Characterizing Large DNS Traces Using Graphs

Charles D. Cranor, Emden Gansner, Balachander Krishnamurthy, Oliver Spatscheck

Abstract—

The increasing deployment of overlay networks that rely on DNS tricks has led to added interest in examining DNS traffic. In this paper we report on a characterization of DNS traffic gathered over a period of several weeks at Internet Gateway Routers (IGRs) in the AT&T Common Backbone. The characterization is carried out using several novel techniques to identify clients, local DNS servers, and authoritative DNS servers. Our techniques include passive and active measurements, graph-based analysis, examination of outliers, and explicit checks against data obtained from several external sources. Our contribution is the reduction of a very large data set (over 1 terabyte of raw data) into a significantly smaller representation that is ideally suited for answering protocol-specific semantic queries quickly. After categorizing the addresses, we use the network aware clustering technique to group local DNS servers. By juxtaposing the DNS server clusters with clusters formed by Web clients obtained from a large portal Web site, we determine the distribution of identified DNS servers in busy clusters. A variety of applications are examined, ranging from identifying suspected zombies to helping Content Distribution Networks in mapping location of DNS servers.

I. Introduction

Among the primary difficulties of carrying out Internet measurements are issues of scale and the representation of large data sets in a format that is suitable for domain-specific queries that can be answered rapidly. Often the scale issue inhibits work, leading to research based on smaller samples of potentially non-representative data. We attack the problem of scale by representing a large amount of Internet data, gathered from peering links, in a condensed graph format and using efficient software tools to answer complex questions swiftly. Although the work described in this paper deals with a specific protocol (DNS), we believe our approach will work just as well for other protocols in the construction and answering of a broad set of domain-specific questions.

We focus on DNS in this paper primarily due to the explosive growth of Web traffic which has led to an corresponding increase in the number of DNS transactions. In a typical Web transaction, a Web client (such as a browser) attempts to obtain a resource identified by a Uniform Resource Locator (URL) by contacting the specified Web site using the HyperText Transfer Protocol (HTTP [1]). Each URL has a server component whose IP addresses has to be determined by using a DNS lookup in order to set up a transport layer connection on which the HTTP message transfer occurs. For example, a Web request for the URL `http://www.example.com/index.html` requires resolution of `www.example.com` to an IP address.

Web traffic is asymmetric by nature—a large number of requests are directed to a few thousand sites. This has resulted in many busy sites offloading frequently requested resources to Content Distribution Networks (CDNs). A CDN is an overlay network that typically uses DNS-based redirection [2] to allow users to fetch resources from 'nearby' caches. The widespread deployment of such overlay networks has led to further increase in DNS traffic.

The increase in Web traffic and the introduction of CDNs led us to seek a broader understanding of the nature and use of the DNS protocol. There have not been many significant measurement studies of DNS traffic in recent research literature. The study described in [3] focuses on the evaluation of the proximity of a small set of clients to 3807 local DNS servers. In contrast, we have been collecting the network flow-level data of DNS traffic at multiple Internet Gateway Routers (IGRs) in the AT&T Common Backbone. In our study we identified 564,077 suspected local DNS server. This large-scale examination of DNS allows us to characterize the current DNS traffic patterns in terms of the participating agents (clients, and local and authoritative DNS servers) and their interaction. It also allows us to examine potential applications of network aware clustering [4].

Our specific goals are as follows:

- Categorize the set of IP addresses seen in the traces with high probability into clients, local DNS servers, authoritative DNS servers, and root servers.
- Validate the correctness of the categorization via a variety of techniques and use the categorization in applications to demonstrate its usefulness. Among the applications considered are identifying anomalous hosts that may have been compromised or were targets of attacks, using the list of suspected local DNS servers in CDN applications, and identifying a lower bound of DNS servers found in busy clusters.

The authors are with AT&T Labs–Research, Florham Park, NJ, USA. email:{chuck, erg, bala, spatsch}@research.att.com Contact author: Balachander Krishnamurthy, fax: 973-360-8077, 180 Park Avenue, Florham Park, NJ 07932

- Be able to handle large data sets with the capability of adding new data sets requiring only incremental work.
- Automate and streamline the process so that it can be repeated at other sites for different applications.

Section II presents a brief introduction to DNS. Section III presents our experimental methodology, discusses the various data elements in our experiment, and conversion of the *netflow* data into graph form. The properties of the resulting graph are examined in Section IV. The characterization of IP addresses gathered is discussed in Section V. Section VI discusses implementation issues associated with the graph representation. The results of our large data gathering experiment is presented in Section VII and Section VIII discusses applications of the categorization. We conclude with a look at other applications and improvements to our implementation and analysis technique.

II. Brief introduction to DNS

Hostnames on the Internet are translated into IP addresses and IP addresses back into hostnames via the Domain Name System (DNS). DNS is a database distributed across a set of servers that handle name and address resolution on a hierarchical basis. The topology of the DNS naming scheme consists of a collection of top-level domains (such as `.com`, `.edu`, and `.dz`) below the root of a hierarchy, organized into separately administered zones (e.g., `att.com`). The zones register the names and IP addresses of a set of authoritative DNS servers with the root servers. Client requests for resolving names are sent by a resolver library to a local DNS server with a cache. A typical DNS transaction consists of request and response sent over UDP with the client using a non-privileged port and the DNS server running on port 53. TCP is used when the UDP attempt fails, either because the data was too large to fit in a datagram or there is a need to send multiple queries [1]. Sites often have more than one local DNS server. A cache failure results in the query being forwarded either to the root server or a domain server (if partial information is available) which returns an authoritative DNS server capable of answering the query. DNS zone transfers are required to replicate and synchronize copies of the zones between the primary and secondary DNS servers; such transfers can be a full copy or incremental. Zone transfers are carried out over TCP since zone transfer data is typically larger than what a UDP datagram can hold and it is important that the information be sent reliably. A DNS query may proceed iteratively or recursively, where the queried server will do the necessary work and return the result. Positive and negative caching with a time to live (TTL) value are routinely employed by DNS servers. The TTL indicates how long the mapping is valid.

