AT&T Labs - Research

subject: **On Network-Aware Clustering of Web Clients**
Work Project No. 3116-16-3010

date: **January 1, 2000**

from: **Balachander Krishnamurthy**
Dept. HA1630000
FP D229
973-360-8648
bala@research.att.com
HA1630000-000101-01TM

Jia Wang
Dept. HA163000
FP D204
973-360-8691
jiawang@research.att.com

TECHNICAL MEMORANDUM

Being able to identify the administrative entities controlling groups of clients (called *clusters*) that are responsible for a significant portion of a Web site's requests can be helpful to both the Web site and the administrative entities. The Web site could consider moving content closer to such clusters, while the administrative entity could consider placing proxy caches in front of its group of clients to take advantage of the traffic pattern. In this work, we propose a novel "network-aware" method to detect client clusters based on readily available routing information. Experimental results show that our entirely automated approach is able to identify clusters for 99.9% of the clients in a wide variety of Web server logs and sampled validation results show that we detect clusters correctly with a probability of over 90%. An efficient self-corrective mechanism increases the applicability and accuracy of our initial approach and makes it adaptive to network dynamics. In addition to being able to detect unusual access patterns made by spiders and (suspected) proxies, our proposed method is useful for content distribution and proxy positioning, and applicable to other problems such as server replication and network management.

1 Introduction

With Web traffic on the rise, Web sites have a particular interest in being able to identify clients that send many requests. If groups of such clients (called *clusters*) under the control of a single administrative entity can be identified, both the administrative entity and the Web site can take advantage of the information. The Web site can have its content replicated nearer to the administrative entity. This lowers the latency perceived by the clients in the cluster under the administrative entity's control as well as the load on the Web server. The administrative entity could also install one or more proxies in front of the client cluster to lower the latency experienced by the clients under its control. Our notion of an administrative entity is generally below that of an Autonomous System (AS) and at or above a collection of machines on a local area network. The administrative entity is responsible for the set of IP addresses in the cluster. This paper presents an entirely automated technique for a novel "network-aware" method of locating sets of interesting client clusters starting with Web server logs and using readily available routing information.

Trace-driven simulation using Web server logs have been widely used in designing and evaluating Web caching systems. Knowing the network topology helps in detecting clusters of hosts that could usefully share proxies. We also need to detect hosts with unusual access patterns, such as spiders or proxies. The only information available in the server logs identifying the client is its IP address. A simple approach to detect client clusters is to assume that if two clients have the same first three bytes in their IP addresses, then they are in the same network, and vice versa. This is the best approach to cluster detection if one relied solely on client information extracted from logs. However, as we show later, this over-simplified assumption may result in an error ratio of 50% in clustering clients. Our method detects client clusters based on routing information. We extract IP addresses from Web server logs and cluster them as close to the administrative entity as possible. The entity administering the network *alone* knows how addresses are apportioned. They can use the cluster information in deciding proxy positioning and in evaluating content distribution mechanisms. The experimental results show that our method is able to detect clusters for 99.9% of the clients. We validate our approach by examining 1% samples of client clusters and demonstrate that our method is able to detect client clusters correctly with a probability of over 90%. We also propose an efficient self-corrective mechanism to increase the applicability and accuracy of our initial approach and make it adaptive to network dynamics. Our method also detects spiders and proxies, if any, in the logs.

The remainder of this paper is organized as follows. Section 2 describes a simple approach to client clustering and gives quantitative measurements on its error ratio. Our approach to detecting client clusters is presented in Section 3. Section 4 addresses several applications of client clustering. After briefly examining related work in Section 5, we conclude with possible improvements to our work in the future.

2 Simple approach

A simple way to detect client clusters is to assume that clients sharing the same first n ($n = 24$) bits in their IP addresses (i.e., prefix length is 24) are on the same network, and vice versa. In other words, it is assumed that only Class C addresses exist (i.e., prefix length is 24). However, this is not grounded in reality. To illustrate this, Table 1 shows the distribution of prefix lengths extracted from the MAE-WEST NAP routing table snapshots taken from July 3 to July 6, 1999 [14] . The corresponding histograms are shown in Figure 1. Although the majority of the prefix length is 24 bits (i.e., around 50% prefixes are of length 24), there are a large number of prefixes that have either longer or shorter lengths. Among non-24 bit prefixes there are more shorter prefixes

	Date			
Prefix length	7/3/1999	7/4/1999	7/5/1999	7/6/1999
8	20	20	20	20
9	1	1	1	1
10	3	3	3	3
11	3	3	3	3
12	12	12	12	12
13	25	24	24	24
14	73	72	72	72
15	111	100	100	100
16	3098	3099	3095	3097
17	333	331	331	332
18	706	702	703	704
19	2092	2091	2074	2084
20	1009	1003	1006	1006
21	1275	1268	1263	1265
22	1805	1797	1798	1794
23	2227	2220	2220	2220
24	13937	14029	14013	14018
25	31	31	31	31
26	34	33	33	33
27	1	1	1	1
28	4	4	4	4
29	3	3	3	3
30	1	1	1	1

Table 1: Distribution of prefix lengths at the MAE-WEST NAP.

(< 24) than longer ones (> 24) mainly due to the allocation of CIDR (*Classless InterDomain Routing*) addresses [1] and route aggregation[2].

Because of the diversity of network prefix lengths, detecting client clusters using the first three bytes of IP addresses has two main drawbacks. First, it mis-detects small networks, which share the same first three bytes of their prefixes, by considering them as one single Class C network (Figure 2(a)). For example, the hosts 151.198.194.17, 151.198.194.34, and 151.198.194.50 share the same first three bytes (i.e., 151.198.194) of IP addresses. The simple approach will consider them to be in a single Class C network with prefix 151.198.194 and netmask length 24. However, in reality they reside in three different networks with prefixes 151.198.194.16, 151.198.194.32, and 151.198.194.48, and netmask length of 28 (Table 2). The names of these hosts are *client-151-198-194-17.bellatlantic.net*, *mailsrv1.wakefern.com*, and *firewall.commonhealthusa.com*, respectively, indicating that they most likely belong to different administrative entities. They are not likely to make common decisions on sharing proxies for example. They may even be located geographically far away from each other in the real world.

Secondly, it is obvious that the simple prefix clustering technique may mis-cluster all Class A, Class B, and

[1] Internet's growth in recent years led to possible exhaustion of Class B network address space and routers not being able to manage the size of routing tables. CIDR was proposed as a mechanism to slow the growth of routing tables and the need for allocating new IP network numbers. For example, instead of an entire block of one Class A, Class B or Class C address being allocated to a typical network, more blocks of Class C addresses can be allocated to a single network (i.e., the network prefix lengths are not necessarily 8, 16, or 24). The Internet address space is allocated in such a manner as to allow aggregation of routing information along topological lines [9, 15] .

[2] With CIDR address allocation mechanism, the routing table can be shrunk by aggregating routing entries with adjacent IP address blocks and same routing path [9, 15] .

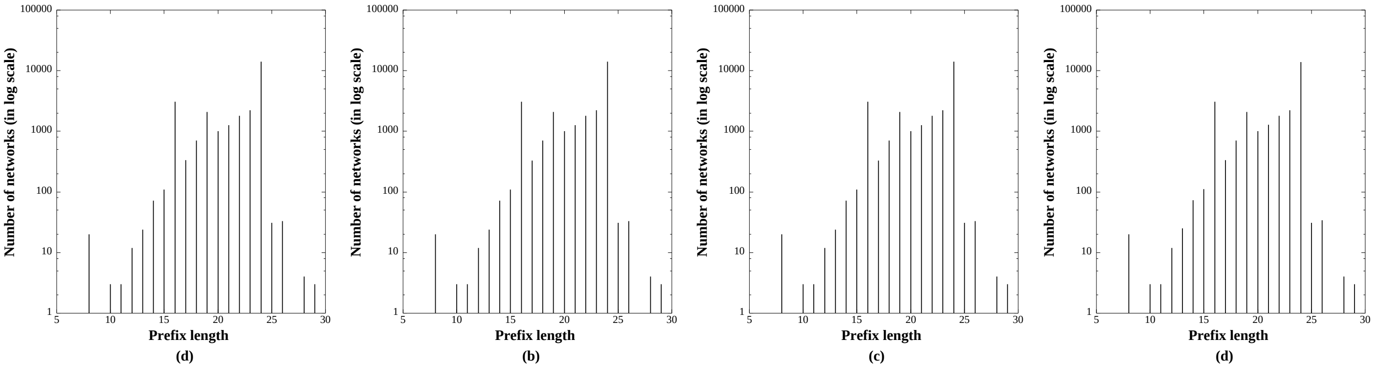

Figure 1: Histogram of prefix lengths at the mae-West NAP: (a) 07/03/1999, (b) 07/04/1999, (c) 07/05/1999, (d) 07/06/1999.

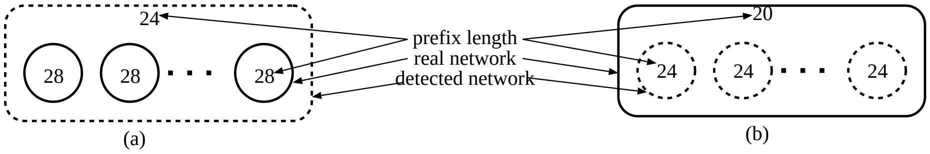

Figure 2: Two cases of mis-detecting client clusters using the simple approach.

IP address	Name	Prefix/Netmask	Routing Tables
151.198.194.17	client-151-198-194-17.bellatlantic.net	151.198.194.16/28	ARIN, NLANR
151.198.194.34	mailsrv1.wakefern.com	151.198.194.32/28	ARIN
151.198.194.50	firewall.commonhealthusa.com	151.198.194.48/28	ARIN, NLANR

Table 2: A counter example of the assumption that the prefix length is 24.

CIDR networks by dividing them into a number of Class C networks (Figure 2(b)). In this case, a trace-driven simulation may underestimate the benefit of Web proxies due to less proxy sharing/co-operating across the mis-clustered networks. In our experiments, as we show in the next section, the over-simplified assumption of the simple approach leads to an error ratio of 50 to 60% in client clustering.

3 Our approach

We propose a novel method to detect client clusters by using the prefixes and netmasks information extracted from the BGP (Border Gateway Protocol [9, 13, 15]) routing and forwarding table snapshots. The Internet consists of a large collection of hosts connected by networks of links and routers, and is divided into thousands of distinct regions of administrative control, referred to as *Autonomous Systems* (AS), ranging from college campuses and corporate networks to large Internet Service Providers (ISPs). An AS typically has very detailed knowledge of the topology within itself and limited reachability information about other ASes. Interdomain routing protocols (such as BGP) control packet forwarding among ASes and interdomain reachability information is maintained at the routers of each AS that speak BGP (Figure 3).

The rationale behind our approach is that the prefixes and corresponding netmasks identify the routes in the routing table which are used by core routers to forward packets to a given destination. Therefore, if the

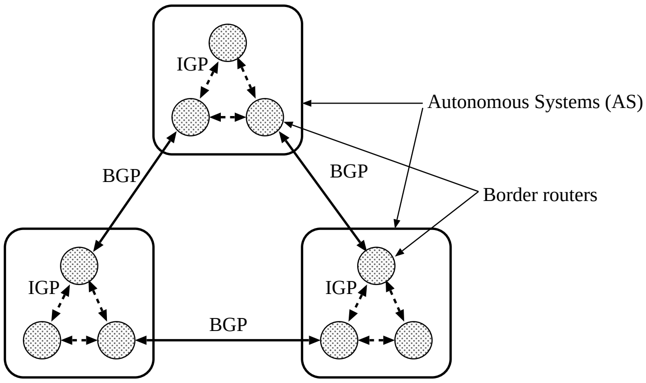

Figure 3: General illustration of AS relationships.

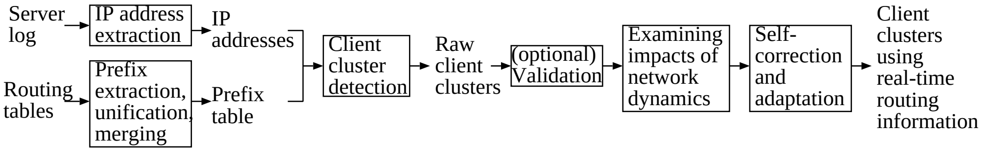

Figure 4: The entire automated process of client cluster detection.

routing table is minimal[3], then each route must correspond to a single destination subnet. Since there has been significant effort to minimize routing table sizes, it is a reasonable approximation to map route entries to subnets. This would be true in all cases due to factors such as route aggregation. Also, as we observed from Table 1, the routing table itself will change for reasons such as change of IP address allocation and link failure. Nevertheless, the reasonableness of our approach is demonstrated with a quantitative assessment later in this section.

Figure 4 gives a graphical view of the five steps in our approach.

1. Network prefix/netmask extraction and extraction of IP addresses from Web server logs;
2. Client cluster detection;
3. (Optional) validation of our method by sampling the resulting client clusters;
4. Examining effect of network dynamics on client cluster detection;
5. Self-correction and adaptation.

The validation step is optional for most applications; we include it here to provide a quantitative measurement of the accuracy of our approach. We discuss each of these steps in this section and also look at other issues affecting client cluster detection.

3.1 Network prefix extraction

The first step of our approach is to generate a merged prefix/netmask entry table. We do it in three steps.

1. Extract prefix/netmask entry from the routing table snapshots;

[3]The router minimizes the size of routing table as much as possible by aggregating outgoing routes.

Name	Date	Size	Comments
AADS	12/7/1999	17K	BGP routing table snapshots updated every 2 hours
ARIN	10/1999	300K	IP network dump
AT&T-BGP	12/15/1999	74K	BGP routing table snapshots
AT&T-Forw	4/28/1999	65K	BGP forwarding table snapshots
CANET	12/1/1999	1.7K	Real-time BGP routing table snapshots
CERFNET	9/29/1999	50K	Real-time BGP routing table snapshots
MAE-EAST	12/7/1999	46K	BGP routing table snapshots taken every 2 hours
MAE-WEST	12/7/1999	30K	BGP routing table snapshots taken every 2 hours
NLANR	11/1997	200K	IP network dump
OREGON	12/7/1999	70K	Real-time BGP routing table snapshots
PACBELL	12/7/1999	25K	BGP routing table snapshots updated every 2 hours
PAIX	12/7/1999	10K	BGP routing table snapshots updated every 2 hours
SINGAREN	12/7/1999	68K	Real-time BGP routing table snapshots
VBNS	12/7/1999	1.8K	BGP routing table snapshots updated every 30 minutes

Table 3: Our collection of BGP routing tables.

2. Unify prefix/netmask entries into a standard format;

3. Merge entries obtained from different routing tables into one big table.

3.1.1 Prefix entry extraction

The BGP routing tables (Table 3) are collected automatically via a simple script from AADS, MAE-EAST, MAE-WEST, PACBELL, PAIX (all from [14]), ARIN [3] , AT&T-Forw and AT&T-BGP (from [4]), CANET [7] , CERFNET [8] , NLANR [16] , OREGON [17] , SINGAREN [18] , and VBNS [19] . We do so by one of two ways:

- Download routing tables from well-known Web sites (e.g., AADS)
- *Telnet* to a particular host to run a script to dump routing tables (e.g., OREGON).

The size of the table depends on the location of the BGP routers and the AS it belongs to. The number of prefix/netmask entries extracted from each routing table are shown in Figure 5 and Table 3. We assembled a total of 391,497 unique prefix/netmask entries[4].

- AADS is a near-real-time snapshot (taken every 2 hours) of the complete, "default-free"[5] Internet routing table as seen by the route servers at AADS NAP. It contains 17K destination networks. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 9.
- ARIN is the October 1999 version of the ARIN `ip_network_dump` file, which gives a numerical representation of the networks registered (IP allocation) at the American Registry for Internet Numbers (ARIN). The ARIN tables are the largest collection of network prefixes/netmasks to the best of our knowledge. There are over 300K unique networks registered at ARIN at the time the dump file was taken. An example snapshot of the routing table is shown in Table 4.
- AT&-BGP contains a dump of BGP tables from a collection of AT&T WorldNet routers, collected on December 15, 1999. There are over 74K destination networks. An example snapshot of the BGP routing table is shown in Table 5.

[4]These include prefix entries collected from both BGP routing table and IP network dumps as explained later in this section.
[5]A default-free table does not contain the default route (e.g. 0.0.0.0/0.0.0.0) in the snapshots.

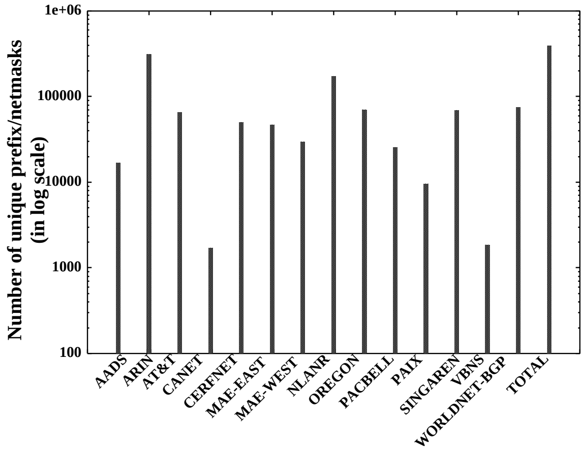

Figure 5: Number of unique prefix/netmask entries extracted from routing tables.

