering.

Delay-tolerant Traffic. The rise of P2P and streaming applications has led to a substantial increase in traffic carried by ISPs and has caused them to take steps ranging from capacity upgrades to the much-maligned selective manipulation of traffic. However, a lot of such traffic is delay-tolerant. For instance, it is common for end-users to download movies only to watch them later. Such traffic can thus be "time-shifted" (delayed across time) to make it more amenable to the network. This, apart from reducing the peak burden on

the ISPs, can offer better performance to end-users. To this effect, I am collaborating with researchers at Telefonica Research towards the use of storage in the network to improve the network friendliness of delay-tolerant bulk traffic. Our current focus is on a CDN-like deployment model that has a symbiotic relationship with ISPs while benefiting the end-users. Looking ahead, I am also interested in an architectural solution to the bulk-traffic problem. For instance, is it feasible for the network to provide storage as a service or is it easier to deal with bulk traffic through a separate low-priority channel that does not interfere with existing (non delay-tolerant) traffic.

Beyond this, I believe that my past work has equipped me with the requisite tools to diversify my research and investigate problems beyond my core interests. For instance, my projects have included analysis, measurement, implementation and deployment and should hold me in good stead for research in networked systems, including enterprise networks, wireless networks, peer-to-peer systems and web services. In the near term, I want to build upon my work in security [3,6] to tackle challenges in securing networked systems. Further, I have a minor in statistics and am interested in game theory. Hence, I am exploring opportunities to apply the corresponding tools to solve network problems.

To summarise, my research will be geared towards building better networked systems with a focus on improving their scalability, manageability, security and reliability. Furthermore, my proposals will retain the dual theme of instant and delayed gratification. To this effect, I want my future research endeavors to track the following roadmap: Given a problem area, the first step will determine the largest subset that can be solved within the existing setup and develop such a solution. Next, the insights from the dirty-slate solution will guide a complete solution that will most likely require infrastructure change. While I do not claim that my proposals will be better at spurring architectural change, I do believe that when the cost of dealing with a problem becomes high enough to justify the costs associated with a change, my solutions will have a better alignment of costs and benefits and hence, a better chance of deployment.

References

[1] **Hitesh Ballani**, Paul Francis, Tuan Cao and Jia Wang, "Making Routers Last Longer with ViAggre," in *Proc. of USENIX Networked Systems Design and Implementation (NSDI)*, April 2009.

[2] **Hitesh Ballani** and Paul Francis, "Fault Management Using the CONMan Abstraction," in *Proc. of IEEE INFOCOM*, April 2009.

[3] **Hitesh Ballani** and Paul Francis, "Mitigating DNS DoS Attacks," in *Proc. of ACM Conference on Communications and Computer Security (CCS)*, October 2008.

[4] **Hitesh Ballani**, Paul Francis, Tuan Cao and Jia Wang, "ViAggre: Making Routers Last Longer!" in *Proc. of workshop on Hot Topics in Networks (Hotnets-VII)*, October 2008.

[5] **Hitesh Ballani** and Paul Francis, "CONMan: A Step towards Network Manageability," in *Proc. of ACM SIGCOMM*, August 2007.

[6] **Hitesh Ballani**, Paul Francis and Xinyang Zhang, "A Study of Prefix Hijacking and Interception in the Internet," in *Proc. of ACM SIGCOMM*, August 2007.

[7] **Hitesh Ballani** and Paul Francis, "A Simple Approach to DNS DoS Defense," in *Proc. of workshop on Hot Topics in Networks (HotNets-V)*, November 2006.

[8] **Hitesh Ballani**, Paul Francis and Sylvia Ratnasamy, "A Measurement-based Deployment Proposal for IP Anycast," in *Proc. of Internet Measurement Conference (IMC)*, October 2006.

[9] **Hitesh Ballani** and Paul Francis, "CONMan - Taking the Complexity out of Network Management," in *Proc. of Sigcomm Workshop on Internet Network Management (INM)*, September 2006.

[10] **Hitesh Ballani**, Yatin Chawathe, Sylvia Ratnasamy, Timothy Roscoe and Scott Shenker, "Off by default!" in *Proc. of workshop on Hot Topics in Networks (Hotnets-IV)*, November 2005.

[11] **Hitesh Ballani** and Paul Francis, "Towards a global IP Anycast service," in *Proc. of ACM SIGCOMM*, August 2005.

[12] **Hitesh Ballani** and Paul Francis, "Towards a Deployable IP Anycast Service," in *Proc. of First Workshop on Real, Large Distributed Systems (WORLDS)*, December 2004.

[13] Tongqing Qiu, L. Ji, D. Pei, J. Wang, J. J. Xu, and **Hitesh Ballani**, "LOCK: Locating Countermeasure-Capable Prefix Hijackers," GeorgiaTech, Tech. Rep. GT-CS-08-04, 2008.

[14] Zheng Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, , and R. Bush, "iSPY: Detecting IP Prefix Hijacking on My Own," in *Proc. of ACM SIGCOMM*, August 2008.

Teaching Statement

Hitesh Ballani

I relish the simple joys and challenges of teaching. Very few things top the satisfaction that I gain from helping hard-working students learn a skill or solve a problem. I distinctly remember my first student lecture at Cornell where I was supposed to explain data structures such as stacks and queues to non-engineering students. While such concepts are second nature to anybody majoring in computer science, I did not want to assume that the same held for, lets say a fine arts student. To account for this, I tried to relate the data structures to their real-life counterparts. For instance, I used a box of Pringles to illustrate how the notion of "Last-In First-Out" captures the operation of a stack and the memory of the entire class appreciating how simple these fundamental programming constructs are has stayed with me ever since.

The above example illustrates one element of my teaching philosophy. In general, I strive to imbibe the following in my teaching:

- I don't treat teaching as a chore. Instead, I see it as an opportunity to interact with young and curious minds and this reflects in the energy and passion with which I teach.
- I try to make my classes, be it a lecture for a hundred students or a review session for just five students, as engaging as possible. I have found that only by involving my students in a discussion do I establish a feedback loop. This allows me to determine what techniques and examples are effective and which ones need to be refined.
- I firmly believe that the "One Size Fits All" approach does not apply to teaching. Based both on my personal experience as a student and as a teacher, I have realised that the best way to get an idea across depends on numerous factors such as the concepts at hand, the familiarity of the students with the topic and even the class size. For instance, I found that it was easier to explain programming language concepts based on mathematical constructs while students better understood network protocols and underlying principles by seeing the protocols in action.
- It is critical to strike a balance between well-established and well-understood concepts against work that represents the cutting-edge technology both in industry and research. This, I believe, is especially true for systems courses. Students get more involved and interested when told about how the algorithms and techniques they have studied are being used in new applications and products. Further, this results in well-rounded students who are better at applying the concepts that they have learnt.
- Teaching is a two-way street. I take my teaching assignments very seriously and try to ensure that students get a complete understanding and appreciation of the subject being taught. On the other side, teaching classes also results in a lot of takeaways for me. For one, explaining ideas to students allows me to understand how other people look at technical issues and by extension, lets me determine the best way to present and portray my research. More importantly, interacting with students who don't have many pre-conceived notions can lead to questions about fundamentals that many of us in the research community have simply accepted on blind faith. Such out-of-the-box discussions help me question my ideas and influence the direction of my research.

As far as *teaching experience* is concerned, I have served as the teaching assistant for two courses, "Programming Practicum" that introduces students to the ways of software engineering and "Computer Networks", an advanced networking course. As a TA, I was responsible for various course tasks, including

project design, review sessions, grading and lectures. For the latter course, I personally designed and implemented a set of programming assignments, including a user-level IP stack. I am very proud of the quality of these assignments which is illustrated by the fact that the same set has been used by the following offerings of the course for the past five years. Further, I also received an "Outstanding TA Award" from the department as a recognition of my contribution to the course and its students. Beyond this, I have given occasional lectures in both undergraduate and graduate-level networking courses.

Another important aspect of the teaching process is *mentoring* students. The focus here is less on imparting knowledge and more on being a *facilitator*. Such facilitation can range from defining and providing direction for a research project to helping out with low-level system idiosyncracies. I have had the pleasure of mentoring two amazing graduate students – Andrey Ermolinskiy (Berkeley) for a brief duration in 2005 and Tuan Cao (Cornell) for the past year. In both cases, I relished the opportunity to guide very bright students who were eager to learn the intricacies of Internet routing. I helped them appreciate why inter-domain routing works the way it works and what can be done to improve it. Seeing them make mistakes similar to the ones that I made when I was starting routing research but spend less time on finding a work-around due to my guidance was indeed very gratifying and contributed to my conviction to pursue an academic career.

I have been fortunate to have had many dedicated teachers who have influenced and inspired me deeply. For instance, it was the way that the teachers at my undergraduate institution piqued and encouraged my interest in systems and networking that convinced me to pursue graduate studies. My thesis advisor, Paul Francis, proved that having a good advisor can go a long way in making graduate school an amazingly enriching experience. I have consciously tried to imbibe the qualities of my teachers and it is their influence and my experience as a teacher and mentor that has shaped the teaching philosophy described above.

Given my research and teaching background, I will be most interested in teaching undergraduate and graduate-level networking courses. I would love to teach seminar and paper-chase courses on a number of special topics including routing, wireless networks, network games, application of game theory to networking problems, network security, etc. I will also be comfortable teaching other systems courses, such as operating systems, distributed systems and software engineering. My teaching vision is to impart my students with not only an understanding of how systems work and the ability to build them but to equip them with a toolset that can be used to solve real-world problems in novel and interesting ways. To this effect, I will strive to ensure that my courses strike the right balance between both "old-school" vs "new-school" concepts and text-based vs hands-on teaching. I also hope to give my students just the right dose of skepticism so that they can evaluate ideas in an objective fashion.

To summarize, I believe that being part of the academic world bestows upon us the very important responsibility of preparing the next generation to face the challenges of the world. As I get ready to pursue a career in academia, I feel fortunate that I will have a chance to mold so many young minds and I am confident that I will be able to guide them to bigger and better things in life.

CONMan: A Step Towards Network Manageability

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

ABSTRACT

Networks are hard to manage and in spite of all the so called holistic management packages, things are getting worse. We argue that the difficulty of network management can partly be attributed to a fundamental flaw in the existing architecture: protocols expose all their internal details and hence, the complexity of the ever-evolving data plane encumbers the management plane. Guided by this observation, in this paper we explore an alternative approach and propose Complexity Oblivious Network Management (CONMan), a network architecture in which the management interface of data-plane protocols includes minimal protocol-specific information. This restricts the operational complexity of protocols to their implementation and allows the management plane to achieve high level policies in a structured fashion. We built the CONMan interface of a few protocols and a management tool that can achieve high-level configuration goals based on this interface. Our preliminary experience with applying this tool to real world VPN configuration indicates the architecture's potential to alleviate the difficulty of configuration management.

Categories and Subject Descriptors: C.2.3 [Network Operations]: Network Management.

General Terms: Management.

Keywords: Management, Abstraction, Configuration.

1. INTRODUCTION

IP networks are hard to manage. Network management (installation, configuration, provisioning, monitoring, testing, debugging) requires detailed knowledge of many different network components, each with its own management interface. To cope, network managers rely on a host of tools ranging from sophisticated centralized network management packages to home-brewed scripts and elementary tools such as ping and traceroute. For instance, Cornell's IT group uses half a dozen different tools, commercial and public domain, and has over 100K lines of scripts for managing the switch and router infrastructure alone (not including email, servers, DNS, DHCP, billing, etc.). In spite of their ever increasing sophistication, management tools seem to be waging a losing battle which is shown by rising management costs and network downtime. A recent survey [18] showed that 80% of the IT budget in enterprises is devoted to maintain just the status quo - in spite of this, configuration errors account for 62% of network downtime.

We believe that the management troubles of the Internet have been aggravated by the lack of research on fundamentals. Instead, there is an increasing reliance on temporary "band-aids". While this has allowed a number of flaws to creep into the way we manage networks, in this paper we focus on one specific shortcoming:

> *Today, protocols and devices expose their internal details leading to a* **deluge of complexity** *that burdens the management plane.*

For instance, it is not uncommon for a network device to have thousands of manageable objects. A review of SNMP MIB modules found more than 13,000 MIB objects in IETF MIBs alone [34]; MIBDepot [49] lists 6200 MIBs from 142 vendors for a total of nearly a million MIB objects. A single router configuration file can consist of more than 10,000 command lines [39]. Encumbering the management plane with all this complexity leads to these problems:

- *Perception differs from reality.* Management applications need to effectively reverse engineer the capabilities and the functionality of protocols and devices from their detailed MIBs. The low-level and non-intuitive nature of these parameters makes this task difficult, if not impossible [28].
- *Error-prone configuration.* Network configuration involves mapping high-level policies and goals to the values of protocol parameters. Since management applications don't have an understanding of the underlying network in the first place, they often resort to a cycle of setting the parameters and correlating events to see if the high level goal was achieved or not. Apart from being haphazard, the noise in measurements and correlations is often the root-cause of misconfigurations and related errors. The inability to understand the network's operation also makes debugging these errors very difficult [21].
- *Fragmentation of tools.* Since devices and their exposed details keep evolving at a frantic pace, management applications tend to lag behind the power curve [26]. Additionally, the inability of standard management interfaces (IETF MIBs) to keep pace with data plane development has led to a plethora of vendor specific MIBs and even vendor specific management applications and has put us in a situation where *no one management approach* suffices. For example, SNMPLink [31] lists more than 1000 management

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

applications, many of them being vendor specific command line or HTML-based tools. Hence, the Internet management plane doesn't have anything analogous to the IP "thin waist" around which the Internet data-plane is built.

- *Lack of dependency maintenance.* Management state is highly inter-dependent. These dependencies are not reflected in the existing set-up; thus, when a low-level value changes, the appropriate dependent changes don't always happen [28]. Instances of improper filtering because the address assigned to some machine changed, or the application was started on some other port are very common. Recent work details the challenges involved in tracking such dependencies in the existing set-up [29] and gives examples of how failure to track them leads to problems in large networks [20].

These shortcomings indicate that an (extreme) alternative worth exploring is to confine the operational complexity of protocols to their implementation. As a matter of fact, we observe that almost all data-plane protocols share some very basic characteristics that should, in theory, suffice for the management of the network. Guided by this observation, we adopt a more modest approach and argue that:

> *The management interface of data-plane protocols should contain as little protocol-specific information as possible.*

This allows all data-plane protocols to have a generic yet simple management interface. While such an approach can be applied to all aspects of management, this paper restricts itself to *Configuration Management*. Hence, in this paper we present the design and implementation of a network architecture, *Complexity Oblivious Network Management* (CONMan), that incorporates this principle for network configuration tasks. In CONMan, all protocols and devices express their capability and their functionality using a generic abstraction. This allows the management plane to understand the potential of the underlying network and to configure it in line with the desired high-level policies without being encumbered by the details of the protocol/device implementation. Having a fixed interface between the management plane and the data plane also allows for independent evolution of the two. To this effect, this paper makes the following *contributions*:

- We present the detailed design of a network architecture that minimizes the protocol-specific information in the management interface of data-plane protocols. We also present protocol-independent configuration primitives that can be used to interact with this interface and hence, configure the network.
- We describe the implementation of the management interface of a few protocols in compliance with the proposed architecture.
- We detail the implementation of a management application that, given the abstraction of the protocols and devices in the network, can achieve high-level configuration goals using the aforementioned primitives.
- The paper presents the use of CONMan in a real-world configuration scenario (VPN configuration) to highlight its advantages over the status quo. Further, we also use a naive but hopefully informative metric to compare the protocol agnosticity of CONMan configurations against today's configurations in three different scenarios (GRE tunnels, MPLS LSPs and VLANs).

Note that CONMan doesn't reduce the total system complexity; it only attempts to correct the skewed division of functionality between management done inside the managed device and that done outside the managed device. While the fact that management applications don't have to deal with myriad protocol details reduces their burden, protocols still need various low-level details in order to operate. With CONMan, it is the protocol implementation that uses the high-level primitives invoked by the management applications and out-of-band communication with other protocols to determine these. This, in effect, puts the responsibility for detailed understanding of protocol operation on the protocol implementor. Since the protocol implementer requires this knowledge in any event, this seems to be a smarter placement of functionality.

2. CONMan ARCHITECTURE

Our architecture consists of *devices* (routers, switches, hosts, etc.) and one or more *network managers* (NMs). A NM is a software entity that resides on one of the network devices and manages some or all of them. Each device has a globally unique, topology independent identifier (*device-id*) that can carry cryptographic meaning (for example, by hashing a public key). Each device also has an internal *management agent* (MA) that is responsible for the device's participation in the management plane. While the rest of the paper talks about a device performing management tasks, in actuality it is the device's MA that is responsible for these. All protocols and applications in devices are modeled as *protocol modules*. Each protocol module has a name as well as an identifier that is unique within the device. Examples of module names include "IPv4", "RFC791", or even a URI (which might be useful for naming applications). Thus, modules can be uniquely referred to using tuples of the form <module name, module-id, device-id>.

2.1 Management Channel

As mentioned in section 1, a number of flaws afflict the way we manage networks. One such flaw is that the existing *management plane depends on the data plane* [6,14]. For example, SNMP operates on top of the data plane and hence, management protocols rely on the correct operation of the very thing they are supposed to manage. In recent work, Greenberg et. al. [14] discuss the implications of this dependency loop and propose a technique for achieving a self-bootstrapping, operationally independent management plane. While such management plane independence can be established using a few other approaches (for instance, a more generalized and self-bootstrapping version of the separate management network that is used by some large ISPs), we agree with their basic hypothesis and in this paper assume the presence of a *management channel*. This management channel should be independent of the data-plane, should not require any preconfiguration and should allow devices in the network to communicate with the NM. However, we do not dictate whether the management channel operates or does not operate over the same physical links as used by the data-plane.

2.2 Overview

Our approach derives from two key observations: First, the main purpose of a network is to provide paths between certain applications on certain hosts while preventing certain other

applications and hosts from using those paths.[1] Second, we observe that most data-plane protocols have some basic characteristics whose knowledge should suffice for configuring the aforementioned paths. For instance, most protocols have the ability to connect to certain other protocols, to switch packets, filter packets, queue packets and so on. We believe that it is these basic characteristics that should serve as the narrow waist for Internet's management plane. Consequently, the management plane only maps the high-level communication goal into the path through the network (i.e. which protocols should connected and how) and the protocols themselves figure out the low-level parameters that they need to operate.

In our proposal, we try to capture these basic characteristics using a generic abstraction called the *Module Abstraction* – all protocol modules in CONMan self-describe themselves using this abstraction. To this effect, we model every protocol module as a node with connections to other nodes, certain generic switching capabilities, certain generic filtering capabilities, certain performance and security characteristics, and certain dependencies (figure 1). Thus, the abstraction describes what the protocol is capable of (*potential*) and what it depends on (*dependencies*). Further, the module can be configured to operate in a certain fashion (*actual*) by manipulating its abstraction using the *CONMan primitives*. Such modeling of protocols using a generic abstraction decouples the data and the management plane so that they can evolve independently of each other.

Each device in the network uses the management channel to inform the NM of its physical connectivity, all modules that it contains, and their respective module abstractions. The module abstraction allows the NM to understand exactly how packets may flow (or not flow) through a given module. This provides the NM with the real picture of the network - it does not need to reverse engineer numerous low-level and non-intuitive parameters.

Given the network's real picture and the high-level goals and policies that need to be satisfied, the NM builds a graph of modules in various devices that satisfy these. This graph captures how each module should operate. The NM can then use the management channel to invoke the appropriate CONMan primitives and configure the modules accordingly. Thus, the NM can configure the entire network from the ground up with (almost) no protocol-specific knowledge. We believe that such as approach would ease network configuration and in general, ameliorate a lot of the problems afflicting network management today.

2.3 Module Abstraction

There are two kinds of modules: data plane modules and control plane modules. Examples of data plane modules (or data modules for short) include TCP, IP, Ethernet, while examples of control plane modules (or control modules for short) include routing algorithms and negotiation algorithms like IPSec's IKE or PPP's LCP and NCPs.

Data modules connect to each other to carry data packets. These connections are called pipes. Control modules also connect to data modules using pipes for delivery services. Data modules may require the use of a control module; we refer to this as a dependency. For instance, in Figure 1, the IPsec module has a (data plane) pipe to IP, and has a dependency on IKE, which in turn has a pipe to UDP. Ultimately, modules, pipes, and dependencies form a graph that in some sense describes the operation of the network. The data modules self-describe themselves using the abstraction shown on the right in figure 1. Below we briefly comment on the components of this abstraction:

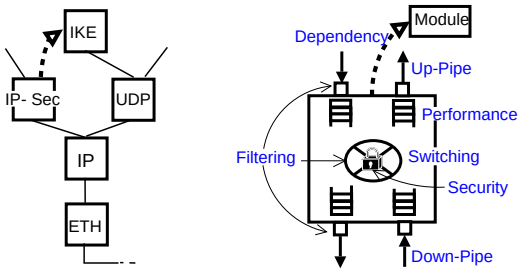


Figure 1: Modules, pipes, and dependencies form a graph that describes the operation of a device (in particular) and the network (in general). The figure on the right denotes the major components of the *module abstraction*.

2.3.1 Pipes

Up and *Down* pipes connect modules to other modules above and below themselves in the same device. Such pipes are point-to-point only. Point-to-point pipes are modeled as unidirectional (and usually come in pairs), though for simplicity we present them as bidirectional. The actual network links are modeled as *Physical* pipes and can be point-to-point or broadcast. Hence, the *path* between two modules in two different devices is the sequence of up-down and physical pipes through which packets travel between the modules. Of these, the NM can create up-down pipes. It cannot create physical pipes, but can discover and enable them. Also, pipes have identifiers which the NM can use to refer to them.

Modules are associated with a list of *connectable-modules*. For example, the connectable-modules for the down pipe of a particular TCP module might be restricted to {IPv4, IPv6} implying that the TCP implementation in question can only operate on top of (have a down pipe to) IPv4 or IPv6.

While modules pass packets between up and down pipes, the end goal is to be able to communicate with modules in other devices. To capture this, each pipe is associated with one or more *peers modules*. For example, the peer module for a down-pipe of a TCP module would be the remote TCP module to which the down-pipe ultimately leads to. Also, each module is associated with a set of *peerable-modules*. For example, the peerable-modules for a TCP module are {TCP} while the peerable-modules for a HTTP-server module are {HTTP-client}.

In effect, the notion of pipes abstracts away the details that protocols need for basic operation. Given a connectivity goal, the NM simply builds the corresponding path by creating pipes while the modules determine the low-level parameters. For instance, creating a down pipe from an IP module to an ETH module might be a part of establishing IP connectivity between two hosts and may cause the IP module to communicate with its peer IP module through the management channel to exchange the MAC address of the ETH module below it.[2] Apart from communication with peer modules, modules

[1]Of course, this is a simplification since the paths must perform adequately, have certain security properties, etc. but the basic argument still applies.

[2]Note that ARP achieves this in the existing set-up and even

may need more help in determining the low-level parameters – they express these as *dependencies* that need to be satisfied before the pipe can be created.

2.3.2 Switch

Switches capture the ability of modules to pass packets between up, down and physical pipes. A switch can be unicast or multicast and can have a small number of basic configurations: packets pass between down and up pipes ([down ⇒ up] and [up ⇒ down] switching, e.g. TCP module), [down ⇒ down] switching (e.g. IP module with forwarding enabled), [up ⇒ up] switching (e.g. IP module with loopback functionality), [up ⇒ phy], [phy ⇒ up] and [phy ⇒ phy] switching (eg. Ethernet module). A module advertises its switching capabilities. The NM uses this and the information about the connectable-modules of each module to build a *potential connectivity* graph for the network. As we show in section 3.3, this allows the NM to determine what paths are and are not possible. For instance, the ETH module in a Layer-2 switching device advertises that it can do [phy ⇒ phy] switching and so, can be used by itself along a path between two devices that the NM is trying to connect. As a contrast, the ETH module in a router would not have [phy ⇒ phy] switching capability and so, the NM must use it in conjunction with the IP module on the router.

When incorporating a module as part of a path, the NM must direct the module as to how packets must be switched between the pipes just created – this is the *actual* switch configuration. Of course, it is not necessary that there be a one-to-one mapping between the pipes. Instead, incoming packets on a pipe may be switched to one of many other pipes and hence, switches may have state which conditions how packets are switched. This switching state can be determined by the module through interaction with its peer module. For instance, the NM, as part of establishing an IP-IP tunnel, may direct an IP module to switch packets between up-pipe P1 (to another IP module) and down-pipe P2 (to the underlying ETH module). The creation of pipe P1 and P2 and the actual switch rule causes the three modules to interact with their peers and determine the parameters needed for a low-level routing rule such as `ip route to 204.9.169.1 dev eth1 nexthop 204.9.168.1`. Alternatively, it also possible that the switching state is generated by control protocols and this is exposed as part of the module abstraction. Section 2.6 discusses these alternatives.

2.3.3 Filters

The filter abstraction allows modules to describe whether and how they can filter packets. Filter rules are described in terms of other abstracted components: pipes, devices, modules or even module types. Note that in configuring a filter, the NM only needs to specify the component names or identifiers that need to be filtered - it is the protocol implementation that is responsible for determining the relevant protocol fields (such as addresses and port numbers). This process and other related issues are detailed in section 2.5.

2.3.4 Performance

Unlike the components above, which are quite specific in nature, performance is harder to specify and manipulate. In our current abstraction, performance is reported in terms of six generic *performance metrics* - delay, jitter, bandwidth, loss-rate, error-rate, and ordering. These encompass most of the IP performance metrics proposed by IETF [36]; though in our architecture the metrics can be used by any module that has the ability to describe its performance, not just the IP module. Additional metrics, such as power, can be added as needed.

Modules and pipes report on their performance with these metrics. They can also advertise the ability to offer performance trade-offs in terms of these metrics. For example, many MAC layer protocols offer optional error correcting checksums which represent a trade-off between error-rate on one hand and bandwidth and delay on the other. Instead of exposing the low-level options and associated parameters, modules specify the trade-offs they can enforce. Just as with filters, the module might allow these trade-offs to be applied to specific traffic classes as specified by the names of modules or pipes and this too is advertised. However, more work is needed towards the way these performance trade-offs can be quantified and what they can capture. Further, protocols modules may have other features such as performance enforcement and security capabilities. Due to space constraints, this paper does not delve into these abstraction components – we refer the interested reader to [4].

with CONMan, the IP module could just as well rely on ARP for the peer's MAC address.

Name	Caller	Callee	Description
showPotential	NM	MA of device	Sec. 2.4
showActual	NM	MA of device	Sec. 2.4
create, delete	NM	MA of device	Sec. 2.4
conveyMessage	Module (Source)	Module (Destination)	Sec. 2.4
listFieldsAndValues	Module (Inspecting)	Module (Target)	Sec. 2.5

Table 1: Functions that are part of the CONMan architecture

2.4 Network Manager (NM)

The management channel allows devices in the network to communicate with the NM. Each device uses this to inform the NM of its physical connectivity, thus allowing the NM to determine the network topology. Beyond this, given the network *potential*, the NM can achieve high level network configuration goals simply by creating and deleting pipes and module components. The following primitives capture the NM's interaction with the devices in the network as part of network configuration. Table 1 shows these and other CONMan primitives offered by the NM and the modules themselves.[3]

(a). *showPotential ()* allows the NM to determine a device's capabilities. The device returns a list of modules with their abstractions. The type of information returned for each module is shown in table 2.
(b). *showActual ()* allows the NM to determine the state of modules in a device. The state of each module includes state for all the pipes, the switch, filters, performance and security enforcement elements. Also returned is a report on the performance parameters. In effect, the NM is presented with the network reality - a module graph and associated information which allows it to understand how the device (and hence, the network) is or should be behaving. By contrast, in the current set up, the NM is presented with all kinds of

[3]We do not give details of the CONMan API. However, we do show the use of these primitives in section 3.

Parameter	What is advertised?
Name	<A,x,y>
Up and Down pipes	Information about up and down pipes such as connectable-modules, dependencies etc.
Physical pipes	Information about the physical pipes (if any) connected to the module
Peerable-Mod.	Set of modules that can be peers of this module
Filter	Classification based on which filtering can be done: what can be filtered and where it can be filtered
Switch	Possible switching between up, down and physical pipes; Is the switch state generated locally or needs to be provided externally
Performance Reporting	Performance metrics that are reported for the module's pipes, filters, switch etc.
Performance Trade-Offs	Traffic classes to which performance trade-offs can be applied and the possible trade-offs
Others	Performance Enforcement and Security Capabilities (not explained)

Table 2: Module abstraction; *showPotential* () describes each module using this abstraction

MIB objects from which it must deduce network behavior.
(c). *create ()* and *delete ()* allow the NM to create and delete pipes, filter-rules, switch-rules and performance enforcement state (queuing structures or service classes). The *showPotential ()* function provides the NM with all the information it needs to create and delete components.

The NM does not need protocol specific knowledge to use these primitives. For instance, it can create up-down pipes simply by satisfying their dependencies and invoking the *create* function. For instance, consider a NM creating a pipe between an IP module and an underlying GRE module. In terms of today's configuration, this amounts to creating a new GRE tunnel which requires a number of low-level parameters to be specified. With CONMan, it is the GRE module that coordinates these parameters with its peer GRE module. For instance, the modules may exchange the tunnel key values to be used, so the NM does not need to know the notion of keys. Since the management channel allows the modules to communicate *only* with the NM, the NM provides:
(d). *conveyMessage ()* allows modules to convey messages to each other through the NM (see detailed example in section 3.2).

2.5 Hiding Complexity

Much of the reduction in management plane complexity comes from the fact that the NM operates in terms of the abstract components, while the protocol modules themselves translate these into concrete protocol objects.

For example, the NM can simply ask a module to filter packets between two given modules - "drop packets from module <IP,B,y> and going to <FOO,C,z>" (where FOO is an application module with up-down pipes to TCP). The protocol module itself is responsible for determining the actual protocol fields. For example, given the high-level specification above, the inspecting module determines that it needs to "drop packets from source address 128.19.2.3 and destined to address 20.3.4.5, port 592". This ensures that the NM, while being opaque to protocol-specific fields, can trace the paths between applications and hence, can reason about its policies regarding a particular application-module.

In some cases, the inspecting module may know what fields and field values to check for on its own. But in other cases, it may not. To address this, CONMan modules provide a *listFieldsAndValues ()* function. This allows other modules to query the target module for the low-level fields and field values corresponding to the identifiers associated with its components. Hence, in the example above, the inspecting module can send queries to the target modules <IP,B,y> and <FOO,C,z> (via the NM), as well as to the modules below them, and ask those modules what field values it should be checking for.

Such an approach also allows for maintenance of network state dependencies – the need to update the dependent state in different modules when some low-level value in a given module changes. To ensure this, the NM tracks the dependencies between component identifiers (that have been resolved) and opaque low-level fields. Also, the NM installs triggers in the target modules telling them to inform the NM when their low-level values change.

However, not all detailed protocol values can be or should be determined by the protocols themselves. For instance, it appears difficult to expect IP modules to chat among themselves and assign IP addresses [12]. This is best done by the NM having explicit knowledge of how to assign IP addresses (as DHCP servers do today). Similarly, tasks like regular expression matching in HTML do not seem amenable to abstraction and should be done by specialized NMs such as Intrusion Detection Systems. Further, there are cases such as P2P protocols where protocol designers don't want to provide the protocol values since they don't want to be filtered. Thus, there are scenarios where the NM will have to deal with protocol-specific details.

2.6 Control Modules

Many data-plane protocols rely on externally generated state for their operation. Today, this may be provided manually as part of the protocol configuration. Alternatively, control-plane protocols can generate some of the state required for data plane operation. For example, routing protocols generate the IP routing table. Similarly, LCP generates PPP configuration state.

In CONMan, data modules can generate this state by interacting with their peer modules based on the create/delete primitives invoked by the NM. While this follows from the general CONMan philosophy, there are cases where such an approach poses challenges regarding the scalability, robustness and responsiveness of the network.

Alternatively, even in CONMan, we may rely on control protocols for the low-level state. However, control modules do not fit into the generic module abstraction presented earlier. Instead, they advertise their ability to provide the state for certain data modules and the NM simply uses them. For example, the PPP module could advertise that it has a dependency on external state (say, X) and the LCP module advertises that it can satisfy dependency X. While relying on control modules suffices in some cases, there are also cases when the control module itself requires quite a bit of configuration. Also, the fact that the NM does not generate this state hinders its ability to understand related network operations and gets in the way of root-cause analysis. Finally, errors in control module operation cannot always be debugged by the NM. For example, the NM does not understand BGP and hence, cannot be expected to debug route flaps and the resulting prefix dampening.

One way to address some of these problems is to let the NM perform the function of the control protocols whereby it uses some high-level goal to generate the required state itself. Of course, this implies that the state generation logic must be embedded into the NM. For example, the 4D research [14]

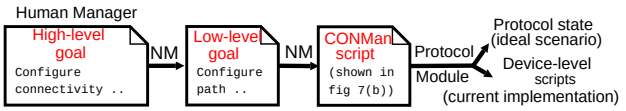


Figure 2: CONMan workflow: from high-level goals to device configuration

argues for the *replacement* of routing protocols, with the NM using its knowledge of the topology to set the switch state for IP modules in devices across the network. A characterization of the scenarios in which state should be generated by the protocols themselves against the ones in which existing control protocols should be used against the ones in which the control protocols should be replaced is an important question in the context of CONMan. However, in order to explore the limits of our proposal (i.e. what can be captured and what cannot be captured), this paper (rather naively) ignores the existence of control protocols. Hence, our implementation, for the most part, involves the protocol modules generating the low-level details.

3. IMPLEMENTATION

In CONMan, human managers don't write device-level scri-pts; instead, they specify high-level configuration goals and it is the NM and the protocol modules that map these to the required low-level configuration. This process is shown in figure 2 and is detailed in various parts of this section.

We implemented four protocols (GRE, MPLS, IP, ETH) as CONMan modules through user-level wrappers around the corresponding existing protocol implementation in Linux (kernel 2.6.14). We also implemented a NM that understands the CONMan abstraction and implements the CONMan NM primitives. In section 3.2, we use the establishment of GRE tunnels as an example to detail our GRE module implementation. This, in effect, describes the mapping of a low-level goal to device-level scripts (see figure 2). In section 3.3, we detail how our NM implementation can map a human-specified high-level goal to low-level goals by describing the configuration of provider-provisioned Virtual Private Networks (VPNs) with CONMan.

3.1 Management Channel

The testbed used for the examples described below comprised of Linux-based PCs operating as end-hosts and routers with Ethernet as the connecting medium. All the PCs were equipped with a separate management NIC and connected to a separate network that served as the management channel for our experiments. Communication between the protocol modules and the NM was done through UDP-IP over this management channel. Note that this is not ideal since the management channel had to be pre-configured; however, this can avoided by using techniques proposed in [14].

3.2 GRE tunneling

GRE is an encapsulation protocol that can be used to encapsulate a network protocol (*payload protocol*) in another network protocol (*delivery protocol*). We focus on GRE with IPv4 as the underlying delivery protocol - *GRE-IP*. Consequently, each tunnel is characterized by a source and a destination IP address. Besides this, GRE also involves key'ing of tunnels - the source and the destination must agree on the key for the tunnel to operate correctly. Configuring such a *GRE-IP* tunnel today requires the management plane to

	Parameter	Value
i	Name	<GRE,device-id,module-id>
ii	Up.Con-Modules (Connectable-Modules)	IPv4
iii	Up.Dependencies	Performance Trade-offs to be specified
iv	Down.Con-Modules	IPv4
v	Down.Dependencies	None
vi	Physical pipes	None
vii	Peerable-Mod.	GRE
viii	Filter	Nil
ix	Switch	[Up ⇒ Down],[Down ⇒ Up]
x	Perf Reporting	Number of recieved and transmitted bytes on each up pipe
xi	Perf Trade-Offs	{[Jitter, Delay] Vs [In-order delivery] | Up-pipe} {[Loss-Rate] Vs [Error-Rate] | Up-pipe}
xii	Perf Enforcement	Nil
xiii	Security	Nil

Table 3: Abstraction exposed by our GRE implementation

provide the IP addresses of the tunnel end-points, the key values, whether to use sequence numbers or not (sequence numbers help with in-order delivery of tunneled packets) and other protocol specific details such as tunnel TTL, the TOS field for tunneled packets, whether to use checksums or not, and whether to use path-mtu-discovery or not.

We have implemented a GRE module conforming to the CONMan architecture. As mentioned earlier, our implementation is based on the Linux GRE kernel module with a user-level wrapper that confines the protocol-specific details to the implementation and exposes a generic abstraction to the NM. This abstraction is shown in table 3 and some of the entries are explained below:

ii). Ideally, GRE can carry any payload protocol and hence, there should not be any restriction on the modules that the GRE module can connect to using an up pipe. However, most implementations restrict the payload protocol to a well defined list of protocols - with our underlying Linux implementation, the only payload protocol possible is IPv4.

iii). To create an up pipe, the NM needs to specify the performance trade-offs (see k below) that apply to pipe.

iv,v). The module is restricted to having IPv4 as the tunneling protocol with no explicit dependencies.

ix). The module can switch packets between an up-pipe and a down-pipe. The switching state is generated by the module on its own.

x). The underlying Linux implementation provides limited performance reporting: the number of bytes transmitted and received on each up pipe.

xi). The module offers the following trade-offs: For a given up-pipe, it can trade-off delay and jitter for in-order delivery. The fact that this is attained by enabling sequence numbers whose use needs to be coordinated with the peer GRE module is not exposed. Similarly, the module can trade-off loss-rate for error-rate for a specified up-pipe through the use of checksums.

We now describe how a NM, based on this abstraction, can use CONMan primitives to achieve the following **low-level goal**:

> *Configure the path between the IP modules <IP,A,a> and <IP,B,a'> labeled as (1) through (12) in figure 3.*

Note that this is equivalent to creating a GRE-IP tunnel between devices **A** and **B** in the existing set-up. Also, as men-

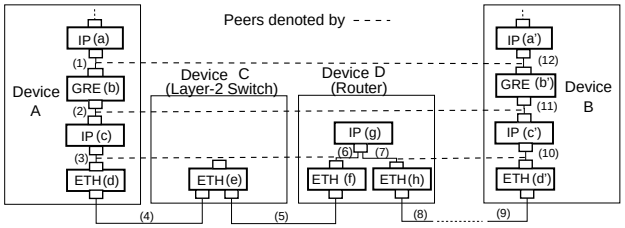


Figure 3: GRE-IP tunnel between devices A and B - the NM needs to build the path labeled from (1) to (12).

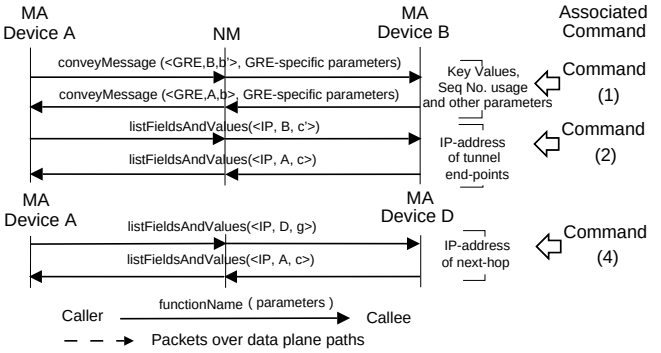


Figure 4: GRE-IP Tunnel establishment between devices A and B - the management plane is simplified by ensuring that protocol complexity is restricted to protocol implementation.

tioned earlier, **the human manager in CONMan is not aware of such low-level goals or the notion of pipes and switches or the CONMan script shown below**. Instead, he/she specifies a high-level goal and the next section describes how our NM implementation maps this high-level goal to the aforementioned low-level goal. This mapping process informs the NM which modules along the path are peers of each other – in figure 3, the dashed line between pipes labelled (1) and (12) indicates that modules *a* and *a'* are peer modules for these pipes (as are modules *b* and *b'*).