III. Methodology

Our experimental methodology consists of the following steps:

1. Extract relevant IP addresses involved in DNS requests and responses from flow records obtained from the AT&T backbone peering links, and aggregate the data over a period of time.
2. Form graphs whose nodes represent IP hosts and whose edges represent DNS transactions between the hosts. Specifically the edges are directed, with the head of the edge representing a host running a DNS server on port 53 either receiving a request or responding to a request. We assume that the tail of the edge represents the IP address that sent a query or a transfer, and/or received a response. We generate two graphs: one representing communication over TCP and another representing UDP traffic. Outliers among nodes and edges that exhibit certain characteristics (e.g., unexpectedly high outdegree but with low edge weights) are identified so that they can be examined further. Such examination currently requires going back to the flow data to extract additional information. The data is also filtered to remove illegal/reserved addresses (e.g., 0.x.x.x and 10.x.x.x addresses). Once outliers and illegal addresses are removed, the merged TCP-UDP graph is analyzed to generate candidate sets of clients, local DNS servers, and authoritative DNS servers.
3. Iteratively increase the probability of proper categorization by performing additional tests specific to each candidate set. This consists of using attributes like fan-in, fan-out, indegree and outdegree of nodes, as well as edge weights. Additionally, external data sets that have authoritative information are used to verify our categorization. Since the complete topology of DNS is not known even when one has administrative access [5], a variety of heuristics have to be used. We also actively probe selected subsets of the DNS hosts identified for verification. Once various nodes are identified as correctly categorized, their fan-in and fan-out set is generated to improve the accuracy of other categorizations.
4. Once the set of clients, local DNS servers, and authoritative DNS servers have been identified with high probability, they can be examined selectively in conjunction with other data sets (such as Web server logs).

Figure 1 shows the various steps involved in our data acquisition, transformation, and representation in a graph.

[1] In the `bind` version of the resolver, the flag RES_USEVC is used to force the query to be sent over TCP. If the flag RES_STAYOPEN is also specified, the resolver will keep the connection open between requests.

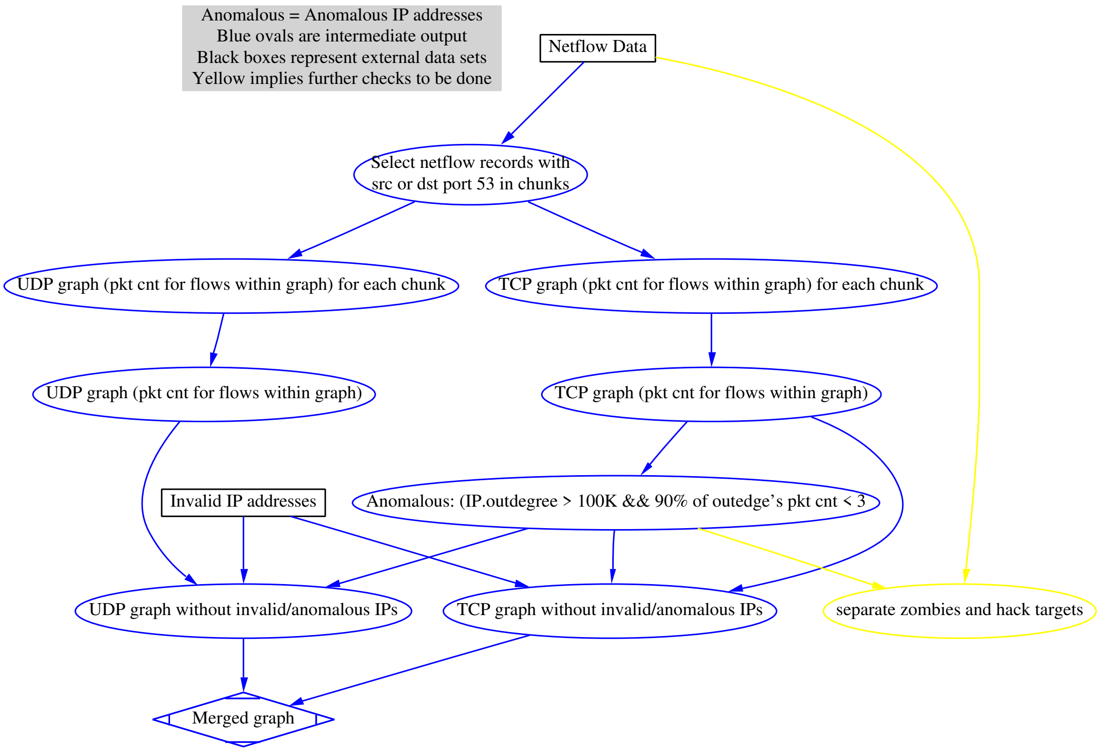

Fig. 1. Steps in converting *netflow* data into a graph

In the rest of this section we discuss the data gathering and the transformation steps involved in generating our graphs.

A. *Data acquisition*

We have been gathering router-level data [6] using Cisco's *netflow* [7] tool. *Netflow* enables accumulation of statistics regarding traffic flows. Our data is gathered at a collection of Internet Gateway Routers that terminate access links in AT&T's backbone network. The flows are *unidirectional* sequences of packets between a particular source and destination device communicating over the same protocol. The most common set of fields recorded are the source and destination IP addresses, source and destination port numbers, the protocol type, and information about the type of service and input interface. A record may be dumped by *netflow* when one of a variety of conditions are met: there was no activity between endpoints for a period of time, a long period of continuous activity has occurred, or the router's cache is full.

The data gathered in our study consist of traffic as shown in Table I. Note that only a portion of the data is actually used due to the reasons explained below. We aggregate data on the default DNS port (53, source or destination) on flows involving either TCP or UDP traffic.

While *netflow* provides us with valuable insight into DNS traffic characteristics, it has several limitations. First, the records are aggregated: the flow abstraction provided by *netflow* obscures details of the intra-record exchanges. A source and destination IP address may have exchanged multiple messages. The number of bytes exchanged and the count of packets may provide hints as to the number of exchanged messages. A single exchange can consist of multiple packets and a large number of bytes. Several short exchanges may have occurred within a single flow. More important, some very long flows may span multiple records. Second, the location where we capture data does not guarantee that we will see the complete flow of traffic; i.e., we might see requests but not responses and vice-versa. The use of hot potato routing implies that response packets may exit via a closer egress point and not use the same path as incoming packets. Third, source and destination ports in a record could both be 53 since some earlier versions of the `bind` software used privileged ports with the sender also using port 53 for the query. The newer versions of `bind` use unprivileged ports that are larger than 1023.

