

Preprocessing DNS Log Data for Effective Data Mining

Mark E. Snyder

Department of Computer Science
Missouri S&T
Rolla, MO 65409, USA
Email: mark.snyder@mst.edu

Ravi Sundaram

Department of Computer and Information Science
Northeastern University
Boston, MA 02115, USA
Email: koods@ccs.neu.edu

Mayur Thakur

Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
Email: mayurthakur@google.com

Abstract—The Domain Name Service (DNS) provides a critical function in directing Internet traffic. Defending DNS servers from bandwidth attacks is assisted by the ability to effectively mine DNS log data for statistical patterns. Processing DNS log data can be classified as a data-intensive problem, and as such presents challenges unique to this class of problem. When problems occur in capturing log data, or when the DNS server experiences an outage (scheduled or unscheduled), the normal pattern of traffic for that server becomes clouded. Simple linear interpolation of the holes in the data does not preserve features such as peaks in traffic (which can occur during an attack, making them of particular interest). We demonstrate a method for estimating values for missing portions of time sensitive DNS log data. This method would be suitable for use with a variety of datasets containing time series values where certain portions are missing.

I. INTRODUCTION

The Domain Name Service (DNS) is an example of a system that is highly susceptible to denial of service (DoS) type attacks. Numerous examples have been documented, including [8]. When analyzing DNS traffic, capturing log data and using data mining techniques to discover trends and other knowledge can lead to the development of tools and techniques for preventing such attacks[4][5]. During the data mining process, care must be taken to verify the quality of the source data. In the case of DNS log data, one deficiency that arises is the presence of holes in the data where for one reason or another the DNS server failed to log the activity during a period of time. Due to the difficulty of obtaining log data, it is often not possible to simply obtain another dataset, and data must be cleaned before it is used to draw conclusions.

Our dataset was obtained from a large company central to the management of the DNS network. The dataset was collected at 26 root servers over a 10 day period in January 2004. For each server, we have log data that includes all DNS requests made from client DNS servers. The log files consist of over 400 gigabytes of raw DNS log data.

A variety of methods for adjusting data to account for these outages are available, such as linear interpolation. However, such methods fail to preserve features common to DNS traffic, namely the correlation of traffic patterns from day to day. The method we illustrate in this paper involves a form of imputation. Aggregating the data first and then adjusting the aggregate values provides a balance between storage, speed in

getting answers to queries, and the level of accuracy desired for our analysis.

II. RELATED WORK

Numerous works have discussed the topic of filling missing values in data sets being used for data mining. In [6] many techniques are described, including multiple imputation and hot deck imputation. These terms refer to generating a value to stand in for the missing value while the data set is being processed, and then continuing to process the data set as if the stand-in was the observed value. This is similar to the process we are using, except that we do not use stochastic or statistical techniques, but rather knowledge of the expected pattern of a DNS server and scaling based on other real observations in the time series to generate replacement values. This can be considered a pattern- or model-based imputation technique.

Other sources evaluate the value of these techniques, such as [2][7]. We could certainly analyze the variance and quality of our process using similar techniques but do not for brevity.

Numerical analysis techniques such as interpolation and linear regression are also a valuable reference, but many of these techniques do not incorporate knowledge of the expected behavior or in their generality become too expensive to incorporate.

Other sampling techniques, such as listwise deletion[1] are not acceptable since nearly every sample has some amount of missing data, and the data we have is judged to be useful in our problem domain even with the holes. Similar issues were discussed in [9].

Statistical methods have been applied to a wide range of problem domains, from the analysis of large sensor networks, the study of financial market trends, and for predictive purpose such as occurrence of diseases such as cancer. Our contribution provides a technique for handling missing data in large datasets using patterns that occur over periodic time intervals with high correlation.

Data mining concepts and techniques are elaborated in [3]. Our efforts focus on the data cleaning stage of knowledge discovery, with elaborations into how the resulting dataset was mined.

III. DATA SUFFICIENCY

There are two reasons why we might perceive a hole in the server data, either (a) the server became unavailable due to power failure, communication failure, or for some other reason, or (b) the server just appeared to be unavailable because we simply do not have the logged data during the time period.