We also assume that the NM has, as part of the mapping process, invoked the *showPotential* primitive at these devices and hence, is aware of the CONMan abstraction for all the modules involved; for instance, table 3 shows the abstraction exposed by the module <GRE-IP,A,b> (or *b* for short). The other modules have similar abstractions that are not shown here. This equips the NM with all the information it needs to create the appropriate pipes and switch state. As a contrast, some manual must be read (either by the implementor of the management application or the system administrator) to gain the equivalent knowledge while configuring *GRE-IP* tunnels today. With this information at hand, the NM can build the segment of the path in device **A** (i.e. $a \Rightarrow b \Rightarrow c \Rightarrow d$) using the following script. A similar script needs to be invoked to build the rest of the path.

```
(1).  P1 = create (pipe, <IP,A,a>, <GRE,A,b>,
          <IP,B,a'>, <GRE,B,b'>,
          trade-off:  order delivery,
          trade-off:  error-rate)
(2).  P2 = create (pipe, <GRE,A,b>, <IP,A,c>,
          <GRE,B,b'>, <IP,B,c'>, None)
(3).  create (switch, <GRE,A,b>, P1, P2)
(4).  P3 = create (pipe, <IP,A,c>, <ETH,A,d>,
          <IP,D,g>, <ETH,D,f>, None)
(5).  create (switch, <IP,A,c>, P2, P3)
(6).  create (switch, <ETH,A,d>, P3, P4)
```

In the script, command (1) creates pipe P1 between the IP module *a* and the underlying GRE module *b*. The fourth and fifth arguments in the command specify the peer IP (*a'*) and GRE (*b'*) modules *for the pipe being created*. Further, the NM satisfies the dependency for creating an up pipe for a GRE module by specifying that it desires in-order delivery of packets and low error-rate. These choices would be based on high-level performance goals specified by the human manager. Similarly, commands (2) and (4) create pipes P2 and P3. Through command (3) the NM specifies that GRE module *b* should switch between pipes P1 and P2. Similarly, commands (5) and (6) configure the switch in modules *c* and *d* respectively.

The simple and structured process described above is all the configuration that the NM needs to do. It is the protocols that incorporate the complexity of determining the low-level parameters. Each module, based on the commands invoked by the NM, interacts with its peer module through the management channel to determine the required protocol specific parameters – this process is briefly described below and illustrated in figure 4.

On invocation of command (1) and the corresponding command on device **B**, modules *b* and *b'* use the *conveyMessage* primitive to exchange the GRE-specific parameters needed for connectivity between them. These include the GRE key values in each direction, the use of sequence numbers, etc. Some of these parameters are based on the trade-off decisions specified by the NM. For example, the NM, as part of command (1), opts for in-order delivery. This causes modules *b* and *b'* to negotiate the use of sequence numbers for the GRE tunnel between them. Similarly, on invocation of command (2), IP modules *c* and *c'* figure out the IP addresses of the tunnel end-points by determining each other's IP address through the use of *listFieldsAndValues*. Command (3) causes the GRE module *b* to generate the actual Linux command to configure the GRE tunnel, the parameters for this command already having been determined:

```
ip tunnel add name gre-P1-P2 mode gre remote 204.9.
169.1 local 204.9.168.1 ikey 1001 okey 2001 icsum
ocsum iseq oseq
```

Similarly, command (4) causes IP modules *c* and *g* to exchange their IP addresses while command (5) causes IP module *c* to generate the low-level routing rule in device **A** such that packets to device **B** are routed through **D**:

```
ip route add to 204.9.169.1 via 204.9.168.2 dev eth2
```

3.3 Virtual Private Networks (VPNs)

VPNs are commonly used to connect geographically distributed enterprise sites across the Internet while offering security and performance comparable to connecting the sites across a dedicated network. As the name suggests, a "provider-provisioned VPN" involves the ISP that provides connectivity to the enterprise sites configuring and maintaining the VPN [3]. We implemented a NM that can be used for such VPN configuration. In the interest of brevity, the discussion below focusses on the configuration sub-task of an ISP trying to ensure that traffic between two sites *S1* and *S2* of a customer *C1* is isolated from other traffic. A complete VPN configuration involves doing the same for all pairs of sites of

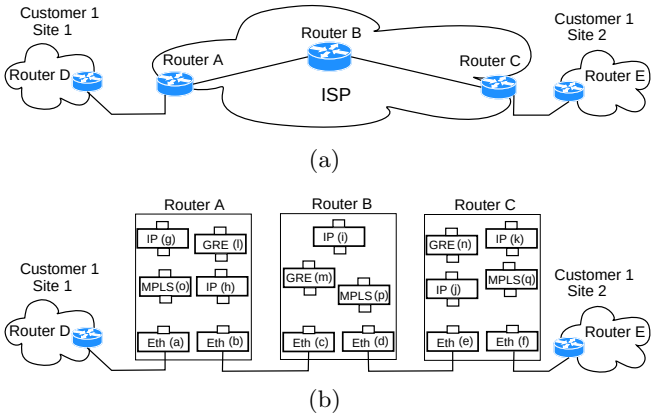


Figure 5: Experimental set-up emulating an ISP and customer sites for the VPN configuration. Figure 5(b) shows the network map and the modules seen by the NM prior to configuration.

each customer needing VPN support. Figure 5(a) shows the relevant part of the set-up in our lab with five Linux hosts (labeled *A*-*E*) serving as two of the ISP's edge routers in different POPs (*A* and *C*), the ISP's core router (*B*) and customer C1's routers at site S1 (*D*) and site S2 (*E*). Specifically, the NM aims to achieve the following **high-level goal specified by the human network manager**:

> *Configure connectivity between sites S1 and S2 of customer C1.*

This is equivalent to the following high-level goal in CONMan terminology:

> *Configure connectivity between the customer-facing interfaces <ETH,A,a> and <ETH,C,f> (see figure 5(b)) for traffic between C1-S1 and C1-S2.*

Ideally, C1-S1 and C1-S2 should be high-level identifiers that get mapped to the IP prefixes for the two sites through communication between the NM of the ISP and the NM of customer C1. However, this paper is restricted to management in a single domain and hence, we provide the NM with this information. Second, we assume that the NM has already assigned IP addresses to the IP modules. Finally, the high-level goal above is imprecise since it does not specify whether traffic between C1-S1 and C1-S2 ought to be isolated or not. Today, this choice is dictated by whether C1-S1 and C1-S2 are public or private IP prefixes and this applies to our implementation too. Consequently, the NM is aware of the notion of public and private addresses. As explained later in the section, this knowledge is used by the NM in the way it finds paths through the network. We admit that this is case of the NM using protocol-specific information. As mentioned earlier, while it is possible to abstract IP addresses, their ubiquity and scarcity combined with the impact of address assignment on routing scalability suggests that it makes engineering sense to let the NM be aware of them and this is what we chose for our implementation.

3.3.1 NM Implementation

All devices in the ISP's network (routers *A, B, C*) inform the NM of their physical connectivity through the management channel. Given the aforementioned goal, our NM implementation invokes *showPotential* at these devices to determine the abstraction for the modules in these devices. Thus, the NM has a network map akin to the one shown in figure 5(b). This also provides the NM with information about how the modules can be connected to each other and how they can switch packets (shown in table 4). Based on this, the NM constructs a graph of *potential connectivity* with modules as "nodes" and up-down and physical pipes as "edges". Figure 6 shows the device **A** part of this graph.

Module	Connectivity and Switching
<ETH,A,a>	Up: {IP, MPLS}, Down: None, Phy: to C1-S1, Switching: [Phy ⇒ Up],[Up ⇒ Phy]
<ETH,A,b>	Up: {IP, MPLS}, Down: None, Phy: to <ETH,B,c>, Switching: [Phy ⇒ Up],[Up ⇒ Phy]
<MPLS,A,o>	Up: {IP}, Down: {ETH}, Phy: None, Switching: [Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down]
<IP,A,g>	Up: {IP, GRE}, Down: {IP, GRE, MPLS, ETH}, Phy: None, Switching:[Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down],[Up ⇒ Up]
<IP,A,h>	Up: {IP, GRE}, Down: {IP, GRE, MPLS, ETH}, Phy: None, Switching:[Down ⇒ Up],[Up ⇒ Down],[Down ⇒ Down],[Up ⇒ Up]
<GRE,A,l>	Up: {IP}, Down: {IP}, Phy: None, Switching: [Down ⇒ Up],[Up ⇒ Down]

Table 4: Connectivity and switching capabilities of the modules in device A.

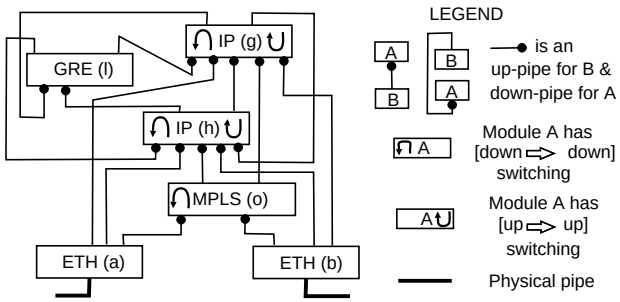


Figure 6: Potential Connectivity sub-graph for device A.

The NM also includes a *path-finder* component that can find all paths between any two modules in such a graph. To do so, the component traverses the graph in a depth-first fashion while avoiding cycles. Further, we made two modifications to the traversal: First, the NM knows that a module encapsulates packets in a protocol header when using [up ⇒ down] and [up ⇒ phy] switching; for example an ETH module adds an Ethernet header to packets that it sends out onto a physical pipe. Similarly, a module decapsulates packets when using [down ⇒ up] and [phy ⇒ up] switching. A module processes the packet header but doesn't remove or add headers when using [phy ⇒ phy], [down ⇒ down] and [up ⇒ up] switching. The traversal keeps track of such encapsulation and decapsulation by the modules along the path and hence, restricts itself to paths that are "sane" in the protocol sense. For instance, assuming that the path shown in figure 7(a) is the path already traversed, this rule implies that the next module should be able to decapsulate or process an IP header and hence, the only possible next module is the IP module in device B, <IP,B,i>. This also allows the NM to determine modules that are peers of each other; in the path above, <ETH,B,c> decapsulates the encapsulation put in by <ETH,A,b> and hence, they are peers.

Second, the NM is aware of the notion of public and private addresses and the traversal uses this information to rule out invalid paths. For instance, the path shown in figure 7(b) is an invalid path as it makes IP modules *g* and *i* peers even

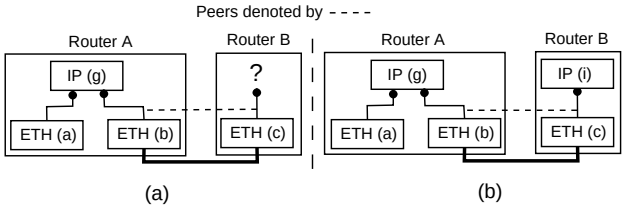


Figure 7: Options explored by the NM's path finder.

though *g* is assigned a private address while *i* is assigned a public address.

For the given goal, the NM directs the path-finder to find paths between modules <ETH,A,a> and <ETH,C,f>. We were expecting the NM to generate the following three paths (we only show the module-id for each module along the path):
1). Using IP-IP tunnel: *a, g, h, b, c, i, d, e, j, k, f.*
2). Using GRE-IP tunnel: *a, g, l, h, b, c, i, d, e, j, n, k, f.*
3). Using MPLS: *a, g, o, b, c, p, d, e, q, k, f.*
However, the NM generated six more paths: IP-IP over MPLS, GRE-IP over MPLS, IP-IP over MPLS only between A and B, IP-IP over MPLS only between B and C, GRE-IP over MPLS only between A and B, and GRE-IP over MPLS only between B and C.[4] While this suggests that we should use more aggressive pruning rules for our traversal, it also shows that the NM can determine the various ways of achieving a high-level goal given the capabilities of the devices in the network. As a contrast, today it is the human managing the network that relies on RFCs and device manuals to determine the options available.

The NM now needs to be able to choose amongst the paths based on high-level directives and/or other metrics. We implemented a very simple algorithm that minimizes the total number of pipes instantiated in the routers. This is, in some sense, akin to minimizing the amount of state on the routers and the communication overhead on the NM. For the scenario in question, the MPLS-based path and the IP-IP tunnel are the best options (our NM implementation prefers the MPLS-based path because the MPLS abstraction mentions that it offers good forwarding bandwidth). We can also think of more sophisticated metrics such as the performance capabilities of the modules along the path or satisfying security constraints. Moreover, while the ability to choose amongst possible configurations without protocol-specific knowledge is critical to the CONMan argument, this is an area that we haven't explored in any detail and is an avenue for future work.

As described in the previous section, once a path is chosen, the NM automatically generates the script of CONMan primitives needed to create the path.

3.3.2 Comparing to the status quo

For each path in the example above, we directed the NM to generate the CONMan primitives needed to create the path. These primitives were invoked at the modules in the devices (routers A, B and C) to configure them. Since the modules are implemented as wrappers around existing protocol implementations, they in turn generate the device-level scripts from the CONMan primitives. It is the management plane that needs to generate these device-level scripts with today's setup. Below we compare the configurations for two of these paths: the *GRE-IP* and the *MPLS* path.

Figure 8(a) shows a Linux configuration snippet at router A that establishes a GRE tunnel to router C and carries traffic between sites S1 and S2 of customer C1. As a contrast, the desired module connectivity and the CONMan commands invoked by the NM at router A to achieve this are shown in figure 8(b). These commands were explained in section 3.2. Similarly, figures 9 shows the Linux and CONMan configuration snippet needed to establish the MPLS path.

Note that while our testbed capabilities constrained us to Layer-3 VPNs, some ISPs establish VPN connectivity at Layer-2. This is typically achieved using Ethernet-over-MPLS or PPP-over-L2TP. Recently, VLAN tunneling has been proposed as another means of doing so [41] and as the use of Ethernet in wide-area networks increases, this could be a future VPN technology. Consequently, we also present the Cisco CatOS and CONMan configuration snippet to establish a VLAN tunnel in figure 10.

The figures show that configuration today requires the management plane to specify a lot of low-level details. As a result, **it is difficult to build management applications that automatically generate these configurations**. Instead, many management applications provide a better user interface and/or some syntactic sugar to the human manager (this is useful in itself). Even with these applications, the human manager still needs to provide the specifics and this leaves the door open for many kinds of errors; for instance, some error possibilities in figure 8(a) include not configuring device A as a router (command 4), misconfiguring the underlying routing so that traffic from the wrong customer goes into a tunnel or the tunneled traffic is delivered to the wrong customer at the other end (commands 5-9), configuring the tunnel end points with the wrong key values (command 2), using tunnel end point IP addresses that are wrong or do not have IP connectivity between them (command 2), etc.

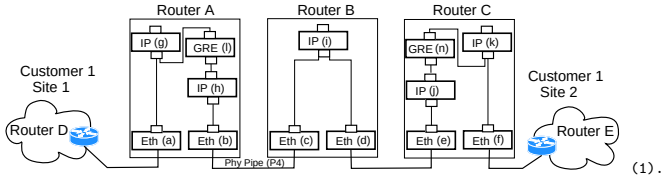
The CONMan scripts do not appear any-less-fragile. However, **the human manager doesn't need to see, much less write, these scripts**. All the identifiers in the script, such as the module and device identifiers, are exposed by the devices themselves and learnt by the NM through *showPotential*. Further, there is very little protocol-specific information in CONMan scripts and hence, **an automated NM can generate the commands and other details algorithmically without incorporating protocol-specific knowledge**. Also, the similarity in the CONMan scripts for three completely different protocols can be seen as retrospective (yet relevant) evidence of CONMan decoupling the management plane from data-plane evolution.

To quantify the protocol-agnosticity of CONMan, we counted the number of protocol-specific commands and state variables in the scripts. Table 5 shows that today's scripts have far more protocol-specific commands and state-variables. As mentioned earlier, the instances of protocol-specific state variables in CONMan scripts (such as *C1-S2* representing the IP prefix for customer1-site2 on line (3) of figure 8(b)) result from the fact that our current effort is restricted to management in a single domain. On the other hand, CONMan scripts have more generic state-variables. This is an outcome of both the verbose nature of the existing CONMan primitives and the fact that CONMan requires the NM to specify a lot of well-structured and systematically learnt generic information which the protocol modules then use to determine

[4]Typically, ISPs use MPLS-over-MPLS [33] or MPLS-over-GRE [40] for VPN support. Both these configurations are not supported by the Linux hosts used for our experiments and hence, the NM cannot propose these paths.

```
#!/bin/bash
# Insert the GRE-IP kernel module
(1) insmod /lib/modules/2.6.14-2/ip_gre.ko
# Create the GRE tunnel with the appropriate key
(2) ip tunnel add name greA mode gre remote 204.9.169.1 local
204.9.168.1 ikey 1001 okey 2001 icsum ocsum iseq oseq
(3) ifconfig greA 192.168.3.1
# Enable Routing
(4) echo 1 > /proc/sys/net/ipv4/ip_forward
# Create IP routing from customer to tunnel
(5) echo 202 tun-1-2 >> /etc/iproute2/rt_tables
(6) ip rule add to 10.0.2.0/24 table tun-1-2
(7) ip route add default dev greA table tun-1-2
# Create IP routing from tunnel to customer
(8) echo 203 tun-2-1 >> /etc/iproute2/rt_tables
(9) ip rule add iff greA table tun-2-1
(10) ip route add default dev eth1 table tun-2-1

(11) ip route add to 204.9.169.1 via 204.9.168.2 dev eth2
```

(a) Configuration "Today"



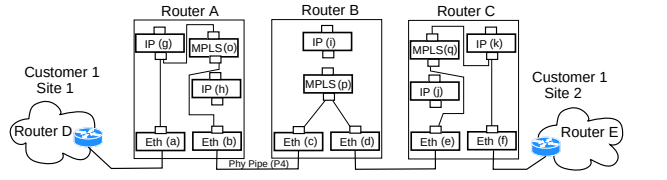

```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
(2). P1 = create (pipe, <IP,A,g>, <GRE,A,l>, <IP,C,k>, <GRE,C,n>,
trade-off: in-order delivery, trade-off: error-rate)
(3) create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
(4) create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
(5). P2 = create (pipe, <GRE,A,l>, <IP,A,h>, <GRE,C,n>, <IP,C,j>, None)
(6). create (switch, <GRE,A,l>, P1, P2)
(7). P3 = create (pipe, <IP,A,h>, <ETH,A,b>, <IP,B,i>, <ETH,B,c>, None)
(8). create (switch, <IP,A,h>, P2, P3)

(9). create (switch, <ETH,A,b>, P3,P4)
```

(b) CONMan configuration

Figure 8: VPN connectivity between sites S1 and S2 of customer C1 through a GRE-IP tunnel between A and C.

```
#!/bin/bash
# Instantiating MPLS kernel modules
modprobe mpls
modprobe mpls4
# MPLS LSP for traffic from S2->S1
mpls labelspace set dev eth2 labelspace 0
mpls ilm add label gen 10001 labelspace 0
KEY-S2-S1=`mpls nhlfe add key 0 mtu 1500 instructions nexthop
eth1 ipv4 192.168.0.1 | grep key | cut -c 17-26`
mpls xc add ilm_label gen 10001 ilm_labelspace 0 nhlfe_key
$KEY-S2-S1
# MPLS LSP for traffic from S1->S2
KEY-S1-S2=`mpls nhlfe add key 0 mtu 1500 instructions push gen
2001 nexthop eth2 ipv4 204.9.168.2 | grep key | cut -c 17-26`
echo 1> /proc/sys/net/ipv4/ip_forward
ip route add 10.0.2.0/24 via 204.9.168.2 mpls $KEY-S1-S2
```

(a) Configuration "Today"




```
P0 = create (pipe, <IP,A,g>, <ETH,A,a>, None, None, None)
P1 = create (pipe, <IP,A,g>, <MPLS,A,o>, <IP,C,k>, <MPLS,C,q>,
None)
create (switch, <IP,A,g>, [P0, dst:C1-S2 ⇒ P1])
create (switch, <IP,A,g>, [P1 ⇒ P0, S2-gateway])
P2 = create (pipe, <MPLS,A,o>, <ETH,A,b>, <MPLS,B,p>, <ETH,B,c>,
None)
create (switch, <MPLS,A,o>, P1, P2)

create (switch, <ETH,A,b>, P2, P4)
```

(b) CONMan configuration

Figure 9: VPN connectivity between sites S1 and S2 of customer C1 using a MPLS LSP through router A, B and C.

	GRE		MPLS		VLAN	
	T	C	T	C	T	C
Generic Commands	1	2	1	2	3	2
Specific Commands	6	0	6	0	4	0
Generic State Var.	9	21	6	18	3	14
Specific State Var.	11	2	8	2	5	1

Table 5: Commands and state variables in Today's (T) and CONMan (C) scripts. The table and the scripts are color/font coded; for instance, the first occurrence of a "Generic Command" in each script appears in Red/Italics and so on.

the protocol parameters. While we admit that these represent very coarse metrics, we see this as a naive yet important step towards quantifying the advantages of having management applications generate CONMan primitives instead of device-level configuration.

4. RELATED WORK

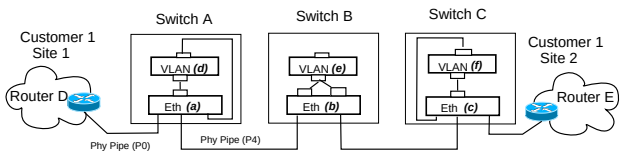
There is a tremendous amount of past work in network management, the most relevant of which we briefly cite here. On the commercial side, SNMPLink [31] lists many existing management tools, from low-end tools like packet analyzers (eg, Wireshark [47]), traffic monitors (eg, MRTG [35]), and SNMP agents (eg, ITM [7]) to high-end managers like OpenView [43].

Zeroconf [51] (and similar efforts like UPnP [46], DLNA [50], etc.) enable "local communication in networks of limited scale" without any configuration [15]. CONMan is more general but there are networks, such as ad-hoc networks, that we don't deal with. Further, with CONMan, the human manager does need to specify a configuration goal, albeit at a high-level. However, there are a number of Zeroconf features, such as address auto-configuration using link-local addresses, that CONMan could gain from. Policy-based management [16] tries to reduce the amount of intricate knowledge required by human managers by allowing management of QoS [2,32] and security [37] based on high-level policies. There are efforts in both research [48] and industry [42–45] with the similar goals. While steps in the right direction, some entity still has to map these policies to the individual device configurations. The complexity of this translation was the major impediment in the adoption of policy-based networking [17].

CONMan does not dictate how data-plane protocols should be implemented. However, there is the vast body of literature that does deal with protocol implementation, i.e. through abstractions [1], specification languages (Estelle, LOTOS, SDL [38]), implementation languages [10,24,25], and modularization (Click [19], [5]). The 4D proposal [14] recognizes the complexity of the Internet's control and management plane and hence, argues for restructuring them. We were motivated by, among other things, 4D's discovery plane. Recently, there has been a spurt of research detailing the reasons for outages and anomalies in IP backbones [22,27], Internet services [28] and BGP routing [11,26]. These studies point to configuration errors as a major culprit. CONMan can reduce these errors, particularly the ones impacting data plane operation. Finally, we believe that CONMan can simplify the cross-layer database and interface proposed in [20], and indeed may provide the basis for the Knowledge Plane objectives laid out by Clark et. al. [9].

```
# put module0 port 9 into VLAN22
# ensure MTU is set properly
set vlan 22 name C1 mtu 1504
set vlan 22 gigabitethernet0/9
# ensure module 0 port 7 is access port
interface gigabitethernet0/7
switchport access vlan 22
switchport mode dot1q-tunnel
exit
vlan dot1q tag native

end
```

(a) Configuration "Today" on Cisco CatOS




```
P1 = create (pipe, <ETH,A,a>, <VLAN,A,d>, <ETH,C,c>,
<VLAN,C,f>)
P2 = create (pipe, <VLAN,A,d>, <ETH,A,a>, <VLAN,B,e>, <ETH,B,b>)
create (switch, <ETH,A,a>, [P0, Tagged ⇒ P1])
create (switch, <ETH,A,a>, [P1 ⇒ P0])
create (switch, <VLAN,A,d>, P1, P2)

create (switch, <ETH,A,a>, P2, P4)
```

(b) CONMan configuration

Figure 10: VPN connectivity between sites S1 and S2 of customer C1 through VLAN tunneling between A and C.

5. DISCUSSION AND FUTURE WORK

In this paper we have presented a network architecture that is amenable to management. Implementation of a few protocols according to the CONMan model and their use in VPN configuration scenarios shows that the approach is worth considering. Though it is too early for us to claim that the abstraction presented here suffices for all data plane protocols and for tasks beyond basic configuration, we do not envision the module abstraction expanding much beyond its current state. As with OSs where we rely on *ioctls* and special-purpose interfaces for things that cannot be accomplished with the file system interface, in cases where protocol features are not captured by the abstraction (some were mentioned in the paper but we hope they will be few and far between), the low-level parameters will have to explicitly be set. Hence, we allow for the possibility of management applications accessing low-level details and provide the relevant hooks. However, we necessitate that any direct changes to the low-level details be appropriately reflected in the protocol's CONMan abstraction. There are many other avenues for future work, some of which we mention below:

– *Scalability*: This paper tests the extent to which management interfaces can be made protocol agnostic. However, it does not address concerns regarding the scalability and the robustness of the proposed approach. For instance, an important concern is the amount of traffic and processing load imposed on the NM, especially as a result of changes in high-level goals or even the network itself. Also, while our current implementation is restricted to lower layer modules and mostly static configuration tasks, scaling would be much more challenging if CONMan were to account for applications too. An extreme resulting scenario would be one where the NM configures modules across the network whenever an application initiates a connection. Note that in such a set-up, the message overhead imposed on the NM(s) would be similar to that imposed on domain controllers in the SANE project [8] and one could use their results to claim that even this can scale.

However, for a lot of tasks, the NM can use existing control protocols. For instance, our current path-finder could easily be modified to use a hierarchical two-step traversal wherein the first step finds paths between devices that have been pre-established using a routing algorithm while the next step finds the complete module-level path given the device-level path. Apart from this, CONMan would certainly benefit from many of the proposals to improve the scalability of automated agents within today's SNMP framework [13,23,30]. Further, as discussed below, the NMs themselves may do specialized jobs and hence, scale by divide-and-conquer.

– *Multiple NMs*: Our current attempt has focussed on a single NM managing a given network. However, multiple NMs may exist. Primary and secondary NMs will be needed for robustness. We can also imagine multiple simultaneously operating NMs. One reason for this might be that NMs do specialized jobs. For example, one is responsible for tunnel creation while another monitors for security violations. Another reason might be that NMs are administratively nested. For example, a high-level NM creates VLANs, but each VLAN has its own NM. Different domains will have their own NM and these may need to communicate.

– *Management Channel*: The aforementioned possibilities pr-esent a number of challenges such as the need for scoped management channels, extending the management channel beyond a single domain, the possibility of conflicting configurations and so on. Consequently, the notion of a management channel needs more thought. However, we would still like to keep the management channel as simple as possible so we don't run into the problem of managing the management channel. Further, the robustness and scalability questions regarding this channel suggest that it should only be used as the basis for low-level configuration. Higher-level management tasks should then rely on the data-plane for communication.

Beyond this, the NM design requires more work. On the user-side, we illustrated a simple high-level goal involving connectivity between two devices. However, we would like to evaluate other high-level goals and their impact on the algorithmic complexity of the NM. An exercise worth pursuing would be to come up with a simple yet precise language for such goals. Challenges for the NM on the the network-side include being able compare the quality of multiple low-level goals that satisfy a given high-level goal, ensuring that the translation process can scale to large networks, etc. Another important question is how to deploy CONMan. It is likely to share IPv6's conundrum: namely that complexity has to be increased over the short-term in order to arrive at reduced complexity over the long-term. However, there is still a lot of work to be done before we can worry about the widespread adoption of CONMan and hence, the path of least resistance towards a manageable, much less a self-managing network.

Acknowledgements

We would like to thank our shepherd, David Maltz, and the anonymous reviewers for their useful feedback. This work was partially supported by NSF Grants 0338750 and 0626978.

6. REFERENCES

[1] M. B. Abbott and L. L. Peterson, "A language-based approach to protocol implementation," in *Proc. of ACM SIGCOMM*, 1992, pp. 27–38.

[2] K. Amiri, S. Calo, and D. Verma, "Policy based management of content distribution networks," *IEEE Network Magazine*, March 2002.

[3] L. Andersson and T. Madsen, "RFC 4026 - Provider Provisioned Virtual Private Network (VPN) Terminology," March 2005.

[4] H. Ballani and P. Francis, "Complexity Oblivious Network Management: A step towards network manageability," Cornell University, Ithaca, NY, US, Tech. Rep. cul.cis/TR2006-2026, 2006.

[5] E. Biagioni, "A structured TCP in standard ML," in *Proc. of ACM SIGCOMM*, 1994.

[6] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform ," in *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[7] Carsten Schmidt, "Interface Traffic Monitor Pro," http://software.ccschmidt.de/.

[8] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "SANE: A Protection Architecture for Enterprise Networks," in *Proc. of Usenix Security*, 2006.

[9] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proc. of ACM SIGCOMM*, 2003.

[10] T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe, "Finally, a Use for Componentized Transport Protocols," in *Proc. of the Fourth Workshop on Hot Topics in Networking*, 2005.

[11] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[12] B. Ford, "Unmanaged Internet Protocol: taming the edge network management crisis," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, 2004.

[13] G. Goldszmidt, Y. Yemini, and S. Yemini, "Network management by delegation: the MAD approach," in *Proc. of the conference of the Centre for Advanced Studies on Collaborative research (CASCON)*, 1991.

[14] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Meyers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communications Review*, October 2005.

[15] E. Guttman, "Autoconfiguration for ip networking: Enabling local communication," *IEEE Internet Computing*, vol. 5, no. 3, 2001.

[16] J. Halpern and E. Ellesson, "The IETF Policy Framework Working Group," Online Charter, http://www.ietf.org/html.charters/OLD/policy-charter.html.

[17] M. Jude, "Policy-based Management: Beyond The Hype," *Business Communication Review*, pp. 52–56, 2001, http://www.bcr.com/bcrmag/2001/03/p52.php.

[18] Z. Kerravala, "Enterprise Networking and Computing : the Need for Configuration Management," Yankee Group report, January 2004.

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, August 2000.

[20] R. R. Kompella, A. Greenberg, J. Rexford, A. C. Snoeren, and J. Yates, "Cross-layer Visibility as a Service," in *Proc. of workshop on Hot Topics in Networks*, 2005.

[21] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, " IP Fault Localization Via Risk Modeling ," in *Proc. of 2nd Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[22] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental Study of Internet Stability and Backbone Failures," in *Proc. of Symposium on Fault-Tolerant Computing (FTCS)*, 1999.

[23] K.-S. Lim and R. Stadler, "Developing Pattern-Based Management Programs," in *Proc. of Conference on Management of Multimedia Networks and Services (MMNS)*, 2001.

[24] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica, "Implementing Declarative Overlays," in *Proc. of ACM SOSP*, 2005.

[25] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan, "Declarative Routing: Extensible Routing with Declarative Queries," in *Proc. of ACM SIGCOMM*, 2005.

[26] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. of ACM SIGCOMM*, 2002, pp. 3–16.

[27] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of Failures in an IP Backbone," in *Proc. of IEEE INFOCOMM*, 2004.

[28] D. Oppenheimer, A. Ganapathi, and D. Patterson, "Why do Internet services fail, and what can be done about it," in *Proc. of USENIX Symposium on Internet Technologies and Systems*, 2003.

[29] P. Bahl et. al., "Discovering Dependencies for Network Management," in *Proc. of workshop on Hot Topics in Networks*, 2006.

[30] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE/ACM Trans. Netw.*, vol. 36, no. 7, 1998.

[31] Pierrick Simier, "SNMPLink," www.snmplink.org/Tools.html.

[32] R. Rajan, D. Verma, S. Kamat, E. Felstaine, , and S. Herzog, "A policy framework for integrated and differentiated services in the internet," *IEEE Network Magazine*, vol. 13, no. 5, September 1999.

[33] E. Rosen and Y. Rekhter, "RFC 4364 - BGP/MPLS IP Virtual Private Networks (VPNs)," February 2006.

[34] J. Schonwalder, "Characterization of SNMP MIB Modules," in *Proc. of International Symposium on Integrated Network Management*, 2005.

[35] Tobias Oetiker and Dave Rand, "MRTG : Multi Router Traffic Grapher," http://mrtg.hdl.com.

[36] H. Uijterwaal and M. Zekauskas, "IP Performance Metrics (ippm)," Online Charter, Jan 2006, http://www.ietf.org/html.charters/ippm-charter.html.

[37] D. Verma, "Simplifying Network Administration using Policy based Management," *IEEE Network Magazine*, March 2002.

[38] G. von Bochmann, "Usage of Protocol Development Tools: The Results of a Survey," in *Proc. of Conference on Protocol Specification, Testing and Verification*, 1987.

[39] G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, and G. Hjalmtysson, "Routing design in operational networks: a look from the inside," in *Proc. of ACM SIGCOMM*, 2004, pp. 27–40.

[40] E. R. Y. Rekhter, R. Bonica, "Use of PE-PE GRE or IP in BGP/MPLS IP Virtual Private Networks," draft-ietf-l3vpn-gre-ip-2547-05, February 2006.

[41] "CISCO 802.1Q Tunneling," http://www.cisco.com/univercd/cc/td/doc/product/lan/c3550/1219ea1/3550scg/swtunnel.htm.

[42] "CISCO Network Management Products," http://www.cisco.com/en/US/products/sw/netmgtsw/index.html.

[43] "HP OpenView," www.openview.hp.com/.

[44] "IBM's Autonomic Computing," http://www-03.ibm.com/autonomic/.

[45] "Microsoft Dynamic Systems Initiative," http://www.microsoft.com/windowsserversystem/dsi/default.mspx.

[46] "UPnP Forum," http://www.upnp.org/.

[47] "Wireshark: A Network Protocol Analyzer," http://www.wireshark.org/.

[48] "IBM Research: Policy-based Networking," , Dec 2006, http://www.research.ibm.com/policy/.

[49] "SNMP MIB Search Engine," , January 2006, www.mibdepot.com.

[50] "Digital Living Network Alliance," Jan 2007, http://www.dlna.org/.

[51] "Zeroconf Working Group," Jan 2007, http://www.zeroconf.org/.

Making Routers Last Longer with ViAggre

Hitesh Ballani*, Paul Francis*, Tuan Cao* and Jia Wang†
*Cornell University †AT&T Labs – Research

To Appear in **NSDI'09** *– This is not the camera-ready version. Please do not distribute. The measurements in section IV-B.4 are in error and will be corrected in the camera-ready version.*

***Abstract*— This paper presents ViAggre (Virtual Aggregation), a "configuration-only" approach to shrinking the routing table on routers. ViAggre does not require any changes to router software and routing protocols and can be deployed independently and autonomously by any ISP. ViAggre is effectively a scalability technique that allows an ISP to modify its internal routing such that individual routers in the ISP's network only maintain a part of the global routing table.**

We evaluate the application of ViAggre to a few tier-1 and tier-2 ISPs and show that it can reduce the routing table on routers by an order of magnitude while imposing almost no traffic stretch and negligible load increase across the routers. We also deploy Virtual Aggregation on a testbed comprising of Cisco routers and benchmark this deployment. Finally, to understand and address concerns regarding the configuration overhead that our proposal entails, we implement a configuration tool that automates ViAggre configuration. While it remains to be seen whether most, if not all, of the management concerns can be eliminated through such automated tools, we believe that the simplicity of the proposal and its possible short-term impact on routing scalability suggest that it is an alternative worth considering.

I. Introduction

The Internet default-free zone (DFZ) routing table has been growing at a rapid rate for the past few years [21]. Looking ahead, there are concerns that as the IPv4 address space runs out, hierarchical aggregation of network prefixes will further deteriorate resulting in a substantial acceleration in the growth of the routing table [31]. A growing IPv6 deployment would worsen the situation even more [29].

The increase in the size of the DFZ routing table has several harmful implications for inter-domain routing.[1] [31] discusses these in detail. At a technical level, increasing routing table size may drive high-end router design into various engineering limits. For instance, while memory and processing speeds might just scale with a growing routing system, power and heat dissipation capabilities may not [30]. On the business side, the performance requirements for forwarding while being able to access a large routing table imply that the cost of forwarding packets increases and hence, networks become less cost-effective [27]. Further, it makes provisioning of networks harder since it is difficult to estimate the usable lifetime of routers, not to mention the cost of the actual upgrades. As a matter of fact, instead of upgrading their routers, a few ISPs have resorted to filtering out some small prefixes (mostly /24s) which implies that parts of the Internet may not have reachability to each other [20]. A recent proprietary conversation with a major Internet ISP revealed that in order to avoid router memory upgrades, the ISP is using a trick that reduces memory requirements but breaks BGP loop-detection and hence, would wreak havoc if adopted by other ISPs too. These anecdotes suggest that ISPs are willing to undergo some pain to avoid the cost of router upgrades.

Such concerns regarding FIB size growth, along with problems arising from a large RIB and the concomitant convergence issues, were part of the reasons that led a recent Internet Architecture Board workshop to conclude that scaling the routing system is one of the most critical challenges of near-term Internet design [30]. The severity of these problems has also prompted a slew of routing proposals [7,8,11,15,19,29,32,40]. All these proposals require changes in the routing and addressing architecture of the Internet. This, we believe, is the nature of the beast since some of the fundamental Internet design choices limit routing scalability; the overloading of IP addresses with "who" and "where" semantics represents a good example [30]. However, the very fact that they require architectural change has contributed to the non-deployment of these proposals.

This paper takes the position that a major architectural change is unlikely and it may be more pragmatic to approach the problem through a series of incremental, individually cost-effective upgrades. Guided by this and the aforementioned implications of a rapidly growing DFZ FIB, this paper proposes *Virtual Aggregation or ViAggre*, a scalability technique that focuses primarily on shrinking the FIB size on routers. ViAggre is a "configuration-only" solution that *applies to legacy routers*. Further, ViAggre can be *adopted independently and autonomously by any ISP* and hence the bar for its deployment is much lower. The key idea behind ViAggre is very simple: an ISP adopting ViAggre divides the responsibility for maintaining the global routing table amongst its routers such that individual routers only maintain a part of the routing table. Thus, this paper makes the following contributions:

[1] Hereon, we follow the terminology used in [39] and use the term "routing table" to refer to the Forwarding Information Base or FIB, commonly also known as the forwarding table. The Routing Information Base is explicitly referred to as the RIB.

- We discuss two deployment options through which an ISP can adopt ViAggre. The first one uses *FIB suppression* to shrink the FIB of all the ISP's routers while the second uses *route filtering* to shrink both the FIB and RIB on all *data-path* routers.
- We analyze the application of ViAggre to an actual tier-1 ISP and several inferred (Rocketfuel [37]) ISP topologies. We find that ViAggre can reduce FIB size by more than an order of magnitude with negligible stretch on the ISP's traffic and very little increase in load across the ISP's routers. Based on predictions of future routing table growth, we estimate that ViAggre can be used to extend the life of already outdated routers by more than 10 years.
- We propose utilizing the notion of prefix popularity to reduce the impact of ViAggre on the ISP's traffic and use a two-month study of a tier-1 ISP's traffic to show the feasibility of such an approach.
- As a proof-of-concept, we configure test topologies comprising of Cisco routers (on WAIL [3]) according to the ViAggre proposal. We use the deployment to benchmark the control-plane processing overhead that ViAggre entails. For one of the presented designs, the overhead is minimal and hence, network properties such as convergence times are not affected. The other design has high overhead due to implementation issues and needs more experimentation.
- ViAggre involves the ISP reconfiguring its routers which can be a deterrent to adoption. We quantify this configuration overhead. We also implement a configuration tool that, given the ISPs existing configuration files, can automatically generate the configuration files needed for ViAggre deployment. We discuss the use of this tool on our testbed.

Overall, the incremental version of ViAggre that this paper presents can be seen as little more than a simple and structured hack that assimilates ideas from existing work including, but not limited to, VPN tunnels and CRIO [40]. We believe that its very simplicity makes ViAggre an attractive short-term solution that provides ISPs with an alternative to upgrading routers in order to cope with routing table growth till more fundamental, long-term architectural changes can be agreed upon and deployed in the Internet. However, the basic ViAggre idea can also be applied in a clean-slate fashion to address routing concerns beyond FIB growth. While we defer the design and the implications of such a non-incremental ViAggre architecture for future work, the notion that the same concept has potential both as an immediate alleviative and as the basis for a next-generation routing architecture seems interesting and worth exploring.

II. ViAggre design

ViAggre allows individual ISPs in the Internet's DFZ to do away with the need for their routers to maintain routes for all prefixes in the global routing table. An ISP adopting ViAggre divides the global address space into a set of *virtual prefixes* such that the virtual prefixes are larger than any aggregatable (real) prefix in use today. So, for instance, an ISP could divide the IPv4 address space into 128 parts with a /7 virtual prefix representing each part (0.0.0.0/7 to 254.0.0.0/7). Note that such a naïve allocation would yield an uneven distribution of real prefixes across the virtual prefixes. However, the virtual prefixes need not be of the same length and hence, the ISP can choose them such that they contain a comparable number of real prefixes.