- AT&T-Frow is a set of forwarding table entries collected from AT&T Worldnet on April 28, 1999. It contains 65K destination networks. An example snapshot of the forwarding table is shown in Table 6.
- CANET contains CA∗Net II routing tables, which have around 1.7K destination networks. The snapshot we used in our experiments was taken on December 1, 1999. An example snapshot of the routing table is shown in Table 7.
- CERFNET is the real-time dump of BGP routing table at CERFnet, which contains approximately 50K unique destination networks (prefix/netmask). The snapshot we used in our experiments was taken on September 29, 1999. An example snapshot of the routing table is shown in Table 8.
- MAE-EAST is a near-real-time snapshot (taken every 2 hours) of the complete, "default-free" Internet routing table as seen by the route servers at Mae-East NAP. It contains 46K destination networks. The snapshot we used in our experiments was taken on June 24, 1999. An example snapshot of the routing table is shown in Table 9.
- MAE-WEST is a near-real-time snapshot (taken every 2 hours) of the complete, "default-free" Internet routing table as seen by the route servers at Mae-west NAP. It contains 30K destination networks. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 9.
- NLANR is the November 1997 version[6] of the InterNIC's netinfo/ip_network_dump file, which gives a numerical representation of the networks registered (IP allocation) at InterNIC. There are around 200K unique networks registered at InterNIC at the time the dump file was taken. An example snapshot of the routing table is shown in Table 4.
- OREGON provides real-time access to BGP routing dumps. The current participants are ANS (Cleveland), ATT (Chicago), BBNPlanet (Palo Alto), CERFnet (San Diego), DIGEX (MAE-EAST), EBONE (EU), ESnet (GA), RIPE NCC (Amsterdam), IAGnet (Chicago), IIJ (Japan), JINX (Johannesburg), LINX (London), MCI (San Francisco), PIPEX (London), Sprint (Stockton), vBNS (Hayward), Verio (MAE-WEST) and blackrose.org (Ann Arbor). It contains 70K destination networks and is considered to be the largest

[6]The most recent version of NLANR's IP network dump is from 1997, but as we only use it as a secondary source of network prefixes in our experiments; its lack of recency has a very minor impact on our results.

IP-Network:	155.21.0.0/16
Created:	19911028050000000
—	
IP-Network:	155.26.0.0/16
Created:	19911028050000000
—	

Table 4: An example snapshot of ARIN and NLANR tables.

			Network	Next hop	Metric	LocPrf	Weight	Path
*	>	i	4.17.1.0/24	12.127.2.254	0	100	0	14045 i
*	>	i		12.127.2.254	0	100	0	14045 i
*		i	4.24.76.224/30	192.205.31.161	0	100	0	i
*	>	i		192.205.31.161	0	100	0	i
*		i	6.0.0.0	12.123.193.15		100	0	7170 1455 i
*	>	i		12.123.193.15		100	0	7170 1455 i
*				192.41.177.8		82	0	7170 1455 i

Table 5: An example snapshot of WORLDNET-BGP routing table.

Prefix	Next hop	Interface
199.164.187.0/24	12.127.45.130	Hssi10/0/1
199.164.211.0/24	12.127.45.130	Hssi10/0/1

Table 6: An example snapshot of AT&T forwarding table.

BGP routing table dump. The snapshots we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 8.

- PACBELL is near-real-time snapshot (taken every 2 hours) of the complete, "default-free" Internet routing table as seen by the route servers at PacBell NAP. It contains around 25K destination networks. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 9.

- PAIX is near-real-time snapshot (taken every 2 hours) of the complete, "default-free" Internet routing table as seen by the route servers at Paix NAP. It contains around 10K destination networks. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 9.

- SINGAREN contains routing tables as viewed at SingAREN BGP peers: `1net`, `abilene`, `apan`, `esnet`, `ihpc`, `imcb`, `korea`, `krdl`, `nanyang`, `napnet`, `ncb`, `nordunet`, `ntu`, `nus`, `startap`, `stc`, `surfnet`, `tanet`, `temasek`, `tgbackup`, and `vbns`, which have around 68K destination networks. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 10.

- VBNS contains all vBNS routes snapshots (about 1.8K destination networks), updated every 30 minutes from `cs.res.vbns.net`. The snapshot we used in our experiments was taken on December 7, 1999. An example snapshot of the routing table is shown in Table 11.

Routes		AS path
134.87.2.0/24	BCnet	6509 271
134.117.0.0	Carleton University	6509 10786
137.82.0.0	University of British Columbia	6509 271
137.122.0.0	University of Ottawa	6509 10786

Table 7: An example snapshot of CANET routing table.

	Network	Next hop	Metric	LocPrf	Weight	Path
*	3.0.0.0	134.55.24.6			0	293 701 80 i
*		206.157.77.11	70		0	1673 701 80 i
*		193.0.0.56			0	3333 286 701 80 i
*		129.250.0.3	0		0	2914 701 80 i
*	4.0.0.0	134.55.24.6			0	293 1 i
*		206.157.77.11	75		0	1673 1 i

Table 8: An example snapshot of CERFNET and OREGON routing tables.

193/255.255.252			
	SPRINT (1239)	N=192.41.177.241	1239 4000 5583 3333 IGP
	EUnet (286)	N=192.41.177.120	286 3333 IGP
	CAIS (3491)	N=192.41.177.96	3491 6660 5378 3333 IGP
	CAIS (3491)	N=192.41.177.85	3491 6660 5378 3333 IGP
	INS (6660)	N=192.41.177.112	6660 5378 3333 IGP
193.0.14/255.255.255			
	SPRINT (1239)	N=192.41.177.241	1239 2529 5459 IGP
	CAIS (3491)	N=192.41.177.96	3491 6660 5378 5459 IGP
	EXODUS (3967)	N=192.41.177.119	3967 2529 5459 IGP
	CAIS (3491)	N=192.41.177.85	3491 6660 5378 5459 IGP
	Verio (2914)	N=192.41.177.196	2914 5413 5459 EGP
	INS (6660)	N=192.41.177.112	6660 5378 5459 IGP
	Verio (2914)	N=192.41.177.121	2914 5413 5459 EGP
	GXNnet (5413)	N=192.41.177.215	5413 5459 IGP

Table 9: An example snapshot of AADS, MAE-EAST, MAE-WEST, PACBELL, and PAIX routing tables.

Prefix	Network	AS path
128.88.0.0	Hewlett-Packard Company	7658 151
24.40.0.0/20	Suburban Cable	5646 1 1239 10643

Table 10: An example snapshot of SINGAREN routing table.

Prefix	Prefix description	Next hop	AS path	Peer AS description
6.0.0.0/8	Army Information Systems Center	cs.ny-nap.vbns.net	7170 1455 (IGP)	AT&T Government Markets
12.0.48.0/20	Harvard University	cs.cht.vbns.net	1742 (IGP)	Harvard University
12.6.208.0/20	AT&T ITS	cs.cht.vbns.net	1742 (IGP)	Harvard University
18.0.0.0/8	Massachusetts Institute of Technology	cs.cht.vbns.net	3 (IGP)	Massachusetts Institute of Technology

Table 11: An example snapshot of VBNS routing table.

Though the routing table also contains interdomain information such as next hop IP address, AS number, and AS path, we have only used the prefix/netmask information in our experiments. The AS number and path information can also provide hints on the geographical location of clients.

It is also worth noting that, although ARIN and NLANR are the two largest tables, they are not generated directly from BGP information, but from a dump of the networks registered at these two sites. The difference between a network dump file and a BGP routing table dump file is that the former usually has a much larger collection of networks. An IP address registered/allocated at these two sites may not necessarily exist and be a routable host on the Internet. However, since our client IP addresses are gathered from real Web server logs, this does not affect us. A network entry in a network dump file is usually larger than one in a BGP routing table (i.e., it has a shorter network prefix length) despite the fact that routes may be aggregated in a BGP routing table. This is because an AS, that requests IP addresses from ARIN and NLANR sites, may further partition and allocate these IP addresses to small network entities. In this case, ARIN and NLANR do not know about such address partitions. This might cause an error in our approach if the network prefix is taken from those IP network dumps. However, this case is relatively rare (less than 1% of clients are clustered by network prefix taken from IP network dump files). We use BGP dump files as primary sources of network prefixes in detecting client clusters and use IP network dump files as secondary sources because each BGP routing table will only have a limited view of the entire network topology. In our experiments, this improves the proportion of the clustered clients from 99% to 99.9%.

3.1.2 Network prefix format unification

The network prefix/netmask entries extracted from different routing tables are in one of three formats as shown in Table 12.

1. $x1.x2.x3.x4/k1.k2.k3.k4$: The format $x1.x2.x3.x4/k1.k2.k3.k4$ is used in routing tables at AADS, MAE-EAST, MAE-WEST, PACBELL, and PAIX, where $x1.x2.x3.x4$ and $k1.k2.k3.k4$ are network prefix and netmask with zeroes dropped at the tail. One such example is 193.1/255.255, which corresponds to 193.1.0.0/255.255.0.0, where 193.1.0.0 and 255.255.0.0 are network prefix and netmask, respectively.

2. $x1.x2.x3.x4/l$: The network prefix/netmask entry may also be in the format $x1.x2.x3.x4/l$ as in routing tables at ARIN, AT&T, CANET, CERFNET, NLANR, OREGON, SINGAREN, VBNS, and WORLDNET-BGP, where $x1.x2.x3.x4$ is the prefix and l is the netmask length. For example, 128.148.0.0/16 stands for 128.148.0.0/255.255.0.0, where 128.148.0.0 and 255.255.0.0 are network prefix and netmask.

3. $x1.x2.x3.0$: The format $x1.x2.x3.0$, which can be found in CANET, CERFNET, OREGON, SINGAREN, and WORLDNET-BGP is an abbreviated representation of $x1.x2.x3.0/k1.k2.k3.0$ (i.e., $x1.x2.x3.0$ is a block of standard Class A, Class B, or Class C addresses and the network prefix is 8, 16, or 24, respectively). For example, 130.15.0.0 is abbreviation of 130.15.0.0/255.255.0.0.

Due to the variety of formats that network prefixes/netmasks may have, it is desirable to unify the prefixes/netmasks extracted from different routing tables into a single format. In our experiments, we arbitrarily chose the first format listed in Table 12 as our standard format and converted prefix/netmask entries of formats $x1.x2.x3.x4/l$ and $x1.x2.x3.0$ into format $x1.x2.x3.x4/k1.k2.k3.k4$. For instance, we converted prefix/netmask examples 128.148.0.0/16, 130.15.0.0, and 192.75.72.0 listed in Table 12 to 128.148/255.255, 130.15/255.255, and 192.75.72/255.255.255, respectively.

Formats	$x1.x2.x3.x4/k1.k2.k3.k4$	$x1.x2.x3.x4/l$	$x1.x2.x3.0$
Routing tables	AADS MAE-EAST MAE-WEST PACBELL PAIX	ARIN AT&T CANET CERFNET NLANR OREGON SINGAREN VBNS WORLDNET-BGP	CANET CERFNET OREGON SINGAREN WORLDNET-BGP
Examples	193.1/255.255 193.0.128/255.255.192	128.148.0.0/16	130.15.0.0 192.75.72.0
Unification	193.1/255.255 193.0.128/255.255.192	128.148/255.255	130.15/255.255 192.75.72/255.255.255

Table 12: The formats of network prefix and netmask in routing table snapshots.

3.1.3 Merging network prefix entries

After unifying network prefix entry formats, we measure the routing table coverage by counting the number of unique prefix/netmask entries extracted from each routing table. As we observed in Figure 5 and Table 3, some routing tables have a better view of network routes than others (i.e., they contains more prefix/netmask entries) and none of them contains complete information of all the prefixes and netmasks. This is mainly due to the fact that not all routes are visible to each router. To resolve this problem, we merge them into one big prefix/netmask table and used this table to cluster clients in server logs.

3.2 Client cluster detection

Once the IP addresses are extracted from the Web server logs (by simply extracting the first column in most logs) and the merged prefix tables is created from routing table snapshots, the second step of our approach is to detect client clusters in the logs. In this section, we describe our methodology of detecting client clusters and then our experiments on various Web server logs to demonstrate the applicability and generality of our approach.

3.2.1 Methodology

Our experiments on detecting client clusters involve three steps:

1. Extract the client IP addresses from the server log;
2. Perform the longest prefix matching (similar to what IP routers do) on each client IP address using the prefix/netmask table we constructed;
3. Classify all the client IP addresses that have the same longest matched prefix into one client cluster, which is identified by the shared prefix.

For example, suppose we want to cluster the IP addresses 12.65.147.94, 12.65.147.149, 12.65.146.207, 12.65.144.247, 24.48.3.87, and 24.48.2.166. The longest matched prefixes of them are 12.65.128.0/19, 12.65.128.0/19, 12.65.128.0/19, 12.65.128.0/19, 24.48.2.0/23, and 24.48.2.0/23, respectively. We then classify the first four clients into a client

Date	Feb 13, 1998
Length of period (days)	1
Total number of requests	11665713
Total number of unique URLs	33875
Total number of unique clients	59582
Number of undetected clients	0
Total number of client clusters	9853
Largest client cluster	1343 clients (134963 requests)
Smallest client cluster	1 client (1 request)
Client cluster URL access	1 URL (2.95203e-05% of total) - 8095 URLs (0.238967% of total)

Table 13: Experimental results of client cluster detection on Nagano server log.

cluster identified by prefix/netmask 12.65.128.0/19 and the last two clients into another client cluster identified by prefix/netmask 24.48.2.0/23. Metrics of client clusters, such as the distribution of number of clients in client clusters, the distribution of number of requests issued from within a client cluster, and the distribution of number of unique URLs accessed from within a client cluster, can be generated for various applications (discussed in Section 4).

3.2.2 Applicability

We use the Nagano server log to demonstrate the applicability of our method. The Nagano log is a 1 day extract from the 1998 Winter Olympic Games server log taken on February 13, 1998[7]. A summary of the client cluster detection result is provided in Table 13. There are a total of 11,665,713 requests in the server log, issued by 59,582 clients accessing 33,875 unique URLs at the server. We extract the IP addresses[8] of these clients, run the cluster detection algorithm and group all of them into 9,853 *client clusters*. The size of client clusters varies from 1 to 1,343 clients, the number of requests issued from within each client cluster varies from 1 to 339,632, and clusters access anywhere from 1 to 8,095 unique URLs.

In order to get more insight on client clusters, we plot the cumulative distribution of clients and requests in a client cluster in Figure 6. Figure 6(a) shows the cumulative distribution of the number of clients in a client cluster. We observe that, although the largest client cluster contains 1,343 clients, more than 95% of client clusters contain less than 100 clients. The cumulative distribution of requests issued by a client cluster is shown in Figure 6(b), which is more heavy-tailed than that of the number of clients in a client cluster as shown in Figure 6(a), implying possible existence of proxies and/or spiders. Around 90% of the client clusters issued less than 1,000 requests. Only a few client clusters are very busy, issuing up to a maximum of 339,632 requests. We should note that such Zipf-like distributions are common in a variety of Web measurements [6] .

Figures 7(a), (b), and (c) show the distributions of number of clients in client clusters, number of requests issued from within client clusters, and number of URLs accessed from within client clusters, respectively. They are plotted in reverse order of number of clients in a cluster (i.e., larger client clusters are on the left side). From Figures 7(b) and (c), we see that, while larger client clusters usually issue more requests and access more URLs

[7] In our approach, we use the network prefix and netmask information in BGP routing tables as an approximation of the collection of IP addresses belonging to each network (or AS)—this rarely changes and most of the changes are incremental. The effect of age difference between the BGP routing table dumps (used in detecting client clusters) and the server logs is fixed in the self-correction and adaptation stage of our approach (discussed later in this section).

[8] Requests issued from IP address 0.0.0.0, an arbitrary address typically used as source address in protocols such as BOOTP when the client doesn't know its own IP address were ignored and client 0.0.0.0 was excluded from our experiments.

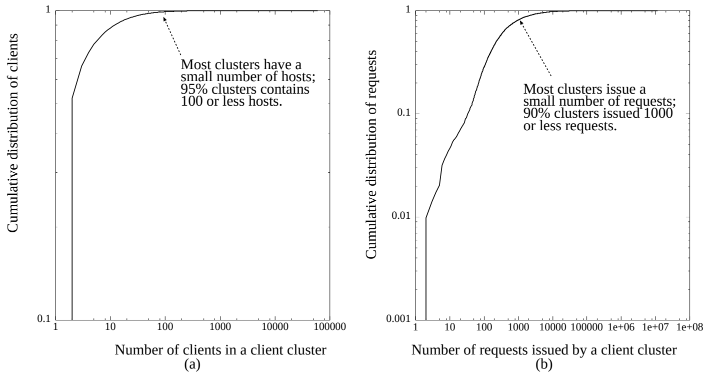

Figure 6: The cumulative distribution of clients and requests in a client cluster for Nagano server log (y axis is in log scale): (a) is the cumulative distribution of number of clients in client clusters; (b) is the cumulative distribution of number of requests issued from within client clusters.

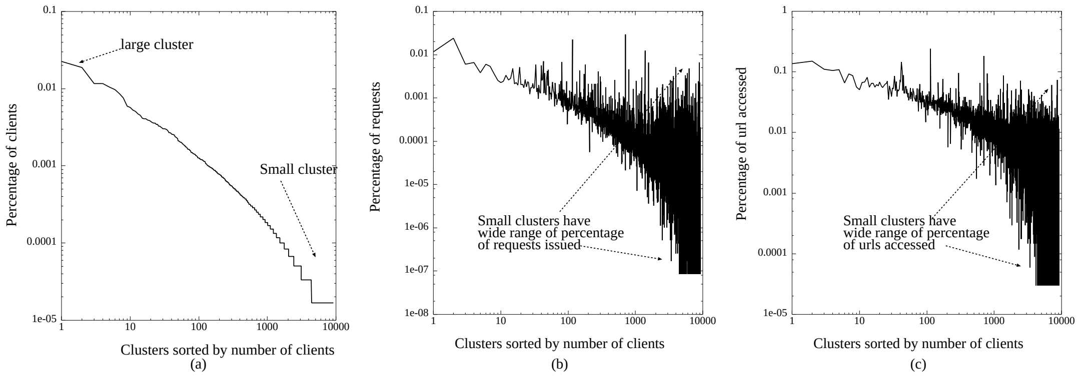

Figure 7: The client cluster distribution for the Nagano server log plotted in reverse order of number of clients in a cluster (both x axis and y axis are in log scale): (a) is the distribution of number of clients in client clusters; (b) is the distribution of the number of requests issued from within client clusters; and (c) is the distribution of the number of URLs accessed from within client clusters. Note that larger client clusters are on the left.

than smaller client clusters, there are a number of relatively small client clusters which issue a significant number of requests ($\sim$ 1% of the total) and/or access a big fraction of URLs ($\sim$ 20% of the total) at the server. Locating such unusual clusters is useful in identifying suspected spiders and proxies.

