An Approach for Computer Network Comparison

S TERRY BRUGGER
University of California, Davis

September 15, 2007

Abstract

Current metrics to quantitatively compare two computer networks are focused on performance aspects (latency and throughput), as these are of vital importance to the network engineering community. Other groups need a more holistic comparison, ideally one that can be quantitatively defined. This paper will examine prior work in the area of network comparison, present applications that require a more holistic comparison than what current techniques provide, and propose an approach for performing such a comparison.

1 Introduction

The network engineering community has a distinguished history of examining computer networks, measuring their performance, and comparing that performance in different situations. The preliminary metrics used for this work are latency (usually measured using the interpacket arrival times), and throughput (usually measured using the number of bytes transfered per second). While these are unquestionably useful measures, taken alone they fail to provide a holistic view of the network. By extension, taken alone, they can not provide a holistic comparison between two network traces. There are a number of applications, such as network forensics, protocol development, or traffic generation, that would benefit from the ability to perform a holistic comparison of two network traces.

This paper will begin by looking at the work to date in network comparison, then it will present six research and development areas that will benefit from a more advanced form of network comparison than what is available today. We will wrap-up by proposing a straw man for a methodology to perform such a comparison. Due to space limitations, we will not go into our initial experiments applying this methodology. This work should be viewed more as a survey designed to spurn further research in this area than a presentation of experimental results.

2 Previous work

There has been surprisingly little research on network simulation for the purposes of testing network based security systems. Most all of the network modeling and simulation work to date has been done by the network infrastructure community for the purposes of developing and testing more efficient queuing algorithms and new protocols. In this arena, the interest is primarily in flow rates and interarrival times. While these things are certainly important when simulating network traffic for security purposes – for instance to ensure that devices can handle normal network loads – it ignores much of the internals of the traffic, for example the application mix. When things like the application mix are considered, the focus is usually on the few applications that make up the bulk of network traffic flow, ignoring the low traffic applications that may be of considerable importance to security devices (such as RPC and SMB services).

When modeling the network flows for internetworking purposes, relatively statistical metrics have been sufficient to validate the models, and hence the correctness of the simulations created by those models. The advances in network modeling have been driven primarily by the use of better statistical measures of the network flows.

From the early days of computer networks, many assumptions were made about network packet flows, primarily that traffic was dominated by bulk transfer operations, which moved a large quantity of data, that interactive applications produced fairly symmetric flows, and that packets followed a Poisson distribution model at any given point in the network. In 1992, Danzig et al. made a number of discoveries that advanced our understanding and ability to model networks. Specifically:

- Different applications produce different packet interarrival times.
- Interarrival characteristics need to be modeled at the application level since network infrastructure changes can change them.
- Interarrival times are only interesting for interactive applications since they will change for bulk transfer applications with network improvements.
- Bulk data is not necessarily large or unidirectional, and interactive traffic isn't symmetric in size in both directions.
- The distribution of network endpoint pairs differs between applications

Danzig et al. measured a large number of traffic characteristics in the course of this research including packet interarrival times and size, session bytes transfered (in each direction and bidirectionally), duration and packets transfered, items per connection for bulk and item sizes, network pair distribution, probability of concurrent connections to same network or host, connection rate by time of day, and MTU.

Despite the large of parameters that they considered, the work is primarily focused on the interests of the network engineering community. In particular, the authors only considered the data portion of the connection, and ignored packets that did not produce a significant load, such as establishment, tear down, ack-only packets, retransmitted packets, and anomalous packets. Such packets are important from a security perspective however, and are likely useful to the network protocol community as well.

The following year Leland et al. released the paper (expanded in 1994) that set the stage for network research for the next decade by showing that Ethernet packet traffic (including traffic introduced from the Internet) had self-similar – or fractal – interarrival times. Specifically, the statistical nature of the interarrival times followed the same long-tail distribution at multiple time scales.