One method of explaining a hole would be to examine the server traffic, specifically the hour of data following a hole. Within the algorithm used by clients to load balance across DNS servers, sometimes referred to as *explore and exploit*, when a DNS server becomes unavailable, clients and client name servers bound to that DNS server will start utilizing other DNS servers instead. The unavailable DNS server will receive a heavy penalty for not responding, but will still be polled periodically. For this reason, the log data should show that when the DNS server becomes available once more, many clients will have begun relying on other DNS servers, but assuming the DNS server comes back online with the same responsiveness it had before, eventually it should begin to receive traffic volumes consistent with the pattern established prior to it becoming unavailable. Thus, if a hole was due to a server becoming unavailable, then the time period following the hole should show a notable reduction in traffic from what a reasonable projection would have predicted. If this is the case, then we should discard the data from this server as not a candidate for analysis. There were no such patterns observed.

Another method would be to examine client traffic, specifically the traffic during the time of the hole. If the hole were caused by reason (a) then the aggregate client traffic across all servers to which it is bound should remain consistent as the client simply looks to other DNS servers to fulfill its needs. However, if the hole were due to reason (b) then there should be a drop in traffic to the server with the hole while the traffic to other servers to which the client is bound should remain statistically consistent with their previous pattern.

IV. FILLING HOLES IN SERVER DATA

We observe a statistical correlation between days of DNS server traffic on all DNS servers. In other words, the behavior on Tuesday is likely to be very similar (above 90% correlation) to Monday. These patterns typically hold on weekends as well, however the volume drops sharply on all servers on Saturday and Sunday of each week. Our technique uses this business knowledge to fill holes. This technique provides better results than simple interpolation because it preserves features such as peaks and valleys that would be lost if we used interpolation.

The algorithm used for filling the holes in the server data requires several steps. The first step was to count all traffic that each server received for each day and hour of the analysis period. For our data, this amounted to 10 days of data.

The next step was to identify which hours provide complete data, which hours are missing, and for hours with partial data, how many minutes of data are present. The method of data capture made this relatively easy. The log files for each server provide a continuous record of all requests while the log file

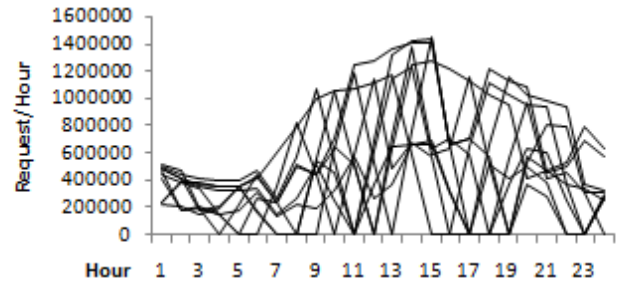


Fig. 1. Diagram showing request volume (unadjusted) for one server.

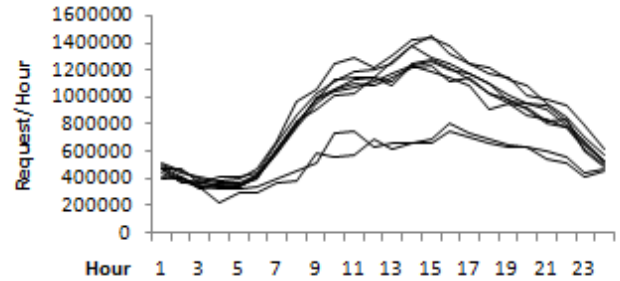


Fig. 2. Diagram showing request volume (adjusted) for one server.

was being created. So each log file was examined and the date-time stamp for the first and last entry was recorded. Once these start and end date-time values were sorted and examined, a value from zero to 60 was assigned to each server-day-hour that represents the number of minutes of data available.

The next step was to sort the days by the overall reliability of the day relative to the rest of the days. In this way, we process days from most reliable to least reliable. A future step will use the pattern established by more reliable days to compute an adjusted value.