We re-plot the same set of data shown in Figure 7 with different sorting of clusters (on x axis)—in reverse order of number of requests. Figures 8(a), (b), and (c) show the distributions of number of requests issued from within client clusters, number of clients in client clusters and number of URLs accessed from within client clusters, respectively. Comparing Figure 8(a) with Figure 7(a), the result further demonstrates that the distribution of number of requests issued from within client clusters is more heavy-tailed than that of number of clients in client clusters, which implies possible existence of proxies and/or spiders. Figures 8(b) and (c) show that busy client clusters usually have a large number of clients and access a big fraction of URLs at the server. We observe that some busy clusters actually have very small number of clients and may access very few URLs—these are also useful measures in identifying spiders and proxies.

Note that Figures 7(a), (b), and (c) are plotted in such a way that the points at the same position on the x axis correspond to the same client cluster. For example, cluster 10 (i.e., x = 10) in Figure 7(a), (b), and (c) refer to the same client cluster. The same is true for Figure 8.

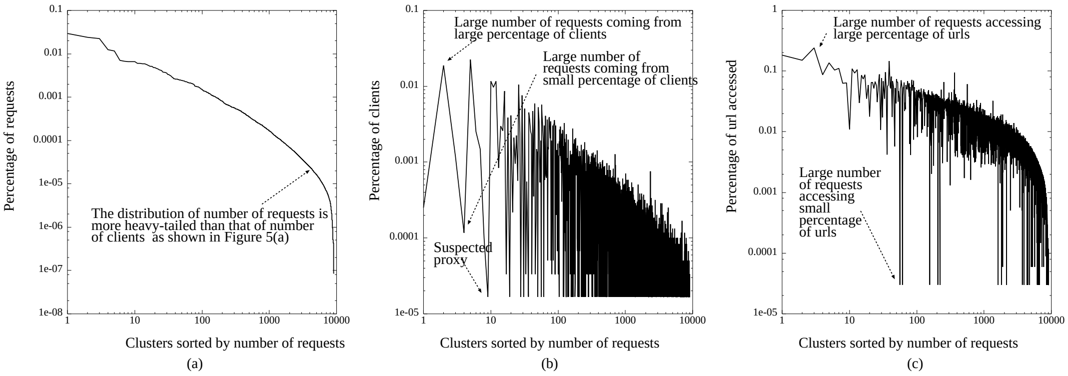

Figure 8: The client cluster distribution for the Nagano server log plotted in reverse order of number of requests (both x axis and y axis are in log scale): (a) is the distribution of number of requests issued from within client clusters (re-plot of Figures 7(b)); (b) is the distribution of number of clients in client clusters (re-plot of Figure 7(a)); and (c) is the distribution of number of URLs accessed from within client clusters (re-plot of Figure 7(c)).

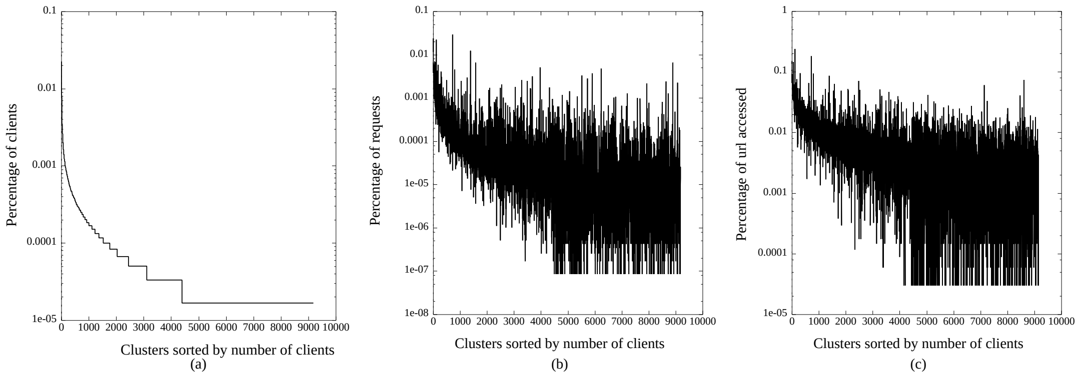

Figure 9: The client cluster distribution for the Nagano server log plotted in reverse order of number of clients (re-plot of Figure 7, only y axis is in log scale) to clarify how client cluster distributions vary along number of requests issued from within client clusters.

To get a clearer view on how the client cluster distributions vary along the number of clients in client clusters we re-plot Figure 7 with only the y axis in log scale—the modified figure is shown in Figure 9. Similarly, to see how the client cluster distributions vary along the number of requests issued from within client clusters, we re-plot Figure 8 with only the y axis in log scale—the modified figure is Figure 10.

3.2.3 Generality of our approach

In order to examine the generality of our approach, we extended our experiments to a wide range of Web server logs. The set of logs on which we experimented range in number of requests (148K to 46 Million), number of clients (40K to 481K), number of unique URLS (340 to 116K), duration (24 hours to 94 days), location (various sites in North and South America), organization (governmental, non-profit, commercial), and nature (transient event logs and popular site logs). The sites included Apache, Unav, EW3, and Sun server logs.

- **Apache server log**

  Apache log is a 49-day Web server log taken from October 1, 1997 to November 18, 1997. A summary of the

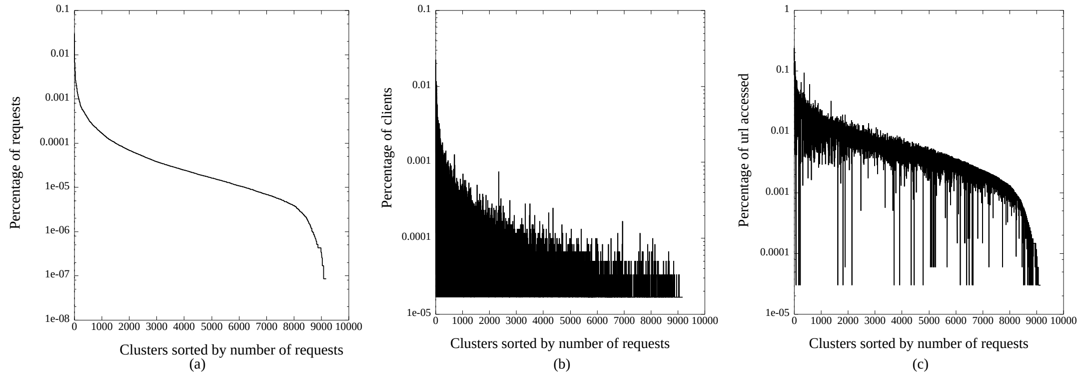

Figure 10: The client cluster distribution for the Nagano server log plotted in reverse order of number of requests (re-plot of Figure 8, only y axis is in log scale) to clarify how client cluster distributions vary along number of requests issued from within client clusters.

client cluster detection result is provided in Table 14. There are a total of 3,461,364 requests in the server log issued by 274,844 clients accessing 51,536 unique URLs at the server. We detected all these clients as 35,563 client clusters. The size of client clusters varies from 1 to 1,680 clients, the number of requests issued from within each client cluster varies from 1 to 78,336, and clusters access anywhere from 1 to 21,207 unique URLs.

We plot the client cluster distributions in Figures 11 and 12. Figure 11(a) shows the cumulative distribution of the number of clients in client clusters. We observe that, although the largest client cluster contains 1,690 clients, more than 95% of the client clusters contain less than 100 clients. The cumulative distribution of requests issued in client clusters is shown in Figure 11(b), which is more heavy-tailed than that of the number of clients in client clusters as shown in Figure 11(a). Around 95% of the client clusters issued less than 1,000 requests. Only a few client clusters are very *busy*, issuing up to a maximum of 78,336 requests. Figures 11(c) and (d) show the distributions of the number of clients in client clusters (in the reverse order of number of clients) and number of requests issued from within client clusters (in reverse order of number of requests issued), respectively, again demonstrating that the distribution of number of requests issued from within client clusters in more heavy-tailed than the number of clients in client clusters.

We also plot the distributions of number of requests issued from within client clusters and the number of unique URLs accessed from within client clusters (in the reverse order of number of clients) in Figure 11(e) and (g), respectively. Again larger client clusters issue more requests and access more URLs than smaller client clusters. Also, there are a number of relatively small client clusters which issue significant amount of requests ($\sim$ 2% of the total) and/or access a big fraction of URLs ($\sim$ 40% of the total) at the server. Again, we plot the distributions of number of clients in client clusters and number of URLs accessed from within client clusters (in the reverse order of number of requests issued) in Figures 11(f) and (h), respectively.

- **Unav server log**

Unav [9] is an event log that was obtained over a thirty-two hour period during December 1999. A summary of the client cluster detection result is provided in Table 15. There are a total of 6,593,672 requests in the server log issued by 28,672 clients accessing 24,150 unique URLs at the server. We detected all these clients as 2,480 client clusters. The size of client clusters varies from 1 to 7,918 clients and the number of requests issued from within each client cluster varies from 1 to 1,507,769.

[9] This event log was provided to me (Bala) under the condition that I do not reveal details associated with the event

Date	October 1, 1999 - November 18, 1999
Length of period (days)	49
Total number of requests	3461361
Total number of unique URLs	51536
Total number of unique clients	274844
Number of undetected clients	0
Total number of client clusters	35563
Largest client cluster	1680 clients (17529 requests)
Smallest client cluster	1 client (1 request)
Client cluster URL access	1 URL (0.000019% of total) - 21207 URLs (0.411499% of total)

Table 14: Experimental results of client cluster detection on the Apache server log.

Date	December 1999
Total number of requests	6593672
Total number of unique URLs	24150
Total number of unique clients	28672
Number of undetected clients	21
Total number of client clusters	2480
Largest client cluster	7918 clients (966788 requests)
Smallest client cluster	1 client (1 request)

Table 15: Experimental results of client cluster detection on the Unav server log.

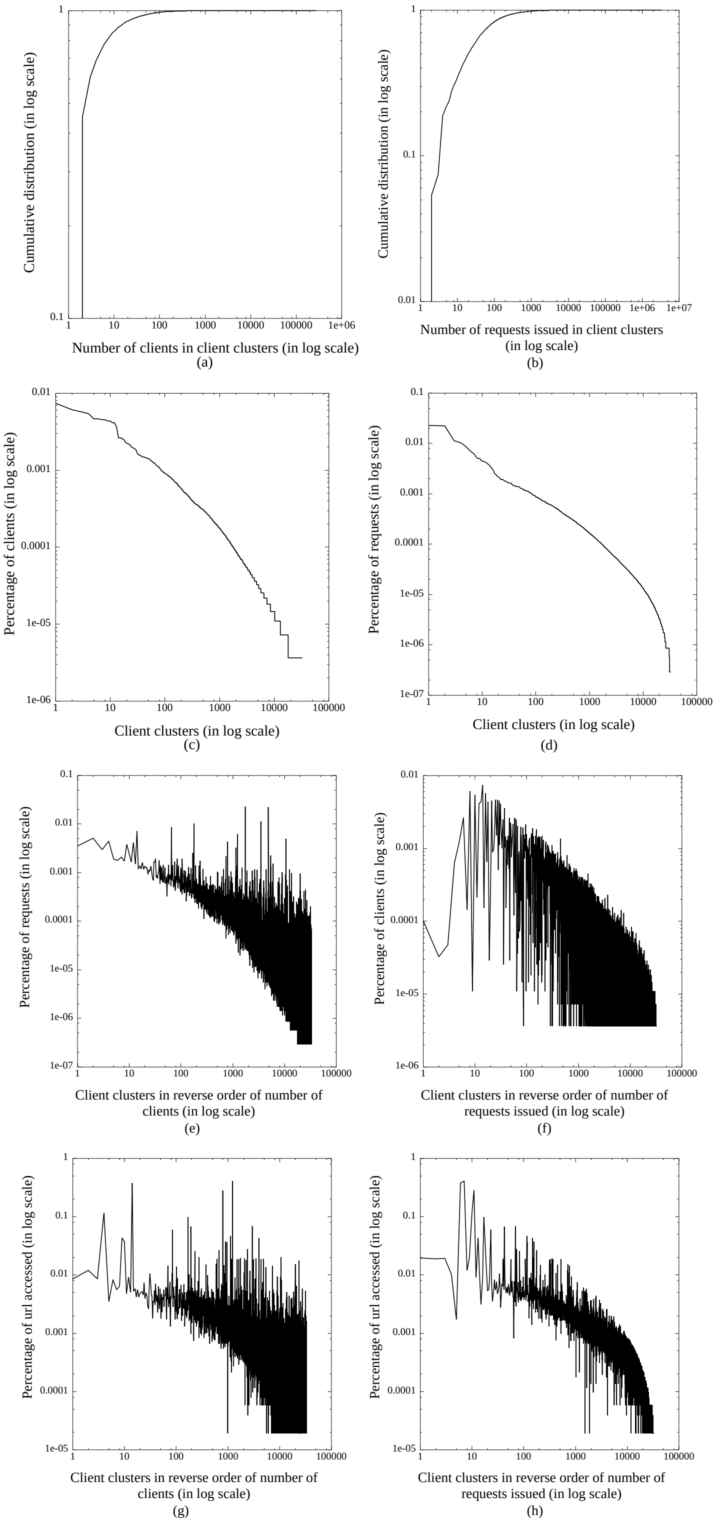

Figure 11: The client cluster distribution for the Apache server log (both x axis and y axis are in log scale): (a) is the cumulative distribution of number of clients in client clusters; (b) is the cumulative distribution of number of requests issued from within client clusters; (c), (e), and (g) are the distributions of number of clients in client clusters, number of requests issued from within client clusters and number of URLs accessed from within client clusters (in reverse order of number of clients), respectively; (d), (f) and (h) are the distributions of number of requests issued from within client clusters, number of clients in client clusters and number of URLs accessed from within client clusters (in reverse order of number of requests), respectively.

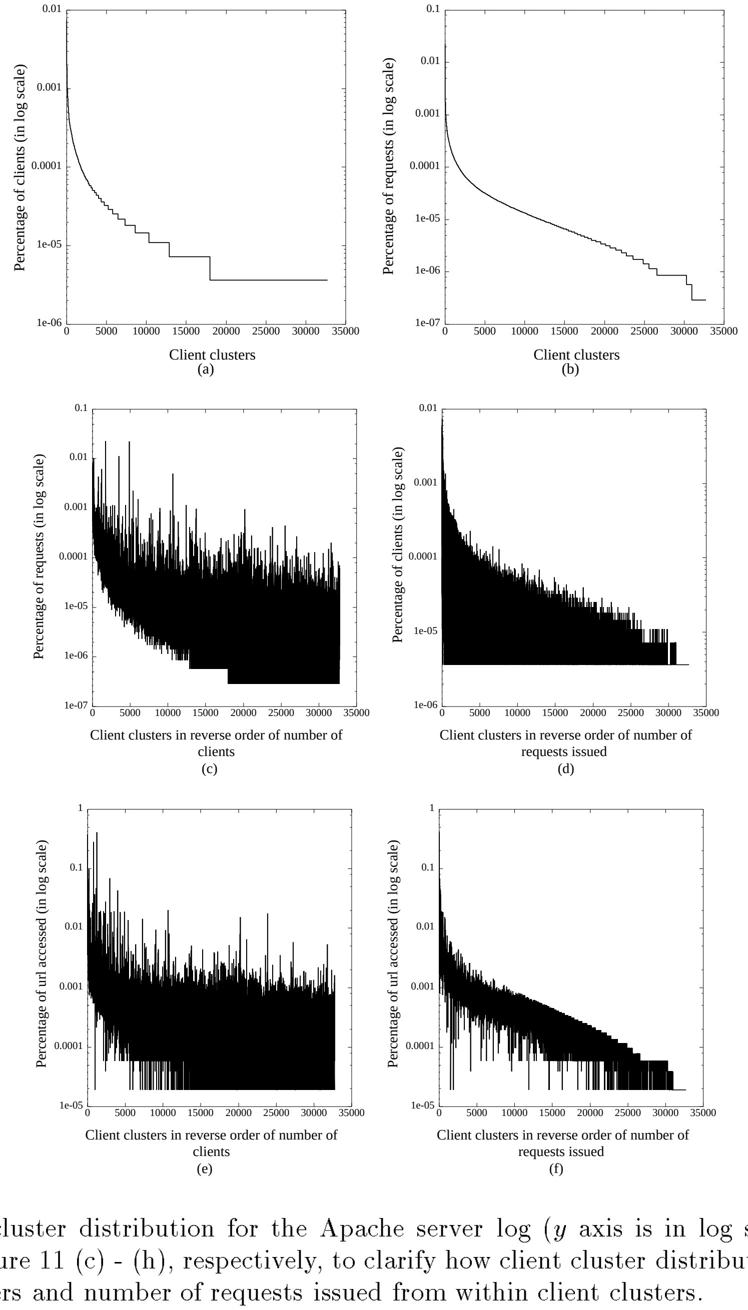

Figure 12: The client cluster distribution for the Apache server log (y axis is in log scale): (a) - (f) are the re-plotted figures of Figure 11 (c) - (h), respectively, to clarify how client cluster distributions vary along number of clients in client clusters and number of requests issued from within client clusters.

- **EW3 server logs**

  We studied five 94-day EW3 logs: EW3-1, EW3-4, EW-6, EW-18, and EW-60. A summary of the client clustering result is provided in Table 16.