Apart from the flow data, we obtained several external datasets that play an important role in our analysis:

1. A set of known root servers (`*.ROOT-SERVERS.net`) as well as authoritative servers for `.net`, `.com`, `.org`, `.mil`, and `.edu`).

TABLE I
DNS TRAFFIC EXAMINED

Protocol	**S**ource **p**ort	**D**estination **p**ort	**U**se
UDP	53	53	Queries between servers and replies
	any	53	Queries from clients/LDs/ADs
	53	any	Replies to above
TCP	any	53	Queries from clients/LDs/ADs expecting long replies, zone transfers
	53	any	Replies to above

2. The collection of NS records in several of the top level generic domains as well as several national domains.
3. A recent large portal Web site's server log.
4. A set of BGP snapshots gathered recently from a variety of places.

We will explain the use of these external data sets in relevant sections.

B. *Converting* netflow *data to graph representation*

We represent the data as a graph whose nodes are the IP addresses and edges are requests or responses between nodes. A *netflow* record can include multiple exchanges between a source and destination IP address. A typical DNS request-response exchange at a high level consists of a host with IP address IP1 sending a UDP DNS packet from a randomly assigned port to port 53 on a DNS server host with IP address IP2. IP2 responds to the request by sending a UDP DNS packet to the requestor's port on host IP1.

A message exchange at the *netflow* level is represented in the high-level DNS graph by an edge whose head points to the IP address associated with port 53. In other words, the edge always points from the IP address generating the DNS request to the IP address servicing the request. Accordingly, there are two cases:

1. IP1–>IP2: *netflow* records either a DNS request message being sent from IP1 to IP2 and/or a DNS reply being sent from IP2 to IP1. The port number associated with IP2 is 53.
2. IP1<>IP2: *netflow* recorded a DNS message whose source and destination packets are both 53. This occurs with older implementations of the `bind` software (prior to version 8.1) which send queries on port 53. A bi-directional edge could imply that either IP1 or IP2 are DNS servers; thus such edges are colored to distinguish them from normal directed edges.

There are two cases with the edge construction process. First, the graphs may have both IP1–>IP2 and IP1<>IP2 since an IP address may be a target of requests at port 53 and may generate requests at a port number other than 53.

Second, although an edge IP1–>IP2 may be present in the graph, this does not imply that the host represented by IP1 initiated the DNS exchange. We may simply be seeing the response from IP2 to IP1 without the request being captured by *netflow*. This is an artifact of the fact that we do not see the complete DNS traffic since we are gathering data only on a subset of peering links.

Along with the edges, we store the weight of the edge (number of packets involved in the exchange). The information about the number of packets on an edge is critical in looking for outliers in the traffic. For example, a full TCP flow is often at least 6 packets long and if we only see one direction of the traffic we should see at least 3 packets. Any TCP flow with a packet count less than 3 is indicative of an anomaly. In addition the TCP flag information can be used to identify if outlier TCP flows occur during the early part of the handshake.

IV. EXAMINING GRAPH PROPERTIES

Once we have the TCP and UDP graphs, we can examine a variety of graph properties. The representation of the traffic data in a graph form was done with two goals in mind:

- We can look for presence or absence of specific patterns based on the semantics of the traffic being examined.
- We can examine if interesting attributes are present in the data based on a generic graph analysis and then determine if they have useful semantic implications.

Among the attributes of the nodes of interest are its indegree and outdegree. Indegree of a node is the number of edges for which it is the head node and outdegree is the number of edges for which it is the tail node. For example, IP addresses that only send DNS requests will have a nonzero outdegree but zero indegree. Recall that our assumption is that an edge in our graph has the requester as the tail and the responder as the head (whose port number is guaranteed to be 53).

Among the attributes of edges, we check for bi-directional edges. In addition, the TCP and UDP graphs have edges in common since some nodes may communicate with each other using both protocols. Such edges can be identified once the graphs have been constructed.

Among graph attributes, we examine the number of connected components, the structure and directed depth of the components, and indegree and outdegree statistics. In some cases, a request over UDP may result in partial information, along with an error of the form "message truncated", since the data did not fit in the packet. The query might then be repeated over TCP. We can examine nodes that repeatedly communicate using UDP and TCP by looking for common edges in the two graphs.

Since data gathering operations are likely to include anomalous or erroneous input which could severely affect inferences, we need to check the data for outliers. Once the *netflow* data has been transformed into a graph, we can examine the graph for presence of outliers and incorrect data elements and separate them.

One outlier characteristic we examined was the outdegree of nodes in the TCP graph. Recall that TCP is used for either zone transfers or for sending queries when UDP request results in a message truncated response. Since almost any TCP flow involves a half dozen packets, a TCP flow with fewer than 6 packets is an anomaly. Due to the use of hot potato routing on our backbone links, however, we may see only one direction of the flow. Thus, if we see less than three packets on a TCP flow, it is an indicator of an anomalous flow. Edges with lower than expected packet counts could be due to errors in *netflow* or because of missed packets.

If we found a very large number of edges with a shorter than expected TCP packet count and the same node involved as initiator or recipient of communication, that would be more significant. Such nodes (addresses) could be viewed as anomalous for several reasons:

1. Machines that have been compromised. The initiator of communication might be a machine being used to probe the network for security weakness. If such a host has been compromised in order to launch an attack, then such hosts are known as *zombies*.
2. Machines that are targets of attacks involving IP spoofing. The recipient of communication with a large number of low TCP packet counts might be the target of an attack.
3. Machines involved in measurements. For example, occasionally a machine may use port 53 on TCP to obtain RTT measurements.
4. Machines that have been misconfigured.

In any of the above cases, we want to isolate the nodes and edges of such hosts and remove them from the TCP graph. The set of nodes can be compared with the *netflow* data (by examining the TCP fields of those packets) to see if the host is a target of attack, or falls in the zombie, measurement, or misconfigured category.

Similarly, examining the prefixes of the IP addresses reveals addresses such as 0.x.x.x, 10.x.x.x, 192.168.x.x, as well as ones in the restricted range of 172.16.0.0 to 172.31.255.255. Such addresses are not valid addresses and have to be removed from the graph. The resulting UDP and TCP graphs are merged into a single graph for further analysis. The process graph shown in Figure 1 shows the merged graph as the final step in the transformation of the voluminous *netflow* data into a compact graph. All applications (other than finding zombies and targeted machines) are carried out by using the merged graph. From this stage on, the *netflow* data is no longer consulted.