The next step was to calculate a scaling factor from zero to one for each server-day-hour, and then compute an adjusted traffic value. This was calculated as the number of minutes of data available divided by 60. For complete hours, this results in a scaling factor of 1.0. For missing hours, the scaling factor is 0.0. One optimization that was applied to the algorithm that is designed to improve accuracy for “light” clients (which could be observed to have only one request per hour in some cases) was to consider any server-day-hour with less than 30 minutes of data available as unreliable and so the scaling factor for these data points was set to 0.0 as well. Once the scaling factor was determined, the adjusted traffic value was simply computed as the scaling factor times the original traffic value.

Processing each day in order from most reliable to least reliable as determined above, the next step is to generate the imputed values. For the first (most reliable) day, we calculate the adjusted traffic value by simply interpolating between known values. If the first (or last) hour of a day was missing, the first (or last) available adjusted value present was used.

Once the first day has been repaired by computing an adjusted traffic volume value, we moved on to successive

less reliable days. For holes in each day, we calculated new volumes as the sum of all non-missing hours this day times the sum of values for this hour for days already processed divided by the sum of all non-missing hours for days already processed.

At this point, the data represents a complete picture of the nature of the traffic for the server with the holes filled in. Figure 1 shows a graph of the original traffic from the available log files for one server. Figure 2 shows a graph of the same server, after the data has been adjusted to allow for the holes. As a pleasant surprise, the graph of the adjusted traffic volume values has fewer and less erratic ripples while still preserving the features one would expect to be present. For example, near hours 14 and 15 on day 8, the original data shows a hole. But in the cleaned data, this range is shown as a peak in the data, which is what we would expect to happen during this range. This feature is present in a more reliable day, and so is extended into the day with missing hours, resulting in a pattern much closer to our business understanding of the way it would look.

As can be seen, since the formula adjusts for the traffic pattern for a day in general, adjusted based on the observed volume for a specific day, even on days with lower traffic, the pattern of traffic is preserved. This is illustrated in Figure 2 by the two relatively lower volume plots, which represent traffic on Saturday and Sunday.

Since clients are predominantly tied to a particular server (or set of servers), a hole in the server data would generally correspond directly with holes in the client data. Another way to say this is to say that if a client is communicating with a server and we are missing data for the server, then it is reasonable to assume that some of the missing traffic came from that client and so its traffic value will need to be adjusted in proportion to the adjustment made to the server traffic.

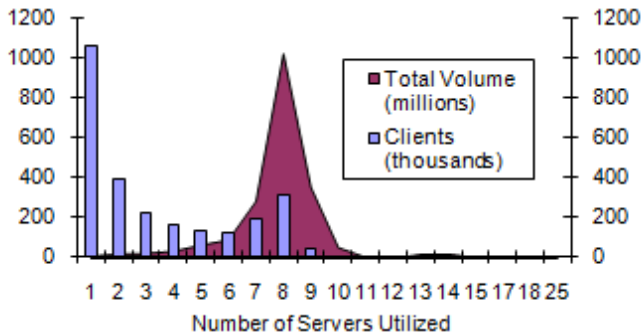


Fig. 3. Diagram of client polygamy (number of servers used by clients).

V. CLIENT TRAFFIC ANALYSIS

With the server traffic volume data in hand, we began to analyze the client traffic. The method used to adjust the client data parallels the adjustments made to the server traffic and uses those results. The primary difference between adjusting the server traffic and adjusting the client traffic is that there are many more clients (over 2.5 million) to process than there

are servers (26), and since each client must be processed for each server to which it is bound (this is where client polygamy becomes an issue), this increases the list to over 8.5 million. This is due to the fact that the bulk of clients talk to as many as 8 different servers for which we have data (see figure 3).