  - **EW3-1**: EW3-1 log is a 94-day Web server log taken from November 21, 1997 to February 22, 1998. There are a total of 13,747,204 requests in the server log, issued by 204,621 clients accessing 562 unique URLs at the server. Among these clients, only 138,879 clients can be identified by IP address, which issued 2,252,912 requests. We detected them as 23,465 client clusters. The size of client clusters varies from 1 to 1,477 clients. Only 2 clients are undetected.
  - **EW3-4**: EW3-4 log is a 94-day Web server log taken from November 21, 1997 to February 22 22, 1998. There are a total of 6,273,792 requests in the server log, issued by 79,623 clients accessing 3,550 unique URLs at the server. Only 52,534 clients can be identified by IP address, issuing 1,079,937 requests. We detected all of them as 13,646 client clusters. The size of client clusters varies from 1 to 803 clients.
  - **EW3-6**: EW3-6 log is a 94-day Web server log taken from November 21, 1997 to February 22 22, 1998. There are a total of 6,281,303 requests in the server log issued by 102,454 clients and access 2,681 unique URLs at the server. Among these clients, only 689,40 clients can be identified by IP address, issuing 1,260,277 requests. We detected all of them as 15,735 client clusters. The size of client clusters varies from 1 to 781 clients.
  - **EW3-18**: EW3-18 log is a 94-day Web server log taken from November 21, 1997 to February 22 22, 1998. There are a total of 3,902,401 requests in the server log, issued by 58,839 clients accessing 340 unique URLs at the server. Among these clients, only 37,049 clients can be identified by IP address, which issued 612,846 requests. We detected them as 10,801 client clusters. The size of client clusters varies from 1 to 886 clients. Only 1 client is undetected.
  - **EW3-60**: EW3-60 log is a 94-day Web server log taken from November 21, 1997 to February 22 22, 1998. There are a total of 45,903,647 requests in the server log, issued by 481,628 clients accessing 16,377 unique URLs at the server. Among these clients, only 274,162 clients can be identified by IP address, issuing 4,647,725 requests. We detected them as 16,688 client clusters. The size of client clusters varies from 1 to 10,609 clients. Only 28 clients are undetected.

  Overall, more than 99.9% of the clients that can be identified by IP addresses in EW3 server logs can be clustered using our method. Since most clients in EW3 server logs are identified by their name instead of IP address, we restricted our experiments to other logs.

  We also studied more recent EW3 server logs, four one-month logs of July 1999 (EW3-a1, EW3-b, EW3-c1, and EW3-c2) and two six-month logs from July 1999 to December 1999 (EW3-a2 and EW3-w).

  - **EW3-a1** EW3-a1 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,199,276 requests in the server log, issued by 21,519 clients accessing 683 unique URLs at the server. We detected all of these clients as 7,754 client clusters. The size of client clusters varies from 1 to 359 clients, while the number of requests issued from within each client cluster varies from 1 to 21,949 (Table 17).
  - **EW3-a2** EW3-a2 log is a 6-month Web server log taken from June 26, 1999 to December 20, 1999. We divided it into 6 sub-logs, one for each month. The detailed information is provided in Table 18.
  - **EW3-b** EW3-b log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,908,761 requests in the server log, issued by 40,156 clients accessing 7,772 unique URLs at the server. We detected all of these clients as 14,895 client clusters. The size of client clusters varies

Log	EW3-1	EW3-4	EW3-6	EW3-18	EW3-60
Length of period (days)	94	94	94	94	94
Total number of requests	13747204	6273792	6281303	3902401	45903647
Total number of unique URLs	562	3550	2681	340	16377
Total number of unique clients	204621	79623	102454	58839	481628
Number of clients identified by IP address	138879	52534	68940	37049	274162
Percentage of clients identified by IP address	67%	65%	67%	62%	56%
Number of requests issued by clients with IP	2252912	1079937	1260277	612846	4647725
Percentage of requests issued by clients with IP	16%	17%	20%	15%	10%
Number of undetected clients	2	0	0	1	28
Total number of client clusters	23465	13646	15735	10801	16688
Largest client cluster (clients)	1477	803	781	886	10609
Smallest client cluster (clients)	1	1	1	1	1

Table 16: Experimental results of client clustering on EW3 server log.

from 1 to 704 clients, while the number of requests issued from within each client cluster varies from 1 to 76,218 (Table 19).

- **EW3-c1** EW3-c1 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 1,496,408 requests in the server log, issued by 160,020 clients accessing 618 unique URLs at the server. We detected all of these clients as 18,571 client clusters. The size of client clusters varies from 1 to 6,989 clients, while the number of requests issued from within each client cluster varies from 1 to 53,217 (Table 20).
- **EW3-c2** EW3-c2 log is a 31-day Web server log taken from July 1, 1999 to July 31, 1999. There are a total of 148,859 requests in the server log, issued by 41,456 clients accessing 769 unique URLs at the server. We detected all of these clients as 16,286 client clusters. The size of client clusters varies from 1 to 824 clients, while the number of requests issued from within each client cluster varies from 1 to 2,410 (Table 21).
- **EW3-w** EW3-w log is a 6-month Web server log taken from June 26, 1999 to December 20, 1999. We divided it into 6 sub-logs, one for each month. The detailed information is provided in Table 22.

To sum up, our experimental results show that all clients in the various EW3 server logs are able to be detected as client clusters using our method. We will provide information on client cluster distributions later.

- **Sun server log**

  Sun log is a 9-day Web server log taken from September 30, 1997 to October 9, 1997. A summary of the client cluster detection result is provided in Table 23. There are a total of 13,871,352 requests in the server log, issued by 219,528 clients and access 116,274 unique URLs at the server. We detected all of these clients as 33,468 client clusters. The size of client clusters varies from 1 to 2,529 clients, the number of requests issued from within each client cluster varies from 1 to 693,955, and clusters access anywhere from 1 to 33,594 unique URLs.

  We plot the client cluster distributions in Figures 13 and 14. Figure 13(a) shows the cumulative distribution of the number of clients in client clusters. We observe that, although the largest client cluster contains 2,536 clients, more than 95% of the client clusters contain less than 100 clients. The cumulative distribution of

Date	July 1, 1999 - July 31, 1999
Period (days)	31
Total number of requests	1199276
Total number of unique URLs	683
Total number of unique clients	21519
Number of undetected clients	0
Total number of client clusters	7754
Largest client cluster	359 clients (19409 requests)
Smallest client cluster	1 client (1 request)

Table 17: Experimental results of client cluster detection on the EW3 server log: EW3-a1.

Date	(1999)	Jun 27 - Jul 26	Jul 27 - Aug 26	Aug 27 - Sep 26	Oct 1 - Oct 31	Nov 6 - Nov 30	Dec 1 - Dec 20
Period	(days)	30	31	31	31	25	20
Total number of requests		1357623	1645975	2180029	2877528	2223590	1847913
Total number of unique URLs		11475	10462	8569	10005	10111	10556
Total number of unique clients		33675	35249	44831	54882	46579	38913
Number of undetected clients		0	0	0	38	82	110
Total number of client clusters		12465	12880	14949	17826	15206	13620
Largest client cluster	(clients)	634	932	760	1001	859	717
	(requests)	16513	26979	25648	42989	27635	24784
Smallest client cluster	(clients)	1	1	1	1	1	1
	(requests)	1	1	1	1	1	1

Table 18: Experimental results of client cluster detection on the EW3-a2 server log over 6 months.

Date	July 1, 1999 - July 31, 1999
Period (days)	31
Total number of requests	1908761
Total number of unique URLs	7772
Total number of unique clients	40156
Number of undetected clients	0
Total number of client clusters	14895
Largest client cluster	704 clients (25328 requests)
Smallest client cluster	1 client (1 request)

Table 19: Experimental results of client cluster detection on the EW3 server log: EW3-b.

Date	July 1, 1999 - July 31, 1999
Period (days)	31
Total number of requests	1496408
Total number of unique URLs	618
Total number of unique clients	160020
Number of undetected clients	0
Total number of client clusters	18571
Largest client cluster	6989 clients (53217 requests)
Smallest client cluster	1 client (1 request)

Table 20: Experimental results of client cluster detection on the EW3 server log: EW3-c1.

Date	July 1, 1999 - July 31, 1999
Period (days)	31
Total number of requests	148859
Total number of unique URLs	769
Total number of unique clients	41456
Number of undetected clients	0
Total number of client clusters	16286
Largest client cluster	824 clients (2410 requests)
Smallest client cluster	1 client (1 request)

Table 21: Experimental results of client cluster detection on the EW3 server log: EW3-c2.

Date	(1999)	Jun 27 - Jul 26	Jul 27 - Aug 26	Aug 27 - Sep 15	Oct 1 - Oct 31	Nov 6 - Nov 30	Dec 1 - Dec 20
Period	(days)	30	31	20	31	25	20
Total number of requests		575451	3753571	2405524	2297309	4506717	2573251
Total number of unique URLs		1705	19210	8569	14982	2069	1526
Total number of unique clients		3659	16202	10587	20788	20165	13452
Number of undetected clients		0	0	0	10	27	36
Total number of client clusters		1517	6187	4391	7321	6863	5111
Largest client cluster	(clients)	389	523	520	574	593	573
	(requests)	39021	145010	103531	45810	114206	82788
Smallest client cluster	(clients)	1	1	1	1	1	1
	(requests)	1	1	1	1	1	1

Table 22: Experimental results of client cluster detection on the EW3-w server log over 6 months.

Date	September 30, 1997 - Oct 9, 1997
Length of period (days)	10
Total number of requests	13871352
Total number of unique URLs	116274
Total number of unique clients	219528
Number of undetected clients	0
Total number of client clusters	33468
Largest client cluster	2529 clients (119231 requests)
Smallest client cluster	1 client (1 request)
Client cluster URL access	1 URL (0.000009% of total) - 33594 URLs (0.288921% of total)

Table 23: Experimental results of client cluster detection on the Sun server log.

requests issued in client clusters is shown in Figure 13(b), which is more heavy-tailed than that of the number of clients in client clusters as shown in Figure 13(a). Around 90% of the client clusters issued less than 1,000 requests. Only few client clusters are very *busy*, issuing up to a maximum of 693,955 requests. Figures 13(c) and (d) show the distributions of number of clients in client clusters (in the reverse order of number of clients) and number of requests issued from within client clusters (in reverse order of number of requests issued), respectively. Again the distribution of number of requests issued from within client clusters is more heavy-tailed than that of number of clients in client clusters.

We plot the distributions of number requests issued from within client clusters and number of unique URLs accessed from within client clusters (in the reverse order of number of clients) in Figures 13(e) and (g), respectively. Larger client clusters issue more requests and access more URLs than smaller client cluster. However, there are a number of relatively small client clusters which issue significant amount of requests ($\sim$ 5% of the total) and/or access a big portion of URLs ($\sim$ 30% of the total) at the server. Finally, we plot the distributions of number of clients in client clusters and number of URLs accessed from within client clusters (in the reverse order of number of requests issued) in Figures 13(f) and (h), respectively.

In summary, our experimental results show that we can group more than 99.9% of the clients into clusters, with very few clients not clusterable (i.e., no network prefixes in our prefix table matches the client IP addresses) due to the lack of proper prefix/netmask information in the routing table snapshots. This is fixed in the self-correction and adaptation stage of our approach (discussed in Section 3.2.6). Additionally, all observations on client clustering made on the Nagano server log also apply to every one of the various other server logs we experimented with.

3.2.4 Validation

We now validate our approach of detecting client clusters. A client cluster may be mis-detected in one of two ways:

1. A client cluster contains more than one network;
2. A network is divided into several client clusters.

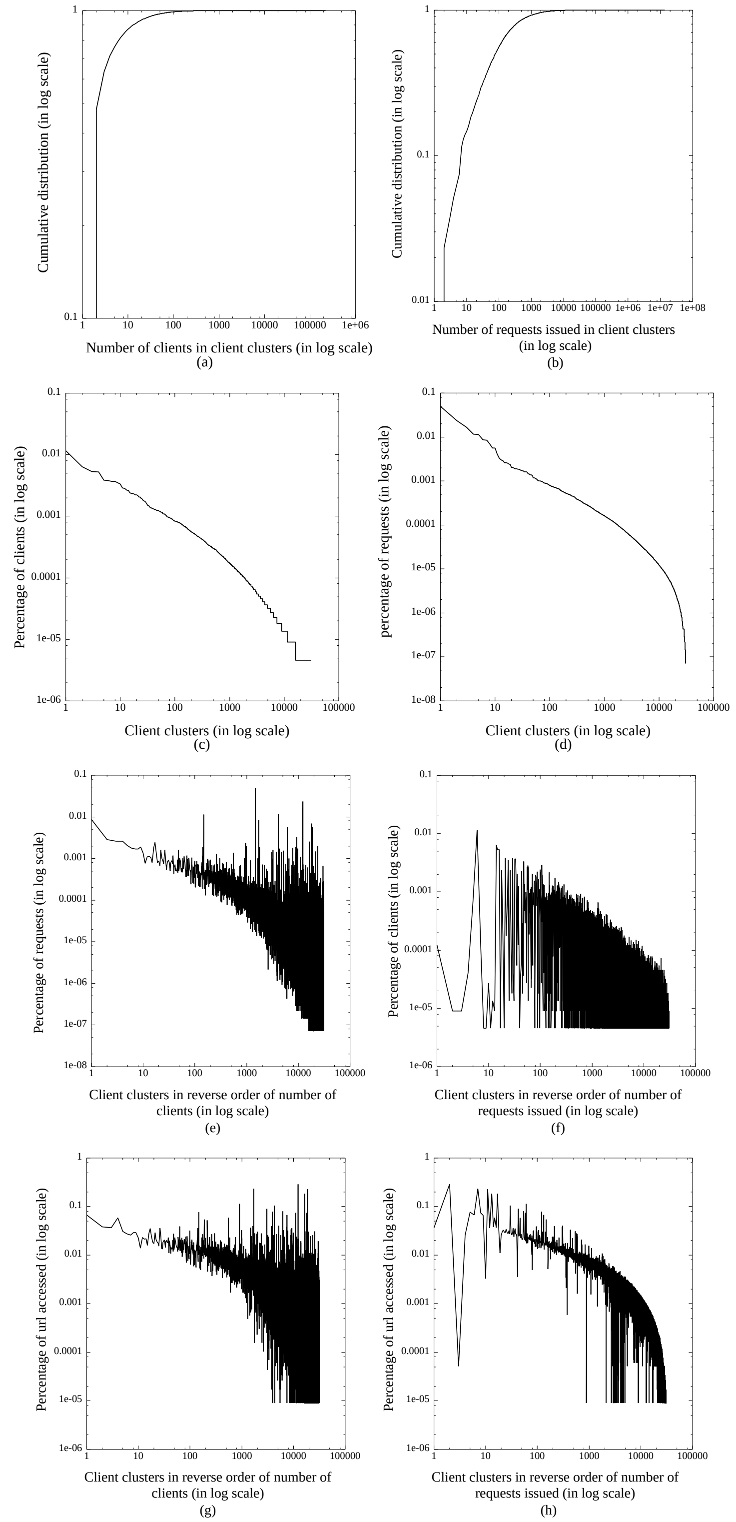

Figure 13: The client cluster distribution for the Sun server log (both x axis and y axis are in log scale): (a) is the cumulative distribution of number of clients in client clusters; (b) is the cumulative distribution of number of requests issued from within client clusters; (c), (e), and (g) are the distributions of number of clients in client clusters, number of requests issued from within client clusters and number of URLs accessed from within client clusters (in reverse order of number of clients), respectively; (d), (f) and (h) are the distributions of number of requests issued from within client clusters, number of clients in client clusters and number of URLs accessed from within client clusters (in reverse order of number of requests), respectively.

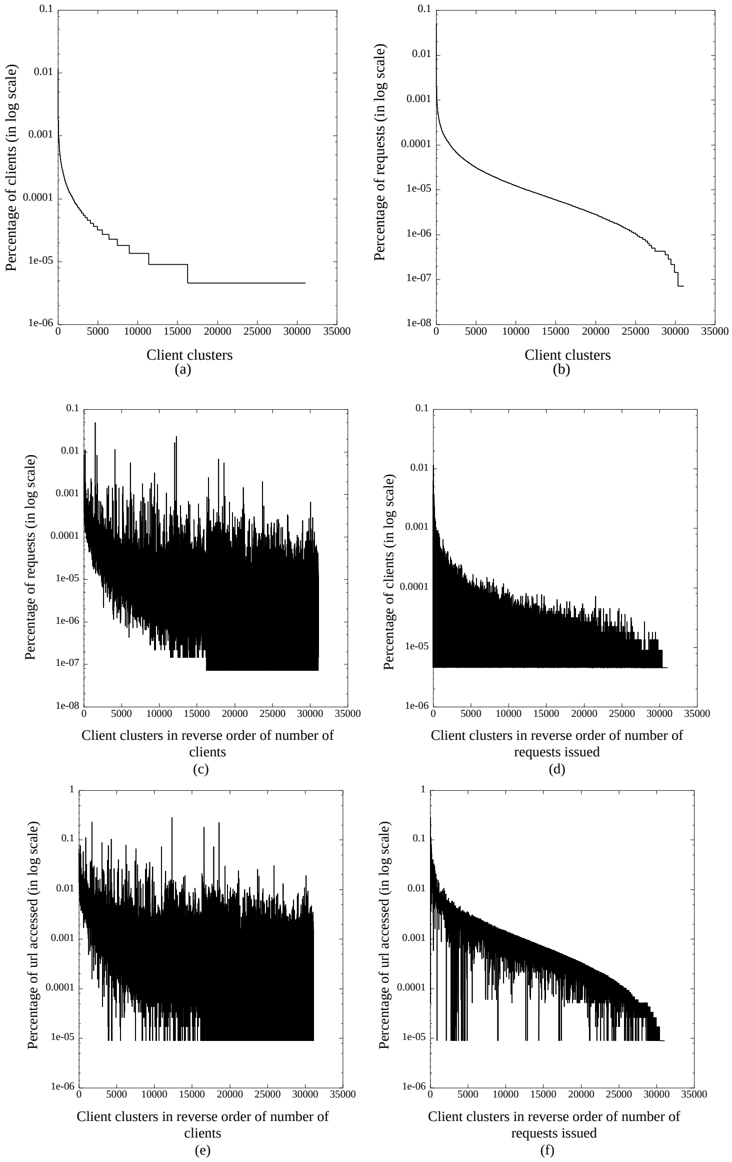

Figure 14: The client cluster distribution for the Sun server log (y axis is in log scale): (a) - (f) are the re-plotted figures of Figure 13 (c) - (h), respectively, to clarify how client cluster distributions vary along number of clients in client clusters and number of requests issued from within client clusters.