The virtual prefixes are not topologically valid aggregates, i.e. there is not a single point in the Internet topology that can hierarchically aggregate the encompassed prefixes. ViAggre makes the virtual prefixes aggregatable by organizing *virtual networks*, one for each virtual prefix. In other words, a virtual topology is configured that causes the virtual prefixes to be aggregatable, thus allowing for routing hierarchy that shrinks the routing table. To create such a virtual network, some of the ISP's routers are assigned to be within the virtual network. These routers maintain routes for all prefixes in the virtual prefix corresponding to the virtual network and hence, are said to be *aggregation points* for the virtual prefix. A router can be an aggregation point for multiple virtual prefixes and is required to only maintain routes for prefixes in the virtual prefixes it is aggregating.

Given this, a packet entering the ISP's network is routed to a close-by aggregation point for the virtual prefix encompassing the actual destination prefix. This aggregation point has a route for the destination prefix and forwards the packet out of the ISP's network in a tunnel. In figure 1 (figure details explained later), router C is an aggregation point for the virtual prefix encompassing the destination prefix and B $\rightarrow$ C $\rightarrow$ D is one such path through the ISP's network.

A. Design Goals

The discussion above describes ViAggre at a conceptual level. While the design space for organizing an ISP's network into virtual networks has several dimensions, this paper aims for deployability and hence is guided by two major design goals:

1) *No changes to router software and routing protocols*: The ISP should not need to deploy new data-plane or control-plane mechanisms.
2) *Transparent to external networks*: An ISP's decision to adopt the ViAggre proposal should not impact its interaction with its neighbors (customers, peers and providers).

These goals, in turn, limit what can be achieved through the ViAggre designs presented here. Routers

today have a Routing Information Base (RIB) generated by the routing protocols and a Forwarding Information Base (FIB) that is used for forwarding the packets. Consequently, the FIB is optimized for looking up destination addresses and is maintained on fast(er) memory, generally on the line cards themselves [31]. All things being equal, it would be nice to shrink both the RIB and the FIB for all ISP devices, as well as make other improvements such as speed up convergence time.

While the basic ViAggre idea can be used to achieve these benefits (section VI), we have not been able to reconcile them with the aforementioned design goals. Instead, this paper is based on the hypothesis that given the performance and monetary implications of the FIB size for routers, an immediately deployable solution that reduces FIB size is useful. Actually, one of the presented designs also shrinks the RIB on routers; only components that are off the data path (i.e. route reflectors) need to maintain the full RIB.

B. Design-I: FIB Supression

This section details one way an ISP can deploy virtual prefix based routing while satisfying the goals specified in the previous section. The discussion below applies to IPv4 (and BGPv4) although the techniques detailed here work equally well for IPv6. The key concept behind this design is to operate the ISP's internal distribution of BGP routes untouched and in particular, to populate the RIB on routers with the full routing table but to suppress most prefixes from being loaded in the FIB of routers. A standard feature on routers today is *FIB Suppression* which can be used to prevent routes for individual prefixes in the RIB from being loaded into the FIB. We have verified support for FIB suppression as part of our ViAggre deployment on Cisco 7300 and 12000 routers. Documentation for Juniper [44] and Foundry [43] routers specify this feature too. We use this as described below.

The ISP does not modify its routing setup – the ISP's routers participate in an intra-domain routing protocol that establishes internal routes through which the routers can reach each other while BGP is used for inter-domain routing just as today. For each virtual prefix, the ISP designates some number of routers to serve as aggregation points for the prefix and hence, form a virtual network. Each router is configured to only load prefixes belonging to the virtual prefixes it is aggregating into its FIB while suppressing all other prefixes.

Given this, the ISP needs to ensure that packets to any prefix can flow through the network in spite of the fact that only a few routers have a route to the prefix. This is achieved as follows:

– *Connecting Virtual Networks.* Aggregation points for a virtual prefix originate a route to the virtual prefix that is distributed throughout the ISP's network but not outside. Specifically, an aggregation point advertises the virtual prefix to its iBGP peers. A router that is not an aggregation point for the virtual prefix would choose the route advertised by the aggregation point closest to it and hence, forward packets destined to any prefix in the virtual prefix to this aggregation point.[2]

– *Sending packets to external routers.* When a router receives a packet destined to a prefix in a virtual prefix it is aggregating, it can look up its FIB to determine the route for the packet. However, such a packet cannot be forwarded in the normal hop-by-hop fashion since a router that is not an aggregation point for the virtual prefix in question might forward the packet back to the aggregation point, resulting in a loop. Hence, the packet must be tunneled from the aggregation point to the external router that was selected as the BGP NEXT_HOP. While the ISP can probably choose from many tunneling technologies, we use of MPLS Label Switched Paths (LSPs) for such tunnels. This choice was influenced by the fact that MPLS is widely supported in routers, is used by ISPs, and operates at wire speed. Further, protocols like LDP [1] automate the establishment of MPLS tunnels and hence, reduce the configuration overhead.

However, a LSP from the aggregation point to an external router would require cooperation from the neighboring ISP. To avoid this, every edge router of the ISP initiates a LSP for every external router it is connected to. Thus, all the ISP routers need to maintain LSP mappings equal to the number of external routers connected to the ISP, a number much smaller than the routes in the DFZ routing table. Note that even though the tunnel endpoint is the external router, the edge router can be configured to strip the MPLS label from the data packets before forwarding them onto the external router. This, in turn, has two implications. First, external routers don't need to be aware of the adoption of ViAggre by the ISP. Second, even the edge router does not need a FIB entry for the destination prefix, instead it chooses the external router to forward the packets to based on the MPLS label of the packet. The behavior of the edge router here is similar to the penultimate hop in a VPN scenario and is achieved through standard configuration.

We now use a concrete example to illustrate the flow of packets through an ISP network that is using ViAggre. Figure 1 shows the relevant routers. The ISP is using /7s as virtual prefixes and router C is an aggregation point for one such virtual prefix 4.0.0.0/7. Edge router D initiates a LSP to external router E with label *l* and hence,

[2]All other attributes for the routes to a virtual prefix are the same and hence, the decision is based on the IGP metric to the aggregation points. Hence, "closest" means closest in terms of IGP metric.

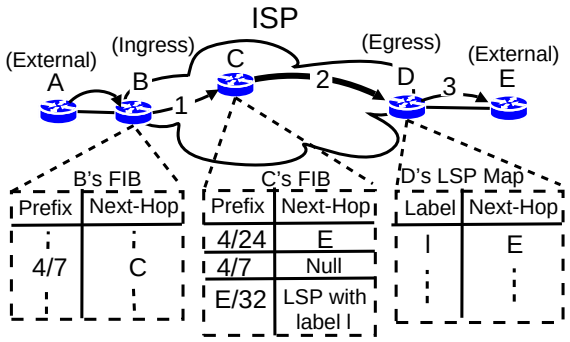


Fig. 1. Path of packets destined to prefix 4.0.0.0/24 (or, 4/24) between external routers A and E through an ISP with ViAggre. Router C is an aggregation point for virtual prefix 4.0.0.0/7 (or, 4/7).

the ISP's routers can get to E through MPLS tunneling. The figure shows the path of a packet destined to prefix 4.0.0.0/24, which is encompassed by 4.0.0.0/7, through the ISP's network. The path from the ingress router B to the external router E comprises of three segments:

1) VP-routed: Ingress router B is not an aggregation point for 4.0.0.0/7 and hence, forwards the packet to aggregation point C.
2) MPLS-LSP: Router C, being an aggregation point for 4.0.0.0/7, has a route for 4.0.0.0/24 with BGP NEXT_HOP set to E. Further, the path to router E involves tunneling the packet with MPLS label *l*.
3) Map-routed: On receiving the tunneled packet from router C, egress router D looks up its MPLS label map, strips the MPLS header and forwards the packet to external router E.

C. Design-II: Route Reflectors

The second design offloads the task of maintaining the full RIB to devices that are off the data path. Many ISPs use route-reflectors for scalable internal distribution of BGP prefixes and we require only these route-reflectors to maintain the full RIB. For ease of exposition, we assume that the ISP is already using per-PoP route reflectors that are off the data path, a common deployment model for ISPs using route reflectors.

In the proposed design, the external routers connected to a PoP are made to peer with the PoP's route-reflector.[3] This is necessary since the external peer may be advertising the entire DFZ routing table and we don't want all these routes to reside on any given data-plane router. The route-reflector also has iBGP peerings with other route-reflectors and with the routers in its PoP. Egress filters are used on the route-reflector's peerings with the PoP's routers to ensure that a router only gets routes for the prefixes it is aggregating. This shrinks both the RIB and the FIB on the routers. The data-plane operation and hence, the path of packets through the ISP's network remains the same as with the previous design.

With this design, a PoP's route-reflector peers with all the external routers connected to the PoP. The RIB size on a BGP router depends on the number of peers it has and hence, the RIB for the route-reflectors can potentially be very large. If needed, the RIB requirements can be scaled by using multiple route-reflectors. Note that the RIB scaling properties here are better than in the status quo. Today, edge routers have no choice but to peer with the directly connected external routers and maintain the resulting RIB. Replicating these routers is prohibitive because of their cost but the same does not apply to off-path route-reflectors, which could even be BGP software routers.

D. Design Comparison

As far as the configuration is concerned, configuring suppression of routes on individual routers in design-I is comparable, at least in terms of complexity, to configuring egress filters on the route-reflectors. In both cases, the configuration can be achieved through BGP route-filtering mechanisms (access-lists, prefix-lists, etc.).

Design-II, apart from shrinking the RIB on the routers, does not require the route suppression feature on routers. Further, as we detail in section V-B, the specific filtering mechanism that we use for FIB suppression on the routers in our deployment leads to high CPU utilization at the peering establishment time and hence, requires more experimentation. However, design-II does require the ISP's eBGP peerings to be reconfigured which, while straightforward, violates our goal of not impacting neighboring ISPs.

E. Network Robustness

ViAggre causes packets to be routed through an aggregation point which leads to robustness concerns. When an aggregation point for a virtual prefix fails, routers using that aggregation point are re-routed to another aggregation point through existing mechanisms without any explicit configuration by the ISP. In case of design-I, a router has routes to all aggregation points for a given virtual prefix in its RIB and hence, when the aggregation point being used fails, the router installs the second closest aggregation point into its FIB and packets are re-routed almost instantly. With design-II, it is the route-reflector that chooses the alternate aggregation point and advertises this to the routers in its PoP. Hence, as long as another aggregation point exists, failover happens automatically and at a fast rate.

F. Routing popular prefixes natively

The use of aggregation points implies that packets in ViAggre may take paths that are longer than native paths.

[3]Note that these will be eBGP multihop peerings since the route-reflector is not directly connected to the external routers.

Apart from the increased path length, the packets incur queuing delay at all the extra hops. The extra hops also result in an increase in load on the ISP's routers and links and a modification in the distribution of traffic across them

Past studies have shown that a large majority of Internet traffic is destined to a very small fraction of prefixes [10,13,34,38]. The fact that routers today have no choice but to maintain the complete DFZ routing table implies that this observation wasn't very useful for routing configuration. However, with ViAggre, individual routers only need to maintain routes for a fraction of prefixes. The ISP can thus configure its ViAggre setup such that the small fraction of popular prefixes are in the FIB of every router and hence, are routed natively. For design-I, this involves configuring each router with a set of popular prefixes that should not be suppressed from the FIB. For design-II, a PoP's route-reflector can be configured to not filter advertisements for popular prefixes from the PoP's routers. Beyond this, the ISP may also choose to install customer prefixes into its routers such that they don't incur any stretch. The rest of the proposal involving virtual prefixes remains the same and ensures that individual routers only maintain routes for a fraction of the unpopular prefixes. In section IV-B.4, we analyze Netflow data from a tier-1 ISP network to show that not only such an approach is feasible, it also addresses all the concerns raised above.

III. Allocating aggregation points

An ISP adopting ViAggre would obviously like to minimise the stretch imposed on its traffic. Ideally, an ISP would deploy an aggregation point for all virtual prefixes in each of its PoPs. This would ensure that for every virtual prefix, a router chooses the aggregation point in the same PoP and hence, the traffic stretch is minimal. However, this is often not possible in practice. This is because ISPs, including tier-1 ISPs, often have some small PoPs with just a few routers and therefore there may not be enough cumulative FIB space in the PoP to hold all the actual prefixes. Hence, the ISP needs to be smart about the way it designates routers to aggregate virtual prefixes and in this section we explore this choice.

A. Problem Formulation

We first introduce the notation used in the rest of this section. Let *T* represent the set of prefixes in the Internet routing table, *R* be the set of ISP's routers and *X* is the set of external routers directly connected to the ISP. For each $r \in R$, P_r represents the set of popular prefixes for router r. *V* is the set of virtual prefixes chosen by the ISP and for each $v \in V$, n_v is the number of prefixes in v. We use two matrices, $D = (d_{i,j})$ that gives the distance between routers i and j and $W = (w_{i,j})$ that gives the IGP metric for the IGP-established path between routers i and j. We also define two relations:

– "BelongsTo" relation *B*: $T \rightarrow V$ such that $B(p)=v$ if prefix p belongs to or is encompassed by virtual prefix v.

– "Egress" relation *E*: $R \times T \rightarrow R$ such that $E(i, p)=j$ if traffic to prefix p from router i egresses at router j.

The mapping relation *A*: $R \rightarrow 2^V$ captures how the ISP assigns aggregation points; i.e. $A(r) = \{v_1 \dots v_n\}$ implies that router r aggregates virtual prefixes $\{v_1 \dots v_n\}$. Given this assignment, we can determine the aggregation point any router uses for its traffic to each virtual prefix. This is captured by the "Use" relation *U*: $R \times V \rightarrow R$ where $U(i, v) = j$ or router i uses aggregation point j for virtual prefix v if the following conditions are satisfied:

$$\begin{array}{lrcll} 1) & v & \in & A(j) & \\ 2) & w_{i,j} & \leq & w_{i,k} & \forall k \in R, v \in A(k) \end{array}$$

Here, condition 1) ensures that router j is an aggregation point for virtual prefix v. Condition 2) captures the operation of BGP with design-I and ensures that a router chooses the aggregation point that is closest in terms of IGP metrics.[4]

Using this notation, we can express the FIB size on routers and the stretch imposed on traffic.

1) Routing State: In ViAggre, a router needs to maintain routes to the (real) prefixes in the virtual prefixes it is aggregating, routes to all the virtual prefixes themselves and routes to the popular prefixes. Further, the router needs to maintain LSP mappings for LSPs originated by the ISP's edge routers with one entry for each external router connected to the ISP. Hence, the "routing state" for the router r, hereon simply referred to as the FIB size (F_r), is given by:

$$F_r = \sum_{v \in A(r)} n_v + |V| + |P_r| + |X|$$

The **Worst FIB size** and the **Average FIB size** are defined as follows:

$$\begin{array}{rcl} \text{Worst FIB size} & = & \max_{r \in R}(F_r) \\ \text{Average FIB size} & = & \sum_{r \in R}(F_r)/|R| \end{array}$$

2) Traffic Stretch: If router i uses router k as an aggregation point for virtual prefix v, packets from router i to a prefix p belonging to v are routed through router k. Hence, the stretch (*S*) imposed on traffic to prefix p from router i is given by:

$$\begin{array}{rcll} S_{i,p} & = & 0, & p \in P_i \\ & = & (d_{i,k} + d_{k,j} - d_{i,j}), & p \in (T - P_i),\ v = B(p) \\ & & & k = U(i, v)\ \&\ j = E(k, p) \end{array}$$

[4]With design-II, a router chooses the aggregation point closest to the router's route-reflector in terms of IGP metrics and so a similar formulation works for the second design too.

The **Worst Stretch** and **Average Stretch** are defined as follows:

$$\text{Worst Stretch} = \max_{i \in R, p \in T}(S_{i,p})$$

$$\text{Average Stretch} = \sum_{i \in R, p \in T} (S_{i,p})/(|R| * |T|)$$

Problem: ViAggre shrinks the routing table on routers by ensuring that individual routers only maintain routes to a fraction of the prefixes and forward packets to an aggregation point for the rest. Thus, through the use of aggregation points, ViAggre trades off an increase in path length for a reduction in routing state. The ISP can use the assignment of aggregation points as a knob to tune this trade-off. Here we consider the simple goal of minimising the FIB Size on the ISP's routers while bounding the stretch. Specifically, the ISP needs to assign aggregation points by determining a mapping *A* that

$$\begin{array}{ll} \min & \text{Worst FIB Size} \\ \text{s.t.} & \text{Worst Stretch} \leq C \end{array}$$

where C is the specified constraint on Worst Stretch. Note that much more complex formulations are possible. Our focus on worst-case metrics is guided by practical concerns – the Worst FIB Size dictates how the ISP's routers need to be provisioned while the Worst Stretch characterizes the most unfavorable impact of the use of ViAggre. Specifically, bounding the Worst Stretch allows the ISP to ensure that its existing SLAs are not breached and applications sensitive to increase in latency (example, VOIP) are not adversely affected.

B. A Greedy Solution

The problem of assigning aggregation points while satisfying the conditions above can be mapped to the MultiCommodity Facility Location (MCL) problem [33]. Using the MCL terminology, this involves routers representing facilities, virtual prefixes being commodities and each router's traffic to virtual prefixes serving as clients. MCL is NP-hard and [33] presents a logarithmic approximation algorithm for it. Here we discuss a greedy approximation solution to the problem.

The first solution step is to determine that if router i were to aggregate virtual prefix v, which routers can it serve without violating the stretch constraint. This is the $can\_serve_{i,v}$ set and is defined as follows:

$$can\_serve_{i,v} = \{j \mid j \in R, (\forall p \in T, B(p) = v, E(i,p) = k, (d_{j,i} + d_{i,k} - d_{j,k}) \leq C)\}$$

Given this, the key idea behind the solution is that any assignment based on the can_serve relation will have Worst Stretch less than C. Hence, our algorithm designates routers to aggregate virtual prefixes in accordance with the can_serve relation while greedily trying to minimise the Worst FIB Size. The algorithm, shown below, stops when each router can be served by at least one aggregation point for each virtual prefix.

$Worst\_FIB\_Size$=0
for all r in R **do**
  for all v in V **do**
    Calculate $can\_serve_{r,v}$
Sort V in decreasing order of n_v
for all v in V **do**
  Sort R in decreasing order of $|can\_serve_{r,v}|$
  repeat
    for all r in R **do**
      if $(F_r + n_v) \leq Worst\_FIB\_Size$ **then**
        A[r]=A[r] $\cup$ v # Assign v to r
        $F_r = F_r + n_v$ # r's FIB size increases
        Mark all routers in $can\_serve_{r,v}$ as served
        if All routers are served for v **then**
          break
    if All routers are not served for v **then**
      # $Worst\_FIB\_Size$ needs to be raised
      for all r in R **do**
        if v $\notin$ A[r] **then**
          # r is not an aggregation point for v
          A[r]=A[r] $\cup$ v
          $F_r = F_r + n_v$
          $Worst\_FIB\_Size = F_r$
          break
  until All Routers are served for virtual prefix v

IV. Evaluation

In this section we evaluate the application of ViAggre to a few Internet ISPs.

A. Metrics of Interest

We defined (**Average** and **Worst**) **FIB Size** and **Stretch** metrics in section III-A. Here we define other metrics that we use for ViAggre evaluation.

1) Impact on Traffic: Apart from the stretch imposed, another aspect of ViAggre's impact is the amount of traffic affected. To account for this, we define **traffic impacted** as the fraction of the ISP's traffic that uses a different router-level path than the native path. Note that in many cases, a router will use an aggregation point for the destination virtual prefix in the same PoP and hence, the packets will follow the same PoP-level path as before. Thus, another metric of interest is the **traffic stretched**, the fraction of traffic that is forwarded along a different PoP-level path than before. In effect, this represents the change in the distribution of traffic across the ISP's inter-PoP links and hence, captures how ViAggre interferes with the ISP's inter-PoP traffic engineering.

2) Impact on Router Load: The extra hops traversed by traffic increases the traffic load on the ISP's routers. We define the **load increase** across a router as the extra traffic it needs to forward due to ViAggre, as a fraction of the traffic it forwards natively.

B. Tier-1 ISP Study

We analysed the application of ViAggre to a large tier-1 ISP in the Internet. For our study, we obtained the ISP's router-level topology (to determine router set *R*) and the routing tables of routers (to determine prefix set *T* and the Egress *E* and BelongsTo *B* relations). We used information about the geographical locations of the routers to determine the Distance matrix *D* such that $d_{i,j}$ is 0 if routers i and j belong to the same PoP (and hence, are in the same city) else $d_{i,j}$ is set to the propagation latency corresponding to the great circle distance between i and j. Further, we did not have information about the ISP's link weights. However, guided by the fact that intra-domain traffic engineering is typically latency-driven [36], we use the Distance matrix *D* as the Weight matrix *W*. We also obtained the ISP's traffic matrix; however, in order to characterise the impact of vanilla ViAggre, the first part of this section assumes that the ISP does not consider any prefixes as popular.

1) Deployment decisions: The ISP, in order to adopt ViAggre, needs to decide what virtual prefixes to use and which routers aggregate these virtual prefixes. We describe the approaches we evaluated.

– *Determining set V.* The most straightforward way to select virtual prefixes while satisfying the two conditions specified in section II is to choose large prefixes (/6s, /7s, etc.) as virtual prefixes. We assume that the ISP uses /7s as its virtual prefixes and refer to this as the "/7 allocation".

However, such selection of virtual prefixes could lead to a skewed distribution of (real) prefixes across them with some virtual prefixes containing a large number of prefixes. For instance, using /7s as virtual prefixes implies that the largest virtual prefix (202.0.0.0/7) contains 22,772 of the prefixes in today's BGP routing table or 8.9% of the routing table. Since at least one ISP router needs to aggregate each virtual prefix, such large virtual prefixes would inhibit the ISP's ability to reduce the Worst FIB size on its routers. However, as we mentioned earlier, the virtual prefixes need not be of the same length and so large virtual prefixes can be split to yield smaller virtual prefixes. To study the effectiveness of this approach, we started with /7s as virtual prefixes and split each of them such that the resulting virtual prefixes were still larger than any prefix in the Internet routing table. This yielded 1024 virtual prefixes with the largest containing 4,551 prefixes or 1.78% of the BGP routing table. We also use this virtual prefix allocation for our evaluation and refer to it as "Uniform Allocation".

– *Determining mapping A.* We implemented the algorithm described in section III-B and use it to designate routers to aggregate virtual prefixes.

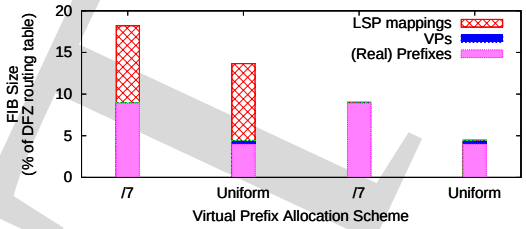
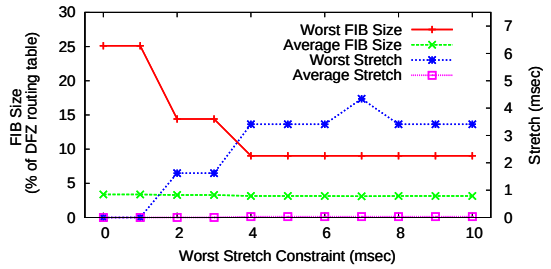


Fig. 2. FIB composition for the router with the largest FIB, C=4ms and no popular prefixes.

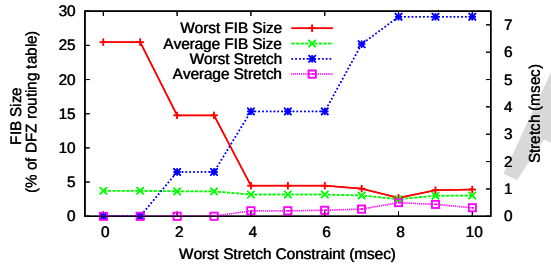
2) Router FIB: We first look at the size and the composition of the FIB on the ISP's routers with a ViAggre deployment. Specifically, we focus on the router with the largest FIB for a deployment where the worst-case stretch (C) is constrained to 4ms. The first two bars in figure 2 show the FIB composition for a /7 and uniform allocation respectively. With a /7 allocation, the router's FIB contains 46,543 entries which represents 18.2% of the routing table today. This includes 22,772 prefixes, 128 virtual prefixes, 23,643 LSP mappings and 0 popular prefixes. As can be seen, in both cases, the LSP mappings for tunnels to the external routers contribute significantly to the FIB. This is because the ISP has a large number of customer routers that it has peerings with.

However, we also note that customer ISPs do not advertise the full routing table to their provider. Hence, edge routers of the ISP could maintain routes advertised by customer routers in their FIB, advertise these routes onwards with themselves as the BGP NEXT_HOP and only initiate LSP advertisements for themselves and for peer and provider routers connected to them. With such a scheme, the number of LSP mappings that the ISP's routers need to maintain and the MPLS overhead in general reduces significantly. The latter set of bars in figure 2 shows the FIB composition with such a deployment for the router with the largest FIB. For the /7 allocation, the Worst FIB size is 23,101 entries (9.02% of today's routing table) while for the Uniform allocation, it is 10,226 entries (4.47%). In the rest of this section, we assume this model of deployment.

3) Stretch Vs. FIB Size: We ran the assignment algorithm with Worst Stretch Constraint (*C*) ranging from 0 to 10 ms and determined the (Average and Worst) Stretch and FIB Size of the resulting ViAggre deployment. Figure 3(a) plots these metrics for the /7 allocation. The Worst FIB size, shown as a fraction of the DFZ routing table size today, expectedly reduces as the constraint on Worst Stretch is relaxed. However, beyond *C*=4ms, the Worst FIB Size remains constant. This is because the largest virtual prefix with a /7 allocation encompasses 8.9% of the DFZ routing table and the Worst FIB Size cannot be any less than 9.02% (0.12% overhead is due to virtual prefixes and LSP mappings). Figure 3(b) plots the same metrics for the Uniform allocation and shows



(a) With /7 allocation



(b) With Uniform allocation

Fig. 3. Variation of FIB Size and Stretch with Worst Stretch constraint and no popular prefixes.

		Today	ViAggre			
	Worst Stretch (ms)	–	0	2	4	8
239K FIB	Quad. Fit	Expired	2015	2020	2039	2051
	Expo. Fit	Expired	2018	2022	2031	2035
1M FIB	Quad. Fit	2015	2033	2044	2081	2106
	Expo. Fit	2018	2029	2033	2042	2046

TABLE I

ESTIMATES FOR ROUTER LIFE WITH VIAGGRE

that the FIB can be shrunk even more. The figure also shows that the Average FIB Size and the Average stretch are expectedly small throughout. The anomaly beyond C=8msec in figure 3(b) results from the fact that our assignment algorithm is an approximation that can yield non-optimal results.

Another way to quantify the benefits of ViAggre is to determine the extension in the life of a router with a specified memory due to the use of ViAggre. As proposed in [22], we used data for the DFZ routing table size from Jan'02 to Dec'07 [21] to fit a quadratic model to routing table growth. Further, it has been claimed that the DFZ routing table has seen exponential growth at the rate of 1.3x every two years for the past few years and will continue to do so [30]. We use these models to extrapolate future DFZ routing table size. We consider two router families: Cisco's Cat6500 series with a supervisor 720-3B forwarding engine that can hold upto 239K IPv4 FIB entries and hence, was supposed to be phased out by mid-2007 [6], though some ISPs still continue to use them. We also consider Cisco's current generation of routers with a supervisor 720-3BXL engine that can hold 1M IPv4 FIB entries. For each of these router families, we calculate the year to which they would be able to cope with the growth in the DFZ routing table with the existing setup and with ViAggre. Table I

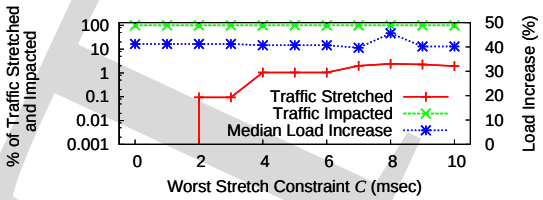


Fig. 4. Variation of the percentage of traffic stretched/impacted and load increase across routers with Worst Stretch Constraint (Uniform Allocation) and no popular prefixes.

shows the results for the Uniform Allocation.

For ViAggre, relaxing the worst-case stretch constraints reduces FIB size and hence, extends the router life. The table shows that if the DFZ routing table were to grow at the aforementioned exponential rate, ViAggre can extend the life of the previous generation of routers to 2018 with no stretch at all. We realise that estimates beyond a few years are not very relevant since the ISP would need to upgrade its routers for other reasons such as newer technologies and higher data rates anyway. However, with ViAggre, at least the ISP is not forced to upgrade due to growth in the routing table.

Figure 4 plots the impact of ViAggre on the ISP's traffic and router load. The percentage of traffic stretched is small, less than 1% for $C \leq 6$ ms. This shows that almost all the traffic is routed through an aggregation point in the same PoP as the ingress. However, the fact that no prefixes are considered popular implies that almost all the traffic follows a different router-level path as compared to the status quo. This shows up in figure 4 since the traffic impacted is ≈100% throughout. This, in turn, results in a median increase in load across the routers by ≈39%. In the next section we discuss how an ISP can use the skewed distribution of traffic to address the load concern while maintaining a small FIB on its routers.

4) Popular Prefixes: Past studies of ISP traffic patterns from as early as 1999 have observed that a small fraction of Internet prefixes carry a large majority of ISP traffic [10,13,34,38]. We used Netflow records collected across the routers of the same tier-1 ISP as in the last section for a period of two months (20^{th} Nov'07 to 20^{th} Jan'07) to generate per-prefix traffic statistics and observed that this pattern continues to the present day. The line labeled "Day-based, ISP-wide" in figure 5 plots the average fraction of the ISP's traffic destined to a given fraction of popular prefixes when the set of popular prefixes is calculated across the ISP on a daily basis. The figure shows that 1.5% of most popular prefixes carry 75.5% of the traffic while 5% of the prefixes carry 90.2% of the traffic.

ViAggre exploits the notion of prefix popularity to reduce its impact on the ISP's traffic. However, the ISP's routers need not consider the same set of prefixes as popular; instead the popular prefixes can be chosen per-

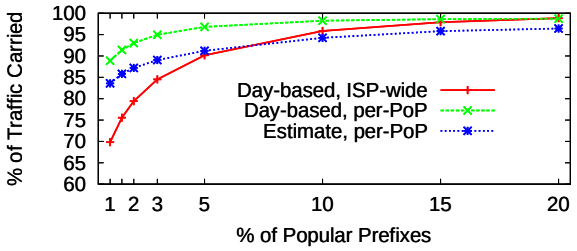
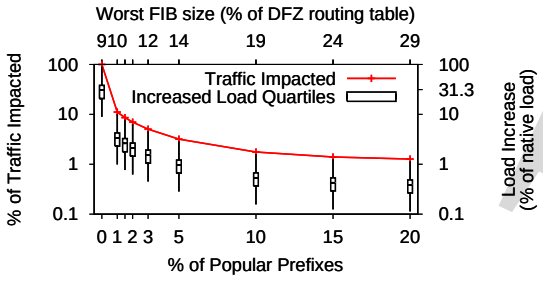
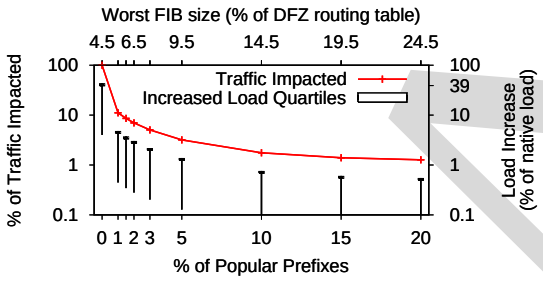


Fig. 5. Popular prefixes carry a large fraction of the ISP's traffic.



(a) With /7 allocation



(b) With Uniform allocation

Fig. 6. Variation of Traffic Impacted and Load Increase (0-25-50-75-100 percentile) with percentage of popular prefixes, C=4ms.

PoP or even per-router. We calculated the fraction of traffic carried by popular prefixes, when popularity is calculated separately for each PoP on a daily basis. This is plotted in the figure as "Day-based, per-PoP" and the fractions are even higher.[5]

When using prefix popularity for router configuration, it would be preferable to be able to calculate the popular prefixes over a week, month, or even longer durations. The line labeled "Estimate, per-PoP" in the figure shows the amount of traffic carried to prefixes that are popular on a given day over the period of the next month, averaged over each day in the first month of our study. As can be seen, the estimate based on prefixes popular on any given day carries just a little less traffic as when the prefix popularity is calculated daily. This suggests that prefix popularity is stable enough for ViAggre configuration and the ISP can use the prefixes that are popular on a given day for a month or so. However, we admit that that these results are very preliminary and we need to study ISP traffic patterns over a longer period to substantiate the claims made above.

[5]We did not have Netflow records for individual routers and hence, were unable to generate router-specific popular prefixes.

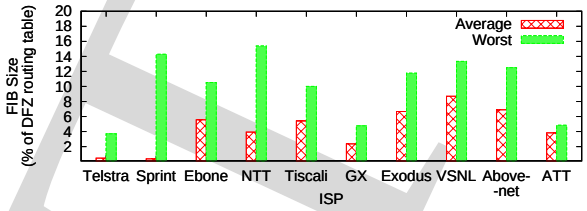


Fig. 7. FIB size for various ISPs using ViAggre.

5) Load Analysis: We now consider the impact of a ViAggre deployment involving popular prefixes, i.e. the ISP populates the FIB on its routers with popular prefixes. Specifically, we focus on a deployment wherein the aggregation points are assigned to constrain Worst Stretch to 4ms, i.e. C = 4ms. Figure 6 shows how the traffic impacted and the quartiles for the load increase vary with the percentage of popular prefixes for both allocations. Note that using popular prefixes increases the router FIB size by the number of prefixes considered popular and thus, the upper X-axis in the figure shows the Worst FIB size. The large fraction of traffic carried by popular prefixes implies that both the traffic impacted and the load increase drops sharply even when a small fraction of prefixes is considered popular. For instance, with 2% popular prefixes in case of the uniform allocation (figure 6(b)), 7% of the traffic follows a different router-level path than before while the largest load increase is 3.1% of the original router load. With 5% popular prefixes, the largest load increase is 1.38%. Note that the more even distribution of prefixes across virtual prefixes in the uniform allocation results in a more even distribution of the excess traffic load across the ISP's routers – this shows up in the load quartiles being much smaller in figure 6(b) as compared to the ones in figure 6(a).

C. Rocketfuel Study

We studied the topologies of 10 ISPs collected as part of the Rocketfuel project [37] to determine the FIB size savings that ViAggre would yield. Note that the fact we don't have traffic matrices for these ISPs implies that we cannot analyze the load increase across their routers. For each ISP, we used the assignment algorithm to determine the worst FIB size resulting from a ViAggre deployment where the worst stretch is limited to 5ms. Figure 7 shows that the worst FIB size is always less than 15% of the DFZ routing table. The FIB size is relatively higher for NTT and Sprint because they have a global footprint with a few small PoPs outside their main area of influence. For instance, Sprint has a few small PoPs in the Asia-Pacific region. The constraint on the worst stretch implies that in many cases, the traffic from these PoPs cannot be routed to an aggregation point in another PoP and so these PoPs must have aggregation points for all virtual prefixes. Consequently, the routers in these PoPs end up with a relatively large FIB. However, the Rocketfuel topologies

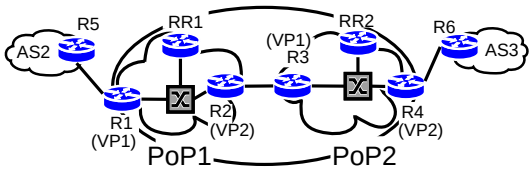


Fig. 8. WAIL topology used for our deployment. All routers in the figure are Cisco 7300s. RR1 and RR2 are route-reflectors and are not on the data path. Routers R1 and R3 aggregate virtual prefix VP1 while routers R2 and R4 aggregate VP2.

are not complete and are missing routers. Hence, while the results presented here are encouraging, they should be treated as conservative estimates of the savings that ViAggre would yield for these ISPs.

D. Discussion

The analysis above shows that ViAggre can significantly reduce FIB size. Most of the ISPs we studied are large tier-1 and tier-2 ISPs. However, smaller tier-2 and tier-3 ISPs are also part of the Internet DFZ. Actually, it is probably more important for such ISPs to be able to operate without needing to upgrade to the latest generation of routers. The fact that these ISPs have small PoPs might suggest that ViAggre would not be very beneficial. However, given their small size, the PoPs of these ISPs are typically geographically close to each other. Hence, it is possible to use the cumulative FIB space across routers of close-by PoPs to shrink the FIB substantially. And the use of popular prefixes ensures that the load increase and the traffic impact is still small. For instance, we analyzed router topology and routing table data from a regional tier-2 ISP (AS2497) and found that a ViAggre deployment with worst stretch less than 5ms can shrink the Worst FIB size to 14.2% of the routing table today.

Further, the fact that such ISPs are not tier-1 ISPs implies they are a customer of at least one other ISP. Hence, in many cases, the ISP could substantially shrink the FIB size on its routers by applying ViAggre to the small number of prefixes advertised by their customers and peers while using default routes for the rest of the prefixes.

V. Deployment

To verify the claim that ViAggre is a configuration-only solution, we deployed both ViAggre designs on a small network built on the WAIL testbed [3]. The test network is shown in figure 8 and represents an ISP with two PoPs. Each PoP has two Cisco 7301 routers and a route-reflector.[6] For the ViAggre deployment, we use two virtual prefixes: 0.0.0.0/1 (VP1) and 128.0.0.0/1 (VP2) with one router in each PoP serving as an aggregation point for each virtual prefix. Routers R1 and R4 have an external router connected to them and exchange routes using an eBGP peering. Specifically, router R5 advertises the entire DFZ routing table and this is, in turn, advertised through the ISP to router R6. We use OSPF for intra-domain routing. Beyond this, we configure the internal distribution of BGP routes according to the following three approaches:

1). **Status Quo.** The routers use a mesh of iBGP peerings to exchange the routes and hence, each router maintains the entire routing table.

2). **Design-I.** The routers still use a mesh of iBGP peerings to exchange routes. Beyond this, the routers are configured as follows:

– *Virtual Prefixes.* Routers advertise the virtual prefix they are aggregating to their iBGP peers.

– *FIB Suppression.* Each router only loads the routes that it is aggregating into its FIB. For instance, router R1 uses an `access-list` to specify that only routes belonging to VP1, the virtual prefix VP2 itself and any popular prefixes are loaded into the FIB. A snippet of this access-list is shown below.

```
! R5's IP address is 198.18.1.200
distance 255 198.18.1.200 0.0.0.0 1

! Don't mark anything inside 0.0.0.0/1
access-list 1 deny 0.0.0.0 128.255.255.255
! Don't mark virtual prefix 128.0.0.0/1
access-list 1 deny 0.0.0.0 128.0.0.0
! Don't mark popular prefix 122.1.1.0/24
access-list 1 deny 122.1.1.0 0.0.0.255
! ... other popular prefixes follow ...

! Mark the rest with admin distance 255
access-list 1 permit any
```

Here, the `distance` command sets the administrative distance of all prefixes that are accepted by `access-list` 1 to "255" and these routes are not loaded by the router into its FIB.

– *LSPs to external routers.* We use MPLS for the tunnels between routers. To this effect, LDP [1] is enabled on the interfaces of all routers and establishes LSPs between the routers. Further, each edge router (R1 and R4) initiates a Downstream Unsolicited tunnel [1] for each external router connected to them to all their IGP neighbors using LDP. This ensures that packets to an external router are forwarded using MPLS to the edge router which strips the MPLS header before forwarding them onwards.

Given this setup and assuming no popular prefixes, routers R1 and R3 store 40.9% of today's routing table (107,943 prefixes that are in VP1) while R2 and R4 store 59.1%.

3). **Design-II.** The routers in a PoP peer with the route-reflector of the PoP and the route-reflectors peer with each other. External routers R1 and R6 are reconfigured to have eBGP peerings with RR1 and RR2 respectively. The advertisement of virtual prefixes and the MPLS

[6]These are used only for the design-II deployment. We used both a Cisco 7301 and a Linux PC as a route-reflector.

configuration is the same as above. Beyond this, the route-reflectors are configured to ensure that they only advertise the prefixes being aggregated by a router to it. For instance, RR1 uses a `prefix-list` to ensure that only prefixes belonging to VP1, virtual prefix VP2 itself and popular prefixes are advertised to router R1. The structure of this prefix-list is similar to the access-list shown above. Finally, route-reflectors use a route-map on their eBGP peerings to change the BGP NEXT_HOP of the advertised routes to the edge router that the external peer is connected too. This ensures that the packets don't actually flow through the route-reflectors.