While certainly an interesting result with unquestionable ramifications for router development, this unfortunately caused an almost singular focus on fractal patterns for interarrival times within the network modeling community:

1. Erramilli et al. (1994) showed how chaotic maps could be used to model and produce the self-similar distributions of packet interarrival times.
2. Paxson and Floyd (1995) established that Poisson processes were useful for modeling the interarrival times of user-initiated sessions, but that other session interarrival times, and the packet interarrival times within all WAN sessions, are best modeled using self-similar processes.
3. Park (1997) classifies the research into the causes of self-similarity. Evidence was found for multiple sources, including packet queuing by network stacks, protocol reliability techniques, application characteristics, and the nature of the data itself being transmitted. Park notes that source based simulation is necessary for proper generation of network traffic because direct packet generation fails to capture the interactions between components which creates the fractal pattern in the first place.
4. Sikdar and Vastola (2001) went on to provide a mathematical formalism for why TCP, in particular its retransmission features, can create interpacket arrival times that follow a fractal pattern, even within an individual connection. In particular, they show that the self-similarity of a connection is directly proportional to the loss rate on the connection. They go on to hypothesize that this behavior may vary – however will not vanish – depending on the TCP implementation used.
5. Racz et al. (2003) expanded the work in (Sikdar and Vastola 2001) by looking at the dominate cause of self-similarity under different conditions. They found that synchronization between sessions was the dominate cause in low-congestion and low-loss networks. We assume that this is due to network queuing effects as noted in earlier papers. In the face of modest congestion, the TCP congestion avoidance protocol quickly becomes the dominate contributor to self-similarity. We find it interesting that Racz et al. looked only at the network layer and above, as the original self-similarity paper by Leland et al. considered all Ethernet traffic, hence implying that the self-similarity was an artifact of the data-link layer. Perhaps unknowingly, Leland et al. imply that

the Ethernet congestion avoidance protocol is a cause of this property when they suggest that the congestion avoidance protocol for ISDN-B be carefully analyzed to be robust in the face of fractal patterned traffic. We wonder if congestion avoidance at the data link layer was only a significant contributor to self-similarity when Ethernet was used in a shared segment topology, as in Leland et al., as opposed to the switched topology most likely used by Racz et al.. An interesting test would be to repeat Racz et al.'s experiments on a data link protocol such as token rink. In any case, once the network starts having a high loss rate, TCP's resend mechanism takes over for the congestion avoidance protocol as the primary contributor to self-similarity.

6. In a departure from the network engineering community, physicists interested in the nature of scale-free networks have become interested in the Internet. In Yook et al. (2002) the authors show that topology of hosts on the Internet follows a fractal (self-similar) pattern [1]. While they don't investigate the result this has on the network traffic, one wonders if the self-similar nature of the network topology is one of the causes of self-similar interpacket times for network traffic.

Floyd and Paxson (2001) followed up their seminal 1995 paper with an excellent examination of why simulating Internet traffic is so hard. The most valuable contribution of this paper is their list of invariants which have held for Internet traffic to date, and which are likely to continue to hold for the foreseeable future. Fortunately, it was here that we begin to see a break from a singular focus on self-similarity which, while certainly a factor in the list of invariants, is not the sole invariant. The paper also succinctly summarizes the important conclusions of the above research that proceeded it.

Mellia et al. (2002) made a huge push to look at the network traffic as more than just the interarrival time between the sessions and packets, and the related packet size, session duration and bytes transfered, or loss rate. They put together a tool (Tstat) that can produce 80 different graphs or plots of network performance characteristics from a network trace. This was a significant advance to allow an analyst to see the effect of various TCP/IP settings on performance; however the focus was squarely on performance and not security, and it did not provide any means to automatically model the network such that new traces could be produced with the same characteristics. For example, no mention is made of measures of the use of IP options or overlapping IP fragments, either of which may be used to attempt to break certain TCP stack implementations. Further, there does not appear to be any correlation between various metrics and the source or source / dest pair, which would be useful to create a trace with the same characteristics, and identify hosts with conflicting measures, such as a host appearing to use both a 64 and a 128 TTL, which may indicate it's attempting to probe the network, or may just be acting as a network address translation gateway.