Server log	Apache	Nagano	Sun
Number of sampled client clusters	382	111	365
Number of sampled clients	2437	307	2217
Number of *nslookup* reachable clients	1470	172	1116
Number of mis-detected client clusters	34	5	22
Number of mis-detected non-US client clusters	21	3	15
Prefix length range	8 - 29	8 - 28	8 - 29
Number of client clusters of prefix length 24	191	57	186

Table 24: Client clustering validation using *nslookup*, of Apache, Nagano, and Sun logs.

While earlier approaches may mis-detect client clusters in either case, most of them fall into the latter one due to the fact that there are more networks of length less than 24 than the ones of length greater than 24 (see Figure 1 and Table 3 in Section 2). Because our approach uses routing table information to detect client clusters, client cluster mis-detecting mainly falls into the former case.

To validate our approach, we take samples (1%) from client clusters of various Web server logs. Two network tools are used to verify the results of client clustering: reverse *nslookup* and *traceroute*. The *nslookup* command queries Internet domain name servers for information about hosts:

```
% nslookup 209.68.10.19
Server: www.whitehouse.gov
Address: 198.137.240.91
Name: tyrant.com
Address: 209.68.10.19
```

Our judgment is made on the assumption that all clients in the same network should have the same suffix in their names, and vice versa[10]. For example, *macbeth.cs.wits.ac.za* and *macabre.cs.wits.ac.za* are in the same network and their names share the same suffix *cs.wits.ac.za*. Furthermore, none of the clients in any other network has the same name suffix as *cs.wits.ac.za*. We label a client cluster to be *incorrect* if either there is *even one* client that does not share the same suffix with others or there is more than one client cluster which share the same name suffix. We run *nslookup* on each client IP address in the sampled client cluster and perform suffix matching on the name of clients belonging to the sampled client cluster. The results of validation are provided in Tables 24– 27. More than 90% of the client clusters are constructed correctly (with our conservative notion of correctness) while only around 50% client clusters can be detected correctly using the simple approach. For instance, among the 111 sampled client clusters in the Nagano server log, 5 client clusters are mis-detected by our method. Each of them contains more than one network. Around 95.4% client clusters are detected correctly. However, only 57 of the total 111 client clusters have prefix length of 24, i.e., only 48.6% client clusters are detected correctly using the simple approach.

One reason for mis-detection in our approach is the existence of suspected national gateways/routers on the Internet (e.g., Croatia, France, Japan), which are recognizable by the client name. In such cases, additional information about the clients/networks behind the national gateways/routers are not available in the routing table. Also, route aggregation in the routing table may cause mis-detection in our approach, and may account for approximately 50% of the mis-detections as observed from Tables 24– 27. However, it is hard to further quantify its actual effect on client clustering.

[10] We make this assumption in validating client cluster detection. While this assumption doesn't hold for some servers, it's reasonable for clients.

Server log	EW3-a1	EW3-b	EW3-c1	EW3-c2
Number of sampled client clusters	86	167	203	177
Number of sampled clients	227	328	495	489
Number of *nslookup* reachable clients	180	271	313	345
Number of mis-detected client clusters	8	5	4	14
Number of mis-detected non-US client clusters	4	1	0	5
Prefix length range	8 - 27	8 - 28	8 - 28	8 - 31
Number of client clusters of prefix length 24	23	54	77	57

Table 25: Client cluster validation using *nslookup*, of the EW3 logs.

Server log	Jul	Aug	Sep	Oct	Nov	Dec
Number of sampled client clusters	140	143	167	193	167	149
Number of sampled clients	417	381	332	558	323	505
Number of *nslookup* reachable clients	348	306	234	350	347	433
Number of mis-detected client clusters	7	4	6	6	4	6
Number of mis-detected non-US client clusters	1	0	1	4	0	2
Prefix length range	8 - 29	8 - 29	8 - 29	8 - 31	8 - 29	8 - 29
Number of client clusters of prefix length 24	40	42	70	76	55	53

Table 26: Client clustering validation using *nslookup*, of the EW3-a2 log over 6 months

Server log	Jul	Aug	Sep	Oct	Nov	Dec
Number of sampled client clusters	13	68	47	83	80	53
Number of sampled clients	417	381	332	468	323	505
Number of *nslookup* reachable clients	10	129	131	358	177	75
Number of mis-detected client clusters	0	1	5	6	2	1
Number of mis-detected non-US client clusters	0	1	0	2	1	0
Prefix length range	8 - 25	8 - 28	8 - 26	8 - 28	8 - 29	8 - 29
Number of client clusters of prefix length 24	4	23	19	27	27	20

Table 27: Client cluster validation using *nslookup*, of the EW3-w log over 6 months

As Tables 24– 27 show, around 50% of clients are not locatable via *nslookup* in our experiments. This is true for all the logs and varies only very slightly from time to time with repeated experiments over a period of time. There are several possible reasons for this:

1. If an organization operates a firewall on its network, then the DNS server will typically not give out the name of the machines behind the firewall.
2. If the machines on the local network acquires dynamic IP addresses via a DHCP server, the IP addresses do not have a one-to-one mapping to a machine on the network. In such cases, the DNS server will not have the registration record for the dynamic addresses.
3. It is also possible that an ISP may not register any names for their customers. In such cases, *traceroute* can be used to get further information.

An alternative way to validate client cluster detection results is *traceroute*, which attempts to trace the route an IP packet follows to an Internet host by launching UDP probe packets with a small maximum time-to-live (Max_ttl variable), and then listening for an ICMP TIME_EXCEEDED response from gateways along the way. Probes are started with a Max_ttl value of one hop, which is increased one hop at a time until an ICMP PORT_UNREACHABLE message is returned. The ICMP PORT_UNREACHABLE message indicates either that the host has been located or the command has reached the maximum number of hops allowed for the trace. By running *traceroute* on each client, a path towards the designated client is discovered. If two clients sit in the same network, then it is very likely that packets routed to them will traverse paths sharing the same suffix, and vice versa. *Traceroute* imposes more traffic load on the Internet and takes more time to discover routes, but yields more information on clients (such as RTT and hop count). We are more interested in either the name of the client or the last few hops on the path towards the client. For faster path resolution and to reduce the traffic load imposed on the Internet, we implemented an optimized *traceroute* (in Linux Red Hat 6.0) with two improvements:

- Instead of having a fixed number (q) of probes sent for each time-to-live (*ttl*) value no matter what ICMP replies are returned, we send only one probe for each *ttl* value. If the first ICMP reply doesn't provide proper information, we send out the second probe, continuing until we get the proper information or the number of probes reaches q. Thus we send out $\leq q$ probes for each *ttl*.
- The initial value of *ttl*, unlike the default one, does not necessarily have to be 1. In fact, we can set the initial value of $ttl = Max\_ttl$ (we set $Max\_ttl = 30$). As such, we only sends one probe with *ttl* of Max_ttl. If the destination is less than max_ttl hops away, *traceroute* returns the destination IP address, name (if available), and round trip time (RTT). Interestingly enough, around 50% of clients can be resolved in this way, which is consistent with our observation on *nslookup* results. This is because *traceroute* doesn't get a ICMP reply packet from the designated router which is either unreachable or unwilling to give out the required information due to firewalls.

The pseudo code of the optimized *traceroute* is shown in Table 28. We define a client to be unresolved if the ICMP reply packet doesn't contain proper information. In our experiments, we set the constant $c = 30$ and $q = 3$. We send probes in serial[11]. The basic idea is that we do a binary search to find the optimal Max_ttl (i.e., the distance to the designated client) instead of the sequential search in the original *traceroute*. We now give an

[11] Although we can send probes in parallel, which makes the resolving process faster than serial probing does, it will impose much more traffic on the network. Because half of the clients can be resolved by sending one probe, we don't gain a significant saving on resolving time comparing to the amount of extra traffic we imposed on the network. We choose to do probing in serial.

estimation on how much time and bandwidth we saved on optimized *traceroute*. Assume the average path length is 15. The original *traceroute* will send out $15 \times 3 \approx 45$ probes and the waiting time (measured by number of hops the probes traverse) is $3 \times (1 + 2 + ... + 15) \approx 360$ hops. We observed that 50% clients can be resolved by sending one probe with $ttl = 30$. We also found that almost all clients are less than 30 hops away. For the purpose of performance estimation, we assume that all the clients are less than 30 hops alway and we only need to do binary search for $ttl < 30$. We further assume that, during binary search, the path length towards clients are randomly distributed with an average of 15 (i.e. average *ttl* is 15) and the average number of probes need to be sent for each *ttl* is 1.5. Therefore, the average number of probes sent during binary search is $log_2 30 \approx 5$. The number of probes we need to send for a client is $50\% \times 1 + 50\% \times 5 \times 1.5 \approx 4.25$. The waiting time using the optimized *traceroute* would be $50\% \times 30 + 50\% \times 5 \times 1.5 \times 15 \approx 71.25$ hops. Thus, we save 90% probes and 80% of the waiting time.

The advantage of using the optimized *traceoute* on validation is that we are able to resolve the name and, if it fails, the path towards the designated client. The resolvability (either name or path) on clients is improved from 50% to 100% in our applications. In addition, the extra traffic imposed on network and time spent on resolving clients have been significantly reduced by our optimized *traceroute*. The time consumed by sending one probe in the optimized *traceroute* is about the same as that by DNS *nslookup*.

We run our optimized *traceroute* on 1% of the sampled client clusters. After resolving all the sampled clients, we applied suffix matching on either the name of the client or on the path towards the client if the client name was not available (as mentioned earlier, we use either the client's name if it was resolved or the last few hops on the path towards the client for validating a cluster). Preliminary results (Tables 29– 32) show that 90% of the client clusters are detected correctly. Moreover, we use the validation information to improve the applicability and accuracy of the cluster detection results. We will discuss this in detail in Section 3.2.6.

The quantification of the error ratio of our approach is very conservative for validation purposes. The validation results give a *worst-case* bound on the accuracy. The actual error ratio should be significantly smaller than what we measured here. Many real applications will be tolerant to a certain degree of inaccuracy and an alternative way to validate is to set a threshold (say 5%) and selectively sample clients. For example, if 95% of the clients inside the cluster are correctly detected, we could consider this cluster to be correct. This selective sampling can be performed in either a client-based or a request-based manner depending on the application's criteria. We plan to study selective sampling techniques in our future work.

Given that the accuracy of the simple approach and our approach are 50% and 90%, respectively, we compare the client cluster distributions of the Nagano server log obtained by the two approaches to illustrate the effect of cluster mis-detection. The number of client clusters detected by our approach is 9,853 as compared to 23,523 of the simple approach. The largest client cluster detected by our approach contains 1,343 hosts (issuing 134,963 requests or 1.15%). However, there are only 63 hosts in the largest client cluster detected by the simple approach (issuing 9,662 requests or 0.08%). We further plot client cluster distributions of the Nagano server log obtained by our approach (as dotted curves) and by the simple approach (as solid curves) in Figure 15. Figure 15(a) and (b) show the distributions of number of clients in client clusters in reverse order of number of clients and in reverse order of number of requests. We observe that the number of client clusters detected by the simple approach is much larger than that of our approach. Note that the maximum number of clients in a client cluster is 256 because of its assumption of prefix length of 24. The average size of client clusters detected by the simple approach is smaller than that of our approach, as is the variance of the size of client clusters. Figure 15(c) and (d) show the distribution of number of requests issued from within client clusters in reverse order of number of requests and in reverse order of number of clients. The average number of requests issued from within client clusters detected by the simple approach is smaller than that of our approach. The significant differences in the

```
Max_ttl = c;
for each client IP address {
    ttl = Max_ttl;
    send one probe packet;
    save ICMP reply;
    while (unresolved) {
        if (Max_ttl is too short) {
            Max_ttl *= 2;
            ttl = Max_ttl;
            n = 0;
            while (no reply and n < q) {
                send one probe packet;
                n += 1;
            }
            save ICMP reply;
        }
        if (Max_ttl is too long) {
            Max_ttl /= 2;
            ttl = Max_ttl;
            n = 0;
            while (no reply and n < q) {
                send one probe packet;
                n += 1;
            }
            save ICMP reply;
        }
    }
    hop_count = ttl;
    if (path needed) {
        for each missing time-to-live value t in [1..hop_count] {
            Max_ttl = t;
            ttl = t;
            send one probe packet;
            save ICMP reply;
        }
    }
}
```

Table 28: The pseudo code of the optimized *traceroute*.

Server log	Apache	Nagano	Sun
Number of sampled client clusters	382	111	365
Number of sampled clients	2437	307	2217
Number of mis-detected client clusters	35	12	33
Number of mis-detected non-US client clusters	20	7	28

Table 29: Client cluster validation using *traceroute*, of Apache, Nagano, and Sun logs.

Server log	EW3-a1	EW3-b	EW3-c1	EW3-c2
Number of sampled client clusters	86	167	203	177
Number of sampled clients	227	328	495	489
Number of mis-clustered client clusters	9	12	22	19
Number of mis-clustered non-US client clusters	6	11	7	11

Table 30: Client clustering validation using *traceroute*, of the EW3 logs.

Server log	Jul	Aug	Sep	Oct	Nov	Dec
Number of sampled client clusters	140	143	167	193	167	149
Number of sampled clients	417	381	332	558	323	505
Number of mis-detected client clusters	11	12	14	16	12	12
Number of mis-detected non-US client clusters	7	9	11	11	7	9

Table 31: Client cluster validation using *traceroute*, of the EW3-a2 log over 6 months

Server log	Jul	Aug	Sep	Oct	Nov	Dec
Number of sampled client clusters	13	68	47	83	80	53
Number of sampled clients	417	381	332	468	323	505
Number of mis-detected client clusters	0	6	5	9	8	4
Number of mis-detected non-US client clusters	0	3	1	5	5	0

Table 32: Client cluster validation using *traceroute*, of the EW3-w log over 6 months

		Network-aware approach	Naive approach
Total number of client clusters		9853	23523
Largest client cluster	(clients)	1343	63
	(requests)	134963	9662
Smallest client cluster	(clients)	1	1
	(requests)	1	9

Table 33: Comparison of the client cluster distributions of Nagano server log obtained by our approach and simple approach.

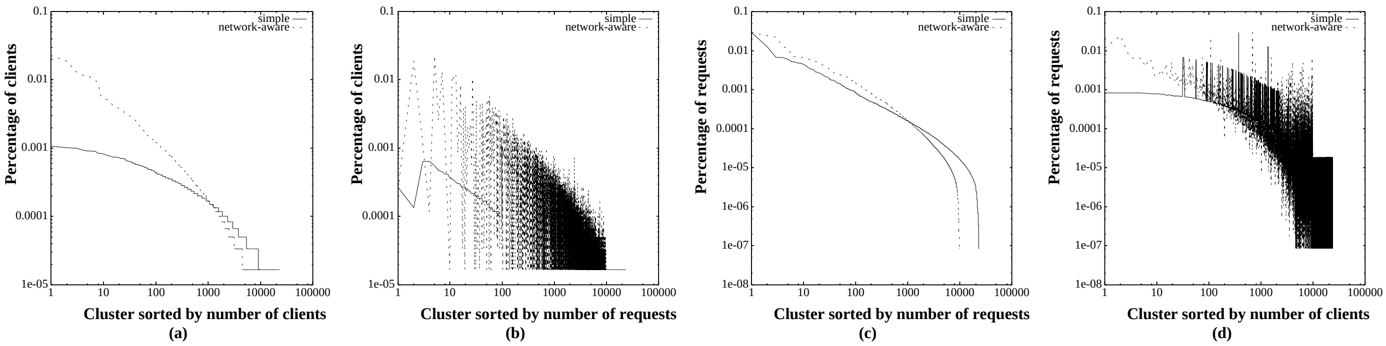

Figure 15: Comparison of the client cluster distributions of the Nagano server log obtained by our approach and the simple approach (both x axis and y axis are in log scale): (a) and (b) are the distributions of number of clients in client clusters in reverse order of number of clients and number of requests, respectively; (c) and (d) are the distributions of number of requests issued from within client clusters in reverse order of number of clients and number of requests, respectively.

client cluster distributions obtained by our approach and the simple approach implies that the simulation results based on different client cluster detection methods may vary a lot.

3.2.5 Effect of BGP dynamics on client cluster detection

An update of BGP routing table is triggered by changes in network reachability and topology, as well as policy changes. BGP dynamics is a well-known phenomenon which affects the performance of Internet applications [12] . We examine the impact of BGP dynamics on client cluster detection results. The experiments were conducted on a timescale of days. We downloaded routing table snapshots daily. In our measurements, we define the *dynamic prefix set* to be the set of prefixes which might change during the entire testing period, i.e., the set of prefixes which are not in the intersection of prefixes of all the routing tables we obtained over the entire testing period. We further define the *maximum effect* to be the size of dynamic prefix set. This is the upper bound of the effect of BGP dynamics on our client cluster detection. The actual effect of BGP dynamics from day to day would be smaller.