A. Configuration Overhead

A drawback of ViAggre being a "configuration-only" approach is the overhead that the extra configuration entails. The discussion above details the extra configuration that routers need to participate in ViAggre. Based on our deployment, the number of extra configuration lines needed for a router r to be configured according to design-I is given by $(r_{int} + r_{ext} + 2|A(r)| + |P_r| + 6)$ where r_{int} is the number of router interfaces, r_{ext} is the number of external routers r is peering with, $|A(r)|$ is the number of virtual prefixes r is aggregating and $|P_r|$ is the number of popular prefixes in r. Given the size of the routing table today, considering even a small fraction of prefixes as popular would cause the expression to be dominated by $|P_r|$ and can represent a large number of configuration lines.

However, quantifying the extra configuration lines does not paint the complete picture since given a list of popular prefixes, it is trivial to generate an access or prefix-list that would allow them. To illustrate this, we developed a configuration tool as part of our deployment effort. The tool is 334 line python script which takes as input a router's existing configuration file, the list of virtual prefixes, the router's (or representative) Netflow records and the percentage of prefixes to be considered popular. The tool extracts relevant information, such as information about the router's interfaces and peerings, from the configuration file. It also uses the Netflow records to determine the list of prefixes to be considered popular. Based on these extracted details, the script generates a configuration file that allows the router to operate as a ViAggre router. We have been using this tool for experiments with our deployment and it is available at [45]. Further, we use *clogin* [42] to automatically load the generated ViAggre configuration file onto the router. Thus, we can reconfigure our testbed from status quo operation to ViAggre operation (design-I and design II) in an automated fashion. While our tool is specific to the router vendor and other technologies in our deployment, its simplicity and our experience with it lends evidence to the argument that ViAggre offers a good trade-off between the configuration overhead and increased routing scalability.

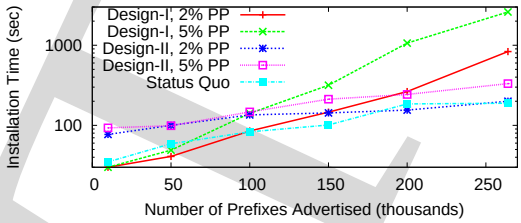


Fig. 9. Installation time with different approaches and varying fraction of Popular Prefixes (PP).

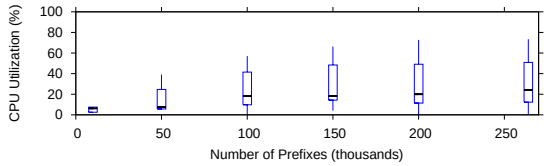
B. Control-plane Overhead

Section IV evaluated the impact of ViAggre on the ISP's data plane. Beyond this, ViAggre uses control-plane mechanisms to divide the routing table amongst the ISP's routers – Design-I uses `access-lists` and Design-II uses `prefix-lists`. We quantify the performance overhead imposed by these mechanisms using our deployment. Specifically, we look at the impact of our designs on the propagation of routes through the ISP.

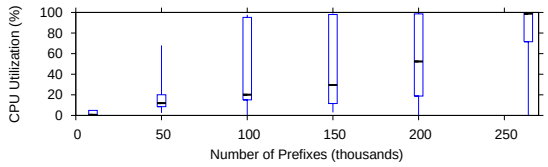
To this effect, we configured the internal distribution of BGP routes in our testbed according to the three approaches described above. External router R5 is configured to advertise a variable number of prefixes through its eBGP peering. We restart this peering on router R5 and measure the time it takes for the routes to be installed into the FIB of the ISP's routers; hereon we refer to this as the *installation time*. During this time, we also measure the CPU utilization on the routers. We achieve this by using a clogin script to execute the "*show process cpu*" command on each router every 5 seconds. The command gives the average CPU utilization of individual processes on the router over the past 5 seconds and we extract the CPU utilization of the "BGP router" process.

We measured the installation time and the CPU utilization for the three approaches. For status quo and design-I, we focus on the measurements for router R1 while for design-II, we focus on the measurements for route-reflector RR1. We also varied the number of popular prefixes. Here we present results with 2% and 5% popular prefixes. Figures 9 and 10 plot the installation time and the quartiles for the CPU utilization respectively.

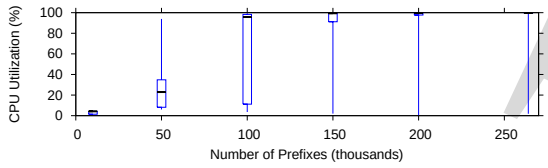
Design-I Vs Status Quo. Figure 9 shows that the installation time with design-I is much higher than that with status quo. For instance, with status quo, the complete routing table is transferred and installed on router R1 in 189 seconds while with design-I and 2% popular prefixes, it takes 834 seconds. Further, the design-I installation time increases significantly as the number of popular prefixes increases. Finally, figures 10(b) and 10(c) show that design-I results in very



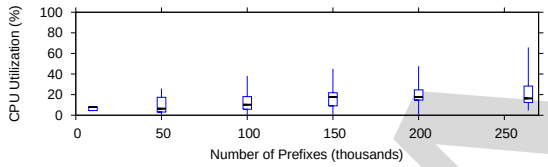
(a) Status Quo, Measured on R1



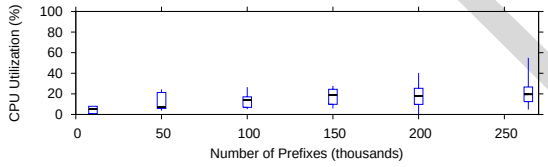
(b) Design-I, 2% PP, Measured on R1



(c) Design-I, 5% PP, Measured on R1



(d) Design-II, 2% PP, Measured on RR1



(e) Design-II, 5% PP, Measured on RR1

Fig. 10. CPU Utilization quartiles (0-25-50-75-100 percentile) for the three approaches and different fraction of Popular Prefixes (PP).

high CPU load during the transfer which increases as more prefixes are considered popular. This results from the fact that access-lists with a large number of rules are very inefficient and would obviously be unacceptable for an ISP deploying ViAggre. While we are currently exploring ways to achieve FIB suppression without the use of access-list, we note that the performance of access-lists has been improved on current generation Cisco routers (12000 and onwards) [41].

Design-II Vs Status Quo. Figure 9 shows that the time to transfer and install routes with design-II is not much higher than status quo, especially with 2% popular prefixes and a large number of advertised routes. For instance, design-II with 2% popular prefixes leads to an installation time of 200 seconds for the entire routing table as compared to 189 seconds for status quo. Figures 10(d) and 10(e) show that the CPU utilization is low with median utilization being less than 20%. We note that the increasing the number of prefixes being advertised increases the number of popular prefixes which, in turn, increases the size of the prefix-list being used. The CPU utilization increases as the number of prefixes advertised increases and then tapers off. Further, the trend is similar to status quo (figure 10(a)). Also note that the utilization shown for design-II was measured on route-reflector RR1 which has fewer peerings than router R1 in status quo. This explains the fact that the utilization with design-II is less than status quo.

C. Failover

As detailed in section II-E, as long as alternate aggregation points exist, traffic in a ViAggre network is automatically re-routed upon failure of the aggregation point being used. We measured this failover time using our testbed. In the interest of space, we very briefly summarise the experiment here. We generated UDP traffic between PCs connected to routers R5 and R6 (figure 8) and then crashed the router being used as the aggregation point for the traffic. We measured the time it takes for traffic to be re-routed over 10 runs with each design. In both cases, the maximum observed failover time was 200 usecs. This shows that our designs ensure fast failover between aggregation points.

VI. Discussion

Pros. ViAggre can be *incrementally deployed* by an ISP since it does not require the cooperation of other ISPs and router vendors. The ISP does not need to change the structure of its PoPs or its topology. What's more, an ISP could experiment with ViAggre on a limited scale (a few virtual prefixes or a limited number of PoPs) to gain experience and comfort before expanding its deployment. None of the attributes in the BGP routes advertised by the ISP to its neighbors are changed due to the adoption of ViAggre. Also, the use of ViAggre by the ISP does not restrict its routing policies and route selection. Further, at least for design-II, the control-plane overhead is minimal and hence, properties such as convergence times are similar. Finally, there is *incentive for deployment* since the ISP improves its own capability to deal with routing table growth.

Management Overhead. As detailed in section V-A, ViAggre requires extra configuration on the ISP's routers. Beyond this, the ISP needs to make a number of deployment decisions such as choosing the virtual prefixes to use, deciding where to keep aggregation points for each virtual prefix, and so on. Apart from such one-time or infrequent decisions, ViAggre may also influence very important aspects of the ISP's day-to-day operation such as maintenance, debugging, etc. All this leads to increased complexity and there is a cost associated with the extra management.

In section V-A we discussed a configuration tool that automates ViAggre configuration. We are also implementing a planning tool that takes as input high-level

constraints specified by the human ISP manager such as constraints on the traffic stretch, router load, router memory used and the robustness of the resulting design. It then uses ILP to solve a multiple-constraint optimization problem to generate VA-specific deployment details such as the assignment of aggregation points. These two tools combined would provide human ISP managers an automated means to adopt ViAggre without needing to delve into ViAggre and configuration-specific details.

It is difficult to speculate about actual costs and so we don't compare the increase in management costs against the cost of upgrading routers. While we hope that our tools will actually lead to cost savings for a ViAggre network, an ISP might just be inclined to adopt ViAggre because it breaks the dependency of various aspects of its operation on the size of the routing table. These aspects include its upgrade cycle, the per-byte forwarding cost, the per-byte forwarding power, etc.

Other concerns. An important concern arising out of the use of ViAggre is the tunneling overhead. However, the extensive use of tunnels (MPLS, GRE-IP, IPSec, VLAN tunneling) in ISP networks has meant that most routers are already equipped with interfaces that have extensive tunneling and detunneling capabilities at line rates [14].

As mentioned earlier, ViAggre represents a trade-off between FIB shrinkage on one hand and increased router load and traffic stretch on the other. The fact that Internet traffic follows a power-law distribution makes this a very beneficial trade-off. This power-law observation has held up in measurement studies from 1999 [10] to 2008 (in this paper) and hence, Internet traffic has followed this distribution for at least the past nine years in spite of the rise in popularity of P2P and video streaming. We believe that, more likely than not, future Internet traffic will be power-law distributed and hence, ViAggre will represent a good trade-off for ISPs.

Other design points. The ViAggre proposal presented in this paper represents one point in the design space that we focussed on for the sake of concreteness. Alternative approaches based on the same idea include

– *Adding routers.* We have presented a couple of techniques that ensure that only a subset of the routing table is loaded into the FIB. Given this, an ISP could install "slow-fat routers", low-end devices (or maybe even a stack of software routers [17]) in each PoP that are only responsible for routing traffic destined to unpopular prefixes. These devices forward a low-volume of traffic, so it would be easier and cheaper to hold the entire routing table. The popular prefixes are loaded into existing routers. This approach does away with a lot of deployment complexity. However, apart from the cost of the additional devices, this leads to concerns similar to the ones that ISPs have regarding routers that cache routes. For instance, attack traffic to unpopular prefixes could lead to a high relative increase in load across the low-end devices.

– *Router changes.* Routers can be changed to be ViAggre-aware and hence, make virtual prefixes first-class network objects. This would do away with a lot of the configuration complexity that ViAggre entails, ensure that ISPs get vendor support and hence, make it more palatable for ISPs. We, in cooperation with a router vendor, are exploring this option [16].

Routers today tend to have multiple blades with each blade maintaining its own copy of the entire routing table. Another approach involving vendor support is to split the routing table amongst router blades using ViAggre and hence, achieve FIB shrinkage with less burden on the ISP itself.

– *Clean-slate ViAggre.* The basic concept of virtual networks can be applied in an inter-domain fashion. The idea here is to use cooperation amongst ISPs to induce a routing hierarchy that is more aggregatable and hence, can accrue benefits beyond shrinking the router FIB. This involves virtual networks for individual virtual prefixes spanning domains such that even the RIB on a router only contains the prefixes it is responsible for. This would reduce both the router FIB and RIB and in general, improve routing scalability. We intend to study the merits and demerits of such an approach in future work.

VII. Related Work

A number of efforts have tried to directly tackle the routing scalability problem through clean-slate designs. One set of approaches try to reduce routing table size by dividing edge networks and ISPs into separate address spaces [7,11,29,32,40]. Our work resembles some aspects of CRIO [40] which uses virtual prefixes and tunneling to decouple network topology from addressing. However, CRIO requires adoption by all provider networks and like [7,11,29,32], requires a new mapping service to determine tunnel endpoints. APT [23] presents such a mapping service. Alternatively, it is possible to encode location information into IP addresses [8,15,19] and hence, reduce routing table size. Finally, an interesting set of approaches that trade-off stretch for routing table size are *Compact Routing* algorithms; see [26] for a survey of the area.

The use of tunnels has long been proposed as a routing scaling mechanism. VPN technologies such as BGP-MPLS VPNs [9] use tunnels to ensure that only PE routers need to keep the VPN routes. As a matter of fact, ISPs can and probably do use tunneling protocols such as MPLS and RSVP-TE to engineer a BGP-free core [35]. However, edge routers still need to keep the full FIB. With ViAggre, none of the routers on the datapath need to maintain the full FIB. Router vendors, if willing, can use a number of techniques to reduce

the FIB size, including FIB compression [35] and route caching [35]. Forgetful routing [24] selectively discards alternative routes to reduce RIB size. [2] sketches the basic ViAggre idea.

In recent work, Kim et. al. [25] use relaying, similar to ViAggre's use of aggregation points, to address the VPN routing scalability problem. The VPN setting involves VPN-specific routing tables and the task of maintaining these can be split amongst PE routers; in our setting there is just the Internet routing table and we use the concept of virtual prefixes to make it divisible. We also have the additional challenge of dealing with networks other than customers since these networks might be advertising the full routing table, which is solved by not installing some routes in the FIB (design-I) or through the use filters on route-reflectors (design-II).

Over the years, several articles have documented the existing state of inter-domain routing and delineated requirements for the future [5,12,28]; see [12] for other routing related proposals. RCP [4] and 4D [18] argue for logical centralization of routing in ISPs to provide scalable internal route distribution and a simplified control plane respectively. We note that ViAggre fits well into these alternative routing models. As a matter of fact, the use of route-reflectors in design-II is similar in spirit to RCSs in [4] and DEs in [18].

VIII. Summary

This paper presents ViAggre, a technique that can be used by an ISP to substantially shrink the FIB on its routers and hence, extend the lifetime of its installed router base. The ISP may have to upgrade the routers for other reasons but at least it is not driven by DFZ growth over which it has no control. While it remains to be seen whether the use of automated tools to configure and manage large ViAggre deployments can offset the complexity concerns, we believe that the simplicity of the proposal and its possible short-term impact on routing scalability suggest that is an alternative worth considering.

References

[1] Andersson, L., Minei, I., and Thomas, B. RFC 5036 - LDP Specification, Jan 2006.

[2] Ballani, H., Francis, P., Cao, T., and Wang, J. ViAggre: Making Routers Last Longer! In *Proc. of Hotnets* (Oct 2008).

[3] Barford, P. Wisconsin Advanced Internet Laboratory (WAIL), Dec 2007. http://wail.cs.wisc.edu/.

[4] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. Design and Implementation of a Routing Control Platform . In *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)* (2005).

[5] Davies, E., and Doria, A. Analysis of Inter-Domain Routing Requirements and History. Internet Draft draft-irtf-routing-history-07.txt, Jan 2008.

[6] de Silva, S. 6500 FIB Forwarding Capacities. NANOG 39 meeting, 2007. http://www.nanog.org/mtg-0702/presentations/fib-desilva.pdf.

[7] Deering, S. The Map & Encap Scheme for scalable IPv4 routing with portable site prefixes, March 1996. http://www.cs.ucla.edu/~lixia/map-n-encap.pdf.

[8] Deering, S., and Hinden, R. IPv6 Metro Addressing. Internet Draft draft-deering-ipv6-metro-addr-00.txt, Mar 1996.

[9] E. Rosen and Y. Rekhter. RFC 2547 - BGP/MPLS VPNs, Mar 1999.

[10] Fang, W., and Peterson, L. Inter-As traffic patterns and their implications. In *Proc. of Global Internet* (1999).

[11] Farinacci, D., Fuller, V., Oran, D., and Meyer, D. Locator/ID Separation Protocol (LISP). Internet Draft draft-farinacci-lisp-02.txt, July 2007.

[12] Feamster, N., Balakrishnan, H., and Rexford, J. Some Foundational Problems in Interdomain Routing. In *Proc. of Workshop on Hot Topics in Networks (HotNets-III)* (2004).

[13] Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., and True, F. Deriving traffic demands for operational IP networks: methodology and experience. *IEEE/ACM Trans. Netw. 9*, 3 (2001).

[14] Francios, P., and Bonaventure, O. An evaluation of IP-based Fast Reroute Techniques. In *Proc. of CoNEXT* (2005).

[15] Francis, P. Comparision of geographical and provider-rooted Internet addressing. *Computer Networks and ISDN Systems 27*, 3 (1994).

[16] Francis, P., Xu, X., and Ballani, H. FIB Suppression with Virtual Aggregation and Default Routes. Internet Draft draft-francis-idr-intra-va-01.txt, Sep 2008.

[17] Gillian, B. VYATTA: Linux IP Routers, Dec 2007. http://freedomhec.pbwiki.com/f/linux_ip_routers.pdf.

[18] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Meyers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A clean slate 4D approach to network control and management. *ACM SIGCOMM Computer Communications Review* (October 2005).

[19] Hain, T. An IPv6 Provider-Independent Global Unicast Address Format. Internet Draft draft-hain-ipv6-PI-addr-02.txt, Sep 2002.

[20] Hughes, D., Dec 2004. PACNOG list posting http://mailman.apnic.net/mailing-lists/pacnog/archive/2004/12/msg00000.html.

[21] Huston, G. BGP Reports, Dec 2007. http://bgp.potaroo.net/.

[22] Huston, G., and Armitage, G. Projecting Future IPv4 Router Requirements from Trends in Dynamic BGP Behaviour. In *Proc. of ATNAC* (2006).

[23] Jen, D., Meisel, M., Massey, D., Wang, L., Zhang, B., and Zhang, L. APT: A Practical Transit Mapping Service. Internet Draft draft-jen-apt-01.txt, Nov 2007.

[24] Karpilovsky, E., and Rexford, J. Using forgetful routing to control BGP table size. In *Proc. of CoNext* (2006).

[25] Kim, C., Gerber, A., Lund, C., Pei, D., and Sen, S. Scalable VPN Routing via Relaying. In *Proc. of ACM SIGMETRICS* (2008).

[26] Krioukov, D., and kc claffy. Toward Compact Interdomain Routing, Aug 2005. http://arxiv.org/abs/cs/0508021.

[27] Li, T. Router Scalability and Moore's Law, Oct 2006. http://www.iab.org/about/workshops/routingandaddressing/Router_Scalability.pdf.

[28] Mao, Z. M. Routing Research Issues. In *Proc. of WIRED* (2003).

[29] Massey, D., Wang, L., Zhang, B., and Zhang, L. A Proposal for Scalable Internet Routing & Addressing. Internet Draft draft-wang-ietf-efit-00, Feb 2007.

[30] Meyer, D., Zhang, L., and Fall, K. Report from the IAB Workshop on Routing and Addressing. Internet Draft draft-iab-raws-report-02.txt, Apr 2007.

[31] Narten, T. Routing and Addressing Problem Statement. Internet Draft draft-narten-radir-problem-statement-02.txt, Apr 2008.

[32] O'Dell, M. GSE–An Alternate Addressing Architecture for IPv6. Internet Draft draft-ietf-ipngwg-gseaddr-00.txt, Feb 1997.

[33] Ravi, R., and Sinha, A. Multicommodity facility location. In *Proc. of ACM-SIAM SODA* (2004).

[34] Rexford, J., Wang, J., Xiao, Z., and Zhang, Y. BGP routing stability of popular destinations. In *Proc. of Internet Measurment Workshop* (2002).

[35] Scudder, J. Router Scaling Trends. APRICOT Meeting, 2007. http://submission.apricot.net/chatter07/slides/future_of_routing.

[36] Spring, N., Mahajan, R., and Anderson, T. Quantifying the Causes of Path Inflation. In *Proc. of ACM SIGCOMM* (2003).

[37] Spring, N., Mahajan, R., and Wetherall, D. Measuring ISP topologies with Rocketfuel. In *Proc. of ACM SIGCOMM* (2002).

[38] Taft, N., Bhattacharyya, S., Jetcheva, J., and Diot, C. Understanding traffic dynamics at a backbone PoP. In *Proc. of Scalability and Traffic Control and IP Networks SPIE ITCOM* (2001).

[39] Y. Rekhter and T. Li and S. Hares, Ed. RFC 4271 - A Border Gateway Protocol 4 (BGP-4), Jan 2006.

[40] Zhang, X., Francis, P., Wang, J., and Yoshida, K. Scaling Global IP Routing with the Core Router-Integrated Overlay. In *Proc. of ICNP* (2006).

[41] Access List Performance Improvements, Oct 2008. http://www.cisco.com/en/US/docs/ios/12_0s/feature/

guide/hw_acl.html.
[42] clogin Manual Page, Oct 2008. http://www.shrubbery.net/rancid/man/elogin.1.html.
[43] Foundry Router Reference, Jul 2008. http://www.foundrynetworks.co.jp/services/documentation/srcli/BGP_cmds.html.
[44] JunOS Route Preferences, Jul 2008. http://www.juniper.net/techpubs/software/junos/junos60/swconfig60-routing/html/protocols-overview4.html.
[45] ViAggre Configuration Tool, Oct 2008. http://www.cs.cornell.edu/~hitesh/va-tools/va_conf_generator.py.

Towards a Global IP Anycast Service

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

ABSTRACT

IP anycast, with its innate ability to find nearby resources in a robust and efficient fashion, has long been considered an important means of service discovery. The growth of P2P applications presents appealing new uses for IP anycast. Unfortunately, IP anycast suffers from serious problems: it is very hard to deploy globally, it scales poorly by the number of anycast groups, and it lacks important features like load-balancing. As a result, its use is limited to a few critical infrastructure services such as DNS root servers. The primary contribution of this paper is a new IP anycast architecture, PIAS, that overcomes these problems while largely maintaining the strengths of IP anycast. PIAS makes use of a proxy overlay that advertises IP anycast addresses on behalf of group members and tunnels anycast packets to those members. The paper presents a detailed design of PIAS and evaluates its scalability and efficiency through simulation. We also present preliminary measurement results on anycasted DNS root servers that suggest that IP anycast provides good affinity. Finally, we describe how PIAS supports two important P2P and overlay applications.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Network communications

General Terms: Design, Measurement.

Keywords: Anycast, Proxy, Overlay, Routing, Architecture.

1. INTRODUCTION

Ever since it was proposed in 1993, IP anycast[1][1] has been viewed as a powerful IP packet addressing and delivery mode. Because IP anycast typically routes packets to the nearest of a group of hosts, it has been seen as a way to obtain efficient, transparent and robust ***service discovery***. In cases where the service itself is a connectionless ***query/reply service***, IP anycast supports the complete service, not just discovery of the service. The best working example of the latter is the use of IP anycast to replicate root DNS servers [2][3] without modifying DNS clients. Other proposed uses include ***host auto-configuration***[1] and using anycast to reach a ***routing substrate***, such as rendezvous points for a multicast tree[4][5] or a IPv6 to IPv4 (6to4) transition device[6].

In spite of its benefits, there has been very little IP anycast deployment to date, especially on a global scale. The only global scale use of IP anycast in a production environment that we are aware of is the anycasting of DNS root servers and AS-112 servers[7][2].

The reason for this is that IP anycast has serious limitations. Foremost among these is IP anycast's poor scalability. As with IP multicast, routes for IP anycast groups cannot be aggregated—the routing infrastructure must support one route per IP anycast group. It is also very hard to deploy IP anycast globally. The network administrator must obtain an address block of adequate size (i.e. a /24), and arrange to advertise it into the BGP substrate of its upstream ISPs. Finally, the use of IP routing as the host selection mechanism means that important selection metrics such as server load cannot be used. It is important to note that while IPv6 has defined anycast as part of its addressing architecture[8], it is also afflicted by the same set of problems.

By contrast, ***application layer anycast*** provides a one-to-any service by mapping a higher-level name, such as a DNS name, into one of a group of hosts, and then informing the client of the selected host's IP address, for instance through DNS or some redirect mechanism. This approach is much easier to deploy globally, and is in some ways superior in functionality to IP anycast. For example, the fine grained control over the load across group members and the ability to incorporate other selection criteria makes DNS-based anycast the method of choice for Content Distribution Networks (CDNs) today.

In spite of this, we believe that IP anycast has compelling advantages, and its appeal increases as overlay and P2P applications increase. First, IP anycast operates at a low level. This makes it potentially useable by, and transparent to, any application that runs over IP. It also makes IP anycast the only form of anycast suitable for low-level protocols, such as DNS. Second, it automatically discovers nearby resources, eliminating the need for complex proximity discovery mechanisms [9]. Finally, packets are delivered directly to the target destination without the need for a redirect (frequently re-

[1]IP anycast is an IP addressing and delivery mode whereby an IP packet is sent to one of a group of hosts identified by the IP anycast address. Where IP unicast is one-to-one, and IP multicast is one-to-many, IP anycast is one-to-any.

[2]anycasted servers that answer PTR queries for the RFC 1918 private addresses

SIGCOMM'05, Aug 22–26, 2005, Philadelphia, Pennsylvania, USA.

quired by application-layer anycast approaches). This saves at least one packet round trip, which can be important for short lived exchanges. It is these advantages that have led to increased use of IP anycast within the operational community, both for providing useful services (DNS root servers), and increasingly for protecting services from unwanted packets (AS112 and DDoS sinkholes [10]).

The *primary contribution* of this paper is the detailed description of a deployment architecture for an IP anycast service that overcomes the limitations of today's "native" IP anycast while adding new features, some typically associated with application-level anycast, and some completely new. This architecture, called **PIAS (Proxy IP Anycast Service)**, is composed as an overlay, and utilizes but does not impact the IP routing infrastructure. The fact that PIAS is an IP anycast service means that *clients* use the service completely transparently—that is, with their existing IP stacks and applications.

PIAS allows an endhost in an anycast group (anycast group member, or **anycast target**) to receive anycast packets for that group via its normal unicast address (and normal protocol stack). The anycast target *joins* the anycast group simply by transmitting a request packet to an anycast address (again, via its unicast interface). The target may likewise *leave* the group through a request packet, or by simply becoming silent.

PIAS utilizes the IP address space efficiently: thousands of IP anycast groups may be identified through a single IP address. It scales well by the number of groups, group size and group churn with virtually no impact on the IP routing infrastructure. It provides fast failover in response to failures of both target hosts and PIAS infrastructure nodes.

PIAS can select targets based on criteria other than proximity to the sending host, notably including the ability to load balance among targets. PIAS has the unique feature that an anycast group member can also transmit packets to other members of the same anycast group. This is in contrast to native IP anycast, where a group member would receive its own packet if it transmitted to the group. This feature makes IP anycast available to P2P applications, something not possible if a host can't both send to and receive from the anycast group.

The remainder of the paper is organized as follows: Section 2 identifies the features of an ideal anycast service. Section 3 spells out the system design together with the goals satisfied by each design feature. Section 4 presents simulations and measurements meant to evaluate various features of the PIAS design. Section 5 discusses related work and section 6 describes a few applications made possible by PIAS. Section 7 discusses other important goals that PIAS must fulfill and section 8 presents our conclusions.

2. DESIGN GOALS

This section specifically lays out the design goals of PIAS, and briefly comments on how well PIAS meets those goals. The subsequent design description section refers back to these goals as needed. The goals are listed here in two parts. The first part lists those goals that are accomplished by native IP anycast, and that we wish to retain. The second part lists those goals that are not accomplished by native IP anycast. In this way, we effectively highlight the weaknesses of IP anycast, and the contributions of PIAS.

1. *Backwards Compatible*: Native IP anycast is completely transparent to clients and routers , and we believe that this transparency is critical to the success of a new IP anycast service. Because PIAS is an overlay technology that uses native IP anycast, it does not change clients and routers.
2. *Scale by group size*: By virtue of being totally distributed among routers, native IP anycast scales well by group size. PIAS has no inherent group size limitation. PIAS is deployed as an overlay infrastructure, and can scale arbitrarily according to the size of that infrastructure.
3. *Efficient packet transfer*: Because native IP anycast uses IP routing, its paths are naturally efficient. As an overlay, PIAS imposes some stretch penalty on the paths packets take. The penalty imposed by PIAS is small (section 4.3), and shrinks as the PIAS infrastructure grows.
4. *Robustness*: Native IP anycast's robustness properties (including packet loss) are similar to IP unicast. PIAS is engineered to be similarly robust.
5. *Fast failover*: Failover speed in Native IP anycast depends on the convergence speed of the underlying routing algorithms, and can be fast (OSPF) or somewhat slow (BGP). PIAS can be engineered to almost always rely on OSPF for certain types of failover (section 3.6). The PIAS overlay exposes additional failover situations that go beyond IP routing, and these are handled accordingly (Section 3.6).

The following are the goals that native IP anycast does not satisfy.

6. *Ease of* joining *and* leaving: Target hosts must not have to interact with IP routing to join and leave.
7. *Scale by the number of groups*: In addition to scaling by the usual metrics of memory and bandwidth, we require that PIAS also make efficient use of the IP address space. PIAS is able to accommodate thousands of groups within a single address by incorporating TCP and UDP port numbers as part of the group address.
8. *Scale by group dynamics*: Globally, IP routing behaves very badly when routes are frequently added and withdrawn. The PIAS overlay hides member dynamics from IP routing, and can handle dynamics caused both by continuous member churn and flash crowds (including those caused by DDoS attacks).
9. *Target Selection criteria*: IP anycast can only select targets based on proximity. At a minimum, we wish to add load and connection affinity as criteria.

3. DESIGN DESCRIPTION

This section gives a detailed description of PIAS. We take a "layered" approach to the description—we start with the core concepts and basic design and then step-by-step describe additional functionality that satisfies specific goals listed in section 2.

PIAS is deployed as an overlay infrastructure. It may be deployed by a CDN company like Akamai, by multiple cooperating ISPs, or even by a single ISP (though the efficacy of proximity discovery would be limited by the ISP's geographic coverage). Multiple distinct PIAS infrastructures may be deployed. In this case, each operates using distinct blocks of IP

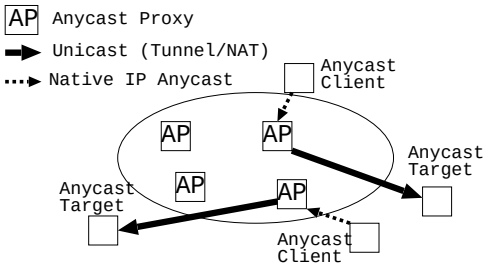


Figure 1: Proxy Architecture: the client packets reaching the proxies through native IP anycast are tunnelled to the targets

anycast addresses, and they do not interact with each other[3]. In the remainder of this document, for simplicity of exposition, we assume a single PIAS infrastructure.

The basic idea of PIAS, illustrated in Figure 1, is very simple. Router-like boxes, hereon referred to as ***anycast proxies*** (AP or simply *proxies*), are deployed at various locations in the Internet, for example at POPs (Point of Presence) of different ISPs. These proxies advertise the same block of IP addresses, referred to as the ***anycast prefix***, into the routing fabric (BGP, IGP). As such, the proxies are reachable by native IP anycast—a packet transmitted to the anycast prefix will reach the closest proxy. However, these proxies are not the actual ***anycast target destinations***(AT)[4]. Rather, true to their name, they proxy packets that reach them via native IP anycast to the true target destinations using unicast IP. This proxying can take the form of lightweight tunnels or NAT. NAT allows for backwards compatibility with the protocol stack at target hosts, but increases processing at the proxy.

This novel combination of native IP anycast with tunnelling to the unicast addresses of the targets allows PIAS to fulfill three critical design goals and drives the rest of the system design. First, it allows for efficient use of the address space as all the IP addresses in the prefix advertised by the proxies can be used by different anycast groups. In fact, PIAS does one better. It identifies an anycast group by the full ***transport address*** (TA), i.e. IP address and TCP/UDP port, thus allowing thousands of anycast groups per IP address. Second, it solves the IP routing scaling problem by allowing many anycast groups to share a single address prefix and hence, fulfills goal 7. Finally, it relieves targets from the burden of interacting with the routing substrate. They can join an anycast group by registering with a nearby proxy that is discovered using native IP anycast. This fulfills goal 6.

The reader may notice two suspicious claims in the last paragraph. First, we claim to ease deployment by running unicast at the target instead of anycast, and yet the proxies still must run anycast. So, how is this an improvement? The benefit is that the difficult work of deploying IP anycast is borne by the anycast provider once, and amortized across many anycast groups. Second, we claim to improve scaling by allowing thousands of IP anycast groups to share a single IP address prefix. All we've really done, however, is to move the scaling problem from the IP routing domain to the PIAS infrastructure domain. This is quite intentional. As we argue later on, the scaling issues are much easier to deal with in the overlay than in IP routing.

PIAS offers two primitives to the members of an anycast group, which involve sending messages to a nearby proxy:

- *join(IP_A:$port_A$,IP_T:$port_T$,options)*: this message instructs the proxy to forward packets addressed to the anycast group identified by the TA IP_A:$port_A$ to the joining node's unicast TA IP_T:$port_T$. The *options* may specify additional information such as the selection criteria (load balance etc.), delivery semantics (scoping etc.), or security parameters needed to authenticate the target host. These are discussed later.
- *leave(IP_A:$port_A$,IP_T:$port_T$,options)*: this message informs the proxy that the target identified by TA IP_T:$port_T$ has left the group IP_A:$port_A$. *options* are the security parameters.

The join and leave messages are transmitted to the anycast address IP_A(that belongs to the anycast prefix) at some well-known port that is dedicated to receiving registration messages. This means that no extra configuration is required for a target to discover a nearby proxy.

Note that we don't specify a "create group" primitive. For the purpose of this paper, we assume that the first join essentially results in the creation of the group. In practice, a subscriber to the service would presumably have entered into a contract with the anycast service provider, which would have resulted in the assignment of anycast TAs to that subscriber. The subscriber would also have obtained authentication information using which targets may join the group. While the issues surrounding this sort of group creation are important, they are not central to the PIAS architecture, and we don't discuss them further.

3.1 The Join Anycast Proxy (JAP)

A target may leave a group either through the leave primitive, or by simply falling silent (for instance, because the target is abruptly shut off or loses its attachment to the Internet). This means that the ***Join AP*** (JAP—the nearby proxy with which the target registers; shown in figure 2) must monitor the health of its targets, determine when they are no longer available, and treat them as having left the group. The proximity of the JAP to the target makes it ideal for this.

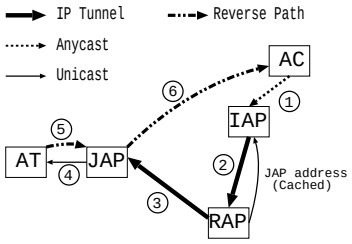
The JAP must also inform zero or more other anycast proxies (APs) of the target(s) that have registered with it. This is because not all APs may be JAPs for a given group (that is, no target joined through them), but **anycast clients** (ACs) may nevertheless send them packets destined for the group. A proxy that receives packets directly from a client is referred to as the ***Ingress AP*** (IAP)[5] for the client. Note that the client-IAP relation is established using native IP anycast. As an IAP, the proxy must know how to forward packets towards a target; even though the IAP may not explicitly know of the target.

One possible way to achieve this would have the JAP spread information about targets associated with it to all proxies. This allows the IAP to tunnel packets directly to clients (as in Figure 1). However, such an approach would hamper PIAS's ability to support a large number of groups. In fact, Figure 1 is conceptual—PIAS's approach for spreading group information is described in the next section and the actual paths taken by packets are shown in Figure 2.

[3]Indeed, a single operator could deploy multiple distinct PIAS infrastructures as a way to scale.

[4]the members of the anycast group; hereon referred to as **anycast targets** or simply **targets**

[5]in figure 1 the proxies in the client-target path are IAPs



No.	Source	Dest	Comment
1	AC:p	AA:g	Native IP Anycast
2	IAP:AC:p	RAP:AA:g	IP-IP tunnel
3	RAP:AC:p	JAP:AA:g	IP-IP tunnel
4	JAP:q	AT:r	Unicast IP JAP:q = NAT(AC:p)
5	AT:r	JAP:q	Unicast IP
6	AA:g	AC:p	Unicast IP AC:p = NAT^{-1}(JAP:q)
7	IAP:AC:p	JAP:AA:g	IP-IP tunnel

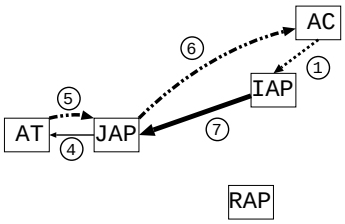


Figure 2: Initial (left) and subsequent (right) packet path. The table shows the various packet headers. Symbols in block letters represent IP addresses, small letters represent ports. AA(Anycast Address) is one address in the address block being advertised by PIAS, AA:g is the transport address assigned to the group the target belongs to, while AT:r is the transport address at which the target wants to accept packets. Here, the target joined the group by invoking join(AA:g,AT:r,options)

3.2 Scale by the number of groups

In the previous section, we mentioned the need for a scheme that would allow PIAS to manage group membership information while scaling to a large number of groups. For any given group, we designate a small number of APs (three or four) to maintain a list of JAPs for the group. When acting in this role, we call the AP a ***Rendezvous Anycast Proxy*** (RAP). All APs can act as RAPs (as well as as JAPs and IAPs).

The RAPs associated with any given group are selected with a consistent hash [11] executed over all APs. This suggests that each proxy know all other proxies, and maintain their current up/down status. This is possible, however, because we can assume a relatively small number of global APs ($\leq$20,000, a number we derive later). We also assume that, like infrastructure routers, APs are stable and rarely crash or are taken out of service. The APs can maintain each other's up/down status through flooding, gossip [12] or a hierarchical structure [13]. The current implementation uses flooding. Such an arrangement establishes a simple one-hop DHT and hence, limits the latency overhead of routing through the proxy overlay.

When a proxy becomes a JAP for the group (i.e. a target of the group registers with it), it uses consistent hashing to determine all the RAPs for the group and informs them of the join. This allows the RAP to build a table of JAPs for the group.

The concept of the RAP leads to a packet path as shown on the left side of Figure 2. When an IAP receives a packet for an anycast group that it knows nothing about, it hashes the group TA, selects the nearest RAP for the group, and transmits the packet to the RAP (path segment 2). The RAP receives the packet and selects a JAP based on whatever selection criteria is used for the group. For instance, if the criteria is proximity, it selects a JAP close to the IAP. The RAP forwards the packet to the selected JAP (path segment 3), and at the same time informs the IAP of the JAP (the RAP sends a list of JAPs, for failover purposes).

The use of RAPs unfortunately introduces another overlay hop in the path from client to target. We mitigate this cost however by having the IAP cache information about JAPs. Once the IAP has cached this information, subsequent packets (not only of this connection, but of subsequent connections too) are transmitted directly to the JAP. This is shown in the right-hand side of Figure 2. The time-to-live on this cache entry can be quite large. This is because the cache entry can be actively invalidated in one of two ways. First, if the target leaves the JAP, the JAP can inform the IAP of this when a subsequent packet arrives. Second, if the JAP disappears altogether, inter-AP monitoring will inform all APs of this event. In both cases, the IAP(s) will remove the cached entries, failover to other JAPs it knows of, or failing this, go back to the RAP. Because of this cache invalidation approach, the IAP does not need to go back to the RAP very often.

Note that in figure 2, the JAP is responsible for transmitting packets to and receiving packets from its targets. The reasoning for this is not obvious and goes as follows. We aim to support legacy clients that expect to see return packets coming from the same address and port to which they sent packets. In general, targets cannot source packets from anycast addresses and so at least one proxy must be inserted into the target-client path. Furthermore, if NAT is being used to forward packets to the target, then the proxy with the NAT state should be the proxy that handles the return packets.

This might argue for traversing the IAP in the reverse direction too, since by necessity it must be traversed in the forward direction. The argument in favor of using the JAP however, boils down to the following two points. First, it is highly convenient to keep all target state in one proxy rather than two or more. Since the JAP in any event must monitor target health, it makes sense to put all target state in the JAP. Second, the JAP is close to the target, so the cost of traversing the JAP in terms of path length is minimal (Section 4.3). Also, by seeing packets pass in both directions, the JAP is better able to monitor the health of the target. For the most part, when a packet passes from client to target, the JAP may expect to soon see a packet in the reverse direction. Rather than force the JAP to continuously ping each target, the lack of a return packet can be used to trigger pings.