Several bits of information are lost in the reduction of *netflow* data to a graph format. Many of the fields are not of interest to this application: the next hop router's IP address; information about the input/output interface; autonomous system and prefix mask bits; the accumulated *or* of TCP flags. In addition, we lose all temporal information as to when the message exchanges occurred or how many occurred during a specific time interval. Since we treat information from all router locations equally, we do not distinguish between the records contributed by different routers. When an edge is added to the graph we do not record if it was added because there was a request, response, or both.

V. Characterization of addresses

A primary goal of gathering DNS traffic is identifying clients, local DNS servers, authoritative DNS servers, and outliers. In this section we discuss our attempts at characterizing the set of addresses seen in the traffic flow. We should note that some categories might overlap: several authoritative DNS servers are also configured to serve as local DNS servers. Gleaning definitive information about such configuration issues requires information that is not publicly available. Similarly, the location of data gathering will have a strong impact on the distribution of clients, local DNS servers, and authoritative DNS servers.

A. Local DNS servers

We begin by attempting to locate local DNS servers. The properties associated with local DNS servers are as follows:

• When a local DNS server starts, it first uses a pre-configured set of root server hints to contact a currently

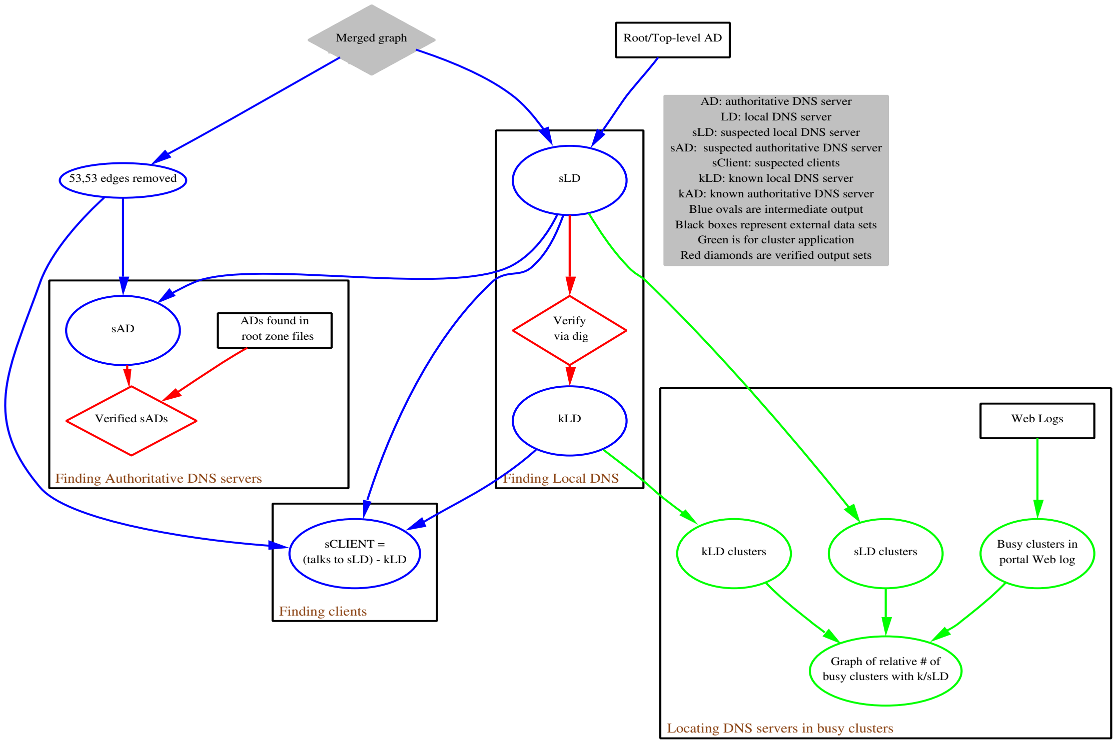

Fig. 2. Using graph to characterize addresses

valid root server in order to obtain a fresh list of root servers. This list is used, on demand, to obtain the addresses of authoritative servers for generic top level domains such as `.com`, `.org`, and `.net`.

- A local DNS server serves as a caching local server for many clients and is thus expected to receive a lot of requests from the set of clients it is responsible for.
- A local DNS server contacts authoritative DNS servers to obtain an answer on a cache miss.

We use each of the above properties of local DNS servers to help identify them in our graph. First we examine the set of 31 known root servers (A.ROOT-SERVERS.net ... M.ROOT-SERVERS.net, as well as authoritative servers for `.net`, `.com`, `.org`, `.mil`, and `.edu`) to see if any of them are the head node of an edge. The tail nodes of each of these edges are candidates to be a local DNS server. We end up with a list of suspected local DNS servers (sLD in Figure 2).

A node with a small indegree is less likely to be a local DNS server assuming we can see all the traffic from the *netflow* data. With increasing probability, the higher the indegree of a node, the more likely it is a local DNS server. However, given the potential of a local DNS server also being configured as an authoritative DNS server, high indegree alone does not uniquely identify a local DNS server. Since there is no automatic way to infer from our data if a suspected local DNS server is indeed a local DNS server, we ran *dig* against a large subset of the suspected DNS servers. Any server that replied is by definition a local DNS server. This set is labeled kLD or known Local DNS servers.

B. Authoritative DNS servers

An authoritative DNS server is authoritative for a set of domains and can give the proper up-to-date response for queries. Authoritative DNS servers (AD) receive many requests but do not generate any queries other than query to the root server at startup time. An AD responds to requests from local DNS servers.

By examining nodes whose outdegree is zero [2] and who receive requests from local DNS servers, we identify suspected authoritative DNS servers. Thus, the heads of edges whose tails are suspected local DNS servers are candidates for authoritative DNS servers. However, in order to generate the set of suspected authoritative DNS servers, we first remove edges in the merged graph that are between IP addresses that both used port 53 to speak to each other. Traffic between two hosts both of which use port 53 implies that the tail or the head of the edge could be a local

[2] It is possible that during our data gathering interval an authoritative server was rebooted and thus might send a request to the root server upon startup, but that is a relatively remote possibility.

DNS server. Including such edges in the graph would end up drawing a lot of clients into the suspected authoritative DNS server collection.