We use AADS, PACBELL, SINGAREN, and VBNS routing tables to illustrate this (Tables 34– 37). We show the measurement results for 1, 4, 7, and 14 day periods on Apache, Ew3, Nagano, and Sun server logs. For example, the AADS routing table entries vary between 16,595 and 17,288 on those days, among which the number of prefixes (*maximum effect*) that are not in the intersection of the routing tables on those days vary between 711 and 1,404. The number of prefixes used to detect client clusters in the Apache server logs on those days vary from 3,932 to 4,035, and the corresponding maximum effects vary from 124 to 227. There are 2,869 client clusters issuing a large number of requests, among which between 605 and 614 client clusters are detected

Period (days)	0	1	4	7	14
AADS prefix	16595	16669	16704	16792	17288
Maximum effect	711	785	820	908	1404
Apache prefix (total 35563)	3932	3935	3929	3950	4035
Maximum effect	124	127	121	142	227
Apache big prefix (total 2869)	605	603	603	605	614
Maximum effect	22	20	20	22	31
EW3 prefix (total 24921)	2592	2588	2582	2604	2682
Maximum effect	75	71	65	87	165
EW3 big prefix (total 2062)	385	386	388	390	412
Maximum effect	4	5	7	9	31
Nagano prefix (total 9853)	663	665	663	673	726
Maximum effect	22	24	22	32	85
Nagano big prefix (total 717)	93	94	93	93	105
Maximum effect	2	3	2	2	14
Sun prefix (total 33468)	3646	3651	3640	3669	3756
Maximum effect	124	129	118	147	234
Sun big prefix (total 2536)	527	525	529	530	542
Maximum effect	15	13	17	18	30

Table 34: The effect of AADS dynamics on client cluster detecting.

Period (days)	0	1	4	7	14
PACBELL prefix	25532	25642	25359	25198	25566
Maximum effect	1746	1856	1573	1412	1780
Apache prefix (total 35563)	6435	6485	6416	6389	6557
Maximum effect	204	254	285	258	226
Apache big prefix (total 2869)	1091	1099	1092	1092	1119
Maximum effect	34	32	35	35	32
EW3 prefix (total 24921)	3895	3907	3874	3857	3985
Maximum effect	140	122	119	132	128
EW3 big prefix (total 2062)	516	513	516	541	589
Maximum effect	14	15	16	17	14
Nagano prefix (total 9853)	954	953	951	964	1047
Maximum effect	37	36	34	37	35
Nagano big prefix (total 717)	132	132	129	134	144
Maximum effect	4	3	2	5	5
Sun prefix (total 33468)	5960	6007	5950	5928	6017
Maximum effect	227	204	217	225	204
Sun big prefix (total 2536)	851	857	854	853	884
Maximum effect	21	27	24	23	24

Table 35: The effect of PACBELL dynamics on client cluster detecting.

using the network prefixes in AADS routing table. The maximum effects vary from 22 to 31. We observe that overall BGP dynamics affects less than 3% of client clusters. This result holds across all the routing tables and all the Web server logs. Therefore, we believe that, although the BGP routing table changes dynamically according to the network environment, its impact on client cluster detection is very minor, and fixed in the self-correction and adaptation process (discussed next).

Period (days)	0	1	4	7	14
SINGAREN prefix	67686	67997	67950	68226	68293
Maximum effect	1958	2269	2222	2498	2565
Apache prefix (total 35563)	15330	15320	15275	15296	15262
Maximum effect	217	207	162	183	149
Apache big prefix (total 2869)	2268	2266	2260	2261	2260
Maximum effect	21	19	13	14	13
EW3 prefix (total 24921)	9931	9912	9906	9906	9888
Maximum effect	109	90	84	84	66
EW3 big prefix (total 2062)	1309	1309	1309	1309	1308
Maximum effect	3	3	3	3	2
Nagano prefix (total 9853)	3730	3724	3718	3718	3712
Maximum effect	41	35	29	29	23
Nagano big prefix (total 717)	471	471	469	469	469
Maximum effect	2	2	0	0	0
Sun prefix (total 33468)	14412	14397	14341	14374	14333
Maximum effect	213	198	142	175	134
Sun big prefix (total 2536)	1956	1956	1951	1950	1949
Maximum effect	12	12	7	6	5

Table 36: The effect of SINGAREN dynamics on client cluster detecting.

Period (days)	0	1	4	7	14
VBNS prefix	1855	1855	1855	1855	1855
Maximum effect	1	1	1	1	1
Apache prefix (total 35563)	675	675	675	675	675
Maximum effect	0	0	0	0	0
Apache big prefix (total 2869)	288	288	288	288	288
Maximum effect	0	0	0	0	0
EW3 prefix (total 24921)	461	461	461	461	461
Maximum effect	0	0	0	0	0
EW3 big prefix (total 2062)	159	159	159	159	159
Maximum effect	0	0	0	0	0
Nagano prefix (total 9853)	164	164	164	164	164
Maximum effect	0	0	0	0	0
Nagano big prefix (total 717)	45	45	45	45	45
Maximum effect	0	0	0	0	0
Sun prefix (total 33468)	698	698	698	698	698
Maximum effect	0	0	0	0	0
Sun big prefix (total 2536)	285	285	285	285	285
Maximum effect	0	0	0	0	0

Table 37: The effect of VBNS dynamics on client cluster detecting.

Period	Number of requests issued	Number of URLs accessed
00:00:00 - 05:59:59	2170801	14415
06:00:00 - 11:59:59	2387396	13665
12:00:00 - 17:59:59	3719796	17942
18:00:00 - 23:59:59	3387720	16090

Table 38: The client access pattern information of the Nagano server log.

3.2.6 Self-correction and adaptation

Besides validating the client cluster results, we use *traceroute* for two other purposes. In Sections 3.2.2 and 3.2.3, we showed that more than 99.9% clients in the Web server logs can be clustered using our method. We have also shown in Section 3.2.4 that both *nslookup* and *traceroute* validation results show that the accuracy of our cluster detecting method is around 90%. Periodic *traceroute* results on sampled clients can be used to further improve the applicability (i.e. detect the undetected clients) and accuracy (i.e., correct the mis-detections) of cluster detection. The periodic sampling can also be used to make client cluster detecting adaptive to network dynamics (i.e., changes of IP address allocations and network topology). For the purpose of cluster detecting undetected clients ($\sim$ 0.1%), we first consider each individual undetected client to be a single client cluster. Then we merge them into bigger client clusters gradually according to *traceroute* sampling information. Self-correction and adaptation is also very important to generate client clusters using real-time routing information and produce real-time client cluster detecting results. By real-time cluster detecting we mean application of cluster detecting techniques to very recent server log data (within the last few minutes).

We consider two cases in the self-correction and adaptation process:

- If there is more than one cluster which belongs to the same network, we merge them into one big cluster and the network prefix and netmask will be recomputed accordingly. For example, if clients in cluster A have the same name suffix as clients in cluster B, then we consider clusters A and B to belong to the same network.
- If there is a cluster which contains clients belonging to more than one network, we partition the cluster into several based on the *traceroute* sampling results.

3.2.7 Other issues

We examine three related issues: partitioning a session into time periods, detecting server clusters in client and proxy logs, and clustering client clusters themselves.

- **Partitioning sessions**

  To examine how requests issued and unique URLs accessed from within each client cluster vary during different time periods, we partition the Nagano server log into four 6-hour sessions. The client access pattern information of each session is provided in Table 38. Detailed information is shown in Figures 16, 17, 18, and 19. Although the first two sessions are less busy than the last two ones, all of them show similar patterns in terms of both the number of requests issued and number of unique URLs accessed by each client cluster. The observations on client cluster distributions obtained from the entire server log still hold for each session indicating that simulations on a sample of server logs might suffice.

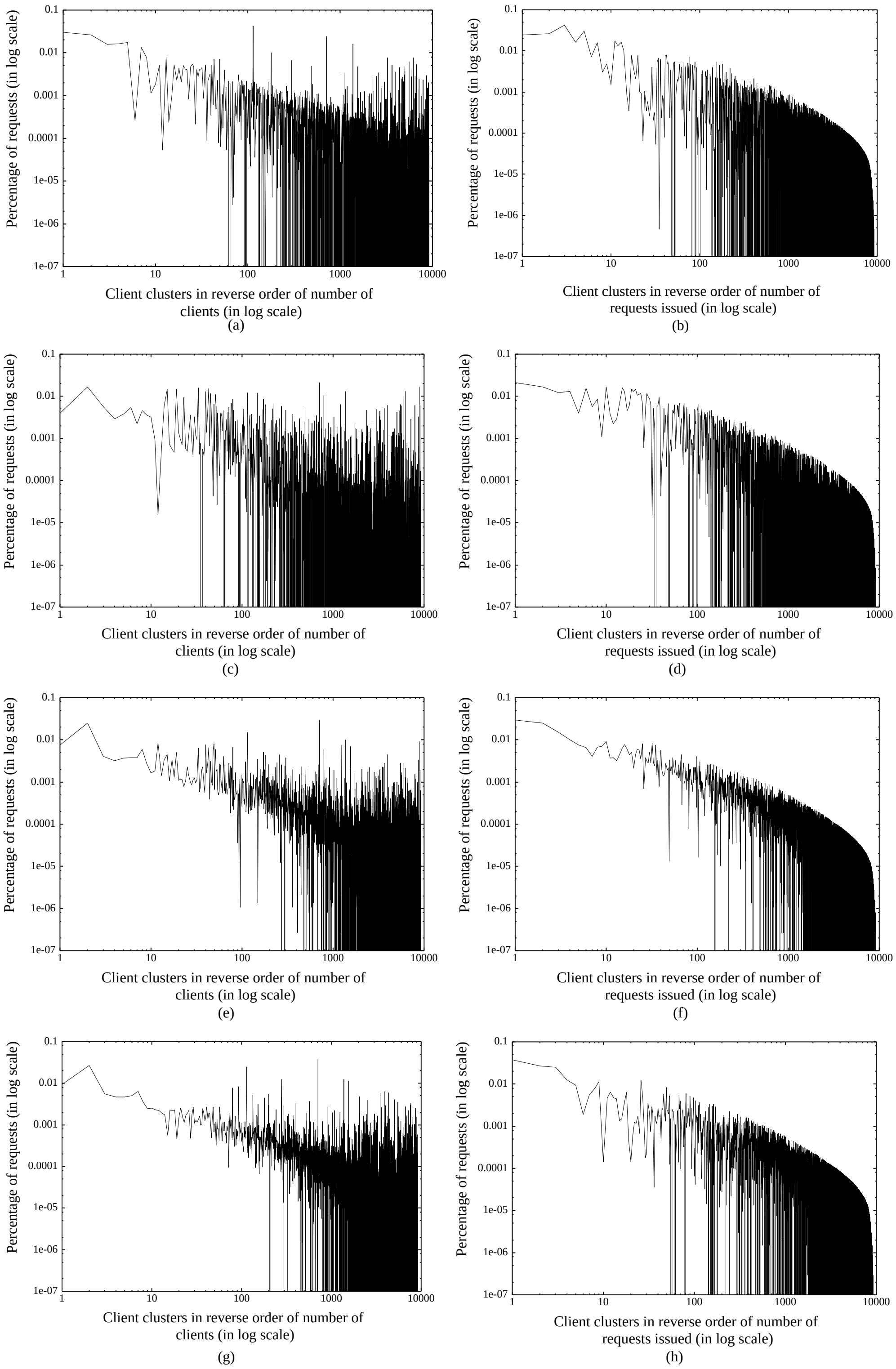

Figure 16: The client cluster requests distribution for the 4 sessions of the Nagano server log (both x axis and y axis are in log scale): (a), (c), (e), and (g) are in the reverse order of number of clients in client clusters for time session 1, 2, 3, and 4; (b), (d), (f), and (h) are in the reverse order of number of requests issued from within client clusters for time sessions 1, 2, 3, and 4.

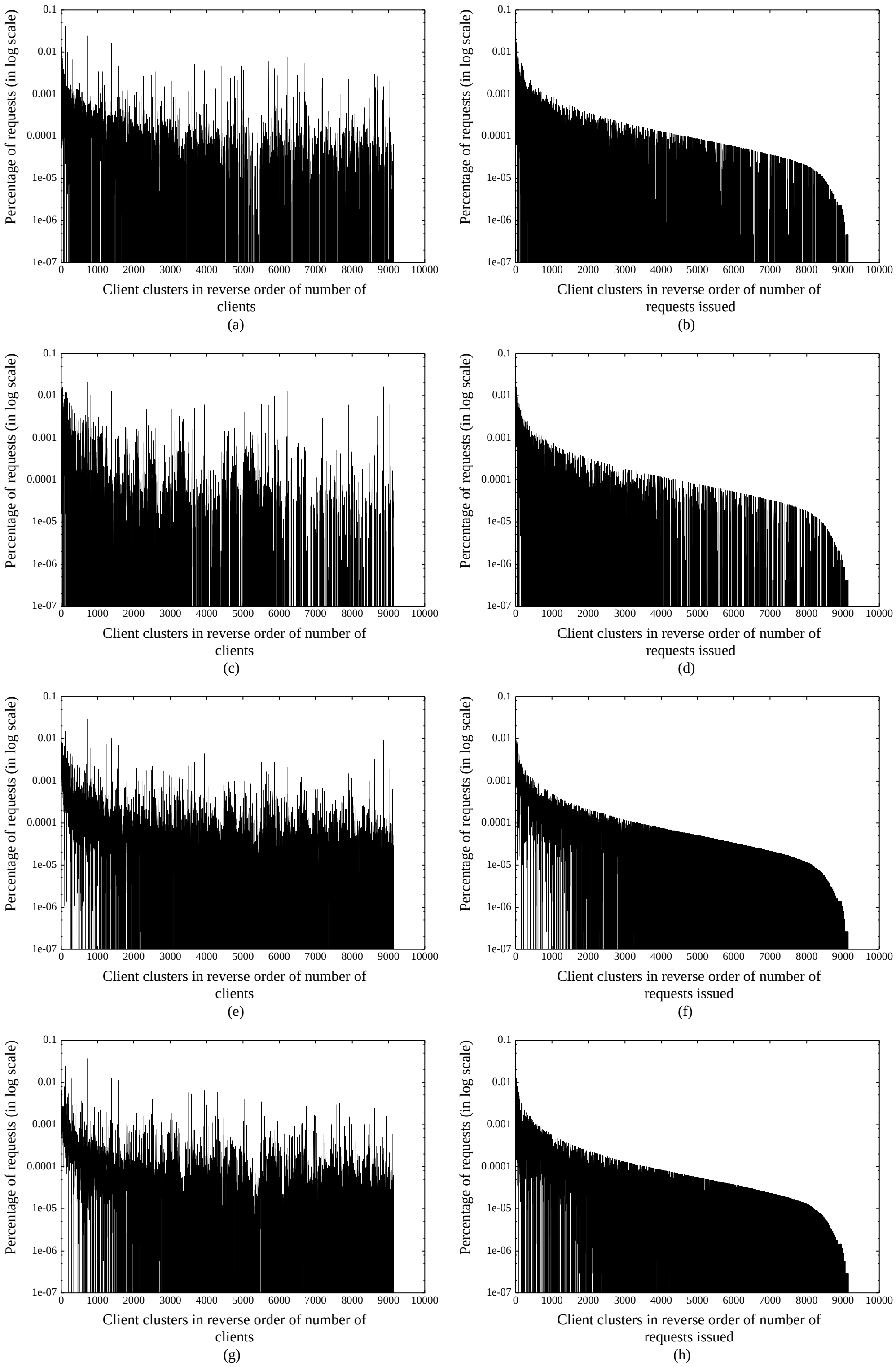

Figure 17: The client cluster requests distribution for the 4 sessions of the Nagano server log (only y axis is in log scale). Note the similarity in number of requests issued and unique URLs accessed by each client cluster across the sessions.

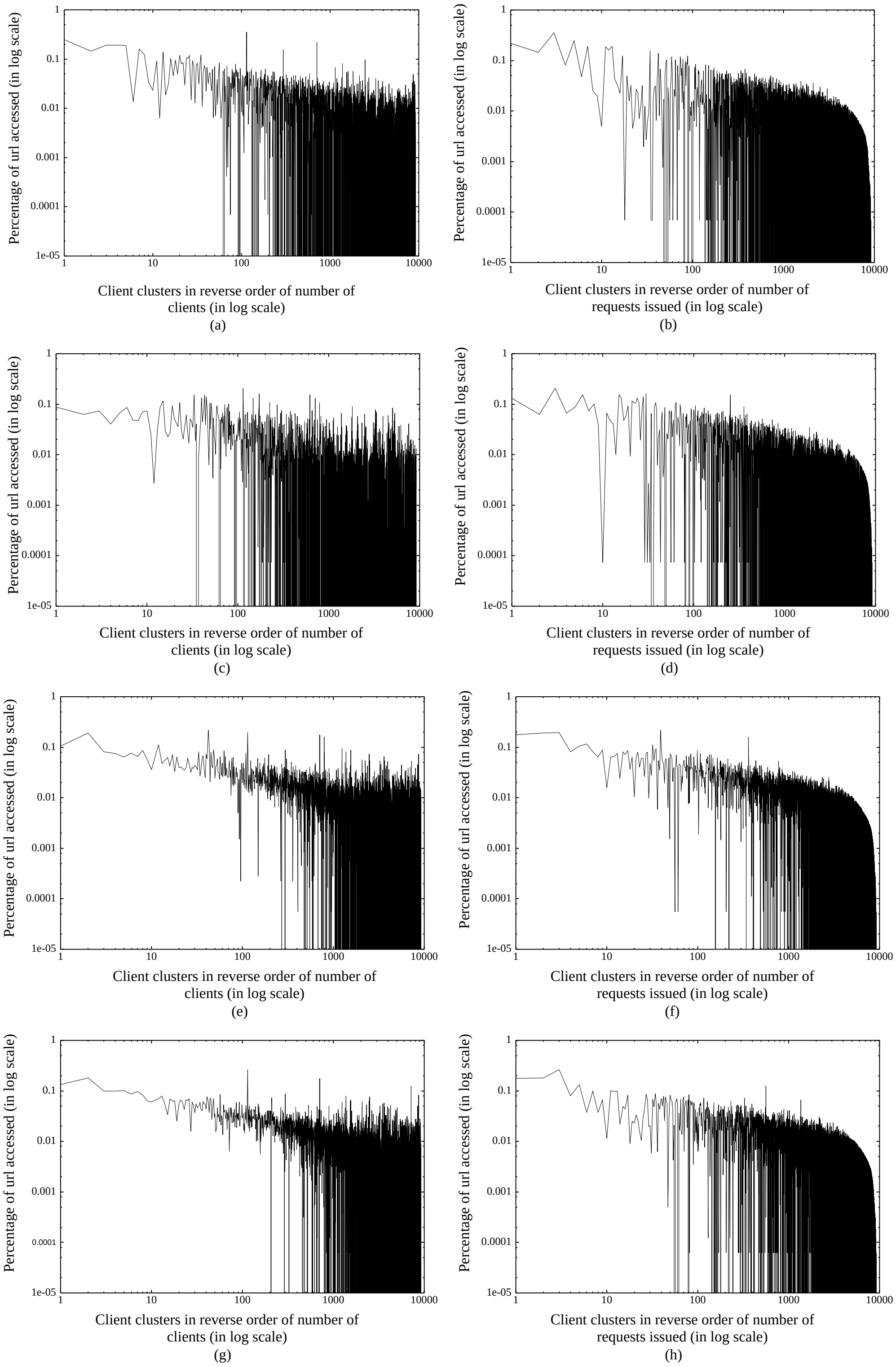

Figure 18: The client cluster URL accessing distribution for the 4 sessions of the Nagano server log (both x axis and y axis are in log scale): (a), (c), (e), and (g) are in the reverse order of number of clients in client clusters for time session 1, 2, 3, and 4; (b), (d), (f), and (h) are in the reverse order of number of requests issued from within client clusters for time session 1, 2, 3, and 4.