The use of proxies implies that the PIAS path (AC⇒IAP⇒JAP⇒AT) might be longer than the direct path (AC⇒AT)[6]. However, the proximity of the client to the IAP and of the target to the JAP should ensure that PIAS imposes minimal stretch and hence fulfills goal 3. This has been substantiated by simulating the stretch imposed by PIAS across a tier-1 topology map of the Internet.

The introduction of the RAP to allow scaling by the number of groups is somewhat equivalent to the extra round-trip imposed by application-level anycast schemes, for instance in the form of the DNS lookup or the HTTP redirect. This is

[6]the PIAS path may actually be shorter as inter-domain routing is not optimal[14]

one aspect of PIAS that falls short of native IP anycast, which has no such extra hop. Having said that, it would be possible for a small number of groups with minimal target churn to operate without RAPS—that is, to spread JAP information among all APs. This might be appropriate, for instance, for a CDN or for 6to4 gateways. By-and-large, however, we can expect most groups to operate with RAPs as described here, and in the remainder of the design section, we assume that is the case.

3.3 Scale by group size and dynamics

If the only selection criteria used by a RAP to select a JAP were proximity to the client, then the RAP could ignore the number of targets reachable at each JAP. In order to load balance across targets, however, RAPs must know roughly how many targets are at each JAP. In this way, RAPs can select JAPs in a load balanced way, and each JAP can subsequently select targets in a load balanced way. Unfortunately, requiring that RAPs maintain counts of targets at JAPs increases the load on RAPs. This could be a problem for very large groups, or for groups with a lot of churn.

We mitigate this problem by allowing the JAP to give the RAP an approximate number of targets, for example within 25% or 50% of the exact number. For instance, if 25% error is allowed, then a JAP that reported 100 targets at one time would not need to report again until the number of targets exceeded 125 or fell below 75. This approach allows us to trade-off the granularity of load-balancing for scalability with group size and dynamics. Indeed, this trade-off can be made dynamically and on a per-group basis. A RAP that is lightly loaded, for instance, could indicate to the JAP that 100% accuracy reporting is allowed (i.e. in its acknowledgement messages). As the RAP load goes up, it would request less accuracy, thus reducing its load. The combination of the two-tiered approach with inaccurate information in a system with 2 groups is illustrated in Figure 3 (the figure assumes that there is just one RAP for each group). Section 4.2 presents simulations that show the benefits of this approach in the case of a large, dynamic group.

In any event, the number of targets is not the only measure of load. Individual targets may be more-or-less loaded due to differing loads placed by different clients. Ultimately, the JAP may simply need to send a message to the RAPs whenever its set of targets are overloaded for whatever reason.

3.4 Scale by number of proxies

Given that we have laid out the basic architecture of PIAS, we can now specifically look at PIAS deployment issues. A central question is, how many proxies may we reasonably expect in a mature PIAS deployment, and can we scale to that many proxies?

A key observation to make here is that the scaling characteristics of PIAS are fundamentally different from the scaling characteristics of IP routing. While the traffic capacity of the Internet can be increased by adding routers, the scalability of IP routing per se is not improved by adding routers. All routers must contain the appropriate routing tables. For instance, all Tier1 routers must contain the complete BGP routing table no matter how many Tier1 routers there are. For the most part, IP routing is scaled by adding hierarchy, not adding routers.

With PIAS, on the other hand, scaling does improve by adding proxies. With each additional proxy, there are lower

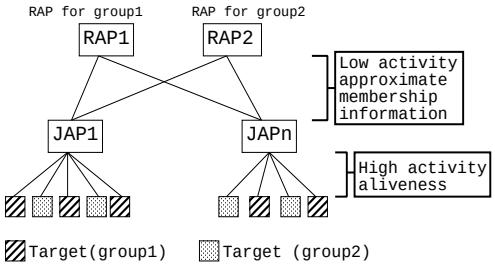


Figure 3: 2-tier membership management: the JAPs keep the aliveness status for the associated targets; the RAP for a group tracks the JAPs and an approximate number of targets associated with each JAP

ratios of target-to-JAP and group-to-RAP. Growth in the number of groups and targets can be absorbed by adding proxies. However, an increase in the number of proxies presents its own scaling challenge. Among other things, every proxy is expected to know the up/down status of every other proxy.

The following describes a simple divide-and-conquer approach that can be used if the number of proxies grows too large. In a typical deployment, a given anycast service provider starts with one anycast prefix, and deploys proxies in enough geographically diverse POPs to achieve good proximity. As more anycast groups are created, or as existing anycast groups grow, the provider expands into more POPs, or adds additional proxies at existing POPs. With continued growth, the provider adds more proxies, but it also obtains a new address prefix (or splits the one it has), and splits its set of proxies into two distinct groups. Because the IP routing infrastructure sees one address prefix per proxy group, and because a proxy group can consist of thousands of proxies and tens of thousands of anycast groups, the provider could continue adding proxies and splitting proxy groups virtually indefinitely.

The size of a mature proxy deployment may be roughly calculated as follows. There are about 200 tier-1 and tier-2 ISPs [15]. An analysis of the ISP topologies mapped out in [16] shows that such ISPs have ∼25 POPs on average. Assuming that we'd like to place proxies in all of these POPs, this leads to 5000 POPs. Assuming 3-4 proxies per POP (for reliability, dicussed later), we get a conservative total of roughly 20,000 proxies before the infrastructure can be split.

While 20,000 proxies is not an outrageous number, it is large enough that we should pay attention to it. One concern not yet addressed is the effect of the number of proxies on IP routing dynamics. In particular, BGP reacts to route dynamics (flapping) of a single prefix by "holding down" that prefix—ignoring any advertisements about the prefix for a period of at most one hour [17]. A naive proxy deployment where each proxy advertises the anycast prefix directly into BGP would imply that a proxy failure necessitates a BGP withdrawal for the prefix (from the site where the proxy is located) that could lead to hold downs. While the proxy stability ensures that such events do not occur often, even the occasional prefix instability and the consequent service disruptions that a large proxy deployment would entail are not acceptable.

Hence, the deployment model involves more than one proxy being placed inside every POP where the proxies are deployed. Such an arrangement is referred to as an ***anycast***

Segment	Failure of	Failover through	Section
AC⇒IAP	IAP	IGP, onto a proxy within the same cluster	3.6
IAP⇒JAP	JAP	proxy health monitoring system	3.6
JAP⇒AT	AT	pings between target and JAP, passive monitoring by JAP	3.1,3.2
AT⇒JAP	JAP	pings routed to a different proxy who becomes JAP	3.6
JAP⇒AC	AC	no failover needed	-

Table 1: Failover along the PIAS forward path (AC⇒IAP⇒JAP⇒AT) and reverse path (AT⇒JAP⇒AC)

cluster[7] and is based on the model used by the anycasted f-root server[18]. The approach involves connecting one or more routers and more than one proxy to a common subnet. All the proxies in the cluster advertise the anycast prefix into IGP while the routers advertise it into BGP and hence, a proxy-failure does not lead to a BGP withdrawal.

3.5 Proximity

The introduction of the proxies into the IP path negates the natural ability of native IP anycast to find the nearest target. Therefore, we require explicit mechanisms in PIAS to regain this capability.

As mentioned before, native IP anycast sets the client-IAP and target-JAP path segments. The RAP, on the other hand, selects the JAP, and therefore sets the IAP-JAP path segment (on forward packets) and the JAP-client path segment (on return packets). To ensure the proximity of the target to the client, the RAP must choose a JAP close to the IAP and hence, every AP must know the distance (in terms of latency) between every pair of APs. This could be accomplished using a proximity addressing scheme like GNP [19] or Vivaldi [20].

Another possibility is to use a simple, brute-force approach whereby every AP occasionally pings every other AP and advertises the minimum measured round trip time (RTT) to all other APs. This is feasible because, with the cluster deployment approach, RAPs only need to know the distance between each pair of clusters. While validating the above claim would require experimentation with the actual deployment, back of the envelope calculations do paint a promising picture for the simple approach.

3.6 Robustness and fast failover

The introduction of proxies between client and target might have a negative impact on the robustness of PIAS as compared to native IP anycast. On the other hand, RON[14] has shown how an overlay structure can be used to improve the resiliency of communication between any two overlay members. Extending the same thought, PIAS, by ensuring the robustness of packet traversal through the proxy overlay, can improve the resiliency of communication between clients and group members. We believe that given the stable nature of the proxies, their deployment in well connected parts of the Internet (tier-1 and tier-2 ISPs) and the engineering that would go into their set-up, PIAS should be able to match, if not better, the robustness offered by native IP anycast.

A related requirement is that of fast fail-over. "E2E" native IP anycast has to achieve failover when a group member crashes, so that clients that were earlier accessing this member are served by some other group member. Given the way native IP anycast works, this failover is tied to IP routing convergence. Specifically, in case of a globally distributed group, the failover is tied to BGP convergence, which in some cases can extend to a few minutes[14]. Since PIAS uses native IP anycast to reach the proxies, it is subject to the same issues. The process of overcoming the failure of a proxy is termed as **proxy failover**. In addition, the proxies must themselves be able to fail over from one target to another which is termed as **target failover**. Thus the failover problem seems worse with PIAS than with native IP anycast; however, this is not the case.

3.6.1 Target failover

As discussed in Sections 3.1 and 3.2, the JAP is responsible for monitoring the aliveness of its targets. It does this through pinging and tracking data packets to and from the target. The JAP is also responsible for directing IAPs to delete their cache entries when enough targets have failed.

3.6.2 Proxy failover

There is still the question of clients failing over onto a different proxy when their IAP crashes, and targets failing over when their JAP crashes. And there are two levels at which this must be achieved: at the routing level and at the overlay level.

At the routing level, the system must be engineered such that when a proxy fails, clients that were using this proxy as an IAP are rerouted to some other proxy quickly. PIAS's deployment of proxies in a cluster means that this failover is across proxies within the same cluster. Also, since the proxies advertise the prefix into IGP, PIAS relies on IGP for convergence after a proxy failure and hence can achieve faster failover. Typically, this is of the order of a few seconds and can be reduced to sub-second times[21].

At the overlay level, to monitor the health of proxies, we use a 2-tier health monitoring system. At the first tier, the proxies within the same proxy cluster are responsible for monitoring each other. At the next level, each proxy in a cluster monitors the health of a small number of other clusters. When either an individual proxy or an entire cluster fails, it is detected quickly and communicated to all remaining proxies.

Section 3.2 had described IAP behavior when a JAP goes down. The only thing left to discuss is target behavior when a JAP goes down. In this case, native IP anycast routing will cause ping packets from the target to reach another JAP, which will ask the target to re-register. Table 1 sums up the way PIAS achieves failover across various segments of the client-target path.

3.7 Target selection criteria

As described earlier, the RAP may select the JAP based on a number of criteria, including proximity, load balancing, and connection affinity[8]. The JAP subsequently selects a target. It is this selection process, divorced from IP routing, that allows PIAS to offer richer target selection criteria

How PIAS achieves load balance and proximity has already been discussed. Connection affinity is discussed later in this section. We wish to point out here that these three important selection criteria are in fact at odds with each other. For

[7]hereon referred to as proxy cluster or simply, cluster

[8]Connection affinity—all packets from a given connection or flow are delivered to the same target.

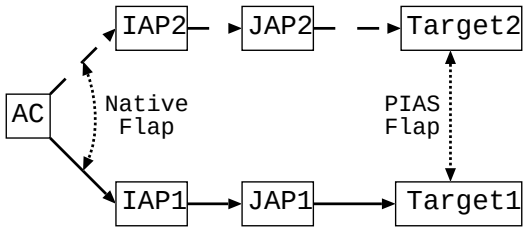


Figure 4: Lack of native IP anycast affinity can cause flaps in the PIAS model

exampl, if both load balance and proximity are important criteria, and the JAP nearest to the IAP is heavily loaded, then one of the other criteria must be compromised. This basic set of trade-offs applies to application-level anycast as well.

By never selecting the source of a packet as the target, PIAS allows a host to be both a target and a client for a given group. Packets sent by the target to the group address would be forwarded to some group target other than the sender. Note that this is not possible with native IP anycast and it allows PIAS to support new P2P applications (section 6.1).

Proxies could potentially base their target selection on various scoping criteria. These selection criteria can be expressed by overloading the transport address, i.e. a group can have separate TAs for each type of scoping. For instance, an anycast packet could be administratively scoped. That is, it could indicate that the target should be in the same site, belong to the same DNS domain, or have the same IP address prefix (or be from different sites, DNS domains, or IP prefixes). While how this would be configured and operated is a good topic for further study, the selection functionality of the RAP allows for the possibility of many such features.

Another form of selection would be to pick a random target rather than the nearest target - the RAP would pick a random JAP who would then pick a random target. Random selection among a group can be useful for various purposes such as spreading gossip [22] or selecting partners in multicast content distribution [23]. Indeed, in the PIAS architecture, there is no reason an anycast packet cannot be replicated by the RAP and delivered to a small number of multiple targets. The salient point here is that, once IP anycast functionality is divorced from IP routing, any number of new delivery semantics are possible if the benefits justify the cost and complexity.

3.7.1 *Connection affinity*

Lack of connection affinity in native IP anycast has long been considered one of its primary weak points. This issue spills over into PIAS. Specifically, the issue is how to maintain affinity when native IP anycast causes a different IAP to be selected during a given client connection. If the same IAP is always used, then packets will be sent to the same JAP that was initially cached by the IAP. However, a change in the IAP could lead to a change in the target the packets are delivered to, as shown by Figure 4. Application-layer anycast doesn't have this problem, because it always makes its target selection decision at connection start time, and subsequently uses unicast.

A simple solution would be to have RAPs select JAPs based on the identity of the client, such as the hash of its IP address. This way, even if IP routing caused packets from a given client to select a different IAP, they would be routed to the same JAP and hence the same target. Unfortunately,

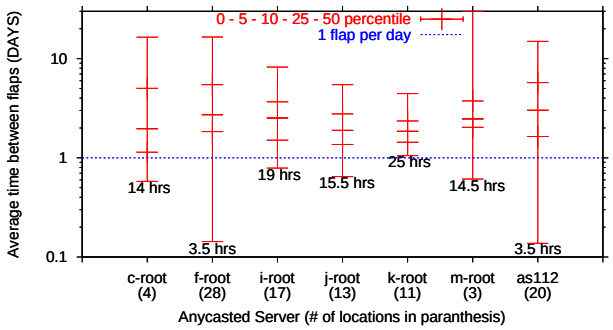


Figure 5: Percentiles for the average time between flaps for all the anycasted destinations

this approach completely sacrifices proximity and load balance. Broadly, another approach would be to modify the host application by making it anycast aware, and redirect the host to the unicast address of a selected target (either PIAS or the target itself could do this redirect). There are some security issues here—the redirect must be hard to spoof—but these are surmountable.

We can also imagine complex schemes whereby JAPs and IAPs coordinate to insure affinity. However, a fundamental question that still has not been answered is, how good or bad is the affinity offered by native IP anycast? It might be the case that the affinity offered by native IP anycast is very good; i.e. the probability that a connection breaks due to a routing flap is very small as compared to the probability of the connection breaking due to other factors. This would imply that we do not need the complex mechanisms stated above. In this regard, we did some measurements to find out the affinity offered by native IP anycast. Our results, while preliminary, suggest that native IP anycast affinity is quite good, and PIAS need not do anything extra to provide reasonable connection affinity. Details of these measurements are presented in section 4.1

4. EVALUATION

In this section we evaluate the PIAS architecture using measurements and simulations. Section 4.1 describes the measurements made using the Planetlab[24] testbed and the anycasted DNS root servers to argue for the sufficiency of the affinity offered by native IP anycast and hence, PIAS. Sections 4.2 and 4.3 present simulation results that show the scalability (by group characteristics) and the efficiency of the PIAS deployment. Finally, section 4.4 discusses our PIAS implementation. We also measured the quality of proximity selection offered by the anycasted DNS server deployments. These are briefly discussed in section 7.

4.1 Connection Affinity measurements

As mentioned earlier, it is important to determine the affinity offered by native IP anycast in order to understand the need for mechanisms to ensure affinity in PIAS. This section presents the results of our measurement study aimed to do so. The goal of the study was to determine how often IP routing selected different locations when sending packets to a native IP anycast address. We used the anycasted root servers and the AS-112 servers as the anycast destinations. For clients, we used 129 Planetlab nodes belonging to 112 sites.

For each anycast destination, the clients probed the associated anycast address every 10 seconds to determine the

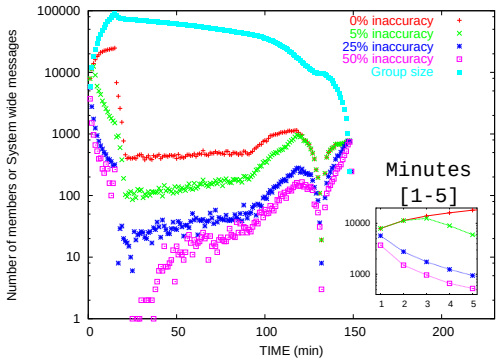


Figure 6: System wide messages from the all the JAPs to the 4 RAPs during the event for varying degrees of inaccuracy

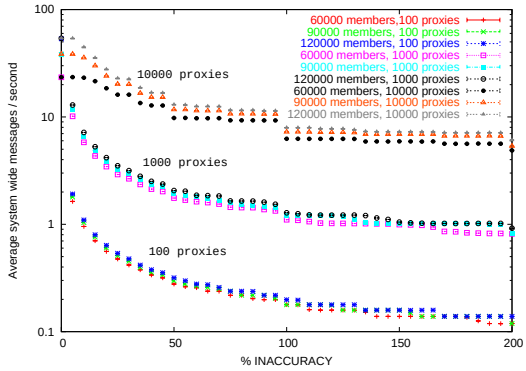


Figure 7: Average system wide messages (per second) versus the percentage of inaccuracy with varying number of proxies and varying maximum group size.

location they are routed too. The servers at different locations have been configured by their operators to respond to a TXT type DNS query with their location[25] and hence, the probes were DNS queries generated using *dig*. This data was collected for a period of 30 continuous days in Dec'04-Jan'05.

The probing of the anycasted destinations reveals changes in routing or 'flaps' that cause packets to be delivered to different locations of an anycasted server. So, a pair of probes from a given Planetlab node switching from the San Jose f-root server to the Palo Alto f-root server[9] would be counted as one flap. Using our measurement data, we determined the average time between flaps to a given root server for each probing node. Figure 5 plots various percentiles for the average time between flaps when probing various anycasted servers. The figure shows that the anycasted services are very stable as viewed from almost all locations. For example, more than 95% of the nodes observed less than a flap per day for all the anycasted destinations. Similarly, ∼48% of the nodes never observed a flap when probing the f-root during the entire 30 day period.

Also, the few nodes that observed frequent flaps (i.e. an average inter-flap duration of less than a day) had their average skewed by tiny bursts of instability in between large periods of stability. For example, the Planetlab node that experienced most flaps (208) over the month when probing j-root was in Leixip, Ireland. Of these, 180 flaps occurred in a 3-hour period. We conjecture that such phenomena can be attributed to ephemeral issues specific to the sites to which these nodes belong. While a more rigorous analysis of the collected data and correlation with BGP-updates for the prefixes representing these anycasted destinations would be needed for determining the causes and patterns amongst these flaps, the overall figures do paint an encouraging picture. These measurements reveal that the probability that a two minute connection breaks due to a flap is about 1 in 4500 and the probability that an hour long connection breaks is about 1 in 150. Note that it is the short connections that, in order to avoid the overhead of anycast to unicast redirect, need to rely on anycast affinity. Long connections can incur the overhead of a redirect and hence, could use anycast for discovery and unicast for the actual communication.

We admit that the limited number(129) and variety of vantage points and the number of locations of the anycast destinations makes our study preliminary. Also, the operators of j-root, based on their observations, have come to the opposite conclusion regarding the ability of native IP anycast to support stateful connections[26]. While their results are being debated by many in the operational community[27], we are trying to acquire the relevant data-sets so as to find the reason for the flapping observed by them (something that the authors of the j-root study have not analyzed).

[9]San Jose and Palo Alto are two locations of the f-root server

4.2 Scalability by group size and dynamics

In this experiment, we evaluate PIAS's ability to handle large and dynamic groups (as described in 3.3). We simulate the load imposed by a large group with high churn on the proxy infrastructure. The dynamics of the simulated group - the arrival rate of group members and the session duration cumulative distribution function - resemble the dynamics of the largest event observed in a study of large-scale streaming applications[28]. Simulation of just one such group is sufficient as the load imposed varies linearly with the number of such groups supported.

The PIAS infrastructure in the simulation has varying number of proxies and maximum group size. We simulate four RAPs per group. We want to measure the number of messages required to keep the 2-tier membership hierarchy updated in face of the group dynamics. This is the number of messages from the JAPs of the group to the 4 RAPs and is referred to as **'system wide messages'**.

Figure 6 plots the system wide messages produced with a proxy deployment of size 1000 and the group size bounded by 90000. The topmost curve in the figure shows how the group size varies with the time. A flash crowd, at a rate of ∼100 members/second, leads to a sudden rise in the group size in the first 10 minutes. The other curves plot the number of messages produced in the corresponding minute (as plotted along the X-axis) for varying degrees of inaccuracy. The degree of inaccuracy, as explained in section 3.3, implies that a JAP only informs a RAP of a change in the number of members associated with it if the change is more than a certain percentage of the last value sent.

The inaccuracy of information offers only a small benefit in the nascent stages of the group (the first minute). This is because no matter what inaccuracy percentage we use, the JAP must inform the RAP of the first group member that contacts it. In the next couple of minutes, as the group increases in

size and more members join their corresponding JAPs, the inaccuracy causes the traffic towards the 4 RAPs to drop rapidly (see the embedded graph in figure 6). Overall, the average number of messages over the duration of the entire event reduces from 2300 per min. with the naive approach to 117 per min. with 50% inaccuracy.

Figure 7 plots the average system wide messages (per second) versus the percentage of inaccuracy for varying number of proxies and varying maximum group size. Each plotted point is obtained by averaging across 20 runs. All curves tend to knee around an inaccuracy mark of 50%−60%. The closeness of the curves for different sized groups (given a fixed number of proxies) points to the scalability of the system by the group size even in the face of high churn.

More interesting is the variation of the load on the RAPs with the number of proxies. As the number of proxies increase, the number of JAPs increase; an offshoot of the assumption that the group members are evenly distributed across the proxy infrastructure. For a given group size, each JAP is associated with lesser number of group members. Hence, there is lesser benefit due to the inaccuracy approach. This shows up as the increase in the average number of messages directed towards the RAPs with the number of proxies.

The figure shows that such an extreme group in a 100 proxy deployment with 100% inaccuracy would require an average of ∼0.18 messages/second. As a contrast the same setup in a 10000 proxy deployment would necessitate an average of ∼7.25 messages/second. The low message overhead substantiates the PIAS scalability claim. Note that a larger number of proxies implies that each proxy is a RAP for a smaller number of groups. The number of targets associated with each proxy (as a JAP) reduces too. Thus, increasing the number of proxies would indeed reduce the overall load on the individual proxies.

4.3 Stretch

PIAS causes packets to follow a longer path (client ⇒ IAP ⇒ JAP ⇒ target). We have argued that the combination of native IP anycast and proxy-to-proxy latency measurements minimizes the effect of this longer path. This section simulates the stretch introduced by PIAS along the end-to-end path.

For the simulation, we use a subset of the actual tier-1 topology of the Internet, as mapped out in the Rocketfuel project [16]. This subset consists of 22 ISPs, 687 POPs, and 2825 inter-POP links (details in [29]). The use of only the tier-1 topology can be justified on two grounds. First, a large proportion of traffic between a randomly chosen client-target pair on the Internet would pass through a tier-1 ISP. Second, such a simulation gives us an approximate idea about the overhead that a PIAS deployment restricted to tier-1 ISPs would entail.

The topology was annotated with the actual distance between POPs (in Kms) based on their geographical locations. We then used SSFNET[30] to simulate BGP route convergence. This allowed us to construct forwarding tables at each of the POPs and hence, determine the forwarding path between any two POPs.

The simulated PIAS deployment involves placing a variable number of proxies at random POPs, one proxy per POP. These POPs are referred to as the *proxy POPs*. For every client-target pair to be simulated, we choose a POP through which the client's packets enter the topology (the *client POP*) and a POP through which the target's packets enter the topology (the *target POP*). The forwarding paths between the client and the target through these POPs represents the *direct path*. The IAP is assumed to be in the *proxy POP* closest to the client POP—this is the *IAP POP*. Similarly, the JAP is in the *proxy POP* closest to the target POP—this is the *JAP POP*. The *PIAS path* comprises of the following three segments: from the *client POP* to the *IAP POP*, from the *IAP POP* to the *JAP POP* and from the *JAP POP* to the *target POP*.

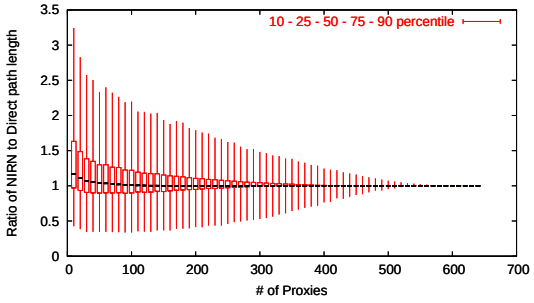


Figure 8: Percentiles for the stretch with varying number of proxies

Figure 8 plots the percentiles for the stretch with varying number of proxies. For a given number of proxies, we simulated 100000 runs. Each run comprised of simulating a client-target pair and finding the *direct* and the *PIAS path* length (in kms). Note that the well-documented non-optimal nature of inter-domain routing[14] is reflected in the cases where the PIAS path turns out to be shorter than the direct path. The figure shows that with a deployment of just 100 proxies (a mature deployment might encompass 50 times more POPs), the median stretch is 1.01 with the 90^{th} percentile being 2.2. Hence, even with a small size deployment, PIAS performs well with regards to the direct path.

4.4 Implementation

We have implemented the PIAS system and are in the process of deploying it. The current implementation of PIAS proxies comprises of a user-space component responsible for the overlay management tasks, such as handling proxy failures, target join/leaves, health monitoring etc. and a kernel-space component responsible for the actual forwarding of packets through the use of Netfilter hooks[31]. This involves tunnelling of the packets when sending them between 2 proxy nodes, and using a NAT when handling packets to/from a target.

5. RELATED WORK

Table 2 summarizes the pros and cons of PIAS, application level anycast, and other related approaches described below.

Partridge et. al. [1] originally proposed the IPv4 anycast service. It involves assigning an otherwise unicast IP address IP_A to multiple hosts, and advertising it into the routing infrastructure from all the hosts. Packets addressed to IP_A will be forwarded to the host nearest to the packet source in terms of metrics used by the routing protocol. Later, IPv6 incorporated anycast into its addressing architecture[8]. It allowed for scoped anycast addresses for groups confined to a topological region, which does not burden the global routing system. However, a globally spread group still poses scalability problems. Besides, IPv6 anycast also inherits all the other

Criterion (related to goal number)	IPv4	IPv6	IP + GIA	App. Level	*i*3	PIAS
Router Modification(1)	No	No	**Yes**	No	No	No
Client Modification(1)	No	No	No	No	**Yes**	No
Scalability by group size(2)	Very Good	Very Good	Very Good	**Poor**	**Poor/Good**[10]	Good
Stretch(3)	No	No	Little/No	No	Little	Little
Robustness(4)	No Issues	No Issues	No Issue	Mixed	Mixed	Mixed[11]
Failover(5)	Fast[12]	Fast[12]	Fast[12]	Fast	Fast	Fast
Target Deployment(6)	**Difficult**	**Difficult**	**Difficult**	Easy	Easy	Easy
Scalability by no. of groups(7)	**No**	**No**	Yes	Yes	Yes	Yes
Scalability by group dynamics(8)	**Poor**	**Poor**	**Poor**	**Poor**	**Poor/Good**[10]	Good
Cost of Proximity(9)	None	None	Small	**Large**	**Large**	Small
Low-level access	Yes	Yes	Yes	**No**	Yes	Yes

Table 2: The Anycast Design Space

limitations of IPv4 anycast. Despite the shortcomings, there has been work detailing the relevance of anycast as a tool for service discovery and other applications, both for IPv4[32] and for IPv6[33].

Katabi and Wroclawski[34] proposed an architecture that allows IP anycast to scale by the number of groups. Their approach is based on the observation that services have a skewed popularity distribution. Hence, making sure that the unpopular groups do not impose any load on the routing infrastructure addresses the scalability issue. However, the need to change routers puts a severe dent on the practical appeal of the approach. Besides, being a router-based approach, it suffers from most other limitations of IPv4 anycast.

Because of the limitations of these approaches, anycast today is typically implemented at the application layer. This offers what is essentially anycast service discovery—DNS-based approaches use DNS redirection while URL-rewriting approaches dynamically rewrite the URL links as part of redirecting a client to the appropriate server. Related proposals in the academic community include [35][36]. The idea behind these is to identify the group using an application level name that, at the beginning of the communication, is mapped to the unicast address of a group member. The reliance on unicast support from the underlying IP layer implies that these approaches circumvent all limitations of IP anycast. The challenge here is to collect the relevant selection metrics about the group members in an efficient and robust fashion.

Another element in this design space is anycast built on top of the indirection architecture offered by i3[37]. i3 uses identifiers as a layer of indirection that generically gives the receiver tremendous control over how it may (or may not) be reached by senders. One of the services i3 can provide is anycast. There are two main advantages of PIAS over i3 for the anycast service. First, PIAS requires no changes in the protocol stack, whereas i3 requires a new layer inserted below transport. A PIAS client, on the other hand, can use PIAS with no changes whatsoever. Second, because PIAS uses native IP anycast, it is easier to derive proximity from PIAS than from i3. PIAS only has to measure distances between proxies—i3 has to measure distances to clients and targets. The main advantage of i3 over PIAS is that it is easier to deploy an i3 infrastructure than a PIAS infrastructure, precisely because i3 doesn't require IP anycast. Indeed, this has been a source of frustration for us—we can't just stick a PIAS proxy on Planetlab and start a service.

As far as the broader notion of indirection is concerned, there is no question that i3 is more general. Its ability for both the sender or receiver to chain services is very powerful. The addressing space is essentially infinite, and hosts can create addresses locally. Finally the security model (that supports the chaining) is elegant and powerful. Having said that, PIAS does provide indirection from which benefits other than just anycast derive. For unicast communications, it could be used to provide mobility, anonymity, DoS protection, and global connectivity through NATs. In the best of all worlds, we'd want something like i3 running over PIAS. But IPv6 and NAT have taught us that you don't always get the best of all worlds, and considering PIAS's backwards compatibility, it may after all be the more compelling story.

6. ANYCAST APPLICATIONS

Given that PIAS offers an easy-to-use global IP anycast service that combines the positive aspects of both native IP anycast and application-layer anycast, it is interesting to consider new ways in which such a service could be used.

6.1 Peer Discovery

Though IP anycast has long been regarded as a means of service discovery, this has always been in the context of clients finding servers. PIAS opens up discovery for P2P networks, where not only is there no client/server distinction, but peers must often find (and be found by) multiple peers, and those peers can come and go rapidly. Examples of such cases include BitTorrent and network games.

One reason that traditional IP anycast has not worked for peer discovery (other than difficulty of deployment), is that an IP anycast group member cannot send to the group—packets are just routed back to themselves. With the right selection characteristics, PIAS can support a wide-range of P2P applications. Random selection would allow peers to find arbitrary other peers, and is useful to insure that unstructured P2P networks are not partitioned. Proximity is obviously also important, but to insure that a peer can find multiple nearby peers (rather than the same peer over and over), a selection service whereby a node can provide a short list of targets to exclude (i.e. already-discovered targets) could be used.

6.2 Reaching an Overlay network

A very compelling application of PIAS would allow a RON[14] network to scale to many thousands of members, and would allow those members to use RON not only for exchanging packets with each other, but with any host on the Internet! What follows is a high-level description of the approach. Assume a set of 50-100 RON "infrastructure" nodes that serve

[10]Note that the way i3 has described their anycast, it wouldn't scale to very large or very dynamic groups, because a single node holds all the targets and receives pings from the targets. It may be possible that i3 could achieve this with a model closer to how they do multicast, but we're not sure.

[11]for reasons described in first paragraph of section 3.6

[12]they can be engineered to be fast by relying on IGP for convergence

many thousands of RON clients. The RON nodes all join a large set of anycast groups—large enough that there is an anycast transport address (TA) for every possible client connection. The RON nodes also partition the anycast TAs so that each TA maps to a single RON node. Clients discover nearby RON nodes (or a couple of them) using one of the anycast groups, and establish a unicast tunnel (for instance, a VPN tunnel) with the RON node. We call this the *RON tunnel*, and the RON node is referred to as the *local RON*.

When a client wishes to establish a connection with some remote host on the Internet, it does so through its RON tunnel. The local RON assigns one of its TAs to the connection using NAT, and forwards the packet to the remote host. When the remote host returns a packet, it reaches a nearby RON node, called the *remote RON*. Because the transport address of the return packet maps to the local RON node, the remote RON node can identify the local RON node. The remote RON tags the packet with its own identity, and transmits the packet through the RON network to the local RON node, which caches the identity of the remote RON, and delivers the packet to the client. Now subsequent packets from the client to the remote host can also traverse the RON network.

This trick isn't limited to RONs. It could also work for route optimization in Mobile IP[13] (for v4 or v6, see [38] for a description of the problem), or simply as a way to anonymize traffic without sacrificing performance.

7. DISCUSSION

In this paper, we have presented the basic aspects of PIAS. A "practical" IP anycast service, however, requires a number of features that we don't have space to describe in detail. For example, the need for scoping whereby packets from clients in a domain (enterprise) are always served by targets within the domain. This can be achieved by deploying a PIAS proxy in the domain, or simply by deploying intra-domain native IP anycast.

Another important issue is security. The IP routing infrastructure is secured router-by-router through human supervision of router configuration. This makes routing security error-prone and unreliable. Since PIAS involves advertising a prefix into inter-domain routing, it is afflicted by the same issues. However, it is important to note that PIAS does not worsen the situation. Also, the fact that from the routing point of view, an anycasted autonomous system is akin to a multi-homed autonomous system implies that any future solution for routing security would apply directly to the PIAS deployment.

PIAS, however, does need to explicitly secure its *join* and *leave* primitives. The fact that these primitives are to be used by group members who have an explicit contract with the anycast service provider implies that we could use standard admission control schemes; for example PIAS could adapt any of a number of network or wireless authentication protocols like EAP [39]. Previous work on using overlays to protect specific targets from DOS attacks [40] described some approaches to allow controlled access to the overlay.

An assumption implicit in PIAS's claim of incurring minimal stretch (section 4.3) is the proximity of the client to the IAP and of the server to the JAP. This assumption is justified by the fact that these relations are discovered using native IP anycast and hence, the distances are small in terms of metrics used by inter-domain routing. However, this does not necessarily imply that the distances are small in terms of latency. As a matter of fact, preliminary measurements done by us show that the assumption does not hold for the j-root server anycast deployment. We found that native IP anycast does not do a great job of selecting close-by locations, at least not for the j-root server deployment. For example, 40% of the measured clients experienced a stretch of more than 4 when accessing the anycasted j-root. The measurement methodology and the results are detailed in [41].

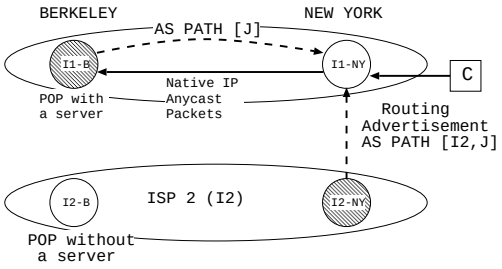


Figure 9: Native IP anycast inefficiency - packets from client C in New York destined to the native IP anycast address are routed to the anycast server in Berkeley, even though there is a server in New York

We believe the inefficacy of anycast when selecting close-by root-servers might be due to the way the j-root servers have been deployed - all 13 anycasted servers for j-root are placed in POPs of different ISPs. A possible problem with this approach is illustrated in figure 9. The figure shows 2 ISP networks- I1 and I2, each having a POP in New York and in Berkeley. It also shows a native IP anycast deployment (AS number J) with two servers - one hosted at the New York POP of I2 (I2-NY) and the other at the Berkeley POP of I1 (I1-B). The figure has these POPs highlighted. The anycast servers have an EBGP relation with the routers of the hosting POP; hence, the anycast prefix is advertised with J as the origin AS. Now, if a client (C) in the New York area sends packets to the anycast address and these reach POP I1-NY, they will be routed to the server hosted at I1-B. This is because the routers in I1-NY would prefer the 1 AS-hop path ([J]) through I1-B to the anycasted server over the 2 AS-hop path ([I2,J]) through I2-NY. Note that the anycasted server hosted at I1-B represents a customer of I1 and so, it would be very uncommon for I1 to steer these packets towards I2-NY due to local policies (local preference values); rather the AS path length would dictate the path.

Although negative, the importance of the result cannot be overemphasized. It brings out the fact that a naive proxy deployment might not achieve low-latency client-IAP and JAP-target paths. Also, an unverified implication of the above analysis is that for good performance, an ISP that is part of the deployment[14] should be sufficiently covered, i.e., there should be clusters at a decent number of POPs of the ISP. For example, deployment of the two servers in the figure at both of the POPs of I1 (I1-NY and I1-B) or I2 (I2-NY and I2-B) would avoid the problem of long paths. We believe that such an approach would ensure that the client-IAP and the target-JAP segments are latency-wise small - something that can only be substantiated when we get the PIAS deployment going

[13]Details withheld for lack of space.

[14]the ISP has at least one POP hosting a proxy cluster

8. CONCLUSIONS

In this paper, we propose a proxy based IP anycast service that addresses most of the limitations of native IP anycast. Specifically, the primary contribution of this paper is the design of PIAS, a practically deployable IP anycast architecture. The unique features of PIAS such as the scalability by the size and dynamics of groups mean that it opens up new avenues of anycast usage. The purported scalability has been substantiated through simulations representing extreme, but real, workloads. Simulations on the real tier-1 topology of the Internet point to the efficiency of our approach.

The fact that PIAS uses native IP anycast means that it can be used as a simple and general means of discovery and bootstrapping. Internet measurements against the anycasted DNS root servers show that the reliance on native IP anycast does not undermine PIAS's ability to support connection oriented services. A PIAS prototype has been built and the deployment efforts are underway. We feel confident that PIAS has the potential of fulfilling the need for a generic Internet-wide anycast service that can serve as a building block of many applications, both old and new.

Acknowledgements

We are grateful to Xinyang Zhang for help with the simulations and to David Anderson for design discussions. We would also like to thank the anonymous reviewers for their feedback. This material is based upon work supported by AFOSR MURI and IAI AFOSR/AFRL under award numbers F49620-02-1-0233 and F49620-02-1-0170 respectively. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the agencies above.

9. REFERENCES

[1] C. Partridge, T. Mendez, and W. Milliken, "RFC 1546 - Host Anycasting Service," November 1993.

[2] T. Hardy, "RFC 3258 - Distributing Authoritative Name Servers via Shared Unicast Addresses," April 2002.

[3] J. Abley, "Hierarchical Anycast for Global Service Distribution," ISC Technical Note ISC-TN-2003-1 www.isc.org/tn/isc-tn-2003-1.html.

[4] D. Kim, D. Meyer, H. Kilmer, and D. Farinacci, "RFC 3446 - Anycast Rendevous Point (RP) mechanism using Protocol Independent Multicast (PIM) and Multicast Source Discovery Protocol (MSDP)," January 2003.

[5] D. Katabi, "The Use of IP-Anycast for Building Efficient Multicast Trees," in *Proc. of Global Telecommunications Conference*, 1999.

[6] C. Huitema, "RFC 3068 - An Anycast Prefix for 6to4 Relay Routers," June 2001.

[7] "AS112 Project Home Page," www.as112.net.

[8] R. Hinden and S. Deering, "RFC 3513 - Internet Protocol Version 6 (IPv6) Addressing Architecture," April 2003.

[9] Akamai Technologies Inc., "Internet Bottlenecks: the Case for Edge Delivery Services," 2000, www.akamai.com/en/resources/pdf/whitepapers/Akamai_Internet_Bottlenecks_Whitepaper.pdf.

[10] B. Greene and D. McPherson, "ISP Security: Deploying and Using Sinkholes," www.nanog.org/mtg-0306/sink.html, June 2003, NANOG TALK.