The most complete system for modeling and validating network traces is presented in (Lan and Heidemann 2002). The authors build application level source models by hand, then parameterize them with distributions automatically built from network traces. The traces produced when the models are used for simulation are then validated using quantitative and qualitative statistical measures. This work pulled together many positive qualities, such as:

- They recognized that the models were the same for different networks, and that it's the parameterization that dictates the unique characteristics.
- They demonstrated the importance of application level parameters (in this case, the distribution of object sizes in HTTP).
- They use multiple quantitative statistical comparisons (Wavelet Scaling Plot and Kolmogorov-Smirnov Goodness of Fit Test) to validate the generated traces.
- For the Kolmogorov-Smirnov test they used the most restrictive critical value of significance so as not to dictate a distribution.
- Instead of just measuring the packet interarrival time, they also used the packet size, session duration, size, and interarrival, and the per-host interarrival and duration, as well as the protocol mix and traffic volume for comparison.
- Additional parameters were used for the models, specifically, TCP window size, Round Trip Time, and bottleneck bandwidth.

Nevertheless, their work left many questions unanswered:

[1] While the authors call it the "physical layout" of the Internet, link-layer protocols such as label-based switching abstract the network layer topology from the physical topology.

- While the use of quantifiable statistical measures for comparison was a large leap ahead of the standard Hurst parameter estimation that most work used in the decade that proceeded it, no justification was given for the use of the Kolmogorov-Smirnov test or the Wavelet Scaling Plot (although there is some precedent for the latter in the network engineering community).
- The presented results – particularly that the inbound and outbound traffic from a network, and the same network at two different times, and two different networks differ above the level of significance – are intuitive. Further, it seems convenient that the same statistical measures validate that the generated traces are below the threshold of significance for difference when compared to the real traces. What is missing is a demonstration of how different the traces must be before they are labeled as different. For instance, the authors show that a one hour trace starting at 14:00 is different from a one hour trace starting at 19:00. Is the one hour trace starting at 14:00 different from the one starting at 15:00? Intuitively, the same sort of activities should be happening on a network at those hours of the afternoon. Baring any reasonable explanation of why they should be different, if the statistical measures say that they're different, then we must question the usefulness of those techniques for trace comparison.
- They note that huge data sets do not exactly follow statistical distributions, so they use a random sample of 10000 data points for their comparisons (an accepted method in the statistical community). It seems that the outliers in network traffic are significant, and that ignoring them – especially if they are significant enough to skew a distribution – stands to loose vital characteristics of the network.
- In addition to the two quantitative comparisons, the authors did a qualitative comparison of the Cumulative Distribution Function (CDF); however humans have a tendency to find patterns where they don't exist. This did seem to serve well as a first order comparison to see if two distributions were obviously different.
- As the authors note, only HTTP and FTP were modeled, and a natural extension to their work would be to add additional protocols. We would be concerned, however, that modeling each protocol by hand does not scale with the growing number of application protocols in use on modern networks, and that failing to model infrequently used protocols, which may be fine when generating traffic for network engineering purposes, will pose problems for the network security community, where an increase of an infrequently used protocol is typically indicative of a new attack. Indeed, the choice to model FTP, with its higher percentage of bytes transfered over DNS, with its higher percentage of packets and sessions, demonstrates this bias in the work as presented.
- Even within the HTTP and FTP models that they used, they chose not to model HTTP clients which use multiple connections per page or the FTP control channel, both of which are bound to have an adverse impact on the use of such models to generate traces for network security testing.
- We also find it odd that their generated HTTP/FTP traffic matched the real traffic when the real traffic was filtered down to just the HTTP and FTP connections. By their own reckoning, a source model is necessary as the individual session traces will behave differently in the presence or absence of other network traffic, so it seems that traces generated in the absence of non-HTTP/FTP traffic should differ from the real traces.

Most recently, Bartoletti and Tang (2005) demonstrated the ability to characterize services based on the differences between vectors to the centroids of clusters representing the services. These representative clusters are produced by clustering thousands of connections to that service into a handful of clusters. Each connection is represented by seven TCP session characteristics. The authors demonstrate that the differences between the centroids of the clusters fall below a standard deviation for clusters within the same service, and are significantly larger between clusters representing different services. Some services are obviously more similar (HTTP and HTTPS) than others (HTTP and FTP). The authors speculate that this method could be used to identify the type of traffic in a tunneled connection, however they do not demonstrate the use of the method for session identification, which is potentially the most interesting application of this approach. Unfortunately, the authors do not have a measure for how many connections are necessary to get a representative cluster. Put another way, they do not know how the method performs given a small number of connections. We would be interested to see how this method compares with our proposed methodology (below), particularly in the area of network (rather than per service) comparison.