We use an external data set available to us to verify some of the authoritative DNS servers. This data set consists of the zone files of top level generic domains (`.edu`, `.com`, `.net`, ...) as well as several country domains. Since this list consists of known authoritative DNS servers we can verify a fraction of suspected authoritative DNS servers. However, there are many authoritative DNS servers whose information is not registered with the top level generic domains. For example, over 230 country domain servers may hold the zone files of several authoritative DNS servers.

C. Clients

Clients are generally expected to be the largest fraction of the IP addresses. Inside an administrative domain where *netflow* data is gathered, one may have a reasonable idea of prefixes owned by the organization against which the suspected list of client IP addresses can be checked.

Clients have a modest outdegree since they typically only have two or three local DNS servers configured. Their communication on port 53 is restricted to talking to these servers. Direct communication to the root DNS servers via tools like *dig* may complicate our assumptions but such traffic is generally a very small fraction originating from only a few clients.

Note that, as in the case with generating suspected authoritative DNS servers, we remove edges whose port numbers on both sides are 53. Clients rarely run using privileged ports (less than 1023). Also, we do not know if the tail or the head was the host on which a DNS server ran. Since there may be some communication between local DNS servers themselves, we shrink the list of IP addresses that talk to suspected local DNS servers (sLD) by the list of known local DNS servers (kLD) and generate our list of suspected clients.

VI. Graph implementation issues

For the class of problems considered, we require a simple type of directed multigraph representation. Graph nodes can be identified with IP addresses with no additional attributes; edges consist of a directed pair of nodes, with an aggregate packet count attribute. Also, as noted above, edges fall into two categories: ordinary and bi-directional.

There are four principal tasks involved in our analysis:

- Compute the outdegree and associated packet count information of a node
- Remove all edges containing certain nodes
- Remove bi-directional edges
- Find neighbors of a node

plus the auxiliary tasks of merging graphs together and splitting them apart. These are all basically simple algorithms, each requiring about 100 lines of code, and with complexity at worst $O(n \log n)$. Most of the tasks differ in only a few lines of code. For the data sizes we are dealing with (cf. Table II), merging data files to create the merged TCP graph takes about 340 seconds; deriving the zombie nodes takes about 13 seconds; stripping zombies and illegal nodes to create the reduced TCP graph takes around 31 seconds; stripping and merging data files to create the reduced UDP graph uses about 10,115 seconds; merging the UDP and TCP graphs takes 1364 seconds; and, finally, removing bidirectional edges from the merged graph requires 108 seconds. A typical neighbor query might take 135 seconds. Basically linear tasks, like this last, are dominated by the time to read the graph. All times reported are based on a 28-processor, 333 MHz Sun SPARC Ultra Enterprise 1000, with 20,480 Mbytes of main memory. These reflect total CPU times. Some of the tasks can be done in parallel, thus significantly reducing wall clock time.

The structure of all the programs is identical: read a collection of graphs; for each edge, compute and store some information; print the stored information. This simplicity is helpful, in that it makes it easy to tailor the data structures of each program to the task at hand. Thus, if we were computing outdegree information, and know that there are no duplicate edges in the input graphs, we only need to maintain a set of nodes, and can ignore the edge structure. Given the size of the graphs involved, minimizing memory usage significantly improves performance.

We use two formats for the external storage of graphs, both edge-oriented. There is a text format, in which each edge is represented as a line of the form "`X.X.X.X->X.X.X.X <packet count>`". For bi-directional edges, we use "<>" rather than "->". This format is used for the intermediate data derived from the *netflow* database, and when humans need to view the graphs. Principally, though, we employ a binary format, with each edge represented by 12 bytes: 4 bytes each for the 32 bits needed for the IP address of the tail and head, followed by 4 bytes for the packet count. In addition, the high-bit of the packet count is set to indicate a bi-directional edge. Table II gives a feel for the file sizes involved, and shows the obvious shrinkage obtained by moving from the text to the binary format. Some of the larger graphs are actually partitioned into several files. This reduces memory usage, allows for parallelism, and can be used to limit the domain of a search.

All the tools use an underlying library for reading and writing the graphs in their several formats. Both formats

TABLE II
PROGRESSIVE GRAPH SIZES

	Edge count	Kilobytes
Daily UDP data	257,339,100	7,832,896
Daily TCP data	22,435,114	691,057
Merged UDP graph	125,253,867	1,503,046
Merged TCP graph	21,949,654	263,395
Reduced UDP graph	125,204,439	1,502,453
Reduced TCP graph	7,842,576	94,110
Merged TCP+UDP graph	132,849,091	1,594,189
Merged TCP+UDP graph without 53-53 edges	76,224,714	914,694

are self-identifying. Thus, a program can give the I/O library a list of file names and the library will provide the correct reader for each file. This library is about 300 lines of C code, and includes a predicate for identifying legal IP addresses. Set operations, when needed, are implemented using a general-purpose dictionary library [8].

To use this approach for an on-going analysis, the tools and file formats can be used incrementally, without rederiving the graphs anew from the *netflow* data. New edges can be added to the basic TCP graph, and a new list of zombies generated. The subset of new edges both of whose nodes are valid can then be merged into the various reduced graphs. In addition, if any new zombies are identified, these associated edges can be stripped from the graphs. The only case requiring more work would be if the changes remove the zombie status of a node, requiring the re-introduction of all its associated edges.

VII. RESULTS

We now present the result of our data gathering and analysis. A total of 20 days worth of *netflow* data (spread over four contiguous periods of five days each in the course of one year) was gathered resulting in over 1 terabyte of data. From this data we extracted traffic on port 53 consisting of nearly 7.5 billion packets of UDP traffic and 149 million packets of TCP. The resulting UDP graph had 5.4 million nodes and 125 million edges while the TCP graph had 19 million nodes and 22 million edges. Table III presents the precise numbers. After being filtered for port 53 traffic and converted to a graph, over one terabyte of raw *netflow* data resulted in a graph of size 1.5 GB.

A. Anomalous and illegal addresses

We closely examined the raw TCP *netflow* data for anomalous IP addresses. We identified 26 addresses whose outdegree was greater than 100K and whose outbound edges had packet counts less than three for at least 90% of the time. These anomalous addresses contain suspected zombies and hacked IP addresses.