[11] D. R. Karger, E. Lehman, F. T. Leighton, R. Panigrahy, M. S. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." in *Proc. of STOC*, 1997.

[12] R. Rodrigues, B. Liskov, and L. Shrira, "The design of a robust peer-to-peer system," in *Proc. of the Tenth ACM SIGOPS European Workshop*, September 2002.

[13] A. Gupta, B. Liskov, and R. Rodrigues, "One Hop Lookups for Peer-to-Peer Overlays," in *Proc. of 9th Workshop on Hot Topics in Operating Systems*, May 2003.

[14] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *Proc. of the eighteenth ACM Symposium on Operating Systems Principles*, 2001.

[15] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points." in *Proc. of INFOCOM*, 2002.

[16] N. Spring, R. Mahajan, and T. Anderson, "Quantifying the Causes of Path Inflation," in *Proc. of ACM SIGCOMM*, August 2003.

[17] Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. of ACM SIGCOMM*, 2002.

[18] J. Abley, "A Software Approach to Distributing Requests for DNS Service Using GNU Zebra, ISC BIND 9, and FreeBSD," in *Proc. of USENIX Annual Technical Conference, FREENIX Track*, 2004.

[19] T. S. E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches." in *Proc. of INFOCOM*, 2002.

[20] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *Proc. of ACM SIGCOMM*, 2004.

[21] C. Alaettinoglu and S. Casner, "Detailed Analysis of ISIS Routing Protocol on the Qwest Backbone," February 2002, NANOG TALK.

[22] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, "SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication," in *Proc. of the Third International COST264 Workshop on Networked Group Communication*, 2001.

[23] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proc. of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[24] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, July 2003.

[25] "ISC F-Root Sites," www.isc.org/index.pl?/ops/f-root/.

[26] P. Barber, M. Larson, M. Kosters, and P. Toscano, "Life and Times of J-Root," www.nanog.org/mtg-0410/kosters.html, October 2004, NANOG TALK.

[27] R. Bush, Mailing list posting www.ripe.net/ripe/maillists/archives/routing-wg/2004/msg00183.html.

[28] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The feasibility of supporting large-scale live streaming applications with dynamic application end-points," in *Proc. of ACM SIGCOMM*, 2004.

[29] X. Zhang, J. Wang, and P. Francis, "Scaling the Internet through Tunnels," pias.gforge.cis.cornell.edu/tbgp.pdf.

[30] "SSFNet," www.ssfnet.org/homePage.html.

[31] "Netfilter," www.netfilter.org.

[32] E. Basturk, R. Haas, R. Engel, D. Kandlur, V. Peris, and D. Saha, "Using IP Anycast For Load Distribution And Server Location," in *Proc. of IEEE Globecom Global Internet Mini Conference*, November 1998.

[33] S. Matsunaga, S. Ata, H. Kitamura, and M. Murata, "Applications of IPv6 Anycasting," draft-ata-ipv6-anycast-app-00, February 2005.

[34] D. Katabi and J. Wroclawski, "A framework for scalable global IP-anycast (GIA)," in *Proc. of ACM SIGCOMM*, 2000.

[35] E. W. Zegura, M. H. Ammar, Z. Fei, and S. Bhattacharjee, "Application-layer anycasting: a server selection architecture and use in a replicated Web service," *IEEE/ACM Trans. Netw.*, vol. 8, no. 4, pp. 455–466, 2000.

[36] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service." in *Proc. of INFOCOM*, 1998.

[37] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet Indirection Infrastructure," in *Proc. of ACM SIGCOMM*, 2002.

[38] "Mobility for IPv6 (mip6), IETF Working Group Charter," www.ripe.net/ripe/maillists/archives/routing-wg/2004/msg00183.html.

[39] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, "RFC 3748 - Extensible Authentication Protocol (EAP)," June 2004.

[40] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: secure overlay services." in *Proc. of ACM SIGCOMM*, 2002.

[41] H. Ballani and P. Francis, "Root-Server Anycast Deployment: A Meaurement Study," pias.gforge.cis.cornell.edu/am.pdf.

A Study of Prefix Hijacking and Interception in the Internet

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

Xinyang Zhang
Cornell University
Ithaca, NY
jzhang@cs.cornell.edu

ABSTRACT

There have been many incidents of prefix hijacking in the Internet. The hijacking AS can blackhole the hijacked traffic. Alternatively, it can transparently intercept the hijacked traffic by forwarding it onto the owner. This paper presents a study of such prefix hijacking and interception with the following contributions: (1). We present a methodology for prefix interception, (2). We estimate the fraction of traffic to any prefix that can be hijacked and intercepted in the Internet today, (3). The interception methodology is implemented and used to intercept real traffic to our prefix, (4). We conduct a detailed study to detect ongoing prefix interception.

We find that: Our hijacking estimates are in line with the impact of past hijacking incidents and show that ASes higher up in the routing hierarchy can hijack a significant amount of traffic to any prefix, including popular prefixes. A less apparent result is that the same holds for prefix interception too. Further, our implementation shows that intercepting traffic to a prefix in the Internet is almost as simple as hijacking it. Finally, while we fail to detect ongoing prefix interception, the detection exercise highlights some of the challenges posed by the prefix interception problem.

Categories and Subject Descriptors: C.2.2 [Network Protocols]: Routing Protocols.

General Terms: Measurement, Security.

Keywords: Routing, BGP, Hijacking, Interception.

1. INTRODUCTION

In the recent past, there have been many instances of "prefix hijacking" in the Internet wherein an Autonomous System "hijacks" routes simply by advertising the corresponding prefixes. Such incidents are regularly reported on the NANOG mailing list [1]; [2–6] report a few specific ones. This, in turn, has prompted a number of proposals to address the problem [3,4,7–21] – some of these target the specific goal of detecting prefix hijack attempts while others strive to improve the general security of inter-domain routing.

Irrespective of whether it is caused by a misconfiguration or a malicious entity, the AS that hijacks a prefix can *blackhole* all the hijacked traffic and thus, cause a denial-of-service attack against the prefix owner [22]. It can also *redirect* the traffic to an incorrect destination and use this for a phishing attack [22]. Spammers have also been known to use hijacked prefixes [23]. In all these cases, the prefix's traffic does not reach the destination. However, it is also possible for an AS to hijack the traffic to a prefix and then *forward this traffic on to the prefix owner* [22,24]. Hence, instead of blackholing the destination's traffic, this would allow the AS to "intercept" the traffic without disrupting the destination's connectivity to the Internet and thus, become a man-in-the-middle. For instance, this may be used by an AS in the USA to transparently capture, record and subsequently deliver IP traffic between end points in Europe and the Middle East.

While these attacks are a bleedingly obvious consequence of the way inter-domain routing operates, their egregiousness cannot be disputed. This is especially true for interception since the intercepted traffic still reaches the proper destination. Consequently, it is less likely that an unsuspecting victim would notice ongoing interception and unlike the other possibilities, this is one case where a prefix could actually be hijacked for a long period. Indeed, it is possible that interception may be happening undetected, on a regular basis, on the Internet today!

However, despite all the incidents and subsequent work in the research community, an actual quantification of the impact of prefix hijacks on the Internet is sorely missing. Motivated by this, in this paper we present an analysis of the impact of an invalid advertisement on ASes in the Internet with specific emphasis on the possibility and practical feasibility of using routing advertisements for traffic interception. To this effect, this paper studies the following aspects of Internet prefix hijacking and interception (with our contributions italicized):

First, we use common routing policies to *analyze the probability of an AS hijacking the traffic to a prefix from another AS.* Note that while hijacking traffic to a prefix simply involves advertising the prefix into inter-domain routing, prefix interception seems trickier because the invalid advertisement originated by the hijacking AS can impact the valid route that it uses to forward the traffic to the prefix's owner. Consequently, we extend our analysis to *determine scenarios where interception is possible and propose a general methodology for prefix interception.* Our analysis shows that a hijacking AS, with high probability, can statically determine the neighbors to which it can safely advertise an invalid route for a prefix while still being able to forward the hijacked traffic back to the prefix owner.

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.

Second, we use routing tables collected at the Route-Views repository [25] to *estimate the fraction of other Route-Views ASes whose traffic to any prefix can be hijacked and intercepted by a given Route-Views AS*. As one would expect, our estimates show that tier-1 ASes can, on average, hijack traffic to any prefix from a significant fraction of ASes (52% to 79%). These estimates also apply to hijacking of popular prefixes that carry a lot of traffic. Further, tier-1 ASes can route all the hijacked traffic back to the owner and hence, can also intercept traffic to any prefix from a significant fraction of ASes. However, these fractions drop off for ASes lower down in the routing hierarchy. For instance, tier-3 ASes and beyond can, on average, hijack traffic to any prefix from 13% to 31% of ASes and intercept traffic from 7% to 17% of ASes. We also *verified our estimates against known prefix-hijacking events on the Internet* and found them to be fairly accurate.

Third, we *implement the aforementioned interception meth--odology and use it to actually intercept traffic to a prefix* belonging to us in five different scenarios. Further, in each scenario, we probe the prefix from >20,000 vantage points to *quantify the fraction of traffic that can be hijacked and the fraction that can be intercepted*. These results, at the very least, provide anecdotal evidence of the claim that a significant amount of traffic to prefixes on the Internet can be intercepted. Moreover, the implementation suggests that intercepting traffic to a prefix in the Internet is almost as simple as hijacking it, requiring changes only in BGP routing policy at the intercepting AS.

Finally, we use a combination of control-plane and data-plane information to *look for actual interception in the Internet*. The study yielded a few unexplained anomalies that could be due to prefix interception. However, our analysis shows that these anomalies can just as well arise from valid routing arrangements. While negative, this result captures some of the challenges in detecting ongoing prefix interception. More generally, the estimates presented in this paper rely on a simplistic model of Internet routing and have several other limitations that we discuss in section 7. However, in spite of these limitations, our quantification and implementation efforts serve to highlight the severity of the problem. In this context, we hope that this paper would bring to the fore the (obvious) possibility of traffic interception in today's inter-domain routing and influence the design of Internet security protocols.

2. METHODOLOGY

ASes in the Internet can use *invalid advertisements* for a *target prefix*, i.e. advertisements with an AS-PATH that does not represent the true AS-PATH to the prefix, to convince other ASs to route traffic for the prefix to itself and hence, hijack the prefix. Among other things, the *hijacking AS* can forward the hijacked traffic to the owner and hence intercept the prefix. Consequently, prefix interception is always preceded by prefix hijacking.

The most obvious form of an invalid advertisement is one where the *hijacking AS*, say X, claims to own the prefix and hence, advertises the prefix with AS-PATH=[X]. We refer to this as an advertisement with an *invalid origin*. However, such an invalid advertisement would lead to a Multiple Origin AS (MOAS) anomaly [26]. The hijacking AS can avoid this by advertising the prefix with AS-PATH=[X, O] where AS O is the owner of the prefix. We refer to this as an advertisement with an *invalid next hop*. Of course, the hijacking AS can

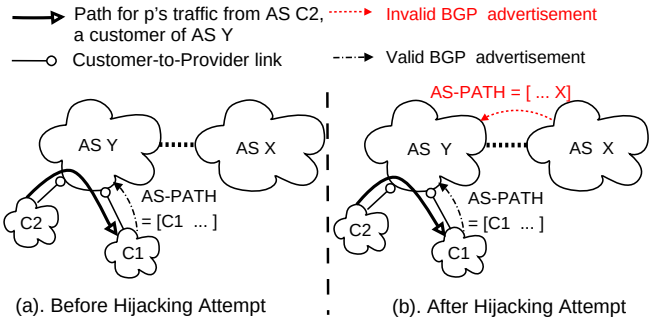


Figure 1: AS Y has an existing customer-route to p and hence, hijacking p's traffic from Y with an invalid provider or peer route is not possible.

advertise the prefix with an even longer AS-PATH but, as we show later in the paper, that would significantly reduce the amount of traffic it can hijack. Hence, we focus on hijacking and interception with routing advertisements that have an invalid origin or an invalid next hop.

Apart from advertising an invalid route for an already rout--able prefix, there are a couple of other approaches that an AS could possibly use for hijacking traffic to a prefix:

(a). An AS could advertise a more specific prefix than the one being advertised by the owner and this would hijack all the traffic to the specific prefix. However, the hijacking AS would not be able route this traffic onto the owner and hence, interception would not be possible.

(b). As AS could advertise a less specific prefix than the one being advertised by the owner. This would hijack traffic to the prefix only when the owner withdraws its advertisements. However, even in that situation, the hijacking AS would not be able to route the hijacked traffic to the owner.

Since the impact of such advertisements can be trivially predicted, we don't study them here. Hence, our estimates for the fraction of traffic that can be hijacked and the fraction that can be intercepted are restricted to hijacking based on advertisement of the same prefix as the one being advertised by the owner.

The discussion in the rest of this section focusses on an AS X trying to hijack (and intercept) the traffic for target prefix p. In the first part of the section we analyze X's ability to hijack p's traffic using an advertisement with an invalid origin (though the arguments can trivially be extended to advertisements with an invalid next hop), while in the second part we study how X ensures that it can forward the hijacked traffic back to p's owner.

2.1 Hijacking Analysis

AS X advertises an invalid route for prefix p with AS-PATH=[X]. We want to evaluate the impact of this advertisement on AS Y that is (n-1) AS hops away from X and thus, receives a route of AS-PATH length n.[1] Specifically, we would like to determine if Y chooses this invalid route over its existing route to p, thus allowing AS X to hijack p's traffic sourced from it. Here, "traffic sourced from AS Y" refers to

[1] Y may be topologically closer to X than (n-1) hops but the shortest path that the invalid advertisement needs to propagate to reach Y comprises of (n-1) ASes.

traffic originating at Y plus "traffic sourced from any of Y's neighbors" that is routed through Y.

Obviously, AS Y's choice depends on both its existing route and the newly-received invalid route to p. We term a route to be a "customer-route" or a "peer-route" or a "provider-route" depending on whether the next-hop AS in the AS-PATH is a customer, a peer or a provider respectively. Since both the existing route and the invalid route could be any of these, there are nine cases to consider. Below we try to answer the aforementioned question for each of these cases, given two assumptions:

(a). The invalid route advertised by AS X reaches AS Y. ISPs are known to install route filters so as to accept advertisements only for specific prefixes from their neighbors, especially if the neighbor is a stub AS [27]. Thus, route filters employed by any AS along the path from X to Y would falsify this assumption. Further, for the invalid advertisement to actually reach Y, it must be accepted and propagated by all ASes along the path. Thus, an implicit assumption here is that X is able to hijack traffic from all ASes along the path from X to Y (in practice, one could verify this assumption by applying the analysis presented below to each AS along the path).

(b). AS Y's choice also depends on its routing policies. Measurement studies in the past have shown that a large majority of ASes on the Internet tend to assign higher local-preference values to customer-routes than to peer-routes than to provider-routes [28]. Since local-preference values are the first step of the BGP decision process [29], ASes prefer customer routes to peer routes to provider routes. We assume that this holds for Y as this lets us analyze the possibility of Y's traffic being hijacked. Further, a part of the analysis also assumes that AS Y assigns the same local-preference value to all its customers, the same value to its peers and the same value to its providers; however, most of the arguments below apply even if this last assumption does not hold. As detailed in section 3.1, we verified these assumptions for tier-1 ASes.

Cases 1-3. *Existing route is customer-route, invalid route is a customer/peer/provider route.* If the invalid route that AS Y receives is a peer or a provider route, irrespective of the attributes (for example, the AS-PATH length) of this route, Y prefers the existing customer-route (assumption (b)). Thus, Y's traffic is not hijacked. Figure 1 shows this scenario.

On the other hand, if the invalid route is a customer-route, AS Y's policy would give equal preference to both routes and hence, the decision is based on the length of the route [29]. If the AS-PATH length of the existing route is less than n, it is preferred. If the AS-PATH length of the existing route is more than n, the invalid route is preferred. Finally, if Y's existing route is n AS-hops long, it must choose between two routes with the same local preference and the same length. This choice is based on other factors such as the IGP metric of the routes [29]. Consequently, some routers belonging to Y may choose to stick with the existing route while others may choose to use the invalid route. Hence, in this case, some fraction of Y's traffic for p may be hijacked. Figure 2 shows this scenario.

Case 4-6. *Existing route is a peer route, invalid route is a customer/peer/provider route.* If the invalid route that AS Y receives is a provider route, it prefers the existing peer-route. Thus, Y's traffic is not hijacked. As a contrast, if the invalid route is a customer-route, Y prefers it and Y's traffic is hijacked.

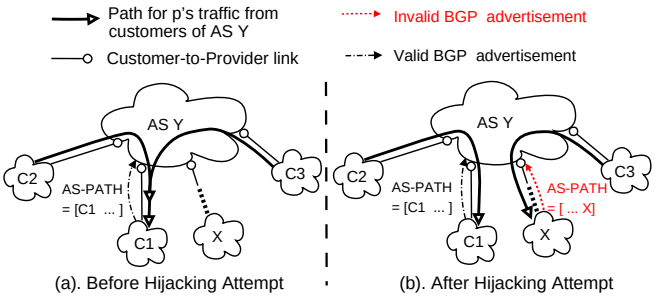


Figure 2: AS Y has an existing customer-route to p and receives an invalid route (advertised by AS X) of equal length through a customer. This causes some fraction of p's traffic to be hijacked.

Invalid route ⇒ Existing route	Length	Customer	Peer	Provider
	<n	✗	✗	✗
Customer	=n	–	✗	✗
	>n	✓	✗	✗
	<n	✓	✗	✗
Peer	=n	✓	–	✗
	>n	✓	✓	✗
	<n	✓	✓	✗
Provider	=n	✓	✓	–
	>n	✓	✓	✓

Table 1: AS Y's traffic to prefix p can (✓), cannot (✗) or can partly (–) be hijacked depending on its existing route and the invalid route.

Finally, if the invalid route is a peer-route, AS Y's policy would give equal preference to both routes and hence, the decision is based on the AS-PATH length of the route [29]. If the length of the existing route is less than n, traffic is not hijacked; if the length is more than n, traffic is hijacked; if the length is n AS hops, some fraction of the traffic may be hijacked.

Case 7-9. *Existing route is a provider route, invalid route is a customer/peer/provider route.* The possibility of hijacking AS Y's traffic in these cases follows from the arguments presented above. Table 1 summarizes the hijacking possibility for all nine cases.

2.2 Interception Analysis

In order to be able to intercept traffic to the prefix p, the hijacking AS needs to forward the hijacked traffic on to p's owner. It can do so by forwarding the hijacked traffic along its existing valid route to p. Figure 3 shows the process by which hijacking AS X hijacks prefix p's traffic from Y (originating at Y's customer $C2$) and then forwards it on to p's owner through its peer W. However, for this to work, X's existing route to p should not be impacted by the invalid route that it advertises. Hence, the hijacking AS X would like to ensure the following *safety*[2] condition:

> *None of the ASes along the route to prefix p used by the hijacking AS should choose the invalid route advertised by it (if they do receive the invalid route) over their existing route to p.*

Note that the obvious way for AS X to satisfy the above

[2]Here, safety refers to the fact that X does not introduce routing instability and is able to route the hijacked traffic to its owner.

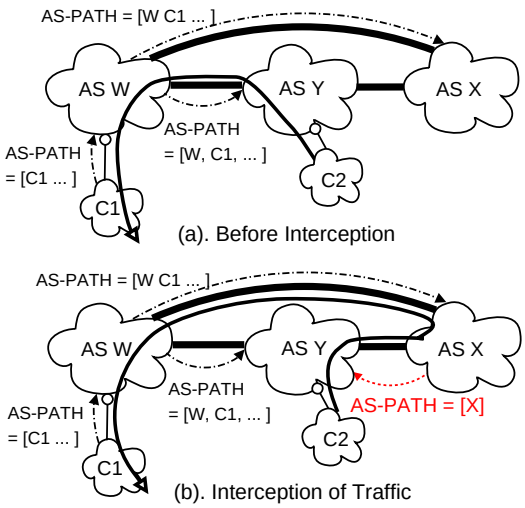


Figure 3: AS X uses an invalid advertisement to hijack traffic from AS Y and then routes the traffic to the owner using its existing route through peer AS W.

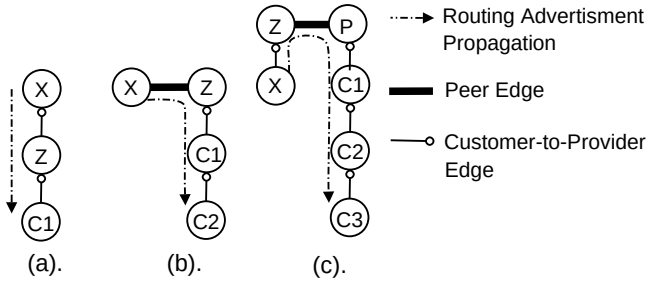


Figure 4: Propagation of the invalid route advertised by AS X to its (a) customer, (b) peer, (c) provider.

condition would be to advertise the invalid route such that the traffic from the ASes along its existing route to p is not hijacked. In theory, the discussion from the previous section applies to the possibility of hijacking from these ASes. However, this observation doesn't have much practical value since X wouldn't know how an invalid route advertised to any of its neighbors would be propagated to these ASes and hence, would not be able to determine if an invalid advertisement can indeed hijack the traffic from a given AS along the path.

Instead, we would like to analyze if a hijacking AS can ensure the safety condition based on local information alone. Specifically, AS X would like to determine if advertising an invalid route for p to a neighboring AS, say Z, can impact its existing route for p. X's existing route to p can be a customer, peer or provider route and Z can be X's customer, peer or provider and hence, there are nine cases to consider. Below we try to answer the aforementioned question, given two assumptions:

(a). As with the hijacking analysis, we assume that ASes prefer customer routes to peer routes to provider routes.

(b). We assume that Internet paths follow the "Valley-free" property [30], i.e. after traversing a provider-to-customer edge or a peer edge, the path cannot traverse another customer-to-provider or peer edge. Analogously, once a routing advertisement traverses a provider-to-customer edge or a peer edge, the advertisement cannot traverse another customer-to-provider or peer edge.

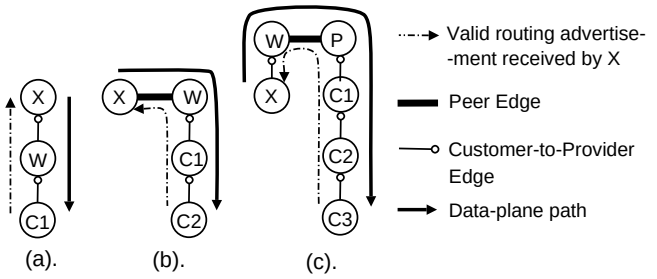


Figure 5: AS X's existing route for prefix p is through a (a) customer, (b) peer, (c) provider.

Consequently, when X advertises the invalid route to a customer, the advertisement can only traverse provider-to-customer edges. Hence, the advertisement is restricted to ASes below X in the AS hierarchy and represents a provider route for these ASes. When X advertises the invalid route to a peer, the advertisement traverses one peer edge followed by provider-to-customer edges only. Hence, other than the peer being advertised to, the advertisement is restricted to ASes below X in the AS hierarchy and represents a provider route for these ASes. Figure 4(a,b) illustrate these scenarios.

When X advertises the invalid route to a provider, each control plane path traversed by the advertisement comprises of one or more customer-to-provider edges followed by zero or one peer edges and zero or more provider-to-customer edges. Hence, the advertisement is propagated to all levels of the AS hierarchy. However, it is important to note that while ASes that are above X in the AS hierarchy may receive the invalid advertisement from a customer, peer or provider, ASes at the same level or below X will always receive the advertisement from a provider (i.e. a provider route). Figure 4(c) illustrates this scenario.

Case 1-3. *X's existing route is a customer route, X advertises the invalid route to a customer/peer/provider.* The fact that X's existing path to p is a customer-route implies that the first edge along this path is a provider-to-customer edge. Further, the valley-free property of Internet paths implies that this is a "downhill path" (as defined by [30]) comprising of a sequence of provider-to-customer edges. Thus, all ASes along the path are below X in the AS hierarchy and use a customer route to p. Figure 5(a) illustrates this scenario. As discussed in assumption (b), irrespective of whether X advertises the invalid route to a customer/peer/provider, the invalid route would appear as a provider route to ASes below X and hence, will not be chosen by them over their existing customer route. Thus, X can advertise the invalid route to all its neighbors.

Case 4-6. *X's existing route is a peer route, X advertises the invalid route to a customer/peer/provider.* The valley-free property implies that X's existing path to p comprises of one peer edge followed by a sequence of provider-to-customer edges. Thus, all ASes along the path use a customer route to p. Figure 5(b) illustrates this scenario. Also, as before, even if the invalid route advertised by X propagates to any of the ASes along the path, it will be a provider or a peer route and hence, will not be chosen over the existing customer route. Thus, X can advertise the invalid route to all its neighbors.

Case 7-9. *X's existing route is a provider route, X advertises the invalid route to a customer/peer/provider.* The valley-free property implies that X's existing path to p comprises of one or more customer-to-provider edges followed by

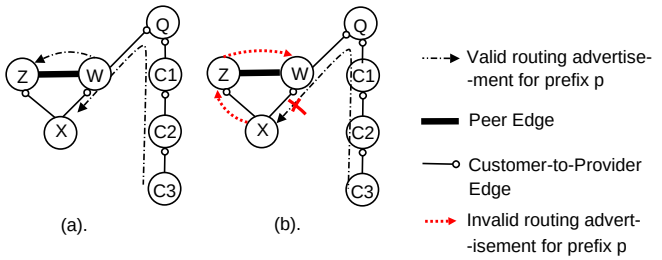


Figure 6: (a) Hijacking AS X has a route for p through provider W. (b) The invalid route advertised by X to another provider Z to intercept p's traffic impacts its existing route for p.

zero or one peer edge followed by zero or more provider-to-customer edges. Hence, ASes along the path may be using a customer or peer or provider route to p. However, any ASes along the path that are at the same level or below X in the AS hierarchy would be using a customer route to p. Figure 5(c) illustrates this scenario.

As discussed in assumption (b), when X advertises the invalid route to a customer or a peer, the advertisement is restricted to ASes at the same level or below X in the AS hierarchy and represents a provider or peer route for them. This implies that the invalid route will not be chosen by these ASes. Hence, X can advertise the invalid route to its customers and peers.

However, when X advertises the invalid route to a provider, the route may be received by ASes above X in the AS hierarchy. For these ASes, both the invalid route and the existing route can be a customer, peer or provider route implying that it is possible they prefer the invalid route. This violates the safety condition and hence, X cannot advertise the invalid route to its providers. Figure 6 shows such a scenario wherein the invalid route advertised by AS X to its provider AS Z impacts its existing route for prefix p.

The analysis presented above implies that an AS trying to intercept traffic for target prefix p can advertise the invalid route to all its neighbors unless its existing route for p is through a provider, in which case the invalid route should not be advertised to other providers of the AS. However, there are a couple of other things to note: First, while our assumptions regarding AS policies and valley-free paths hold for a majority of ASes on the Internet, exceptions certainly do exist. Hence, the aforementioned policy for advertising invalid routes *ensures safety with a high probability*; an AS advertising invalid routes may still cause routing instability and needs to account for it. It can do so by observing if its existing route for p changes as a result of advertising the invalid route. If such a change occurs, the hijacking AS can pin point the anomaly-causing neighbor based on the recently received advertisements for p and hence, stop advertising the invalid route to this neighbor.

Second, even when the hijacking AS's existing route for p is through a provider, advertising the invalid route to another provider may not necessarily impact the AS's route for p. Hence, it is possible to imagine the hijacking AS using an aggressive approach by advertising the invalid route to all neighbors and then stopping the advertisement to specific neighbors if route instability arises.

Based on the description above, the following pseudo-code represents the conceptual process by which hijacking AS X can intercept traffic to target prefix p from its neighbors:

```
If (existing route to p is through a provider)
then
 Advertise to all peers and customers a route
 for prefix p with AS-PATH [X];
else
 Advertise to all neighbors a route for prefix
 p with AS-PATH [X];
endif
If (the invalid advertisement causes the
  existing route for p to change)
then
    Stop the advertisement to the
    anomaly-causing neighbor;
endif
```

3. HIJACKING AND INTERCEPTION ESTIMATES

Given the methodology described in the previous section, we can estimate the fraction of ASes in the Internet whose traffic to a *target prefix* that can be hijacked and intercepted by any given AS.

3.1 Hijacking by tier-1 ASes

Here we focus on hijacking by tier-1 ASes and determine the fraction of other tier-1 ASes whose traffic to a prefix can be hijacked and intercepted by a tier-1 AS in the Internet today. A tier-1 AS is an AS with no providers and a peering with all other tier-1 ASes [31]. Hence, tier-1 ASes are at the top of the routing hierarchy. We used CAIDA's AS ranking tool [31] and commercial reports on AS ranking [32] to come up with a list of 15 highly ranked ASes that are considered as tier-1 ASes in this paper. Note that we treat hijacking by tier-1 ASes as a special case since we can verify the two assumptions made by the analysis presented in section 2.1:

Assumption (a). *An invalid route advertised by the hijacking AS reaches the AS whose traffic ought to be intercepted.* The fact that all tier-1 ASes peer with each other makes this trivially true. Further, it is unlikely that the invalid route advertised by a hijacking tier-1 AS would be filtered out by any of the other tier-1 ASes [27].

Assumption (b). *Tier-1 ASes prefer customer-routes over peer-routes and give the same preference to routes from different peers.*[3] A lot of tier-1 ASes offer publicly-accessible route-servers and policy guides which let us determine their import policies expressed in the form of local-preference values. We were able to do this for nine of the fifteen tier-1 ASes. While we don't show the actual local-preference values in the interest of brevity, we found that this assumption was satisfied for all the nine ASes.

This validation of the assumptions improves our confidence in the accuracy of the estimates presented here. The actual estimates were guided by two observations. First, the fact that tier-1 ASes don't have any provider routes implies that they can safely advertise the invalid route to all neighbors. Consequently, (almost) all traffic that can be hijacked by a tier-1 AS can also be routed back to its owner.

Second, from the point of view of other tier-1 ASes, the invalid route advertised by hijacking AS X is a peer route one AS-hop long. This, combined with table 1, implies that X can hijack all traffic for prefix p from a peer AS if the

[3]Tier-1 ASes don't have provider routes. Also, we focus on hijacking from other tier-1 ASes that are peers of the hijacking AS and hence, their preference amongst routes from different customers is not relevant.

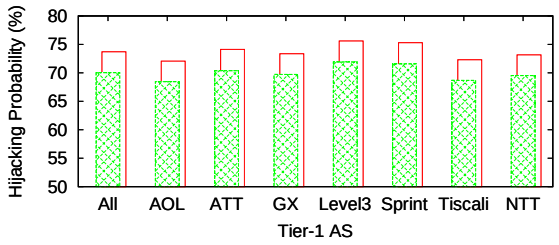


Figure 7: Probability of prefix hijacking on average and for each of the tier-1 ASes that serve as hijacking ASes in our estimation.

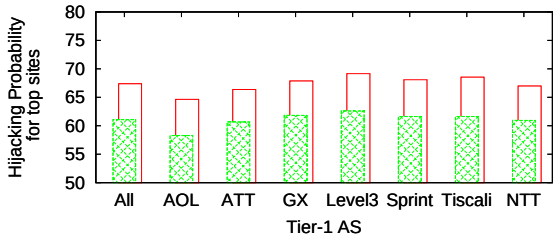


Figure 8: Probability of prefix hijacking for prefixes corresponding to top-100 sites.

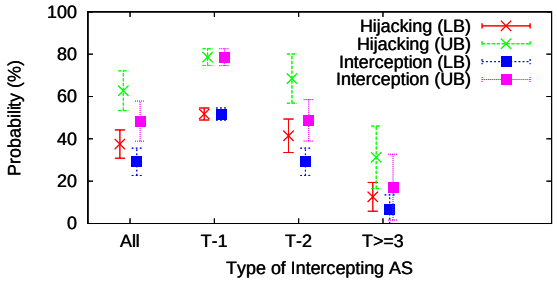


Figure 9: Probability of prefix hijacking and prefix interception for ASes in the RV-set.

peer's existing route for p is a provider route or a peer route of length more than one AS hop. AS X can intercept some fraction of the traffic if the peer's existing route for p is a peer route of length one. However, this fraction depends on both the intra-domain metrics of the peer AS and the locations at which X peers with it. Given that we lack this information, we define the upper and the lower bounds of hijacking; the lower bound assumes that none of p's traffic from such peers is hijacked while the upper bound assumes that all of p's traffic from such peers is hijacked.

Overall, we can determine if X can hijack traffic for prefix p from a peer tier-1 AS based on the peer's existing route for p. Since the Route-Views repository collects routes from 7 of the 15 tier-1 ASes (AOL, ATT, Global Crossing, Level3, Sprint, Tiscali and NTT), we focussed on these seven ASes and for each of them, determined the prefixes in the Internet routing table whose traffic from the other six tier-1 ASes (and their customers) can be hijacked. For ease of exposition, we hereon refer to the fraction of other ASes whose traffic is hijacked by the hijacking AS (averaged across all prefixes) as the *probability of hijacking*. The *probability of interception* is defined analogously. Thus, we were able to determine the probability of hijacking for each of the seven ASes. The fact that the hijacking AS is a tier-1 AS implies that the interception probability is the same.

Figure 7 shows the lower and the upper bound for probability of prefix hijacking on average and for individual ASes. The figure shows that a tier-1 AS can, on average, hijack the traffic for a prefix from another tier-1 AS with ≈70-75% probability. The fact that the ability to hijack a prefix's traffic from a peer depends only on the peer's existing route for the prefix shows up in that the hijacking probability does not vary much across tier-1 ASes. Further, this implies that the estimate also applies to hijacking of tier-1 traffic by multiple colluding tier-1 ASes.

While we have focussed on the probability of prefix hijacking, another important question is the amount of traffic that can be hijacked. Note that the fact that a small number of prefixes carry a majority of the Internet's traffic [33] implies that the probability estimates can be misleading. To address this, we focussed on the top 100 web-sites in terms of the traffic carried according to the Alexa's web-site rankings [34]. We mapped these sites to the corresponding prefixes and determined if a tier-1 AS can hijack traffic for these popular prefixes from its peers. Figure 8 plots the hijacking probability for the popular prefixes. As can be seen, a tier-1 AS can hijack traffic for these prefixes with a probability of ≈60-70%, which is close to the overall estimate.[4] This suggests that our estimates should closely approximate the fraction of traffic that a tier-1 AS can intercept from its peers.

3.2 Hijacking by any AS

We now try to estimate the probability of prefix hijacking and the probability of prefix interception for ASes in general, not just tier-1 ASes. To this effect we focus on all the 34 ASes that contribute to the Route-Views repository – these ASes are hereon referred to as the RV-set. This includes 7 tier-1 ASes, 19 tier-2 ASes and 8 other ASes (tier≥3). For each AS in the RV-set, we determined the prefixes in the Internet routing table whose traffic from the other ASes in the set can be hijacked and routed back to the prefix owner. Specifically, the estimation procedure required us to answer the following questions: Can an AS in the RV-set, say X, hijack traffic for target prefix p from another AS, say Y, in the RV-set? If yes, can it route this hijacked traffic on to p's owner?

As described in section 2.1, AS X's ability to hijack p's traffic from AS Y depends on both Y's existing route for p and the invalid route received by Y. The Route-Views data provides Y's existing route for each prefix p. As far as the propagation of the invalid route advertised by X is concerned, we determined a prefix owned by X (i.e. the origin AS in the AS-PATH for the prefix is X) and used Y's route to this prefix as an approximation of the invalid route that Y would receive. Lets assume that the AS next to the origin AS in the AS-PATH for this route is Z. Hence, AS X advertises the invalid route to its neighbor Z and this propagates onto AS Y. Section 2.2 detailed that the safety of X advertising the invalid route to its neighbor Z depends on both X's existing route for p and X's relation with Z. As before, the Route-Views data provides X's existing route for each prefix p. Finally, we used CAIDA's AS relationship data [35] to determine X's relation with Z.

Using this basic methodology, we estimated the upper (UB) and lower bound (LB) for the probability of hijacking and the probability of interception for the ASes in the RV-set. These

[4]In practice, their popularity suggests that these prefixes are well engineered and monitored and hence, we believe that it is unlikely that an AS will attempt to hijack or intercept their traffic.

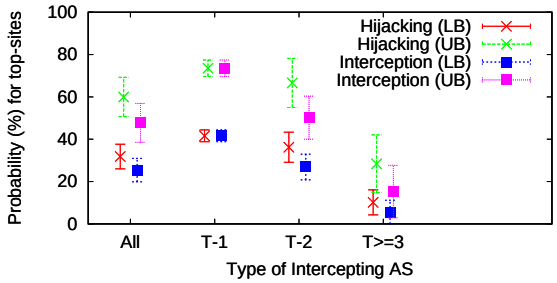


Figure 10: Probability of prefix hijacking and prefix interception for popular prefixes.

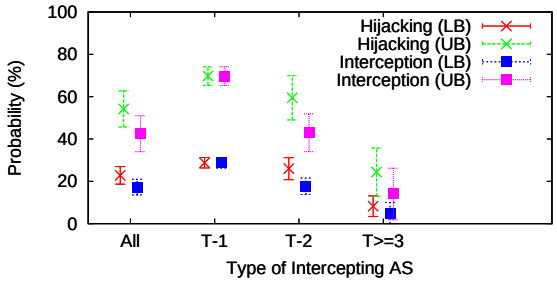


Figure 11: Probability of prefix hijacking and prefix interception with routes that have an invalid next-hop.

are plotted in figure 9 – the error bars in the figure represent the 95% confidence interval for the corresponding bound. The graph shows that the overall probability of hijacking a prefix varies between 38% and 63% while the probability of intercepting a prefix varies between 29% and 48%. Also plotted are the probabilities for ASes of different kinds. As mentioned earlier, for tier-1 ASes, the hijacking and interception probabilities are the same and these vary between 52% and 79%. Note that this encompasses the range in the previous section. However, as one would expect, both the hijacking and the interception probabilities drop off for tier-2 ASes and onwards. Also, such ASes have a higher variance and hence, a larger confidence interval for the various bounds.

As before, we also determined these probabilities for popular prefixes corresponding to the top-100 sites. These are plotted in figure 10. The figure shows that both the hijacking and interception probabilities for the popular prefixes are only slightly lower (≈5-10%) than for all prefixes. Overall, our results show that ASes higher up in the AS hierarchy (tier-1 and some tier-2 ASes) can both hijack and intercept any prefix with a high probability (>50%). However, invalid routes advertised by ASes lower down in the hierarchy wouldn't have as significant an impact.

Note that the estimates in this and the previous section have assumed that the hijacking AS advertises routes with an invalid origin. As mentioned earlier, this can lead to a MOAS anomaly and the hijacking AS can avoid this by advertising routes with an invalid next hop. This would increase the length of the invalid route and hence, reduce the amount of traffic that can be hijacked (and intercepted) but would make detection harder. We measured the hijacking and interception probabilities for ASes in the RV-set with such advertisements. These are plotted in figure 11. The figure shows that using advertisements with an invalid next-hop reduces the hijacking and interception probabilities by ≈10-20% with the probabilities for tier-1 ASes ranging between 30% to 70%.

Prefix	Owner (AS name)	Hijac--ker	Estimated Hijacking LB-UB %	Actual Hijack--ing (%)
64.233.161.0/24	Google	Cogent	35.5-64.5	45.2
12.173.227.0/24	MarthaStewart Living	ConEd.	36.4-84.9	42.4
63.165.71.0/24	Folksamerica	"	39.4-72.7	39.4
64.132.55.0/24	OverseasMedia	"	18.2-51.5	18.2
65.115.240.0/24	ViewTrade	"	27.2-54.5	21.2
65.209.93.0/24	LavaTrading	"	39.4-72.7	45.5
66.77.142.0/24	Folksamerica	"	90.9-90.9	90.9
66.194.137.0/24	MacKayShields	"	18.2-57.5	27.3
66.207.32.0/20	ADI	"	45.5-66.7	63.6
69.64.209.0/24	TheStreet.Com	"	72.7-81.8	84.8
160.79.45.0/24	RhodesASN	"	27.3-75.8	51.5
160.79.67.0/24	TheStreet.Com	"	60.6-75.8	69.7
192.251.16.0/24	T&TForex	"	27.3-57.6	27.3
198.15.10.0/24	TigerFund	"	0-1	60.6
204.13.72.0/24	FTENNY	"	93.9-93.9	75.8
216.223.46.0/24	SDSNY	"	51.5-78.8	18.2

Table 2: Comparing our estimates for known prefix hijacking events with the actual hijack probability.