The traffic for different clients can exhibit very different patterns. For the client traffic analysis, it is helpful to aggregate numerous clients together in order to start to see similar patterns form. The trick is to try to be clever so that clients with a similar pattern are grouped together. Intuition tells us to start by grouping high-volume clients together, and low-volume clients together. Later, we can verify our intuition by analyzing the variance of certain clients from the aggregate pattern developed. The approach we took for grouping clients involves examining the traffic volumes calculated for each client and assigning a percentile rank. From there, several options are possible for aggregating the clients. First, we could sort the clients based on the total volume each client generates and then divide them into 10 even groups where each group contains the same number of clients. The problem with this approach is that the highest-volume clients will be disproportionately represented over the lower volume clients. The grouping we used involves dividing the groups into deciles (groups of 10 percentiles) based on cumulative traffic volume (see table I). Each decile group represents roughly 10% of the total volume of requests. Even though the number of clients in each group varies, each group is equally represented based on volume of traffic.

Decile	Clients Per Decile	Min Volume	Max Volume	Aggregate Volume
1	2477307	0	1379	202886210
2	80322	1380	4708	202806972
3	28813	4709	10618	202825584
4	13958	10619	20051	202841854
5	7756	20052	34542	202830513
6	4639	34543	55950	202843126
7	2930	55951	87116	202820052
8	1860	87117	138966	202893117
9	1177	138967	215926	202803433
10	440	215927	6901682	202833014

TABLE I
CLIENT DECILE GROUPS EACH REPRESENTING (ROUGHLY) EQUIVALENT CUMULATIVE TRAFFIC VOLUME.

Once the client decile groups have been established, we aggregate the traffic for each client decile group by server. At this point, each server must be checked for a hole. If a hole exists, then the volume for that hour must be prorated to account for the missing traffic (from the discussion previously on data sufficiency, we are only working with data from servers with holes we believe are due to missing log data). The entire algorithm is outlined below.

Step 1: Aggregate volume for the clients in the client decile group by server and for each server, look up minutes of data and scaling factor values.

Step 2: Process servers individually. First, sort days based on server minutes of data available.

Step 3: Examine each decile-server-day-hour value and compute adjusted volume. As before, the first day is handled differently than the others. Where a scaling factor is greater than zero, we multiply the scaling factor by the volume to get the adjusted volume. If hours are missing at the beginning and the end of the first day, we simply copy values from the first available or last available hour, respectively. If hours are missing in the middle of the data for the first day, we simply interpolate the missing values from the prior and next available hours.

For subsequent days, we revisit the algorithm and results from server traffic analysis. There are two cases to consider when examining a given client-server-hour volume value. If the server-day-hour scaling factor is non-zero, this means that the traffic value we have for this decile-server-day-hour must be multiplied by the scaling factor to get an adjusted decile-server-day-hour value. The volume could be zero if the clients in this decile group generated no traffic to that server during that hour, in which case the adjusted value will still be zero.

If the server-day-hour scaling factor is zero, this indicates that either we have no data from the server for this hour or the number of minutes of data for that server-hour is less than our established threshold of 50%, in which case we considered this an unreliable hour and treated it as if it was a missing hour. This means that we imputed the value for the server traffic so we must likewise impute the value for the decile-server-day-hour. For a particular decile-server-day-hour value, the formula to compute the adjusted value is to add up the decile-server-day-hour values for this day where the server scaling factor is not zero, multiply that times the sum of the adjusted server volumes for this hour, then divide the result by the sum of the adjusted server volumes for hours with non-zero scaling factors this day. As with the server adjustment process, this gives an updated set of decile-server-day-hour traffic volume values that takes the volume of traffic of the day as well as the relative traffic volume of this hour of the day from other days into consideration. In order to ensure reliability, we processed each day in order from most reliable to least reliable based on total minutes of log data available for each day.

As with the server analysis, there is a dramatic improvement in regularity between the adjusted and unadjusted datasets. Figure 4 shows the aggregate traffic for one client decile group (consisting of the clients that generate the top 10% of the total volume of traffic) across all of the servers to which it is bound (for which we have data).

As before, we next examine figure 5 which shows the traffic for the same group of clients once it has been adjusted to account for the holes in the server data using the technique described above.