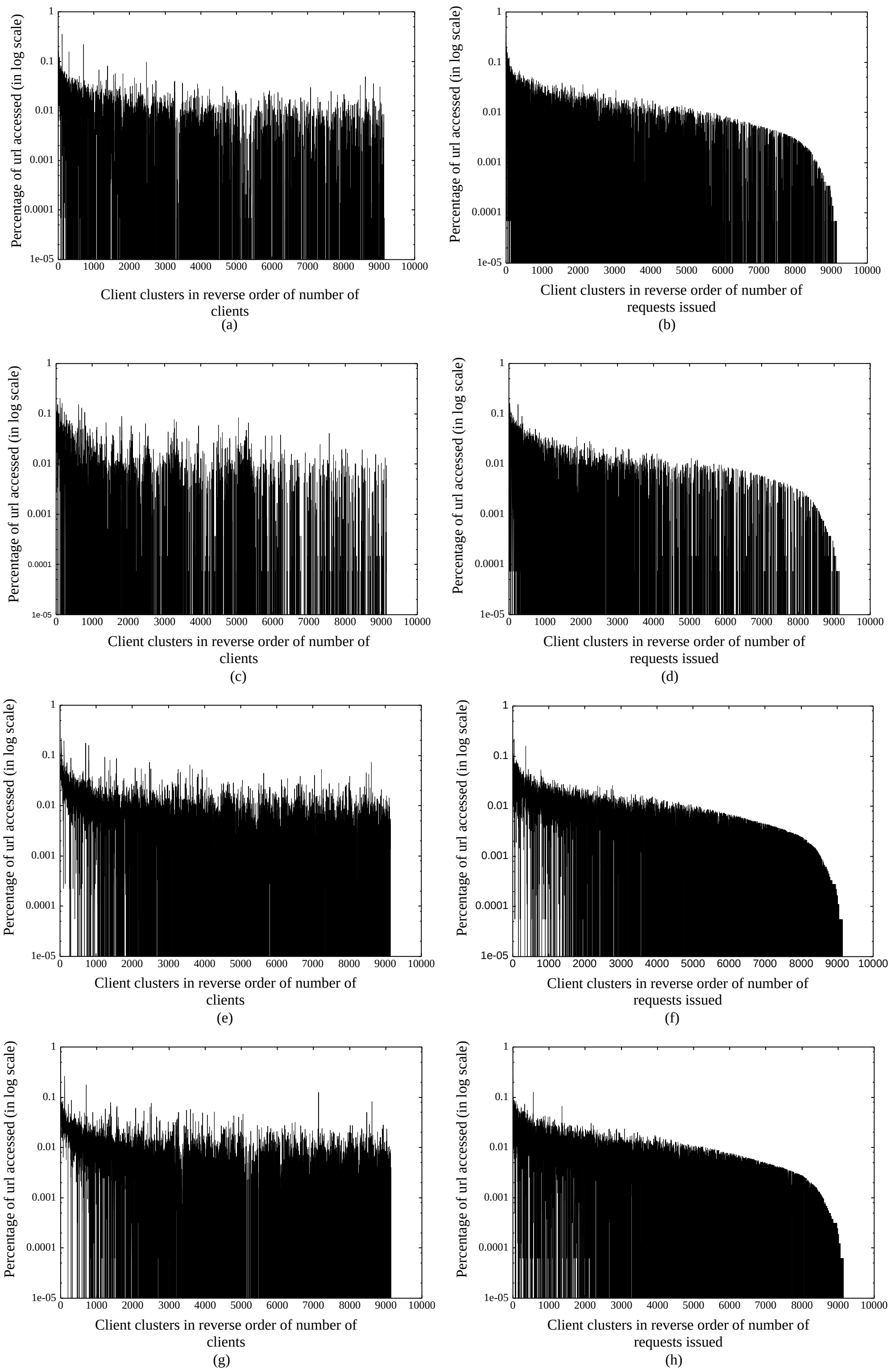

Figure 19: The client cluster URL accessing distribution for the 4 sessions of the Nagano server log (only y axis is in log scale). Note the similarity in number of requests issued and unique URLs accessed by each client cluster across the sessions.

Length of period (days)	11
Total number of unique servers	94674
Total number of unique servers identified by IP address	69192
Number of undetected servers	153
Total number of server clusters	17192
Largest server cluster (servers)	729
Smallest server cluster (servers)	1

Table 39: Experimental results of server cluster detection on WorldNet client trace.

- **Web server cluster detection**

  Our method can also be used to detect *server* clusters in proxy logs. In a set of servers accessed over a 11-day period in a AT&T Worldnet client trace we found 69,192 unique server IP addresses, with only 153 ($\sim$ 0.2% of the total) that were not clusterable (due to lack of proper network prefix/netmask information) (Table 39). Roughly 4% (729 out of 17,192) of the server clusters received 70% of the 12.4 Million requests. Detecting client *and* server clusters can aid in engineering and shaping traffic in a content distribution application.

- **Clustering of client clusters**

  After detecting client clusters based on the BGP routing table information, we can further cluster nearby client clusters (i.e., networks) into *network clusters*. We use *traceroute* to do the higher level clustering. Typically, we run *traceroute* on a number of ($r \geq 1$) randomly selected clients in each network and do suffix matching on path towards each destination network. The second level clustering is useful in applications such as selective content distribution, proxy placement, and load balancing.

4 Applications of client clustering

The client cluster detection mechanism is applicable to Web caching, server replication, content distribution, server-based access prediction, and network management. The real-time client clustering information (i.e., the client clusters results detected from recent data logs using real-time routing information) gives the service provider a global view of where their customers are located and how their demands change from time to time. This is crucial information for service providers to enhance their services, reduce costs, and extend global presence. We present one application of clustering—Web caching.

4.1 Web caching simulation

We examine the usefulness of network aware clustering in a Web caching simulation and contrast it with the simple approach. We present our approach to identifying spiders and proxies and then describe proper placement of proxies between clients and servers.

4.1.1 Client classification

We classify clients into *visible clients* (visible to the Web server, representing a majority), *hidden clients* (hidden behind proxies and thus not visible to the server), and *spiders*. All visible clients belonging to the same client

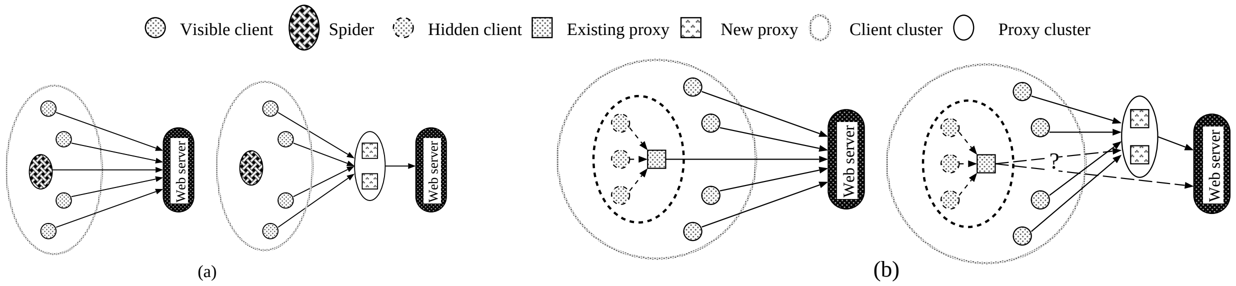

Figure 20: Eliminating spiders and existing proxies from the server logs: (a) spiders; (b) existing proxies.

cluster will share one set of proxies in the Web caching system. The goal of our Web caching system is to improve the Web access quality perceived by *both* visible and hidden clients. First, we identify spiders and eliminate them from server logs. Spiders blindly visit (almost) every resource in a Web site. Clients in the same cluster with a spider will not benefit from a proxy in front of the cluster (as shown in Figure 20(a)) since the spider will issue most of the requests and a majority of the URLs accessed by it will not be accessed repeatedly. Next, we locate existing proxies since (Figure 20(b)) hidden clients behind them may suffer longer latencies due to being an additional hop away from the origin server. These hidden clients will only benefit from a new proxy when most of their requested pages, which are not available at the existing proxy, are cached at the new proxy (i.e., those pages are also requested by normal clients in the same cluster as the existing proxy).

4.1.2 Identifying spiders/proxies

A spider cluster often issues a very large amount of requests within a short period and the host(s) within that client cluster issuing a large percentage of the requests is suspected to be a spider. The overall access pattern of clients (consisting of the number of unique URLs accessed, the arrival time of the requests, and the request distribution inside a client cluster) is shared by a proxy but not by a spider.

The number of unique URLs accessed can identify some spiders, but may not be enough to identify all of them. The spider in the Sun server log, which is in a cluster of 27 hosts, issued 692,453 requests and accessed 4,426 out of 116,274 unique URLs. We can distinguish spiders from clients/proxies by examining the request arrival time. From Figure 21(c), we don't see any similarity between the access pattern of the spider and that of the entire server log shown in Figure 21(a). There are certain correspondences between the access pattern of the proxy shown in Figure 21(b) and that of the entire server log. Each spike observed in the proxy access pattern matches the daily spike in the access pattern of the entire log.

If a client cluster contains both spiders and normal clients, the requests issued by hosts within the cluster will have an uneven distribution among them and can help identify a spider. The request distribution of the cluster containing a spider is shown in Figure 22 and Table 40. Almost all the requests are issued by the spider. We thus need to combine examination of request arrival time and the distribution of requests within a cluster to identify spiders. There are no spiders in the Nagano server log—unsurprising given that it is a single day's log of a transient event site.

Unlike differentiating a spider from a proxy and a client, it is harder to identify proxies among a group of clients. A proxy will mimic the access pattern of clients behind it (see Figures 21 and 23). One difference between a proxy and a client is that the proxy may issue more requests and have a shorter "think" time between requests than a client does. For example, we detected a client cluster in the Sun server log issuing 326,566 requests. The

IP address	Number of requests
130.191.5.101	22
130.191.13.11	692493
130.191.17.113	68
130.191.17.124	6
130.191.17.189	6
130.191.17.212	2
130.191.18.200	51
130.191.25.10	598
130.191.26.16	6
130.191.26.43	17
130.191.26.72	17
130.191.57.14	9
130.191.57.16	12
130.191.117.95	74
130.191.117.110	13
130.191.118.1	38
130.191.137.87	20
130.191.147.10	4
130.191.154.185	22
130.191.156.126	25
130.191.156.127	6
130.191.157.80	1
130.191.165.33	67
130.191.166.1	112
130.191.182.120	2
130.191.219.184	8
130.191.226.108	256
Total	693955

Table 40: The client requests distribution of a client cluster containing a known spider (130.191.13.11) in the Sun server log which issues 692,493 requests (99.79% of all requests).

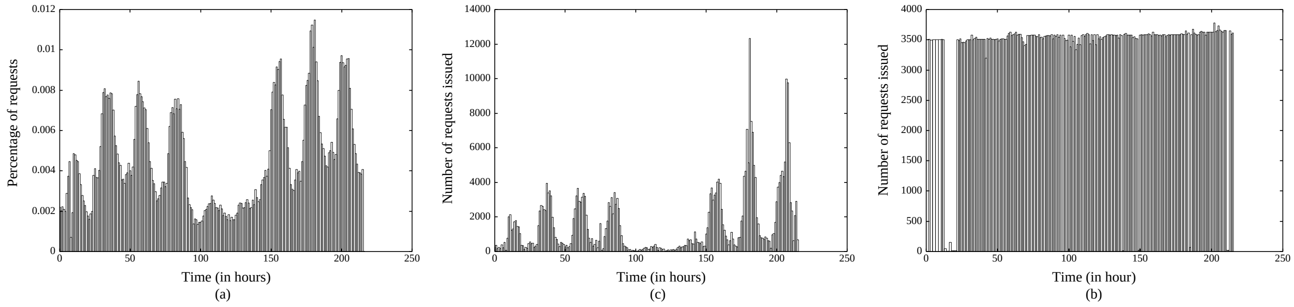

Figure 21: Histogram of the requests in the Sun server log: (a) the entire server log, (b) a client cluster containing a proxy, (c) a client cluster containing a spider.

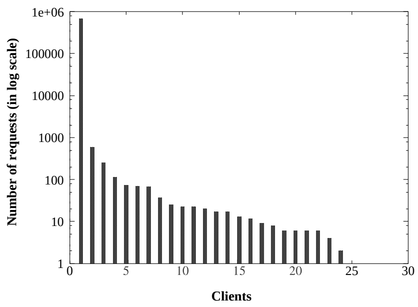

Figure 22: The client requests distribution of a client cluster containing a spider in the Sun server log which issues 692,453 requests (99.79% of all requests in the cluster).

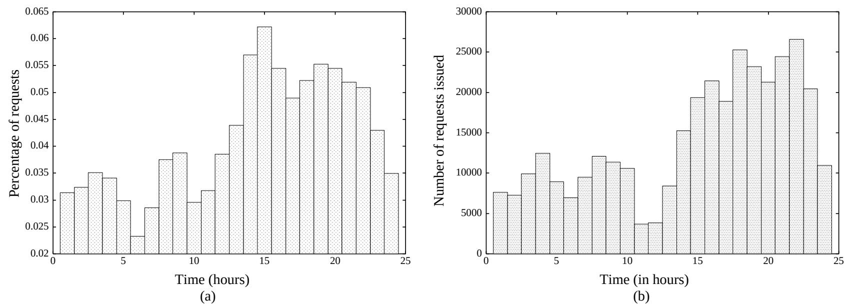

Figure 23: Histogram of requests in the Nagano server logs: (a) the entire server log, (b) a client cluster containing only a proxy.

cluster has only two clients issuing 2,699 and 323,867 requests, respectively. We suspect that the second client is a proxy. In the Nagano server log a client cluster containing only one client issued 77,311 requests. Proxies can also be seen in Figures 7(b) and 8(b) in Section 3. However, our method depends on the number of clients sharing the proxy and their access pattern. We have not found a solution guaranteed to locate all proxies correctly.

Besides using access patterns, the `User-Agent` field of the request, if present in a server log, is also useful in differentiating proxies. The `User-Agent` field includes information about the particular browser, Operating System version and hardware used by the client initiating the request. If there are a lot of different `User-Agent` fields from the same host responsible for a lot of requests, then it is very likely that the host is a proxy.

Approach	Network-aware	Simple
Total number of client clusters	9853	23523
Threshold (requests per client cluster)	2744	696
Number of busy client clusters	717 (32691 clients, 8167590 requests)	3242 (30774 clients, 8167335 requests)
Busy client clusters (requests)	2744 - 339632 (1 - 1343 clients)	696 - 339632 (4 - 63 clients)
Less-busy client clusters (requests)	1 - 2741 (1 - 46 clients)	1 - 695 (1 - 4 clients)

Table 41: Experimental results of thresholding client clusters on the Nagano server log.

Total number of client clusters	35563
Threshold = 0.7 (requests per client cluster)	199
Number of busy client clusters	2869 (167825 clients, 2424554 requests)
Busy client clusters (requests)	199 - 78336 (1 - 1680 clients)
Less-busy client clusters (requests)	1 - 198 (1 - 63 clients)

Table 42: Experimental results of thresholding client clusters on the Apache server log.

Total number of client clusters	2480
Threshold = 0.7 (requests per client cluster)	42779
Number of busy client clusters	24 (17446 clients, 4631370 requests)
Busy client clusters (requests)	42779 - 1507769 (2 - 7918 clients)
Less-busy client clusters (requests)	1 - 36976 (1 - 352 clients)

Table 43: Experimental results of thresholding client clusters on the Unav server log.

4.1.3 Thresholding client clusters

After identifying and eliminating possible spiders and proxies from the server log, we filter out uninteresting client clusters by thresholding on the number of requests issued from within a client cluster. After reverse sorting based on the number of requests issued from within client clusters, we retain *busy* client clusters—clusters whose requests added up to at least 70% of the total requests in the server log. Such a thresholding reduced the number of client clusters dramatically with the smallest client cluster included issuing at least 2,744 requests. In the Nagano log there were only 717 busy client clusters out of 9,853 (see Table 41). These busy client clusters have 32,691 clients and issue 8,167,590 requests. The size of busy client clusters ranges from 1 to 1,343 clients and the number of requests issued by busy client clusters ranges from 2,744 to 339,632. The size of less-busy client clusters, filtered out, ranges from 1 to 46 clients and the number of requests issued by less-busy client clusters ranges from 1 to 2,741. The results of thresholding client clusters of Apache, Unav, EW3, and Sun server logs are shown in Tables 42– 50.

Similar thresholding on clusters detected using the simple approach leads 3,242 busy client clusters (out of 23,523) in the Nagano log, with the smallest cluster issuing 696 requests. The busy clusters have 30,774 clients, issuing 8,167,335 requests (70% of the total). The size of busy client clusters ranges from 4 to 63 clients and the number of requests issued by them ranges from 696 to 339,632. The size of less-busy client clusters ranges from 1 to 4 clients and the number of requests issued by less-busy client clusters ranges from 1 to 695. The results are very different from that of the network-aware approach, which implies the simple approach does not do a good job on grouping clients.