3.3 Verifying against known events

We now verify our estimates against known prefix hijack events. For instance, Cogent (AS 174) hijacked a prefix (64.233.161.0/24) belonging to Google (AS 15169) on May 07, 2005 through an advertisement with an invalid origin [5]. According to BGP updates collected at the Route-Views repository, Cogent started advertising the prefix on May 07, 2005 14:37:56 and this caused 14 of the 31 (45.2%) distinct ASes part of the RV-set at that time to choose the invalid route. It is not known if the hijacked traffic was blackholed or actually routed back to Google. We ran our analysis on a routing table collected earlier that day (before the hijack) and estimated that the probability that an invalid route for the prefix advertised by Cogent would hijack traffic from ASes in the RV-set ranges between 35.5% and 64.5%. Further, the fact that Cogent is a tier-1 AS implies that the same applies to the probability of interception. As can be seen, our estimate encompasses the fraction of ASes from which traffic was actually hijacked. Further, amongst the 11 ASes whose traffic our analysis predicted would be surely be hijacked (i.e. they were included in the lower bound), only one was not hijacked in reality.

We performed the same exercise for other known hijack events. Since we did not have BGP routing tables from the hijacking AS in these cases, we were only able to predict the probability of hijacking. Table 2 shows the results. As can be seen, our estimate encompasses the actual hijacking probability for 11 of the 16 prefixes analyzed, in 3 cases we over-estimate, in 1 case we under-estimate while in 1 case our estimate provides no information. Note that the assumption that the invalid route actually reaches the ASes in the RV-set cannot be verified and this is a frequent cause for over-estimation. More importantly, the outliers show that Internet routing is certainly more complex than the simplified model used for our analysis. However, the proportion of cases where our estimates were accurate and the exercise in the next section fortify our confidence in the results presented.

4. INTERNET TRAFFIC INTERCEPTION

There have been instances of prefix hijacking in the Internet. However, we are not aware of incidents where the hijacked traffic was still being routed to the owner. While

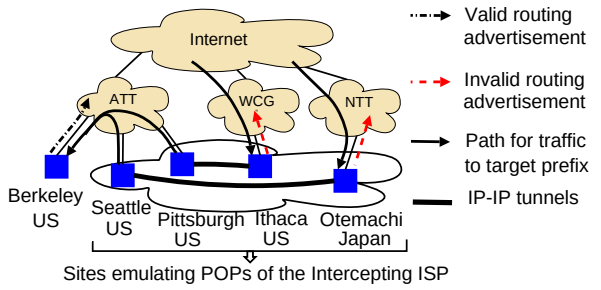


Figure 12: Intercepting traffic from the prefix owner at the Berkeley site. The four other sites emulate an ISP, use invalid routes to hijack traffic and route it back to the owner.

the fact that ISPs can use invalid routing advertisements to intercept traffic is pretty obvious, we still wanted to attempt interception in practice. Apart from serving as a proof of concept, our hope was to derive insights from this exercise into the practicality of intercepting traffic within the existing routing framework. In this section, we detail our deployment and implementation efforts for intercepting traffic in the Internet. We used this to actually intercept a prefix's traffic (of course, the prefix belonged to us).

For these experiments, we deployed hosts at five different sites and used the Quagga software router [36] on these hosts to establish EBGP peerings with different ISPs. Effectively, this allowed us to advertise our prefix (204.9.168.0/22) into the Internet through the peerings. These sites and the upstream ISP at each site are shown in figure 12. The idea behind the experiments was to use our prefix as the *target prefix* with one of the sites serving as the owner of the prefix and the four other sites serving as the geographically distributed POPs of an ISP trying to intercept the prefix. We used IP-IP tunnels between these sites for any intra-domain communication between the POPs of our emulated ISP. Figure 12 shows one such set-up with the site in Berkeley acting as the prefix owner. Invalid routes for the prefix are advertised through the sites at Ithaca and Otemachi. These invalid advertisements hijack traffic for the target prefix which is tunneled to the other two sites and is then routed to its owner.

For hijacking the target prefix's traffic from a given site, we simply advertised the prefix through all the four other sites. However, for interception, the traffic ought to be routed back to the owner. This is tricky since all our sites are effectively stub sites peering with providers and hence, all outgoing edges for the ISP emulated by our sites are customer-to-provider edges. Consequently, the existing route used by the ISP for the target prefix is bound to be a provider route. Also, the invalid route can only be advertised through a provider. As we detailed in section 2.2, this can lead to a routing instability impacting the ISP's existing route for the target prefix. Hence, we manually determined the optimal way of advertising the invalid route so that the ISP is still able to route the hijacked traffic to the designated owner.

We used recursive DNS nameservers across the Internet to generate actual traffic destined to the prefix. To this effect, we collected a list 23,858 of recursive nameservers belonging to 7,566 of the 18,391 routable ASes on the Internet (based on a BGP routing table obtained from the RouteViews repository). We also pointed the NS record for a domain name under our control (`prefix.anycast.guha.cc`) to

Ber	Pit	Sea	Ith	Ote	% of traffic Hijacked	% of traffic Intercepted
O	✗	✗	✓	✓	91.7	78.8
✗	O	✗	✓	✓	68.8	67.5
✗	✗	O	✓	✓	97.4	66.2
✗	✗	✗	O	✓	66.0	47.3
✓	✓	✓	✗	O	76.1	23.4

Table 3: Percentage of Traffic Hijacked and Intercepted. Each row corresponds to a scenario with one site acting as the prefix owner (O) and the four other sites emulating the Intercepting ISP – some of these sites advertise the invalid route (✓) while others don't (✗).

point to an address in the prefix. Thus, a query for a name such as `query.prefix.anycast.guha.cc` to a nameserver in the aforementioned list causes it to send a DNS packet to our prefix and thus, allows us to probe our prefix from the nameserver. We loosely term the fraction of probes received at a given site as the "fraction of traffic" received at the site.

The probing methodology described above was used to measure the fraction of traffic that can be hijacked and intercepted from individual sites in our deployment. Table 3 shows these results. For our deployment, the fraction of traffic hijacked varies between 66% and 97.4% while the fraction of traffic intercepted varies between 23.4% and 78.8%. This, at the very least, provides anecdotal evidence that a significant fraction of traffic to prefixes on the Internet can be intercepted.

More importantly, our proof-of-concept implementation, as described below, represents one approach that ISPs might use to intercept traffic to a prefix with existing routers and routing framework. Given a target prefix, the hijacking AS can determine the next hop AS for its existing valid route to the target prefix - let this be the *preferred AS*. The routers of the hijacking AS that peer directly with the preferred AS and thus, receive valid BGP advertisements for the target prefix are left with unmodified configurations. All other routers are configured with static routes to send traffic destined to the target prefix to one of the unmodified routers. Also, these routers are configured to advertise this internal static route through BGP to the external routers they peer with (while satisfying the advertisement constraints discussed in section 2.2). This ensures that all neighbors of the hijacking AS receive a one AS-hop route to the target prefix while the hijacking AS can forward the hijacked traffic to the destination. All this can be achieved with standard management interfaces and tools used by ISPs today. Thus, intercepting traffic to a prefix in the Internet is almost as simple as hijacking it.

5. INTERCEPTION DETECTION

We wanted to determine if traffic to *any* prefix is being intercepted in the Internet today. Note that there has been work towards detecting prefix hijacks [3,15,18–21] and since the interception of a prefix necessarily involves hijacking it, these would seem to apply. However, they either look for anomalies in routing advertisements [15,18] and hop count changes [21] which are not effective for detecting ongoing interception or use fingerprinting to detect blackholing/redirec--tion of the hijacked traffic [20] and hence, would not work for prefix interception. Alternatively, MyASN [19] uses BGP updates collected at route-repositories and information provided by a prefix owner about the origin AS of the prefix to alert the owner of any attempts to hijack their prefix through

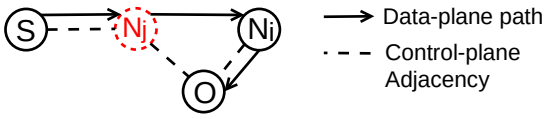


Figure 13: Next-hop Anomaly: a signature for Internet interception. Here, AS N_j uses fake advertisements to claim to be a next-hop for origin AS O and routes intercepted traffic for prefix p through AS N_i.

advertisements with an invalid origin. PHAS [3] is a similar service. These services are guided by the observation that it is the prefix owner that can authoritatively distinguish valid prefix advertisements from invalid ones [37] and hence, require proactive participation of prefix owners. Here we explore the possibility of detecting ongoing prefix interception in the Internet without pro-active participation by prefix owners.

Note that detecting interception (and hijacking) based solely on control plane information is not possible. For instance, a change in the origin AS for a prefix is a frequent occurrence in the Internet [3] and hence, a MOAS conflict cannot be used as an indicator of hijacking based on routes with an invalid origin. Guided by this observation, we attempted to use a combination of control-plane and data-plane information from a number of vantage points to detect interception scenarios in the Internet.

5.1 A Signature for Internet Interception

The key insight guiding our approach for interception detection is that the intercepting ISP relies on its existing route for the target prefix to send the prefix's traffic to its owner. Consider a prefix p with origin AS O and with next-hop ASes $N_1, \ldots, N_n$. Here, *next-hop AS* refers to an AS that appears next to O in the control-plane AS-level paths to p. Given this, in all likelihood, a packet destined to p that reaches AS N_j should be routed directly to the origin AS O. Thus, a data-plane trace wherein packets to p traverse AS N_i after traversing AS N_j (j≠i) would suggest that AS N_j is not a next-hop AS for prefix p and is advertising a route with an invalid next-hop to intercept the prefix's traffic. Figure 13 illustrates this scenario – we refer to such an occurrence as a *next-hop anomaly* and use it as a signature for interception on the Internet. The following sections detail our study of such anomalies in the Internet.

5.2 Data Sources

For control-plane information, we used the BGP routing tables collected at the Route-Views repository. This provides us with a view of the Internet's routing state from a total of 43 vantage points belonging to 34 distinct ASes. For the analysis on any given day, we used a routing table collected on that day to determine the set of next-hop ASes for each routable prefix.

For data-plane information, we use the traceroutes collected as part of the IPlane project [38]. This includes daily traceroutes to ≈100,000 routable prefixes from ≈200 PlanetLab nodes [39].[5] Thus, our data-set for each day of analysis comprised of ≈20 million IP-level traceroutes. We processed these traces to map the IP-level traceroutes to the corresponding AS-level traceroutes by mapping IP addresses to their origin ASes based on BGP routing tables.

[5]Instead of traceroutes to all routable prefixes, the data set contains traceroutes only to one prefix in each BGP atom [40]. However, this suffices for the detection exercise.

	Oct 31	Nov 25	Dec 2	Dec 4
Anomalous Prefixes	5977	6125	4760	4904
Anomalous Clusters	834	749	545	619
After accounting for IP-to-AS mapping errors	440	392	306	348
After validation based on data-plane information	32	26	27	28
After validation based on *whois* information	11	11	10	12
After e-mail survey	9	11	10	11

Table 4: Number of next-hop anomalies at various stages of our analysis.

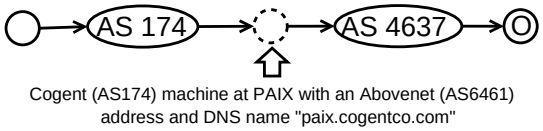


Figure 14: Erroneous AS-level paths due to presence of IXP machines. Here, Next-Hop ASes for O = {6461, 4637}, Original AS-level Path = { .., 174, 6461, 4637, O} and Rectified AS-level Path = { .., 174, 4637, O}

5.3 Detecting Next-hop Anomalies

We used the AS-level traceroutes and the next-hop information extracted from the routing tables to determine instances of next-hop anomalies on four days in Oct-Dec, 2006. The number of prefixes for which we detected next-hop anomalies on each of these days are shown in the first row of table 4. To make the analysis of these anomalies more manageable, we clustered them using triples of the form {N_j, N_i, O}. Thus, anomalies involving the same next-hop ASes (N_j and N_i) and the same origin AS (0) were clustered as one. It is reasonable to assume that anomalies that are clustered together occur due to the same root-cause. The second row of table 4 displays the number of anomalous clusters on each day.

However, a majority of these anomalies are due to errors in IP-to-AS mappings based on BGP routing tables. These are similar to the errors that Mao et. al. [41] had to account for as part of their AS-level traceroute tool. A brief explanation of these error possibilities and how we accounted for them is given below:

(a). *Internet Exchange Points (IXPs)* refer to locations that host a number of ISPs who can, in turn, peer with each other on top of the IXP infrastructure. Since IXP-hosted machines are typically assigned addresses from address space of the IXP or one of the participating ISPs, this can lead to an additional AS along the data-plane AS-level path. If this additional AS happens to be a next-hop of the prefix being traced, the trace would be falsely flagged as being anomalous. Figure 14 illustrates a scenario at Palo Alto Internet Exchange (PAIX) using AS 6461's (Abovenet) address space that causes Abovenet to be erroneously flagged as an Intercepting ISP.

We detect such errors based on the DNS names of the IXP machines. In figure 14, the DNS name for the IXP machine suggests that it belongs to a participant ISP, AS 174 (Cogent). Consequently, the rectified data-plane AS-level path does not include AS 6461 and hence, is not anomalous.

(b). *Sibling ASes*: ASes from sibling organizations may share their address space and may also have cooperative routing arrangements. Thus, next-hop anomalies wherein the two

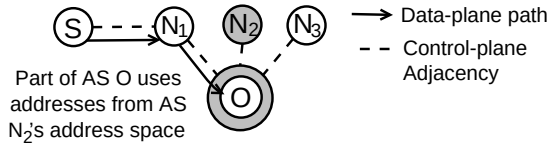


Figure 15: AS O uses part of its provider N_2's address space and this leads to erroneous AS-level paths. Here, Original AS-level Path = {S, N_1, N_2, O} and Rectified AS-level Path = {S, N_1, O}

.

next-hop ASes are sibling ASes should not be flagged as such. We achieve this by utilizing the similarity in the DNS names for IP hops in the two ASes, though in some cases we had to directly feed the sibling relationships to the analysis.

(c). *Using Provider Address space*: In many scenarios, an ISP will provide its customer with a small part of its address space that the customer ends up using for its peerings with others ISPs too. For instance, in figure 15, AS N_2 assigns its customer O with a part of the address space announced by it and is used by O for its peerings with N_1 and N_3 too. In this scenario, the AS-level path of packets routed to O from N_1 will include N_2 and will be erroneously flagged as a next-hop anomaly.

As before, we detect such errors based on the DNS names of the IP hops involved - the IP hops attributed to N_2 would have the same DNS name suffix as the IP hops belonging to O. In cases where the reverse name lookup for the IP hops in O failed, we looked for similarity between the DNS names of the IP hops attributed to N_2 and the AS name for O.

Thus, by utilizing ownership information encoded in DNS names and AS names we were able to account for almost all the IP-to-AS mapping errors in an automated fashion. The number of anomalous clusters after this step of the analysis are shown in the third row of table 4.

5.4 Anomalies due to Traffic Engineering

Apart from active interception by an ISP, a next-hop anom--aly may also result due to traffic engineering by the ASes involved. For instance, the data-path shown in figure 13 may arise if O is a stub-AS multihomed to two providers and is using one as its primary provider (AS N_i) while the other as a backup (AS N_j). As described below, such a primary-backup arrangement can be achieved using a number a techniques and some of these can result in next-hop anomalies.

First, the origin AS O may advertise the prefix p to provider N_i while advertising a less specific prefix that covers p to provider N_j. The more specific advertisement to N_i ensures its primary status. However, when determining the next-hop ASes for the destination being traced, we use only the routing table entries for the longest prefix that matches the destination address. Thus, with such specific advertisements, our analysis would consider only AS N_i as AS O's next-hop for prefix p and so the data-path shown in figure 13 would not be flagged as a next-hop anomaly.

Second, the origin AS O may use *AS-Path prepending* to advertise a longer path for prefix p to N_j than to N_i. This can lead to scenarios where a part of N_j chooses to route packets destined to p directly to O (and hence, it emanates a routing advertisement claiming to be a next-hop for O) while the rest of N_j routes the packets through N_i. Finally, a number of ISPs offer customers *community-attribute based control* over how their prefix advertisements are propagated by the ISP [42]. For instance, AS O may advertise prefix p to AS N_j and direct N_j to propagate this advertisement only to specific peers. As before, such inbound traffic control can result in different parts of N_j using different routes to p.

To account for traffic-engineering induced anomalies and any remaining mapping errors, we use the following tests to verify if AS N_j has direct data-path connectivity to origin AS O:

(a). We utilize the fact that our data-plane information for a given prefix includes probes from a large number of vantage points. If the trace from any of our vantage points indicates that AS N_j can indeed route packets for p directly to AS O, we have conclusive evidence that N_j is a next-hop AS for O and we assume that N_j cannot be an intercepting ISP for p. The fourth row of table 4 shows the number of anomalous clusters after validation of the anomalies based on data-plane information.

(b). Some ASes publish information about their peers and their route import/export policies as part of the *whois* registries. As before, a *whois* entry for AS O indicating that it peers with N_j would imply that N_j cannot be an intercepting ISP for p. The fifth row of table 4 shows the number of anomalous clusters after accounting for such *whois* information.

Thus, we were able to attribute a majority of the observed anomalies to traffic engineering by the origin. More importantly, the fact that the interception signature used here can also result from valid scenarios in the Internet implies that we have to rely on the prefix owners for conclusive evidence of interception. Consequently, for the remaining anomalies, we conducted an e-mail survey asking the prefix owner if they had a peering relation with the next-hop AS suspected of interception. We received only three responses; in all three cases the prefix owner was indeed peering with the next-hop AS in question.

5.5 Unexplained Anomalies

The analysis above yielded a total of thirteen distinct next-hop anomalies that were not explained by any of the heuristics described above. Interestingly, the *whois* entries for the origin ASes in five of the anomalies included information about the ASes they peer with and this did not include the next-hop AS suspected of interception. However, this could just be a result of the *whois* information being outdated.

Further, we manually inspected these anomalous traces and while they look like interception scenarios, we can just as well imagine them resulting from traffic engineering arrangements. These could also result from routing events that impact the link connecting the suspected next-hop AS and the origin AS. Since our control-plane information consists of a routing table snapshot on the same day as the trace, such a routing event is not captured in our next-hop calculations.

Overall, we are unable to conclusively classify any of the unexplained anomalies as actual prefix interception. Fundamentally, this is because other than observing the links traversed by the probes from our vantage points, there is no way for us to verify the data-plane adjacency of two ASes as claimed by the corresponding control-plane advertisements. However, this surely does not rule out ongoing prefix interception. For instance, our study focussed only on interception through advertisement of a route with an invalid next-hop. It is also possible for the intercepting ISP to pose as the origin AS or as an AS that is two or more hops away from the

origin. Further, our study also makes a number of rather simplistic assumptions about the behavior of the intercepting ISP and hence, could have missed interception scenarios. For instance, we assume that the intercepting ISP does not manipulate the responses to traceroute-based probes to evade detection – something as a simple as the intercepting ISP configuring its routers to stop generating ICMP responses would defeat our detection. In spite of these limitations, we think that this simple attempt at detection highlights some of the challenges posed by the interception detection problem.

6. RELATED WORK

A lot of recent work has focussed on BGP security with particular emphasis on preventing the hijacking of prefixes. Some of these efforts use cryptography to secure BGP [7–13], while others propose new protocols [14], non-cryptographic additions to BGP [17] or rely on route characteristics [4,16] such as the stability of routes [4]. Wendlandt et. al. [43] argued that securing data delivery is more important than securing routing for secure communication. Our interception estimates show that communication confidentiality can be breached even when data delivery is secured. As discussed in section 5, there have also been efforts towards detecting prefix hijacks in the Internet [3,15,18–21].

The possibility of traffic interception by using invalid advertisements has been discussed by [22,24]. In recent work, Lad et. al. [44] estimate the impact of prefix hijacks through simulations across the Internet's AS-level topology. Such an approach allows them to evaluate the impact of hijacks by a much larger set of ASes than considered in this paper. On the other hand, by restricting ourselves to the ASes that contribute to the Route-Views repository, we observe each AS's actual route for any given prefix and don't need to simulate route propagation. As a matter of fact, the authors of [45] argue that it is difficult to accurately predict Internet routes through simulation over topologies where ASes are represented as nodes.

Apart from specification of attacks on BGP [46], past research has also shown the possibility of invalid advertisements resulting from misconfigurations [26,47]. Feamster et. al. [48] studied the presence of advertisements for unallocated prefixes in Internet routing. Ramachandran et. al. [23] analyzed the use of short-lived invalid routing advertisements by spammers.

7. DISCUSSION

The estimates in section 3 are based on ASes contributing to Route-Views. Further, the analysis itself relies on a rather simplistic model of Internet routing. For instance, the assumptions regarding routing preferences and the valley-free nature of routes don't always hold. The analysis does not account for special arrangements between ASes such as sibling ASes, mutual transit, etc. Also, ASes apply ingress-filters to restrict the prefixes that their neighbors can advertise to them. However, such filtering of advertisements varies greatly with the AS's size [49], relationship with the neighbor [27] and even the AS's location (for example, ASes in Europe are known to use filters aggressively [47]). Overall, a majority of the ASes struggle to maintain up-to-date filters or any filters at all [27,47,49]. More generally, the fact that these assumptions hold in the common case indicates that our estimates should closely reflect the actual amount of hijacking possible and this claim is fortified by our verification efforts.

It seems unlikely that an AS would intentionally hijack a prefix and then blackhole or redirect the hijacked traffic since this would impact the destination's connectivity and hence, would be immediately noticed. Misconfigurations or router compromises are more likely to lead to such an occurrence; to the best of our knowledge, this was the case for all prefix hijacking incidents reported in the past. In this context, it is important to note that our hijacking estimates implicitly assume that the hijacking AS advertises an invalid route to all its neighbors. However, by the very nature of BGP, both misconfigurations at and compromises of only a few (or even a single) well-placed routers can cause the ISP to advertise an invalid route to all of its neighbors and thus, our hijacking estimates capture an extreme yet realistic scenario.

The more interesting scenario is that of prefix interception since the hijacked traffic still reaches the destination. Consequently, it is less likely that an unsuspecting prefix owner would notice the interception which may have been going on for a long period. On the other hand, the presence of easily accessible route-repositories and router-servers implies that an informed prefix-owner can detect most interception attempts. Still, it wouldn't be a stretch to imagine ASes intentionally intercepting the traffic to a not-well-monitored prefix. For instance, this would (for good or for bad) ease lawful interception [50] since law enforcement agencies wouldn't necessarily need to go to different ISPs on a case-by-case basis.

In the past, ARP poisoning, DNS spoofing and other attack vectors have been proposed for man-in-the-middle (mitm) attacks in the Internet [51–53]. The increasing use of encryption for Internet communication would seem to alleviate the privacy concerns arising from such attacks. However, the use of a number of security protocols in the Internet leaves a lot to be desired and hence, the fact that traffic can be intercepted in the Internet does magnify the scope of the problem. For instance, launching a mitm attack on SSL through self-signed certificates leads to an "invalid certificate" warning on most browsers but these are often disregarded not just by common users [52] but by well-informed technical users too [53]. This and other social issues are compounded by technical problems such as frequent warnings resulting from multiple trusted authorities and even flaws in browsers that allow certificates to be forged and hence, allow for attacks where the user is not even warned [52]. All this suggests that even small ISPs that can intercept a small fraction of traffic from other ASes can cause a lot of damage.

8. CONCLUSION

This paper presents a study of Internet prefix hijacking and interception. We estimate that ASes higher up in the routing hierarchy can both hijack and intercept traffic to any prefix from a significant fraction (>50%) of ASes in the Internet. More surprising and perhaps more egregious is that even small ASes can hijack and intercept traffic from a non-negligible fraction of ASes. Further, we implemented the proposed interception methodology and used it for actually intercepting traffic to our prefix. Our experience suggests that it is indeed very simple for ASes to intercept traffic for prefixes within the existing routing set-up. Finally, we conducted a simple study to detect ongoing prefix interception. The study neither detected interception nor did it determine that there is no interception in the Internet; however, it did shed light on some of the issues involved in detecting prefix interception.

On a broader note, while our hijacking and interception estimates are (mostly) along expected lines, the notion of being able to intercept traffic in the Internet has far reaching implications for all aspects of Internet security, both at a technical level and a social level, and we hope that this paper will force a rethink on some of these issues.

Acknowledgements

We would like to thank the anonymous reviewers for their useful feedback. This work was partially supported by AFOSR under Award No. F49620-02-1-0233 and PA8750-05-C-0268 and by a grant from Intel.

9. REFERENCES

[1] "Nanog Mailing List," http://www.nanog.org/mailinglist.html.

[2] "7007 Explanation and Apology," Apr 1997, http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html.

[3] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "PHAS: A prefix hijack alert system," in *Proc. of USENIX Security symposium*, 2006.

[4] J. Karlin, S. Forrest, and J. Rexford, "Pretty Good BGP: Improving BGP by Cautiously Adopting Routes," in *Proc. of ICNP*, 2006.

[5] T. Wan and P. C. van Oorschot, "Analysis of BGP Prefix Origins During Google's May 2005 Outage," in *Proc. of Security in Systems and Networks*, 2006.

[6] P. Boothe, J. Hiebert, and R. Bush, "Short-Lived Prefix Hijacking on the Internet," NANOG 36 meeting, 2006, http://www.nanog.org/mtg-0602/pdf/boothe.pdf.

[7] Y.-C. Hu, A. Perrig, and M. Sirbu, "SPV: secure path vector routing for securing BGP," in *Proc. of ACM SIGCOMM*, 2004.

[8] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," *IEEE Journal on Selected Areas in Communication*, vol. 18, no. 4, 2000.

[9] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz, "Listen and whisper: Security mechanisms for BGP," in *Proc. of USENIX/ACM NSDI*, 2004.

[10] T. Wan, E. Kranakis, and P. van Oorschot, "Pretty Secure BGP, psBGP," in *Proc. of NDSS*, 2005.

[11] R. White, "Architecture and Deployment Considerations for Secure Origin BGP (soBGP)," draft-white-sobgp-architecture-01, Nov 2005.

[12] W. Aiello, J. Ioannidis, and P. McDaniel, "Origin authentication in interdomain routing," in *Proc. of conference on Computer and communications security (CCS)*, 2003.

[13] B. Smith and J. Garcia-Luna-Aceves, "Securing the Border Gateway Routing Protocol," in *Proc. of Global Internet*, 1996.

[14] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin, "Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing," in *Proc. of NDSS*, 2003.

[15] C. Kruegel, D.Mutz, W. Robertson, and F. Valeur, "Topology-based Detection of Anomalous BGP Messages," *LNCS*, 2003.

[16] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Protecting BGP Routes to Top Level DNS Servers," in *Proc. of ICDCS*, 2003.

[17] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Detection of Invalid Routing Announcement in the Internet," in *Proc. of DSN*, 2002.

[18] S. T. Teoh, K. Zhang, S.-M. Tseng, K.-L. Ma, and S. F. Wu, "Combining visual and automated data mining for near-real-time anomaly detection and analysis in BGP," in *Proc. of ACM workshop on Visualization and data mining for computer security*, 2004.

[19] "RIPE MyASN service," http://www.ris.ripe.net/myasn.html.

[20] X. Hu and Z. M. Mao, "Accurate Real-time Identification of IP Prefix Hijacking," in *Proc. of IEEE Security and Privacy (Oakland)*, 2007.

[21] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A Light-Weight Distributed Scheme for Detecting IP Prefix Hijacks in Realtime," in *Proc. of ACM SIGCOMM*, August 2007.

[22] O. Nordstrom and C. Dovrolis, "Beware of BGP attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, 2004.

[23] A. Ramachandran and N. Feamster, "Understanding the Network-Level Behavior of Spammers ," in *Proc. of ACM SIGCOMM*, 2006.

[24] J. Kim, S. Y. Ko, D. M. Nicol, X. A. Dimitropoulos, and G. F. Riley, "A BGP Attack Against Traffic Engineering," in *Proc. of WSC*, 2004.

[25] "Route Views Project Page," May 2006, www.route-views.org.

[26] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "An analysis of BGP multiple origin AS (MOAS) conflicts," in *Proc. of ACM SIGCOMM IMW*, 2001.

[27] C. Labovitz, A. Ahuja, R. Wattenhofer, and V. Srinivasan, "The Impact of Internet Policy and Topology on Delayed Routing Convergence," in *Proc. of IEEE INFOCOM*, 2001.

[28] F. Wang and L. Gao, "On Inferring and Characterizing Internet Routing Policies," in *Proc. of ACM SIGCOMM conference on Internet measurement*, 2003.

[29] "BGP Best Path Selection Algorithm," July 2006, http://www.cisco.com/warp/public/459/25.shtml.

[30] L. Gao, "On Inferring Autonomous System relationships in the Internet," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, 2001.

[31] B. Huffaker, "CAIDA AS Ranking Project," July 2006, http://www.caida.org/analysis/topology/rank_as/.

[32] "Tier 1 network - Wikipedia entry," July 2006, http://en.wikipedia.org/wiki/Tier_1_network.

[33] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. of Internet Measurment Workshop*, 2002.

[34] "Alexa Top Sites," http://www.alexa.com/site/ds/top_sites?ts_mode=global.

[35] A. Ma, "CAIDA AS Relationships," July 2006, http://www.caida.org/data/active/as-relationships/.

[36] "Quagga Routing Suite," Apr 2006, http://www.quagga.net/.

[37] G. Huston, "Auto-Detecting Hijacked Prefixes?" RIPE 50 meeting, 2005, http://www.ripe.net/ripe/meetings/ripe-50/presentations/index.html.

[38] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani., "iPlane: An Information Plane for Distributed Services," in *Proc. of OSDI*, 2006.

[39] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, July 2003.

[40] A. Broido and kc claffy, "Analysis of RouteViews BGP data: policy atoms," in *Proc. of network-related data management (NRDM) workshop*, 2001.

[41] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, "Towards an accurate AS-level traceroute tool," in *Proc. of ACM SIGCOMM*, 2003.

[42] "SprintLink's BGP Policy," May 2006, http://www.sprintlink.net/policy/bgp.html.

[43] D. Wendlandt, I. Avramopoulos, D. G. Andersen, and J. Rexford, "Don't Secure Routing Protocols, Secure Data Delivery," in *Proc. of workshop on Hot Topics in Networks*, 2006.

[44] M. Lad, R. Oliveira, B. Zhang, and L. Zhang, "Understanding Resiliency of Internet Topology Against Prefix Hijack Attacks," in *Proc. of IEEE/IFIP DSN*, 2007.

[45] W. Mühlbauer, A. Feldmann, O. Maennel, M. Roughan, and S. Uhlig, "Building an AS-topology model that captures route diversity," in *Proc. of ACM Sigcomm*, 2006.

[46] S. Convery, D. Cook, and M. Franz, "An Attack Tree for the Border Gateway Protocol," draft-convery-bgpattack-01, July 2001.

[47] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. of ACM SIGCOMM*, 2002, pp. 3–16.

[48] N. Feamster, J. Jung, and H. Balakrishnan, "An empirical study of "bogon" route advertisements," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 1, 2005.

[49] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. of Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[50] F. Baker, B. Foster, and C. Sharp, "RFC 3924 - Cisco Architecture for Lawful Intercept in IP Networks," Oct 2004.

[51] "Content Verification - Man in the Middle Attack," Jan 2007, http://www.contentverification.com/man-in-the-middle/index.html.

[52] "Mattias Eriksson, An Example of a Man-in-the-middle Attack Against Server Authenticated SSL-sessions," Jan 2007, http://www.cs.umu.se/education/examina/Rapporter/MattiasEriksson.pdf.

[53] K. Fujiwara, "DNS Process-in-the-middle Attack," ICANN Presentation, 2005, http://www.icann.org/presentations/dns-attack-MdP-05apr05.pdf.

Mitigating DNS DoS Attacks

Hitesh Ballani
Cornell University
Ithaca, NY
hitesh@cs.cornell.edu

Paul Francis
Cornell University
Ithaca, NY
francis@cs.cornell.edu

ABSTRACT

This paper considers DoS attacks on DNS wherein attackers flood the nameservers of a zone to disrupt resolution of resource records belonging to the zone and consequently, any of its sub-zones. We propose a minor change in the caching behavior of DNS resolvers that can significantly alleviate the impact of such attacks. In our proposal, DNS resolvers do not completely evict cached records whose TTL has expired; rather, such records are stored in a separate "stale cache". If, during the resolution of a query, a resolver does not receive any response from the nameservers that are responsible for authoritatively answering the query, it can use the information stored in the stale cache to answer the query.

In effect, the stale cache is the part of the global DNS database that has been accessed by the resolver and represents an insurance policy that the resolver uses only when the relevant DNS servers are unavailable. We analyze a 65-day DNS trace to quantify the benefits of a stale cache under different attack scenarios. Further, while the proposed change to DNS resolvers also changes DNS semantics, we argue that it does not adversely impact any of the fundamental DNS characteristics such as the autonomy of zone operators and hence, is a very simple and practical candidate for mitigating the impact of DoS attacks on DNS.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Reliability, Availability.

General Terms: Reliability, Security.

Keywords: DNS, Denial of Service, stale cache.

1. INTRODUCTION

In the recent past, there have been many instances of flooding attacks on the Domain Name System (DNS) aimed at preventing clients from resolving resource records belonging to the zone under attack [26-29]. While these attacks have had varying success in disrupting the resolution of names belonging to the targeted zone, the threat posed by them to DNS operation is obvious. As a matter of fact, DNS's pivotal role as a precursor to almost all Internet services implies that such attacks represent a severe threat to the Internet in general.

In response to such attacks, some of the DNS root-servers and top-level domain (TLD) servers have been replicated through IP Anycast [10]. Lately, a number of research efforts have proposed new architectures for the Internet's naming system. The key insight behind these proposals is to decouple the distribution of DNS data from the hierarchy of authority for the data [8,9]. Once this decoupling is done, several mechanisms can be used to make the data distribution infrastructure highly robust and to ensure its availability in the face of attacks. For instance, efforts arguing for centralized data distribution [8] and peer-to-peer based data distribution [7,9,22,24] represent the two extremes of the design space for such a robust distribution infrastructure.

However, we are not convinced of the need for a new DNS architecture involving a new dissemination mechanism to ensure DNS operation when nameservers are unavailable. Rather, we argue that a complementary and a much more modest tack to handle DoS attacks on DNS infrastructure is to do away with the need for 100% availability in the existing architecture. In this paper, we follow this argument and show that the need for nameserver availability in the *existing DNS framework* can be reduced simply through a minor modification in the caching behavior of DNS resolvers.

Today, DNS resolvers cache the responses they receive from nameservers to improve lookup performance and reduce lookup overhead. A resolver can use the cached responses to answer queries for a duration specified by the *time-to-live* (TTL) value associated with the response. We propose to modify the operation of resolvers such that they do not expunge cached records whose TTL value has expired. Rather, such records are evicted from the cache and stored in a separate "*stale cache*". Given a query that cannot be answered based on the cached information, resolvers today traverse down a hierarchy of DNS zones by querying the authoritative nameservers for the zone at each step. However, this resolution process fails if all the nameservers for the zone at any step of this traversal are unavailable. In such a scenario, we allow resolvers to use the information stored in their stale cache to answer the query for the unavailable zone and thus, allow the resolution process to continue.

Modifying DNS resolvers as specified above results in normal DNS operation when resolvers are able to access nameservers; only when all the nameservers for a zone do not respond to the queries from a resolver does the resolver resort to using records for the zone from its stale cache (*stale records*). This modification implies that DNS resolvers store the part of the global DNS database that has been accessed by them and use it when the relevant DNS servers are unavailable. Consequently, while attackers may be able to flood nameservers and overwhelm them, resolvers would still

CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.

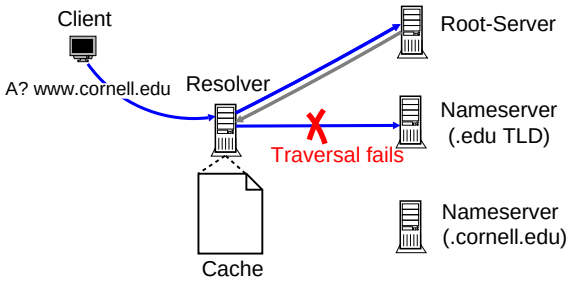


Figure 1: Traversal down the DNS hierarchy during the resolution of the A-record for *www.cornell.edu* fails if the *.edu* TLD nameservers are under attack.

have the stale records to rely upon. To this effect, this paper makes the following contributions:

- We present a simple modification in the caching behavior of DNS resolvers that would make nameserver availability less critical than it is today and hence, mitigate the impact of DoS attacks on DNS infrastructure.
- We discuss some details concerning the implementation of a stale cache in a DNS resolver. Further, our scheme has a number of practical advantages with regards to protection against flooding attacks that we discuss in section 4.1.
- We analyze a 65-day DNS trace to quantify the benefits of having a stale cache under different attack scenarios and find that the stale cache can be used to resolve a significant fraction of client queries even under severe attacks of long duration.
- Using trace-based simulation, we determine the memory footprint of the stale cache and find that maintaining even a month's worth of stale records requires a small amount of memory.
- While DNS resolvers rely on their stale cache *only* when the relevant nameservers are unavailable, the fact that the TTL-value for stale records has expired implies that it is possible that these records may not be the same as those returned by the actual nameservers (had they been available). We use the aforementioned trace to quantify this possibility and find that the probability of inaccurate records being returned in case of an attack is very small ($<$0.5%).

On the flip side, our proposal changes DNS semantics. For example, zone owners cannot expect the records served by their nameservers to be completely evicted by all resolvers within one TTL period. We analyze problems that may arise due to such semantic changes; the impact of this and other drawbacks of our scheme are discussed in section 4.2. This analysis leads us to conclude that the scheme does not adversely impact any of the fundamental DNS characteristics such as the autonomy of zone owners. Hence, we believe that the proposed resolver modification represents a very simple and practical candidate for alleviating the impact of DoS attacks on DNS.

2. A SIMPLE IDEA

2.1 DNS Resolvers Today

Clients rely on DNS primarily to map service names to the IP addresses of the corresponding servers. Typically, clients issue their queries to a local DNS resolver which maps each query to a matching resource record set (hereon simply referred to as a matching record) and returns it in the response.[1] Each record is associated with a time-to-live (TTL) value and resolvers are allowed to cache a record till its TTL expires; beyond this, the record is evicted from the cache. Given a query to resolve, a resolver executes the following actions[2]:

1. Look up the cache for a matching record. If a matching record is found, it is returned as the response.
2. If a matching record is not found in the cache, the resolver uses the DNS resolution process to obtain a matching record. This involves:
   (a) Determine the closest zone that encloses the query and has its information cached (if no such zone is cached, the enclosing zone is the root zone and the resolver resorts to contacting the DNS root-servers). For example, given an *A*-record query for the name *www.cornell.edu*, the resolver determines if records regarding the authoritative nameservers for the zones *.cornell.edu*, or *.edu* (in that order) are present in its cache.
   (b) Starting from the closest enclosing zone, traverse down the DNS zone hierarchy by querying subsequent sub-zones until the zone responsible for authoritatively answering the original query is reached or an error response from a zone's nameservers implies that the traversal cannot proceed. In either case, the resolver returns the appropriate response to the client. Also, all responses (including negative responses indicating error) during this resolution process are cached by the resolver.
3. In case the resolution process in (2.b) *fails* due to the inability of the resolver to contact all the nameservers of the relevant zone at any step of the traversal, return a response indicating the failure. Note that the term "failure" refers only to the scenario when the traversal is not completed due to the unavailability of the nameservers of a zone. Figure 1 illustrates this scenario.

2.2 DNS Flooding Attacks

We consider DoS attacks on DNS servers where attackers flood the nameservers of a zone to disrupt the resolution of records belonging to the zone and consequently, any of its sub-zones. In general, *flooding attacks* aimed at denying service to clients take advantage of the skewed distribution of functionality between clients and servers. In the case of DNS, the fact that the nameservers for a zone are completely responsible for serving the zone's records and in turn, for the operation of any sub-zones implies that their availability is critical and makes them an attractive target for flooding attacks.

2.3 Proposed Resolver Modification

We argue that changing the caching behavior of DNS resolvers so that they shoulder more of the resolution burden, especially when nameservers are unavailable, is an effective way to address DNS flooding attacks. Further, such a modification is possible within

[1]Note that the matching record may not answer the query; for example, it may reflect an error condition due to which the query cannot be answered. Hence, the term "response" includes both positive and negative responses.

[2]This is a simplification of the algorithm used by resolvers but suffices for the purpose of exposition. See [14] for a more detailed version.