VI. DOMAIN TRAFFIC ANALYSIS

Next, we analyze the traffic for each domain name requested. The method used to aggregate and adjust the domain name data is nearly identical to the adjustments made to the client traffic and again uses the results of the server analysis. The only difference between adjusting the domain name traffic

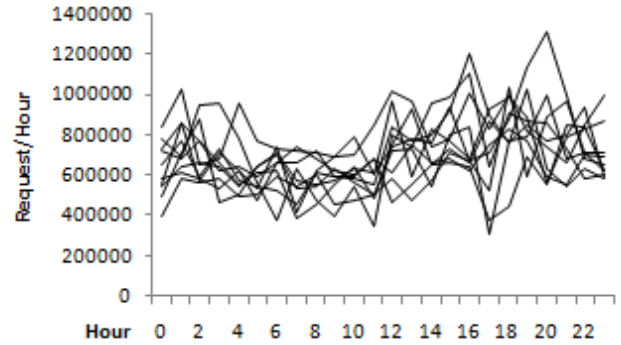


Fig. 4. Traffic for heaviest client decile group.

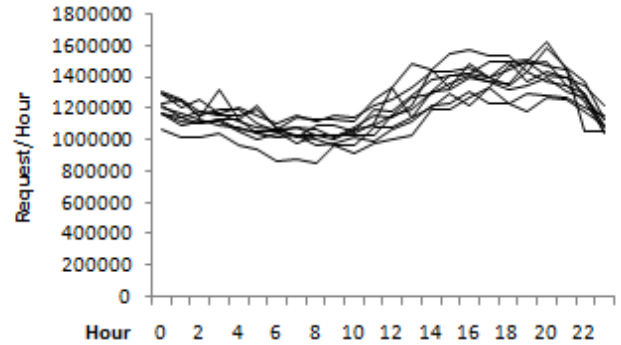


Fig. 5. Adjusted traffic for heaviest client decile group.

and adjusting the client traffic is using the domain name column instead of the client IP address column in the database. As with the client processing, there are many domain names (over 3 million) to process.

For the domain name traffic analysis, it is once again helpful to aggregate numerous domain names together in order to examine the patterns between like domain names. As with the client processing, we group high-volume domain names together and low-volume domain names together. The grouping we used involves dividing the domain names into deciles (groups of 10 percentiles) based on cumulative traffic volume (see table II). Each decile group represents roughly 10% of the total volume of requests. As with the client decile groups, the number of domain names in each group varies, but each group is equally represented based on volume of traffic. A slight difference with the domain names, however, is that there is more of a discrepancy between the volume requested of high frequency domain names and low frequency domain names. In the table, the top three deciles are dominated each by only one domain name, and the fourth decile aggregates only two domain names. This leaves a disproportionate number of domain names in the tenth decile. However, for our purposes, this doesn't constitute a significant problem and we could always reportion the domain names into alternate groups manually.

Once the domain name decile groups have been established, we aggregate the traffic for each domain name decile group

Decile	Domains Per Decile	Min Volume	Max Volume	Aggregate Volume
1	1	265940087	265940087	265940087
2	1	142193910	142193910	142193910
3	1	121219666	121219666	121219666
4	2	94370237	116432237	210802474
5	4	42150988	91905649	252276144
6	7	27443752	37836078	217956083
7	9	18378494	27057672	203110531
8	20	6499397	15942293	208851647
9	50	2095633	6496745	201621665
10	3044001	1	2062561	204411668

TABLE II
DOMAIN NAME DECILE GROUPS EACH REPRESENTING (ROUGHLY)
EQUIVALENT CUMULATIVE TRAFFIC VOLUME.

by server. At this point, each server must be checked for a hole. If a hole exists, then the volume for that hour must be prorated to account for the missing traffic (from the discussion previously on data sufficiency, we are only working with data from servers with holes due to missing log data). The entire algorithm is omitted, due to the fact that it is exactly the same as the algorithm used in client processing.

At the end of processing each of the server-day-hour values above we can begin to examine the traffic pattern for a “normal” heavy domain name. As with the client and server analyses, there is a dramatic improvement toward expectations between the adjusted and unadjusted datasets. Figure 6 shows the aggregate traffic for one domain name decile group (consisting of the domain name in the first decile that generates the top 10% of the total volume of traffic) across all of the servers from which the domain name was requested (for which we have data).