Total number of client clusters	7754
Threshold = 0.7 (requests per client cluster)	60
Number of busy client clusters	1600 (13262 clients, 839552 requests)
Busy client clusters (requests)	160 - 21949 (1 - 359 clients)
Less-busy client clusters (requests)	1 - 159 (1 - 11 clients)

Table 44: Experimental results of thresholding client clusters on the EW3 server log: EW3-a1.

Total number of client clusters		12465	12880	14949	17826	15206	13620
Threshold = 0.7	(requests per client cluster)	146	182	209	260	215	184
Number of busy client clusters		1533	1456	1752	1767	1804	1764
	(clients)	17517	18203	24120	28585	25485	20961
	(requests)	951196	1153002	1527640	2015302	1554956	1292315
Busy client clusters	(requests)	146 - 100140	182 - 145970	209 - 116806	260 - 132009	215 - 81695	184 - 86585
	(clients)	1 - 634	1 - 932	1 - 760	1 - 1001	1 - 859	1 - 717
Less-busy client clusters	(requests)	1 - 145	1 - 181	1 - 208	1 - 259	1 - 214	1 - 183
	(clients)	1 - 13	1 - 25	1 - 22	1 - 21	1 - 21	1 - 13

Table 45: Experimental results of thresholding client clusters on the EW3-a2 server log over 6 months.

Total number of client clusters	14895
Threshold = 0.7 (requests per client cluster)	153
Number of busy client clusters	2494 (22312 clients, 1336783 requests)
Busy client clusters (requests)	153 - 76218 (1 - 704 clients)
Less-busy client clusters (requests)	1 - 152 (1 - 13 clients)

Table 46: Experimental results of thresholding client clusters on the EW3 server log: EW3-b.

Total number of client clusters	18571
Threshold = 0.7 (requests per client cluster)	325
Number of busy client clusters	224 (120956 clients, 1047571 requests)
Busy client clusters (requests)	325 - 53217 (1 - 6989 clients)
Less-busy client clusters (requests)	1 - 324 (1 - 38 clients)

Table 47: Experimental results of client clustering on the EW3 server log: EW3-c1.

Total number of client clusters	16286
Threshold = 0.7 (requests per client cluster)	11
Number of busy client clusters	3146 (22718 clients, 106870 requests)
Busy client clusters (requests)	11 - 2410 (1 - 824 clients)
Less-busy client clusters (requests)	1 - 10 (1 - 9 clients)

Table 48: Experimental results of thresholding client clusters on the EW3 server log: EW3-c2.

Total number of client clusters		1517	6187	4391	7321	6863	5111
Threshold = 0.7	(requests per client cluster)	293	528	458	443	600	432
Number of busy client clusters		451	1554	1197	1201	1594	1323
	(clients)	2423	10243	6682	10974	12884	8568
	(requests)	404278	2629217	1677253	1608847	3152799	1799285
Busy client clusters	(requests)	293 - 39021	528 - 151229	458 - 104183	443 - 107791	600 - 167907	432 - 99176
	(clients)	1 - 389	1 - 523	1 - 520	1 - 574	1 - 593	1 - 573
Less-busy client clusters	(requests)	1 - 292	1 - 527	1 - 457	1 - 442	1 - 599	1 - 431
	(clients)	1 - 8	1 - 17	1 - 9	1 - 17	1 - 16	1 - 10

Table 49: Experimental results of thresholding client clusters on the EW3-w server log over 6 month.

Total number of client clusters	33468
Threshold = 0.7 (requests per client cluster)	934
Number of busy client clusters	2536 (121838 clients, 9712211 requests)
Busy client clusters (requests)	934 - 693955 (1 - 2529 clients)
Less-busy client clusters (requests)	1 - 933 (1 - 57 clients)

Table 50: Experimental results of thresholding client clusters on the Sun server log.

4.1.4 Proxy placement

One way to place proxies is to assign one or more proxies for each client cluster based on metrics such as the number of clients, number of requests issued, the URLs accessed, or the number of bytes fetched from server. The proxies assigned to clients in the same client cluster form a proxy cluster and would co-operate with each other. Alternatively, we can place a proxy in front of each client cluster and further group proxies into proxy clusters according to their AS numbers and geographical locations. All proxies belonging to the same AS and located geographically nearby will be grouped together to form a proxy cluster. The first approach is easier to implement while the second one, though more practical, is complicated. Here, we only address the first approach in our experiments.

4.1.5 Experimental results

We use the client cluster results in the trace-driven simulation to evaluate the benefit of proxy caching. The big picture of the Web caching simulation is shown in Figure 24. We place proxies in front of each client cluster. We implement the Piggyback Cache Validation [10] scheme with a fixed TTL (time-to-live) expiration period at each proxy cache. By default, a cached resource is considered stale once a period of one hour has elapsed. When the expiration time is reached for this resource, a validation check is piggybacked on a subsequent request to its server. If the resource is accessed after its expiration, but before validation, then a `GET If-Modified-Since` request is sent to the server for this resource. We use LRU as the cache replacement policy. Our purpose is not to evaluate the PCV scheme, but to illustrate the applicability of client clustering on Web caching simulation and to determine the effect of different client cluster detection approaches on simulation results.

In our simulation, we set TTL to be 1 hour, vary cache size at each proxy and examine two performance metrics: hit ratio and byte hit ratio. Varying TTL to 5, 10, and 15 minutes yields similar results. We compare the results of simulations on the client clusters obtained by our approach and the simple approach. Our simulations

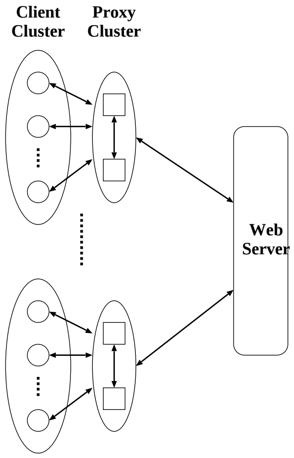

Figure 24: Clustering clients and proxies.

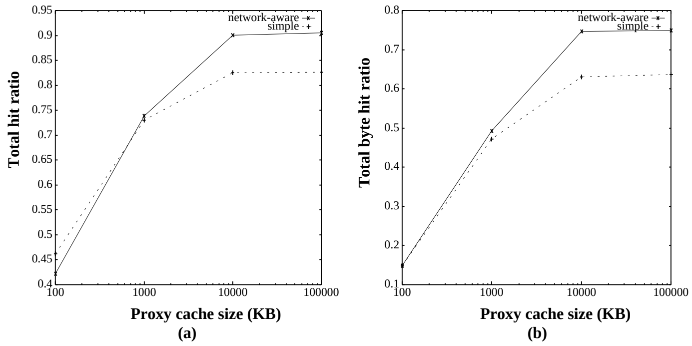

Figure 25: Simulation results on Web server performance vs proxy cache size of the Nagano server log (x axis is in log scale): (a) is the total hit-ratio, (b) is the total byte hit ratio.

are conducted on the Nagano log and the EW3 logs though we only show the results on the Nagano server log[12].

Our evaluation of simulation results consists of two parts: server performance and proxy performance. To evaluate server performance, we vary the cache size at each proxy from 100 KB to 100 MB and examine total hit ratio and byte hit ratio observed at server. Figure 25 shows the simulation results of server performance on the Nagano server log. The curve with a '*' marker shows the results of our approach and the curve with a '+' marker shows the results of the simple approach. Figure 25(a) and (b) show the total hit ratio and byte hit ratio observed at server (i.e., the ratio and byte ratio of requests served by local proxies), respectively. Both hit ratio and byte hit ratio increase as the proxy cache size increases. The results show that the simple approach under-estimate (around 10%) both hit ratio and byte hit ratio observed at server when local proxy cache size is large (e.g., > 700KB). The hit ratio (i.e., up to 60 $\sim$ 75%) obtained from our simulation is greater than typical cache hit rates of around 40% [2] due to the proxies in our simulation being dedicated to one typical server. The client access and resource updating pattern of the Nagano event log are different than other logs. We did't observe a high hit ratio on all the logs.

To evaluate proxy performance, we fix the cache size as infinite and examine the hit ratio and byte hit ratio at each proxy (Figures 26). The solid curves show the results of our approach and the dotted curves show the results of the simple approach. We only show the results of top 100 client clusters in the reverse order of number

[12] In our simulation, we ignore all requests to resources accessed by clients less than 10 times.

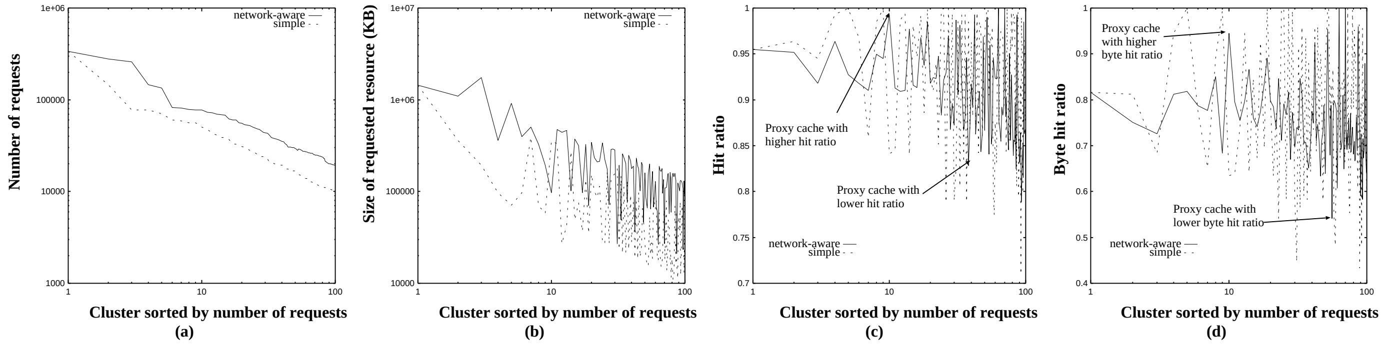

Figure 26: Simulation results on proxy cache performance of the top 100 client clusters in the Nagano server log (x axis is in log scale): (a) and (b) are the number of requests and size of requested resources in client clusters in reverse order of number of requests, respectively; (c) and (d) are the proxy cache hit ratio and byte hit ratio of client clusters in reverse order of number of requests, respectively.

of requests (i.e., the first 100 client cluster if we ordered them in the reverse order of number of requests issued). Figure 26(a) and (b) show the number of requests and size (in KB) of requests issued in client clusters in reverse order of number of requests, respectively. Figure 26(c) and (d) show the hit ratio and byte hit ratio observed at each proxy. The great differences of the client requests and proxy hit ratio results got from our approach and those obtained from the simple approach demonstrate that the simple approach fails to properly evaluate potential benefit of proxy caching.

To summarize, the significant differences shown in simulation results demonstrate that the simple approach is not able evaluate benefits of Web caching schemes or the corresponding overhead (both at server and at proxies) well, and hence, fails to serve as a guide on solving problems such as proxy placement. The simulation results from our network-aware approach are more realistic, and is useful for designing and evaluating Web caching systems. For example, knowing the location of clients and their demands, a Web site can better provision its service. While we only address simulation of Web caching system with one server and multiple proxies, we can also simulate multiple servers and multiple proxies by merging more server logs collected at the same time.

4.2 Other applications of client clustering

Besides Web caching, client clustering is also useful in several other applications. Real-time clustering enables content distribution service providers to deliver the right content to the right customer at the right time. As a service replication mechanism, better service can be provided by placing replicated servers (e.g., mirror sites) at hot spots to accommodate customers' demands. Also, with the knowledge of customer's location and the traffic load origination, the service providers can to do better load balancing by provisioning in advance.

5 Related work

We are not aware of any work that uses BGP routing information to do clustering of clients for the range of applications we have examined. Simply using *nslookup* to do clustering is both expensive and unlikely to yield full results. A clustering model based on distance between servers and clients [5] was used to lower the cost of document dissemination; the implementation used *traceroute*. Using *traceroute* along the lines of [11] (although they use it for a different purpose) or [5] is also more expensive than using routing table information. In particular it is not feasible for using it under real-time considerations, e.g., when a Web event is in progress. Several efforts are underway at research and product levels in improving Web caching and content distribution. At the DNS layer some companies (e.g., Akamai [1]) use client's location to serve content from a nearby site to lower load on the server and user perceived latency. The clustering methodology could be used by such companies.

6 Conclusion and future work

We have proposed a novel way to detect client clusters using BGP routing information. Experimental results show that our method is able to group more than 99.9% of the clients captured in a wide variety of server logs into clusters. Sampling validation results demonstrate that our method has an accuracy of 90%. A self-correction and adaptation mechanism was also proposed to improve the applicability and accuracy of the initial cluster detection results. The entire cluster detection process can be done in an automated fashion—moving from a server log to a set of interesting client clusters. The cluster information can be used in applications such as content distribution, caching, and network management. A useful by-product identification of spiders and proxies among Web clients by examining the access patterns of corresponding client clusters. Our methodology is immune to BGP dynamics, independent of the range and diversity of server logs, and computationally non-intensive. Our future work includes using information on ASes to reduce the error ratio of client cluster detection.

Acknowledgment

We would like to thank several people who answered questions including Jennifer Rexford, Anja Feldmann, Craig Labovitz, Vern Paxson, Thomas Narten, Bill Manning, Tim Griffin, and Steven Bellovin.

We would also like to thank those who read an earlier version of this document and offered suggestions including Jennifer Rexford, Chuck Cranor, Tim Griffin, Oliver Spatscheck, Sylvia Ratnaswamy, Walter Willinger, Craig Wills, and Phong Vo. Discussions with Emden Gansner and Nick Duffield were also beneficial.

Balachander Krishnamurthy

FP-HA1630000-BK/JW **Jia Wang**

Copy to
Members of HA163
Members of HA171
Members of HA173
Dir. HA16

References

[1] Akamai, http://www.akamai.com.

[2] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the www. Technical Report TR-96-11, Boston University Computer Science Department, November 1996.

[3] American Registry for Internet Numbers (ARIN) IP network dump, October 1999. ftp://rs.arin.net/netinfo.

[4] AT&T Forwarding Table Snapshot, April 1999.

[5] Azer Bestavros and C. Cunha. Server-initiated Document Dissemination for the WWW. In *IEEE Data Engineering Bulletin*, September 1996.

[6] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web Caching and Zipf-like distributions: Evidence and Implications. In *Proceedings of IEEE Infocom'99*, March 1999.

[7] Canada Internet Transit Service, December 1999. http://enfm.utcc.utoronto.ca/cgi-bin/c2/c2routes.pl?pop=toro

[8] AT&T Cerfnet BGP Route Viewer, September 1999. route-server.cerf.net.

[9] B. Halabi. *Internet Routing Architectures.* Cisco Press, 1997.

[10] Balachander Krishnamurthy and Craig E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proc. USENIX Symp. on Internet Technologies and Systems*, pages 1–12, December 1997, http://www.usenix.org/events/usits97.

[11] P. Krishnan, Danny Raz, and Yuval Shavitt. The cache location problem. Under submission.

[12] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *Proceedings of ACM SIGCOMM*, September 1997. http://www.acm.org/sigcomm/sigcomm97/program.html.

[13] K. Lougheed and Y. Rekhter. A Border Gateway Protocol. RFC 1163, IETF, June 1990.

[14] Merit Internet Performance Measurement and Analysis Project, 1999. http://www.merit.edu/~ipma/routing_table.

[15] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol.* Addison-Wesley, 1998.

[16] NLANR network analysis infrastructure, November 1997. http://moat.nlanr.net/IPaddrocc.

[17] Oregon Exchange BGP Route Viewer, December 1997. route-views.oregon-ix.net.

[18] SingAREN BGP routing table, December 1999. http://noc.singaren.net.sg/netstats/routes.

[19] Vbns route information, December 1999. http://www.vbns.net/route/index.html.

Contents

AT&T

Document Cover Sheet
for Technical Memorandum

Title: On Network-Aware Clustering of Web Clients

Authors	Electronic Address	Location	Phone	Company (if other than AT&T)
Balachander Krishnamurthy	bala@research.att.com	FP D229	973-360-8648	
Jia Wang	jiawang@research.att.com	FP D204	973-360-8691	

Document No.	Work Project No.
HA1630000-000101-01TM	3116-16-3010

Keywords:
Web, clustering, networks, proxy, spider

MERCURY Announcement Bulletin Sections

Abstract

Being able to identify the administrative entities controlling groups of clients (called *clusters*) that are responsible for a significant portion of a Web site's requests can be helpful to both the Web site and the administrative entities. The Web site could consider moving content closer to such clusters, while the administrative entity could consider placing proxy caches in front of its group of clients to take advantage of the traffic pattern. In this work, we propose a novel "network-aware" method to detect client clusters based on readily available routing information. Experimental results show that our entirely automated approach is able to identify clusters for 99.9% of the clients in a wide variety of Web server logs and sampled validation results show that we detect clusters correctly with a probability of over 90%. An efficient self-corrective mechanism increases the applicability and accuracy of our initial approach and makes it adaptive to network dynamics. In addition to being able to detect unusual access patterns made by spiders and (suspected) proxies, our proposed method is useful for content distribution and proxy positioning, and applicable to other problems such as server replication and network management.

Pages of Text 49 Other Pages 5 Total 54
No. Figs. 26 No. Tables 50 No. Refs. 19

Mailing Label

Complete Copy	Cover Sheet Only
Members of HA163	L. R. Rabiner
Members of HA171	Members of HA16
Members of HA173	Members of HA17
Dir. HA16	DHs HA16

Future AT&T Distribution by ITDS Release to any AT&T employee (excluding contract employees)

Author Signatures

Balachander Krishnamurthy Jia Wang

For Use by Recipient of Cover Sheet:

Computing network users may order copies via `http://attlis.att.com/services/proprietary.html`.