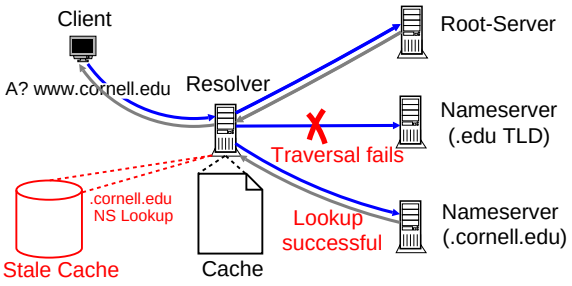


Figure 2: Resolution of the A-record for *www.cornell.edu* succeeds: a stale NS record for *.cornell.edu* allows the traversal to continue even though the *.edu* TLD nameservers are inaccessible.

the existing DNS framework. To this effect, DNS resolvers should store the responses of the queries they resolve beyond the TTL values associated with the respective responses and use stale information if all the authoritative nameservers for a zone are unavailable. Thus, the resolvers have the stale information to rely on, in case the authoritative servers for a zone are overwhelmed due to a flood of requests. More concretely, we propose the following change in the operation of DNS resolvers–

Stale Cache: Resolvers do not completely expunge cached records whose TTL value has expired. Rather, such records are evicted from the cache and stored in a separate *stale cache*. In effect, the stale cache together with the resolver cache represents the part of the global DNS database that has been accessed by the resolver.

Resolving Queries: In our proposal, the first two steps executed by a resolver when resolving a query are the same as before. Hence, given a query, the resolver attempts to respond to it based on the cached information or through the resolution process. The third step is modified as follows:

3) In case the resolution process in (2.b) fails due to the inability of the resolver to contact all the nameservers of the relevant zone at any step of the traversal, search the stale cache for the required record. If such a record is found, the resolution process in (2.b) can continue based on this stale record. Figure 2 illustrates this scenario.

This modification implies that when (and only when) the authoritative nameservers for a zone are unavailable, the resolver can resort to using responses from a previously resolved query.

Stale Cache clean-up: Existing resolvers cache the responses to the queries made during the resolution process in step (2.b). In our proposal, these responses are also used to evict the corresponding stale records from the stale cache. For example, during the resolution of the *A* record for the name *www.cornell.edu*, the resolver may query the authoritative nameservers of the zone *.edu* for the authoritative nameservers of the sub-zone *.cornell.edu*. When a response containing records regarding these nameservers is received, it is cached and is also used to evict any nameserver records for *.cornell.edu* present in the stale cache. Note that this newly cached response will be evicted to the stale cache upon expiration of its TTL value. Also note that all responses (including negative responses) are used to evict the stale cache. For example, a NXDOMAIN response from the nameserver for *.edu* indicating that the sub-zone *.cornell.edu* no longer exists will also lead to eviction of the existing nameserver record for *.cornell.edu* in the stale cache. Hence, this clean-up process ensures that a record stored in the stale cache always corresponds to the latest authoritative information that the resolver received.

2.4 Stale Cache Details

From an implementation point of view, a resolver can perform steps (2.b) and (3) of the query lookup concurrently. For instance, continuing the earlier example, while the resolver queries the zone *.edu*'s nameserver for the nameservers of the sub-zone *.cornell.edu*, it can lookup its stale cache for information regarding the nameservers for *.cornell.edu*. As mentioned earlier, the information from the stale cache is used only if the resolver is unable to contact all the nameservers for *.edu* and hence, the latency of the stale cache lookup is not critical. Consequently, the stale cache can even be maintained on the resolver's disk. However, as we show in section 3.3, even a month's worth of stale records require a small amount of storage space and hence, we envision resolvers maintaining their stale cache in memory.

3. EVALUATION

In order to evaluate the advantages of a stale cache, we collected DNS traffic at the link that connects the Cornell Computer Science department's network to the Internet. The network comprises of ≈1300 hosts. The trace was collected for a period of 65 days – from 21^{st} Nov, 2007 to 24^{th} Jan, 2008. It consists of 84,580,513 DNS queries and 53,848,115 DNS responses for a total of 4,478,731 unique names. Each collected packet was anonymized to preserve the privacy of the network's clients. This included anonymizing the source and destination IP addresses and the names and addresses in the DNS part of packet. The fact that the trace was collected at the network's border router and not at the resolvers (i.e., the caching nameservers) that reside inside the network implies that we do not see all the queries generated by clients. Specifically, client queries that can be answered based on the cached contents of the resolvers do not appear in our trace. This quirk of the collection process has important implications for the results presented here and we discuss these later in the section.

Given the trace, we can simulate the operation of a stale cache serving clients in the network under different attack scenarios. Such a simulation is governed by two key parameters:

- *Stale cache size*: A stale cache size of *x* days implies that stale records are kept in the stale cache for a maximum of *x* days. In our simulations, we vary the stale cache size from 1 to 30 days. Further, in section 3.3 we measure the actual memory footprint for a stale cache of *x* days.

- *Attack duration*: This allows us to evaluate the operation of the stale cache under attacks of varying durations. For any given type of attack, we simulate the attack lasting for a duration of 3, 6, 12 and 24 hours.

Hence, to simulate the operation of a 7-day stale cache under an attack lasting 3 hours, we populate the stale cache using the DNS queries and responses in the first 7 days of the trace. We then simulate an attack every 3 hours while ensuring that the stale cache contains trace data for the past 7 days. This allowed us to have 464 simulation runs ((65-7) days * 8 simulations per day) for a 3-hour attack while using a 7-day stale cache. Thus, we were able to simulate a number of attacks for any given stale cache size and attack duration.

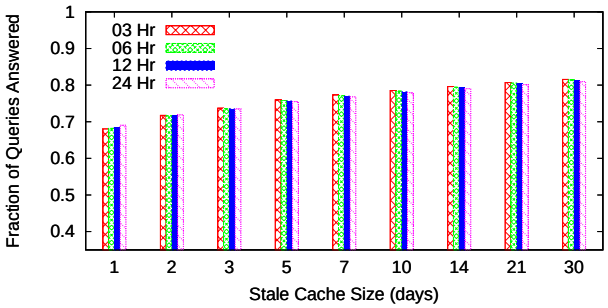


Figure 3: Fraction of Queries Answered using a stale cache of varying size during an attack wherein none of the nameservers are operational.

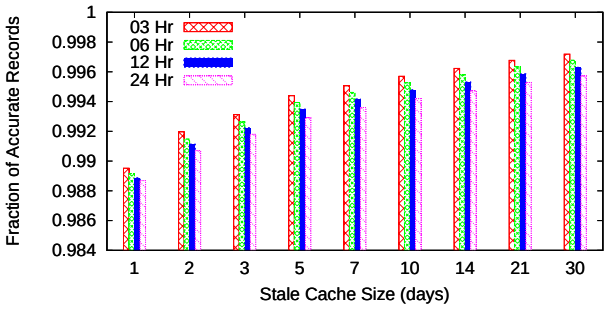


Figure 4: Fraction of Accurate Records in responses based on a stale cache of varying size during an attack wherein none of the nameservers are operational.

3.1 Is history useful?

We wanted to determine if there is any value to maintaining historical information in the form of DNS records beyond their TTL-values. To this effect, we consider an attack wherein none of the DNS nameservers are operational and hence, all queries that cannot be answered based on the information cached at the resolvers rely on the simulated stale cache. Note that this is does not represent a realistic flooding attack; instead, the objective here is to use an extreme scenario to test the limits of the value of keeping around stale DNS information.

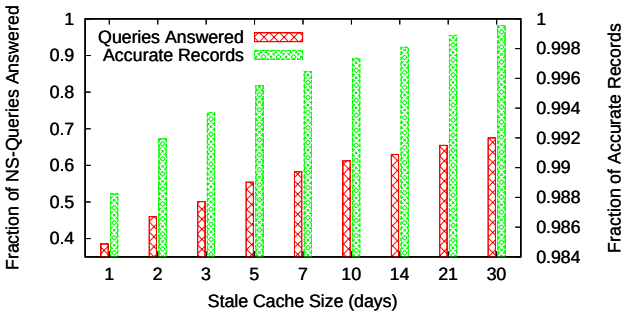
We simulated the attack scenario described above for varying attack durations and varying stale cache sizes. Here we focus on those queries that cannot be answered based on the resolver cache. Figure 3 plots the fraction of such queries that can be answered based on the stale cache. The figure shows that a 1-day stale cache can be used to answer 68% of such queries over the course of a 3-hour attack. The fraction of queries answered increases with the stale cache size; for instance, a 3-day stale cache can answer 73.7% and a 14-day stale cache can answer 79.6% of the queries. However, increasing the stale cache size beyond 14 days yields diminishing returns; for instance, a 21-day stale cache can answer 80.7% and a 30-day stale cache can answer 81.5% of the queries.[3] Past studies have found that the popularity of DNS names follows a zipf distribution [11] and the diminishing returns from increasing the stale cache size appear to be a consequence of this.

The variation of the fraction of queries answered with the attack duration for a given stale cache size is a little more complicated. For a small stale cache (<=2 days), the fraction of queries answered increases with attack duration. While non-intuitive, this can be explained based on the facts that, 1). for attacks of short-duration, many queries can be answered based on the resolver's cache and 2). the focus here is on queries that can be answered using the stale cache. Consequently, for an attack lasting 3 hours, many queries can already be answered based on the resolver's cache and the probability that a query whose answer is not cached can be answered based on the stale cache is small. For attacks of longer duration, most of the cached records have expired and hence, the stale cache is able to answer more queries.

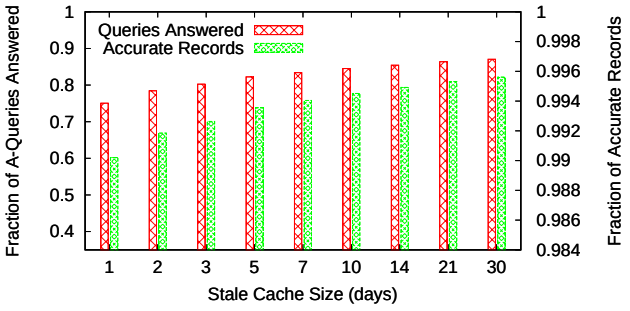
However, this effect diminishes for larger stale caches. The figure shows that for larger stale cache sizes, there is a small reduction in the fraction of queries answered as the attack duration increases. For instance, a 14-day stale cache can be used to answer 79.4% of the queries during an attack lasting 6 hours and 79% of the queries during an attack lasting 24 hours.

As mentioned earlier, an important thing to note is that the trace does not include queries that are answered based on the cached contents of the network's resolvers. This implies that the numbers regarding the fraction of queries answered (and similar numbers in the rest of this section) vastly underestimate the actual fraction of client queries that succeed in case of an attack with a stale cache in place.

However, the fact that the TTL-value for the stale records has expired implies that responses to client queries based on the stale cache may not be the same as the responses that would be received in case the actual nameservers were operational. This leads to the notion of *accurate* and *inaccurate* records. Note that using an inaccurate record in the resolution process does not necessarily imply that the name being queried is resolved to a wrong address. In-



(a) NS-queries



(b) A-queries

Figure 5: For (a) NS-queries and (b) A-queries, Fraction of Queries Answered and Accurate Records when using a stale cache during an 3-hour attack.

[3]For clarity, the X-axis in figure 3 and the figures in the rest of this section is limited to some chosen stale cache sizes.

stead, in spite of the use of an inaccurate record, a name may be resolved properly or may not be resolved at all – we discuss these possibilities and their implications in section 4.2.

Our trace-based simulation allows us to determine the accuracy of the DNS records in responses that utilize the stale cache. Specifically, for each query received during a simulated attack, we compare the response based on the stale cache and the actual response from the nameserver had it been accessible (we get this information from the trace) and all matching records are counted as accurate records. Figure 4 plots the fraction of accurate records for varying stale cache size and attack duration. The accuracy percentage increases with increasing stale cache size. This results from the fact that as the stale cache increases in size, it can answer more and more queries for NS records that tend to be more stable and hence, the increase in accuracy.[4] However, the increase tapers off beyond a stale cache size of 10-14 days; a 10-day stale cache yielded 99.6% accurate records during a 3-hour attack. Also, the fraction of accurate records reduces by a small amount as the attack duration increases.

Next, we focussed on different kinds of queries. Specifically, we studied queries for NS-records (i.e. NS-queries) and queries for A-records (i.e. A-queries) and determined the fraction of such queries that can be answered using the stale cache and the accuracy of the corresponding stale records. In case of an attack lasting 3-hours, the values for these fractions are plotted in figure 5. The figures show that while the fraction of queries answered increases with increasing stale cache size in both cases, the fraction of NS-queries answered is much less than the fraction of A-queries answered. For instance, a 14-day stale cache can answer 63% of NS-queries and 85.4% of A-queries.

This results from the fact that NS-records tend to have higher TTL values as compared to A-records (especially A-records for names not belonging to nameservers). Consequently, most of the NS queries can be answered using the resolver cache. Further, if a NS-query cannot be answered through the resolver cache, it is more likely that the corresponding NS-records weren't queried for in the past and hence, would not be present in the stale cache too. This also implies that the fraction of NS-queries answered hits the point of diminishing returns much later than the fraction of A-queries answered. The figure also shows that, as expected, the accuracy of NS-records is higher than that of A-records. In both cases, the accuracy of the stale records returned to clients increases with the stale cache size and is >99.5% with a stale cache of more than 10 days.

Overall, these results show that even in the extreme attack scenario considered here, the stale cache can answer a significant fraction of the client queries in a surprisingly accurate fashion.

3.2 Performance under different attack scenarios

We now evaluate the performance of the stale cache under three different attacks scenarios. The first attack involves the root-servers not being accessible to the clients. Today, such an attack would cause any queries for NS records corresponding to the top-level domains (TLDs) to fail.[5] However, in case of the trace-based simulation of a stale cache, all such queries succeeded. This is because, for the query and response pattern captured in our trace, the NS records for all the TLDs were present either in the cache or the stale cache at all times. Thus, the stale cache would have ensured that all

[4]We explain the increase in the fraction of NS-queries answered later in the section.

[5]This assumes that the NS records are not present in the cache of the network resolvers or have expired since the attack started.

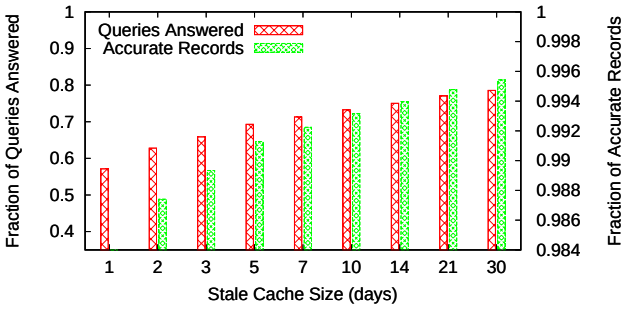


Figure 6: Fraction of Queries (for two-level names) Answered and Accurate Records when using a stale cache during an 3-hour attack on the TLD nameservers.

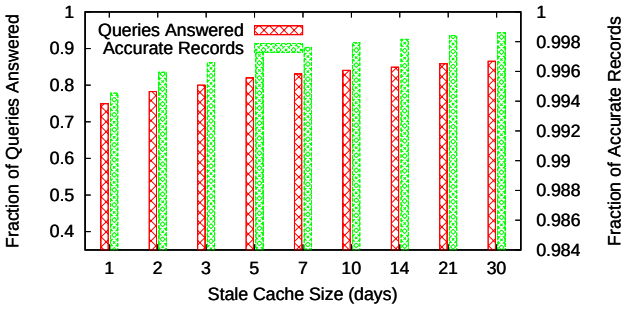


Figure 7: Fraction of Queries (for three-level names) Answered and Accurate Records when using a stale cache during an 3-hour attack on second-level nameservers.

names would still resolve and hence, would effectively shield the network from an attack on the root-servers.

The second attack involves clients not being able to access TLD nameservers. Today, this would cause queries for any records corresponding to two-level names such as *a.com* to fail. Further, any queries for longer names that rely on the resolution of two-level names would fail too. Here we restrict ourselves to the queries for two-level names that cannot be answered based on the resolver cache. Figure 6 plots both the fraction of such queries that the stale cache can answer and the fraction of records in these responses that are accurate in case of a 3-hour attack. The trends for longer duration attacks are similar. The fraction of queries answered increases with an increasing stale cache size though it tapers off for a stale cache of more than 14 days. A 14-day stale cache can answer 75% of the queries for two-level names. The reason for the lower fraction of queries answered is that clients typically access the NS records for names such as *a.com* and these records tend to have a high TTL-value. As explained earlier, this implies that most such queries are answered based on the resolver cache and if a record is not cached, there is a higher probability that it has not been queried at all and hence, is not present in the stale cache too. Of course, the fraction of total client queries that succeed when using the stale cache is much higher. The graph also shows that records from a 14-day stale cache are 99.4% accurate and accuracy increases with the stale cache size too.

Similarly, the last attack scenario involves second-level nameservers being inaccessible. This would cause queries for any records corresponding to a three-level name such as *b.a.com* to fail. We focus on such queries that are not cached by the network's re-

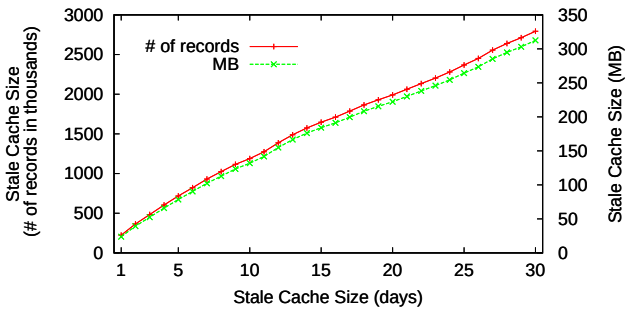


Figure 8: Stale cache memory footprint

solvers. Figure 7 plots the fraction of such queries answered using stale records during a 3-hour attack and the accuracy of these stale records. As before, the fraction of queries answered increases with an increasing stale cache size. However, in this case, the returns from increasing the stale cache size are diminished much sooner that the previous attack scenario. A 14-day stale cache can answer 85% of the queries for three-level names. Both A and NS records for names such as *b.a.com* are accessed by clients and these tend to have lower TTL-values than the records for two-level names. This explains the higher percentage of queries answered. The graph also shows that records from a 4-day stale cache are 99.8% accurate.

3.3 Memory Footprint

We now evaluate the memory requirements of the stale cache. Figure 8 plots both the number of DNS records and the actual memory used by a stale cache of size 1 to 30 days. As one would expect, the memory requirements of the stale cache increase as the number of days increase. Note that the simulated stale cache stores DNS records without any encoding and hence, there is scope for further reducing the memory required for the stale cache. More importantly, the figure shows that even for a network with 1300 hosts and a query-response rate of $\approx$25 DNS packets per second, the stale cache memory footprint is very small. For instance, maintaining stale records for a period of 30 days given the query pattern in our trace requires $<$313MB of storage space.

Of course, the stale cache memory requirements depend on the number of clients being served by the resolver and their query patterns. Also, the evaluation in the previous section shows that the gains to be obtained from stale records older than two weeks are minimal. These factors suggest that, in practice, resolvers will keep stale records only for a configurable number of days, for example stale records for the past couple of weeks. Further, the resolver will be limited to at most a certain amount of memory for the stale cache. In case the stale cache fills up, the resolver would evict records based on some criterion. For instance, the resolver could use the query pattern of clients to evict the least recently used DNS records (LRU eviction). However, given the amount of memory on modern machines, we believe that resolvers should easily be able to maintain a stale cache containing records for a couple of weeks and there shouldn't be a need for more complex eviction algorithms. In section 4.2 we discuss how placing a limit on the duration for which stale records are kept addresses some of the practical concerns arising out of the use of stale information.

4. DISCUSSION

There have been a number of "clean-slate" proposals to make the availability of specific nameservers less critical for the operation of Internet's naming system. These proposals [7-9,12,22,24] decouple the ownership of names from the task of distributing them and try to architect a robust mechanism for distributing the names. However, such an approach could increase the total DNS overhead many times over, especially in the face of the use of DNS for load balancing purposes. On a more general note, while most of us agree that DNS is afflicted by a few problems, we think that a majority of them can be attributed to misconfigurations, improper implementations, violations of best current practices, or even a lack of motivation to address them and not to major architectural flaws. For example, problems regarding high lookup latency can mostly be attributed to misconfigurations (i.e. broken and inconsistent delegations) [22] and the long timeouts used by resolvers in case of errors [19]. Consequently, despite a number of proposals arguing to the contrary, we do not see a pressing need for an architectural change. Guided by this observation, our proposal represents an exercise in showing how minor operational modifications can address DNS problems; specifically, modifying the caching behavior of DNS resolvers can reduce the impact of flooding attacks on DNS.

In the rest of this section we discuss the advantages of the proposed modification and a few possible objections to it.

4.1 Pros

DNS Robustness. The proposed modification ensures that resolvers can respond to queries for a zone even if the zone's authoritative nameservers are unavailable, assuming that the resolver has queried the zone at some point in past and the previous response is present in the resolver's stale cache. The evaluation in the previous section showed that a stale cache can indeed make DNS more robust to DoS flooding attacks. Further, while past attempts such as the anycasting of DNS nameservers provide nameserver operators with a mechanism, albeit a very expensive one, to protect the name resolution for their zones, our modification represents an insurance policy that can be adopted by the resolver operators and hence, provides some control to the client.

Simplicity. The biggest argument in favor of the stale cache as a means of increasing DNS robustness is its simplicity. The proposed scheme:

- *Does not change the basic protocol operation and infrastructure*; only the caching behavior of resolvers is modified.
- *Does not impose any load on DNS*, since it does not involve any extra queries being generated.
- *Does not impact the latency of query resolution*, since the stale cache is utilized only when the query resolution fails.

Incremental Deployment. Any single resolver can adopt the modifications proposed in this paper and achieve significant protection from attacks against the DNS servers it and its clients access. Hence, the proposal can be incrementally deployed.

Motivation for Deployment. Modifying a resolver is beneficial for the clients being served by the it since the resolver can resolve queries for zones that have been accessed by it in the past even if the nameservers for the zones are not available. Hence, there is motivation for the resolver operators to switch to the modified resolver.

4.2 Objections

DNS caching semantics and the possibility of inaccurate information being used. The biggest objection against the proposed modification is that it changes the semantics of DNS caching. With the current DNS specifications, a zone operator can expect the reco-rds served by the zone's authoritative nameservers to be completely

expunged by resolvers within TTL seconds. With our proposal, such records would be evicted to the stale cache. The problem with such an approach is best explained through an example. Let's consider a zone whose records have been updated. Also, consider a resolver that has accessed the zone but not since the update and so the zone's records in the resolver's stale cache are obsolete or inaccurate. Given this, if the resolver needs to resolve a query for the zone at a time when all the zone's authoritative nameservers are unreachable, it would resort to using the inaccurate records present in its stale cache.

The problematic scenario described above arises only when two conditions are met:

1. The DNS records for the zone in question have been updated since the last access by the resolver.

2. The nameservers for the zone are currently inaccessible.

Condition (1) can arise due to several reasons: for instance, the nameservers for a zone have been moved, the service itself has migrated or there have been address space changes, DNS based load-balancing across the nameservers or the application servers, etc. We consider these below.

First, if the nameservers have been moved, the name resolution may fail while if the service migrates, the name may be resolved to the wrong address. Both these are undesirable scenarios. However, restricting the duration for which resolvers can keep records in their stale cache helps us avoid these. Specifically, to account for this, a nameserver/service needs to be run on both its old and new address for a couple of weeks after migration. This allows for the old records to be flushed from the stale cache of resolvers. Note that zone operators anyway need to do this today since a large number of misbehaving resolvers disregard TTL values and use expired records even when the nameservers for a zone are available [32,34].

Second, if the DNS records have been changed to balance client load, the name would probably resolve properly but this might interfere with the load across the servers. In a recent study, Poole et. al. [21] found that name-to-IP mappings tend to be very stable with less than 2% of DNS names changing IP addresses more than once a week. Further, most of these names can be attributed CDNs like Akamai trying to balance client load across their servers.[6] This implies that not only is condition (1) rare, in a vast majority of cases where it does occur, using the stale records would not lead to wrong resolution. While this is far from perfect, the small possibility of load imbalance across the servers when they are under attack (in which case the load balancing isn't working anyway) seems like a small price to pay for the robustness offered by a stale cache. Also, the possibility of a resolver using inaccurate records for a zone is much less for zones that the resolver frequently accesses.

Further, resolvers may choose to apply the modified caching sch--eme to infrastructure records only. Infrastructure records, as defined by [17], refer to records used to navigate across delegations between zones and include the *NS* records (and the corresponding *A* records) for zones. Past studies show that such records change even more infrequently [9,17] than other DNS records and hence, this would further reduce the possibility of resolvers using inaccurate records while still providing a large robustness gain.

Finally, it is also possible to make changes on the client-side DNS software to make applications aware of the use of stale records. A resolver could use the RCODE field in the DNS header (a 4-bit field; values 5-16 for this field have been reserved for future use [14]) to inform the querying client that the response is based on the stale cache. Similarly, the client `gethostbyname` and the relevant `libresolv` functions could be modified to interpret the new RCODE value and inform applications of the same. With these changes in place, applications would have the flexibility of being able to account for the possibility of inaccurate records and decide whether to use stale records or not based on application semantics and/or user choice. However, most applications that need to make sure that they are accessing the right resource use application-specific authentication anyway; for instance, financial web-sites commonly use personalized site-keys for this purpose [33]. This, combined with the fact that the possibility of stale records being inaccurate (especially ones that lead to wrong resolution of names) is miniscule, implies that we don't feel that the overhead of modifying the DNS-software at all clients is justified.

Autonomy for zone operators. Another important concern is that the proposed modification would seem to move autonomy away from zone operators to resolver operators. Allowing resolvers to store records after their TTL value has expired suggests that zone operators do not control the access to their sub-zones; for instance, they could not kill off their sub-zones when they wish to.

However, this is not the case. The fact that we don't modify DNS's hierarchical resolution process implies that resolvers still need to go through the nameservers for a zone in order to access its sub-zones and hence, the autonomy of zone operators is not affected. For instance, let's assume that the operator for the zone *.com* needs to kill off the sub-zone *.rogue.com*. Typically, this would involve *.com*'s zone operator configuring the zone's authoritative nameservers to respond to any queries regarding *.rogue.edu* with a NXDOMAIN, implying that no such domain exists. Consequently, a resolver trying to resolve a query like the *A* record for *www.rogue.- -com* by traversing down the DNS zone hierarchy would receive a NXDOMAIN response from one of the *.com* nameservers and would forward this to the client that originated the query. Further, this response would be cached and eventually be evicted to the stale cache. Thus, if there are any such future queries at a time when all the *.com* nameservers are unavailable, the resolver would still return a NXDOMAIN response.

Attackers attempting to force the use of inaccurate information. Apart from the possibility of inaccurate data being used, there is also the possibility of attackers taking advantage of the stale cache maintained by resolvers to force the use of inaccurate records. Attackers may keep track of updates to the records of a zone and start flooding the authoritative nameservers for the zone as soon as some of the records are updated. If the attack overwhelms the zone's nameservers, resolvers trying to resolve the zone's records would rely on the obsolete data stored in their stale cache. In effect, attackers can now flood the nameservers for a zone in order to delay the propagation of updates to the zone's records for the duration of the attack. While we cannot imagine many cases where such an attack could be used, one scenario where it does appear to be harmful is to undermine the autonomy of zone-operators. In the example above, the owners of the *.rouge.com* zone may flood the *.com* nameservers to force the use of stale NS records for their zone and hence, prevent their zone from being killed. The bigger problem here is that there is incentive to flood the nameservers of a zone to prevent sub-zones from getting killed. This problem captures an inherent trade-off that the use of stale records exposes: when a zone's nameservers are being flooded, all sub-zones, including sub-zones that were deleted in the recent past, are accessible. While this is certainly a serious concern, it is important to note that the sub-zones will stay alive only as long as the zone's nameservers are inaccessible. Given that measures to counter flooding attacks on

[6]The fact that the actual DNS names in our trace have been anonymized implies that we cannot determine if the changed mappings observed by us can also be attributed to CDNs.

nameservers, such as filtering by ISPs, are usually applied within a day or two of the attack, the sub-zones would be able to stay alive for not too long a duration.

Privacy Concerns. With our proposal, DNS resolvers store DNS records long beyond their TTL-values. This leads to privacy concerns in case the resolver is broken into. Specifically, if a resolver were to be compromised, the attacker would gain access to all the stale cache records and hence, would have a heap of information about what the resolver's clients have been querying and in turn, their web-access patterns. However, the stale cache would not provide the attacker with information about queries from individual clients. Also, this is certainly no worse than other DoS mitigation proposals that require DNS resolvers to query entities other than a zone's authoritative nameservers to resolve the zone's records and hence, leak out private information as an integral part of their operation.

Resolution latency in the face of an attack. In our proposal, if a resolver is unable to reach the authoritative nameservers of a zone, it resorts to using the zone's records in the stale cache. Consequently, the resolver must query each of the nameservers for the zone, wait for the query to timeout (and possibly retry) before it can use the stale cache. With the current timeout values used by resolvers, this would entail a high lookup latency in the face of attacks (i.e. when the nameservers for a zone are unavailable). For example, the default configuration for the BIND8 resolver [31] involves sending queries to each nameserver for 30 seconds with an exponentially increasing period between consecutive retries. So, clients accessing a zone with two authoritative nameservers at a time when both of them are unavailable would need to wait for 60 seconds before receiving a reply. However, most resolvers allow the retry and timeout values to be configured and hence, the lookup latency problem can be solved by using aggressive values for these timers. As a matter of fact, past work has already suggested that these timer values are major contributors to the high lookup latency when errors are encountered [19].

DoS'ing the application servers. The proposed modification does not reduce the vulnerability of nameservers to DoS attacks. Consequently, attackers can still flood them so that they are unable to serve (and update) the records of the corresponding zones. Rather, the modification makes the availability of DNS nameservers less critical and hence, significantly reduces the impact of DoS attacks on DNS.

Further, the proposal does not address the general DoS problem and attackers can deny service to clients by attacking the application servers instead of the corresponding DNS nameservers. As a matter of fact, a flooding attack that chokes the network bottleneck for a zone's nameservers is also likely to hamper the availability of the zone's application servers. In such a scenario, there isn't much value to being able to resolve the names for the application servers since clients would not be able to reach them anyway.[7] In effect, this concern boils down to how common is it for application servers and their nameservers to share a network bottleneck. We intend to measure this for nameservers on the Internet as part of our future work.

Interaction with DNSSec. The proposal does not have any harmful interactions with or implications for DNSSec. In case the resolver cannot reach the nameservers of a zone and relies on the corresponding records in the stale cache, the records ought to be classified as "Undetermined" by the resolver.[8] Hence, any DNSSec policies expressed by the resolver operator for undetermined records naturally apply to the stale records.

[7]Note that there is still a lot of value to being able to access the sub-zones when a zone's nameservers are being flooded. For example, being able to access the rest of the name system when the root-servers are being flooded.

5. RELATED WORK

A number of recent efforts [7-9,22,24] have proposed new architectures for the next generation Internet naming system that address DNS's performance and robustness problems. Other proposals to change the DNS architecture include multicasting the global DNS database to specialized servers to reduce the response time for clients [12] and augmenting the DNS structure with additional pointers that can be used to access sub-zones and hence, increase DNS robustness against flooding attacks [25]. [20] argues for taking advantage of site multihoming by spreading the identity of end hosts and rate-limiting name resolution requests to mitigate DoS attacks. Balakrishnan et al. [1] propose to replace the hierarchical DNS (and URL) namespace with flat identifiers. We show that a minor operational change to resolvers in the *existing DNS framework* can significantly mitigate the impact of DoS attacks on DNS.

The use of caches and more generally, of stale data to improve system availability shows up in many aspects of computer science. Examples include using stale data to improve availability of services [13] and even shared memory multiprocessors [23]. [15] proposes and evaluates the use of stale data to reduce the measurement overhead for placement of services on the Internet. This paper evaluates the efficacy of stale data in increasing DNS availability.

Pappas et al. [17] argue for the use of long TTL values for infrastructure DNS records as a means of alleviating the impact of DoS attacks on DNS. We share with their proposal the basic notion of using records already present in the resolver cache for a longer period. While our proposal involves changing the caching behavior of resolvers, using longer TTL values for a zone's records involves a minor configuration change at the zone's nameservers and hence, does not necessitate any software update. However, using long TTL values represents a technique that can be used by nameserver operators. Also, long TTL-values make it harder for operators to update their records. In subsequent work [18], the authors augment their proposal and argue for resolvers proactively renewing the infrastructure records present in their cache as a means of mitigating attack impact. This scheme has an important advantage over the use of stale records: it does not modify DNS caching semantics. However, as shown in [18], proactive renewal of DNS records by resolvers, when used in isolation, increases DNS traffic many times over. Further, the overhead of such an approach implies that it cannot be used for non-infrastructure DNS records, a large fraction of which don't change very rapidly.

In past work [2], we discuss the use of stale DNS records as a DoS mitigation mechanism. This paper follows up on that proposal and quantifies the advantage of a stale cache and the possibility of using obsolete information through trace-based simulations. Nonamed [35] is a quasi DNS resolver that provides users the option of using stale DNS information which maybe be useful for operation when disconnected from the Internet. We argue for the use of a zone's stale records only when all nameservers for the zone are unavailable. Cohen and Kaplan [6] propose the use of stale DNS records for improving DNS performance. This involves fetching data based on the stale records and issuing a DNS query to refresh the stale record concurrently. CoDNS [19] is a cooperative DNS lookup service designed to alleviate client-side DNS problems. We share with their proposal the notion of client-side (i.e. resolver-

[8]Undetermined records correspond to records resulting from a non-DNSSec lookup [30].

side) changes to address DNS problems. While CoDNS involves resolvers co-operating amongst each other to mask resolver-side issues, we propose that resolvers use local storage to insure themselves (and their clients) against DoS attacks on DNS.

There have also been studies to determine the characteristics of the existing DNS architecture. Jung et. al. [11] use DNS traces to study client-perceived DNS performance and the effectiveness of client caching. They found name accesses to be heavy-tailed which also shows up in our measurements as the diminishing returns of increasing the stale cache size. [16] studied both the deployment patterns and the availability of DNS name servers while [4] measured the performance of the E root-server and observed instances of DoS attacks wherein the root-server was used as reflector.

6. FUTURE WORK

This paper presents a very simple modification to the caching behavior of DNS resolvers. A preliminary evaluation based on DNS-traces collected at Cornell University shows that stale records can be quite effective in mitigating the impact of DoS attacks on DNS. While the proposed modification certainly has some drawbacks, the cost-benefit ratio, especially given the frequency and the impact of DoS attacks, appears to favor the use of the stale cache. However, a few aspects of our proposal require more work. For instance, privacy concerns implied that we had to anonymize the collected DNS traces and hence, were not able to study the DNS records that would have been inaccurate had they been used as stale records in the face of an attack. Specifically, we would have liked to determine if this was due to load-balancing across nameservers and if the clients would still have been able to access the desired resource. We are in the process of obtaining the relevant part of the unanonymized trace to answer this and similar questions.

We are currently implementing the proposed modification into dbjdns [3], a popular DNS resolver. We also intend to explore the possibility of implementing this as an add-on to the CoDNS resolution service [19] running on PlanetLab [5]. Apart from clearing up the implementation issues, such an exercise would help us analyze the advantages of maintaining a stale cache in the face of actual attacks (which occur frequently enough to make this exercise worthwhile!).

Acknowledgements

We would like to thank Larry Parmelee at CFS for his help and patience with the DNS collection process. We are also grateful to Paul Vixie at ISC for helpful discussions on why this proposal should "not" be incorporated in DNS resolvers.

7. REFERENCES

[1] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A Layered Naming Architecture for the Internet," in *Proc. of ACM SIGCOMM*, 2004.

[2] H. Ballani and P. Francis, "A Simple Approach to DNS DoS Mitigation," in *Proc. of workshop on Hot Topics in Networks*, Nov 2006.

[3] D. J. Bernstein, "djbdns: Domain Name System Tools," Apr 2008, http://cr.yp.to/djbdns.html.

[4] N. Brownlee, k claffy, and E. Nemeth, "DNS Measurements at a Root Server," in *Proc. of Globecom*, 2001.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, July 2003.

[6] E. Cohen and H. Kaplan, "Proactive Caching of DNS Records: Addressing a Performance Bottleneck," in *Proc. of Symposium on Applications and the Internet*, 2001.

[7] R. Cox, A. Muthitacharoen, and R. T. Morris, "Serving DNS using a Peer-to-Peer Lookup Service," in *Proc. of IPTPS*, 2002.

[8] T. Deegan, J. Crowcroft, and A. Warfield, "The Main Name System: An Exercise in Centralized Computing," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, 2005.

[9] M. Handley and A. Greenhalgh, "The Case for Pushing DNS," in *Proc. of Hotnets-IV*, 2005.

[10] T. Hardy, "RFC 3258 - Distributing Authoritative Name Servers via Shared Unicast Addresses," April 2002.

[11] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, 2002.

[12] J. Kangasharju and K. W. Ross, "A Replicated Architecture for the Domain Name System," in *Proc. of INFOCOM*, 2000.

[13] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, "Providing high availability using lazy replication," *ACM Trans. Comput. Syst.*, vol. 10, no. 4, 1992.

[14] P. Mockapetris, "RFC 1035, DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION," Nov 1987.

[15] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat, "Service placement in a shared wide-area platform," in *Proc. of the USENIX '06 Annual Technical Conference*, 2006.

[16] J. Pang, J. Hendricks, A. Akella, R. D. Prisco, B. Maggs, and S. Seshan, "Availability, usage, and deployment characteristics of the domain name system," in *Proc. of Internet Measurement Conference*, 2004.

[17] V. Pappas, B. Zhang, E. Osterweil, D. Massey, and L. Zhang, "Improving DNS Service Availability by Using Long TTLs," draft-pappas-dnsop-long-ttl-02, June 2006.

[18] V. Pappas, D. Massey, and L. Zhang, "Enhancing DNS Resilience against Denial of Service Attacks," in *Proc. of Conference on Dependable Systems and Networks (DSN)*, 2007.

[19] K. Park, V. Pai, L. Peterson, and Z. Wang, "CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups," in *Proc. of USENIX OSDI*, 2004.

[20] D. S. Phatak, "Spread-Identity mechanisms for DOS resilience and Security," in *Proc. of SecureComm*, 2005.

[21] L. Poole and V. S. Pai, "ConfiDNS: leveraging scale and history to improve DNS security," in *Proc. of the 3rd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, 2006.

[22] V. Ramasubramanian and E. G. Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," in *Proc of ACM SIGCOMM*, 2004.

[23] D. Soring, "Using lightweight checkpoint/recovery to improve the availability and designability of shared memory multiprocessors," Ph.D. dissertation, University of Wisconsin-Madison, 2002.

[24] M. Theimer and M. B. Jones, "Overlook: Scalable Name Service on an Overlay Network," in *Proc. of ICDCS*, 2002.

[25] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang, "HOURS: Achieving DoS Resilience in an Open Service Hierarchy," in *Proc. of Conference on Dependable Systems and Networks (DSN)*, 2004.

[26] "Microsoft DDoS Attack, NetworkWorld," Jan 2001, http://www.networkworld.com/news/2001/0125mshacked.html.

[27] "Root Server DDoS Attack, RIPE Mail Archive," Nov 2002, https://www.ripe.net/ripe/maillists/archives/eof-list/2002/msg00009.html.

[28] "Akamai DDoS Attack, Internet Security News," Jun 2004, http://www.landfield.com/isn/mail-archive/2004/Jun/0088.html.

[29] "UltrDNS DDoS Attack, Washington Post," May 2005, http://blog.washingtonpost.com/securityfix/2006/05/blue_security_surrenders_but_s.html.

[30] "CISCO DNSSEC page," Aug 2006, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-2/dnssec.html.

[31] "Internet Systems Consortium," Aug 2006, http://www.isc.org/.

[32] "SLASHDOT: Providers Ignoring DNS TTL?" Aug 2006, http://ask.slashdot.org/article.pl?sid=05/04/18/198259&tid=95&tid=128&tid=4.

[33] "SiteKey at Bank of America," Jul 2007, http://www.bankofamerica.com/privacy/sitekey/.

[34] "DNS - What do big sites do?" Aug 2008, http://forum.powweb.com/archive/index.php/t-54961.html.

[35] "nonamed - Man page," Aug 2008, http://www.minix3.org/previous-versions/Intel-2.0.3/wwwman/man8/nonamed.8.html.