We extracted the full *netflow* records for packets in which the anomalous IP address appeared in the source or destination field and examined the TCP flags field. The TCP flags field in a *netflow* record is the bitwise *or* of all TCP flag fields of all TCP packets counted in that particular *netflow* record. For example, a TCP flag field entry of "RST, SYN" for two TCP packets indicates that at least one of the TCP packets had the RST bit set and at least one of them had the SYN bit set.

Table IV shows the result of this investigation for two anomalous IP addresses (actual addresses have been masked). IP1 is a host which sent at least 1,021,268 SYN requests and received at least 949,839 datagrams with the RST and/or ACK bit set. Since not a single datagram with the SYN ACK bit set was sent to IP1, we determine that IP1 is a host that used TCP port 53 for probing via at least 1,021,268 SYN requests.

IP2 is a host that our records show contacted 2,556,908 distinct IP addresses using TCP port 53. IP2 sent 6,170,090 SYN packets to which at least some machines responded with a packet which has the SYN bit set. Since we do not see all traffic for a particular IP address we do not know the *exact* number of TCP replies with the SYN bit set which were received by IP2. However, we can give an upper bound of 3,989 of such packets recorded in our *netflow* data. This is the total packet count of all *netflow* records sent to IP2, for which the bitwise *or* of the TCP flags contains the SYN bit.

All other anomalous IP addresses we isolated before creating the merged graph were similar in that they originated a large volume of traffic but received only a very few replies. Therefore, we can conclude that all suspect IPs we isolated were probing the network or were misconfigured. None of the addresses were targets for a denial of service attack using IP spoofing.

1915 IP addresses were removed as illegal since they began with 0.x.x.x, 10.x.x.x, etc. In all, 6,793,678 IP addresses and a corresponding 7,842,576 edges were removed as illegal in the TCP graph. The UDP graph with illegal nodes (and corresponding edges) removed had 5.4 million nodes and 125 million edges. The TCP graph with illegal nodes (and corresponding edges) removed had 6.8 million nodes and 7.8 million edges. A single suspected zombie alone probed 5,153,666 different IP addresses. Since this suspected zombie presumably did random probing, the ratio of DNS servers in that set is very small. Most of those 5 million nodes were removed when the suspected

TABLE III
DNS GRAPH PROTOCOL BREAKDOWN

	nodes	edges	packets
UDP raw	5,391,685	125,253,867	7,520,127,483
Without illegal IPs	5,374,094	125,204,439	7,506,734,902
% Reduction	0.33	0.04	0.18
TCP raw	19,732,072	21,949,654	149,220,358
Without illegal, anomalous IPs	6,793,678	7,842,576	124,327,321
% Reduction	66	64	17

TABLE IV
ANOMALOUS ADDRESSES EXAMINED

	Pkts with IP in `src` field	Pkts with IP in `dst` field
IP1 total	1,021,268	949,839
SYN	1,021,268	0
RST ACK	0	949,839
IP2 total	6,177,723	33,891
SYN	6,170,090	1
SYN ACK	46	214
ACK	121	343
FIN SYN ACK	6,906	3,526
RST	144	341
RST ACK	5	29,110

Note that for IP2, the table does not show several infrequently used bit settings such as FIN|SYN|RST|ACK.

TABLE V
CHARACTERIZING ADDRESSES

Suspected LDNS	564,077
SLDNS tested via *dig*	224,348
Known LDNS	58,527
Suspected Authoritative	2,919,455
SAD with outdegree=0	2,717,025
Known AD	217,721
Suspected Client	3,178,981
SClient with Known LDNS removed	3,129,037
SClient with Known LDNS removed and with indeg=0, outdeg < 4	2,298,988

zombie was removed from the TCP graph. This explains the dramatic percentage reduction in the TCP graph.

The merged graph had 11.7 million nodes and 132.8 million edges. The number of bi-directional edges in the merged graph was 56,624,377, involving 1,257,983 nodes.

B. Characterizing addresses

Once the merged graph has been obtained we used a variety of graph queries to partition the set of addresses into clients, local DNS servers, authoritative DNS servers. Since there is no way to categorize the addresses directly, we begin with corresponding *suspected* sets and with increasing confidence group them into the *known* sets. Table V presents the actual numbers extracted from our merged graph.

B.1 Characterizing local DNS servers

By looking for the set of edges whose heads were in the set of known top level authoritative servers or root servers, we identified nearly half a million IP addresses as suspected local DNS servers (sLD in Figure 2). Checking via *dig* nearly half of this suspected set led to confirming around 60 thousand, about one in four. These confirmed addresses are the known local DNS servers (kLD). Local DNS servers may refuse to respond to *dig* if the request came from a set of clients it does not recognize. Thus the absence of response does not imply that the server is not a local DNS server. As a way of cross checking, we selected four sets of 5,000 addresses at random (for a total of 20,000) from the hitherto untested set and ran

```
dig @%s A.ROOT-SERVERS.NET
```

where %s represents the IP address of the server tested. All servers which returned an A record were marked as known local DNS servers. In each of the four sets, nearly 1 in 4 IP address was confirmed as known local DNS servers.