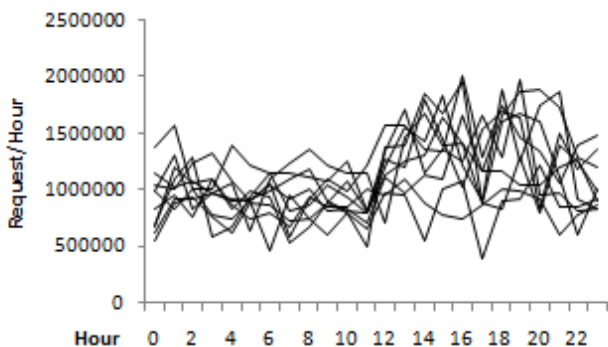


Fig. 6. Traffic for heaviest domain name decile group.

As before, we next examine figure 7 which shows the traffic for the same domain name once it has been adjusted to account for the holes in the server data using the technique described above.

VII. CONCLUSION

We started with a set of raw DNS logs capturing volumetric data regarding frequency of requests of a set of DNS servers. However, this data was not useful in its raw form because

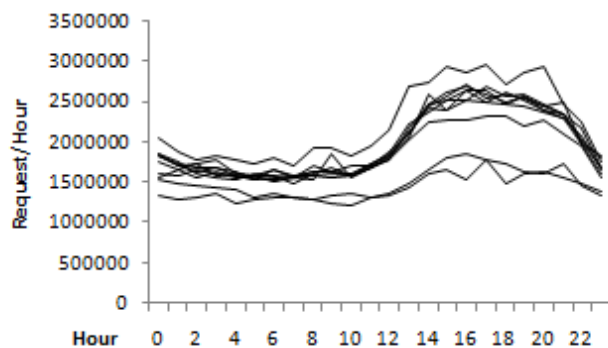


Fig. 7. Adjusted traffic for heaviest domain name decile group.

of gaps in coverage. After preprocessing the data using the demonstrated model-based imputation technique we were able to perform robust, useful analysis against the cleaned data. This method can be applied to other problem domains involving time series data. We believe portions of this technique can also be adapted to real-time monitoring of streaming data that encounter periodic gaps or outages in coverage. By detailing a reproducible method for analyzing server log data to identify habitual patterns for the traffic processed by DNS servers, we have demonstrated a technique to assist in detecting attacks on Domain Name Service (DNS) servers that rely on statistical analysis.

REFERENCES

- [1] Paul D. Allison. *Missing data*. Sage Publications, Inc., Thousand Oaks, CA, USA, 2002.
- [2] Marvin L. Brown and John F. Kros. The impact of missing data on data mining. pages 174–198, 2003.
- [3] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2000.
- [4] Seong Soo Kim and A. L. Narasimha Reddy. Statistical techniques for detecting traffic anomalies through packet header data. *IEEE/ACM Trans. Netw.*, 16(3):562–575, 2008.
- [5] Keunsoo Lee, Juhyun Kim, Ki Hoon Kwon, Younggoo Han, and Sehun Kim. Ddos attack detection method using cluster analysis. *Expert Syst. Appl.*, 34(3):1659–1665, 2008.
- [6] Roderick J A Little and Donald B Rubin. *Statistical analysis with missing data*. John Wiley & Sons, Inc., New York, NY, USA, 1986.
- [7] Ingunn Myrtveit, Erik Stensrud, and Ulf H. Olsson. Analyzing data sets with missing data: An empirical evaluation of imputation methods and likelihood-based methods. *IEEE Trans. Softw. Eng.*, 27(11):999–1013, 2001.
- [8] R. Naraine. Massive ddos attack hit dns root servers. www.internetnews.com/dev-news/article.php/1486981, October 2002.
- [9] W. Eric Wong, Jin Zhao, and Victor K. Y. Chan. Applying statistical methodology to optimize and simplify software metric models with missing data. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1728–1733, New York, NY, USA, 2006. ACM.