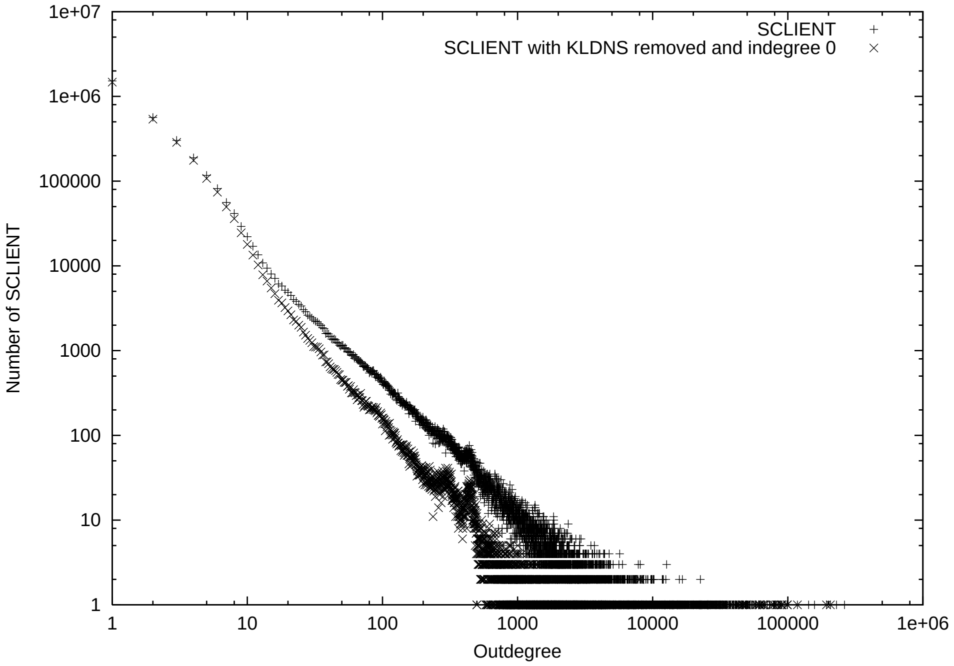

Fig. 3. Outdegree distribution of suspected clients

B.2 Characterizing authoritative DNS servers

Removing 56,624,377 edges with both port numbers 53 from the merged graph and examining the set of edges whose tails are in the sLD set yields 2.9 million addresses that form our suspected authoritative DNS server set (sAD in Figure 2). Since we do not expect authoritative DNS servers to make requests except during startup, we expect that a vast majority of the ADs to have an outdegree of 0. With this stipulation, we have 2.7 million suspected authoritative DNS servers. We extracted 485,661 addresses from the zone files of `.arpa`, `.com`, `.edu`, `.net`, `.org`, and `.root`, and 42,501 addresses from several country domains. 217,721 addresses (over 40%) of the merged set of 527,273 authoritative DNS servers (with duplicates removed) were found in our suspected authoritative DNS server list.

In the intersection of suspected local DNS servers and suspected authoritative DNS servers we found nearly 150,000 servers that are configured to support both roles.

B.3 Characterizing DNS clients

Removing 56,624,377 edges with both port numbers 53 from the merged graph and examining the set of edges whose head are in the sLD yielded nearly 3.2 million addresses. Most clients using DNS are configured to use one to three local DNS servers and nearly 73% of the suspected clients (with known local DNS servers removed) have an outdegree less than 4 and an indegree of 0. Figure 3 shows the distribution of outdegrees of the suspected clients. As can be seen, a vast majority has a low outdegree as expected but there are several outliers. Clients whose outdegrees are greater or equal to 4 are likely to be local DNS servers that talked to another local DNS server, or machines that are configured to run as local DNS servers and have users generating DNS queries, or hosts that are involved in probing activity.

C. Other graph analysis

Besides the basic analysis, we also did a preliminary exploration of other properties of the graphs at hand, such as degree statistics, connected components and graph structure. Two characteristics are obvious. One is the similarity to large random graphs, in which there are many connected components, most small with the numbers trailing off as the size of the components increase, followed then by one large “monster” component containing most of the graph. The other characteristic is, not surprising, the prevalence of fan-like structures among the small components.

VIII. APPLICATIONS

Once we have categorized the IP addresses, we can explore a variety of applications that can be built on top of the lists. The identification of suspected local DNS servers alone is extremely valuable in the context of Content Distribution Networks (CDNs). Most CDNs base their redirection decision on the location of the local DNS server of a client (and not based on the location of the client). This

TABLE VI
WEB/DNS DATA EXAMINED FOR CLUSTERING APPLICATION

	Server log	Suspected LDNS	Known DNS
IP addresses	7,652,670	564,077	58,527
# of Clusters	114,928	69,923	20,736
# of busy clusters	43,164	–	–

requires CDNs to map the location of such servers within the Internet. Knowing a superset of local DNS servers in the Internet greatly reduces the number of IP addresses which have to be mapped. Using our data it seems sufficient to precisely map 564,077 IP addresses, in contrast to all IP addresses present in the Internet.

One application is dynamically choosing the best local DNS server. Currently a client selects a local DNS server either by using a static file (typically `/etc/resolv.conf`) or by using a DHCP (Dynamic Host Configuration Protocol) server. Although DHCP provides a way to dynamically choose a set of local DNS servers, it typically treats all servers in the selected set equally. On a large network with a diverse set of local DNS servers this may not be the right action. A mechanism that can help clients choose an appropriate DNS server, based on a set of criteria, would be beneficial. Helping clients choose a local DNS server close in terms of network proximity is obviously beneficial. Clients often make DNS queries followed by other actions, such as fetching a Web page. The administrator of the client system could use information gathered to make the proper choice of local DNS server and thus speed up client requests.

We use the network aware clustering technique outlined in [4] to cluster the set of IP addresses representing clients in a large portal Web site server log. A total of 104,018,140 requests were received at the Web site from over 7.6 million unique IP addresses. The IP addresses were clustered using 441,230 unique BGP prefixes gathered by merging 14 separate routing table snapshots in May 2001. We examine only *busy* clusters: clusters that were responsible for a reasonable fraction of the requests to the Web site. By sorting the number of requests emanating from each cluster and selecting the top few clusters we end up with a large fraction (70%) of the overall number of requests. A total of 43,164 busy clusters generated over 72.8 million requests.

We then cluster the IP addresses in the suspected local DNS server list (sLD) and the known local DNS server list (kLD) separately. We examine how many of the clusters of the suspected (known) local DNS servers are found in the set of busy clusters of the Web clients. Each match indicates the presence of at least one suspected (known) DNS server in the busy clusters. Table VI presents the raw numbers of the cluster experiment. By plotting the percentage of matches found in increasing fractions of the set of busy clusters of Web clients we can get a relative measure of busy clusters that include a suspected (known) local DNS server. Figure 4 shows the relative match between both the suspected and known DNS server clusters against clusters of the Web clients. This is a lower bound on the number of local DNS servers present in these clusters.

The bump in known local DNS servers graph is not present in the suspected local DNS servers. We surmise that the top few of the busiest clusters are likely to have proxies or spiders (which typically generate a lot more requests than ordinary clients). Such servers often also serve as local DNS servers and are typically configured to not respond to probe queries from outside. The fact that the *netflow* data sees a lot of DNS servers belonging to the busy clusters is understandable. Clients belonging to busy clusters access a wide variety of servers and at least some of their DNS traffic will traverse the backbone and are likely to be seen in our *netflow* data. The steady decline in number of local DNS servers that we see in the less busy clusters is also not surprising: their clients' Web traffic to the portal site is low and thus their DNS requests are less likely overall to traverse our backbone.

IX. SUMMARY

We have presented a methodology of large scale Internet measurement and analysis that greatly benefits from efficient processing software, a graph representation, and a set of generic library routines that helps query the graph for several DNS-specific properties. Our goals were to convert the large amount of data obtained by the *netflow* tool to a terse graphical representation that serves as a base for a variety of applications. The applications, such as locating local and authoritative DNS servers, can be constructed efficiently. Answers to a variety of questions about the structure of the graph, and properties of nodes and edges, that have domain-specific meaning can be obtained quickly. We use simple attributes of nodes, like in- and outdegree, to easily characterize the set of IP addresses. There is little need to resort to the much larger and somewhat unwieldy format of the output of *netflow*. The reduction in size of the data without loss of significant information and the ability to mine the graph rapidly suggests that this paradigm of traffic mining might be useful for other network traffic datasets as well.

With a high degree of probability we have characterized

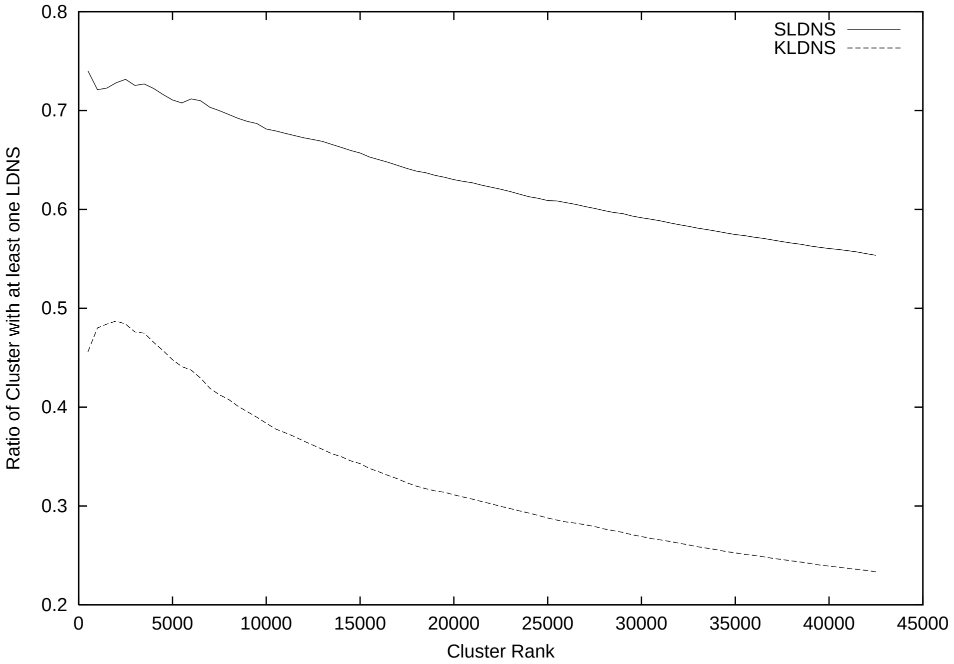

Fig. 4. Looking for DNS servers in busy clusters

the set of IP addresses involved in the flow to be one of a DNS client, local DNS server, authoritative DNS server, or an outlier such as a suspected zombie. Using available tools (such as *dig*) or external datasets (such as zone files of top level generic domains) we have verified that our characterization is correct. Using Web server logs from a portal site and the network aware clustering technique, we have been able to place local DNS servers in busy clusters, with applications in deployment of content distribution networks.

Our entire process is automated with scripts used to convert the *netflow* data to graphs and a tailored graph library software to efficiently mine the graph. A variety of other applications are being constructed using the graph.

At present, the size of the graphs, and the characteristics of the problems and the machines we are using make our simple approach feasible. As the graphs grow, other tactics will become necessary. Some are simple, such as making use of 64-bit hardware, using the monotonic and uniform memory allocation pattern to avoid unnecessary space overhead, or reducing the number of intermediate (especially, text) representations. More sophisticated improvements include the use of external memory algorithms (e.g.,[9]) and variable record formats to reduce disk space usage.

The approach presented here can be generalized to deal with other analyses of network data for which one can capitalize on the duality of relational data and attributed graphs. If the relevant questions deal primarily with an underlying binary relation and some associated attributes, and, by its nature, this is often true of network data, then one can extract the associated graph, reducing the data size, and then apply the appropriate graph operations. In the case of *netflow* data, given the fixed data schema and uniformity of the analysis tasks, it should be possible to parameterize the process for other analyses. Thus, one should be able to describe a desired model, requiring certain data fields in a *netflow* record, and the actions on the model, and have the extraction and analysis tools generated semi-automatically.

In future work, we plan to locate invalid delegates (also known as lame delegates in DNS parlance), machines that either do not exist or have been improperly registered as an authoritative DNS server for a domain. Currently, there is no known technique to automatically identify these. However, the graph representation will help generate a narrower set of suspected lame delegates when we use the sequencing of request information.

X. ACKNOWLEDGMENTS

We thank Randy Bush for several useful conversations that clarified important operational issues and oddities that we observed. We thank Carsten Lund for his initial assistance with the *netflow* data.

REFERENCES

[1] B. Krishnamurthy and J. Rexford, *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measure-*

ment. Addison-Wesley, May 2001. ISBN 0-201-710889-0.

[2] B. Krishnamurthy, C. Wills, and Y. Zhang, "On the use and performance of content distribution networks," June 2001. Under submission.

[3] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proceedings of IEEE Infocom 2001*, 2001.

[4] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *Proceedings of ACM SIGCOMM*, August 2000. `http://www.acm.org/sigcomm/sigcomm00/program.html`.

[5] M. Grossglauser and B. Krishnamurthy, "Looking for science in the art of network measurement," in *International Workshop on Data Communications*, September 2001. Taormina, Sicily, Italy.

[6] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," in *Proc. ACM SIGCOMM*, (Stockholm, Sweden), August 2000.

[7] "Netflow." `http://www.cisco.com/warp/public/732/netflow/index.html`.

[8] K.-P. Vo, "Cdt: A Container Data Type Library," *Software Practice and Experience*, vol. 27, pp. 1177–1197, 1997.

[9] J. M. Abello and J. S. Vitter, eds., *External Memory Algorithms*, vol. 50 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, DIMACS, 1999.