3 Usefulness of a More Holistic Comparison

There are numerous applications that would benefit from a more holistic quantitative comparison of two network traces:

1. Network Forensics
2. Network Administration
3. Network Application Development
4. Network Protocol Development
5. Network Traffic Generation
6. Network Intelligence

We'll look at each briefly.

3.1 Network Forensics

Given two traces of two different attack incidents, a holistic comparison of the traces will allow us to compare and contrast the attacks at a network level. Hopefully, this will allow some insight into the modus operandi of the attacks. This will go beyond a "Network Ballistics" type comparison (as in (Bartoletti 2004; Parno and Bartoletti 2004), which focuses on the tools used and their settings, to consider the set of tools used for the entire attack, and human factors such as typing latency if a shell connection is made.

3.2 Network Administration

Administrators of large networks can have a hard enough time dealing with issues such as network reliability and policy enforcement; frequently they don't have enough time to consider how – and more importantly, why – their network is changing. Some quick measurements from their gateway might tell them that bandwidth usage is up X% this month, but without knowing why, it's difficult to plan appropriately. A method to quantifiably compare a trace of the network from say, a month ago, to a trace today, could tell not only how much the network has changed, but what factors accounted for that change. Perhaps the change is due to a new file sharing protocol that policy dictates should be blocked. Or perhaps the change is due to an increase in video teleconferencing, which must take latency into account as much as bandwidth when planning for future capacity.

3.3 Network Application Development

Network applications are constantly growing in importance. For numerous reasons, application developers are moving away from monolithic desktop applications to server based applications, many of which are accessible from standard web browsers. Additionally, the number of applications that retrieve – particularly those that stream – data from the Internet, continues to increase. Many network centric applications are sensitive to changes in the network they're operating on, and may themselves change the behaviour of that network. It is likely that the application developer won't be able to see these changes on their local network that they use for development. Instead, they will need to take a snapshot of the network with and without their application running, in order to compare them to gain an understanding of how their application is affecting the network. Such a comparison tool would be of additional use in allowing the developer to see how changes to the application change the way that it interacts with the network.

3.4 Network Protocol Development

Closely related to network application development, is the development of network protocols. This may mean either the development of new protocols or modifications of existing protocols, at anywhere from the network to the application level. One might think of the difference as application development is concerned with how the use of established protocols impacts the network, whereas in protocol development we're concerned with how changes to the protocol itself will impact the network. This is something

that Floyd and Kohler (2003) and Medina et al. (2005) tackle in Floyd and Kohler and Medina et al., respectively. Both papers discuss what measurements are useful when doing network protocol development and how the choice of measurements has impacted protocol development. Our approach, as we will discuss, is to include all possible characteristics and determine relative relevance as one of the steps when quantifying similarity.

3.5 Network Traffic Generation

Just as protocol development was closely related to application development, so too is traffic generation closely related to protocol development, primarily because simulation provides a support on which much protocol development is done. Traffic simulation and generation are used for more than just protocol development though: they're used to test all sorts of things that interoperate with the network, spanning the gambit from software applications (such as intrusion detection systems), to hardware (such as routers).

It was actually this area that spawned the author's original interest in network comparison. Specifically, the generation of test data for intrusion detection systems. This is chronicled in (Brugger and Chow 2007).

3.6 Network Intelligence

Numerous organizations are increasing interested in understanding what is going on inside various computer networks. We'll dub this general category "Network Intelligence". Such work may be interested in identifying the type of traffic traveling over an encrypted connection[2], or perhaps we have a trace of some subnet and we want to see what other subnet it is most similar to for provisioning purposes.

There is also a much deeper, philosophical argument for Network Intelligence, in that understanding how networks work is an end in itself. While computer science at its core is interested in understanding how computers operate, network research at its core must be interested in understanding how networks operate at the network layer and above. Being able to compare two network traces in a quantitative manner is a step in this direction.

4 Proposed Methodology

Now that we've established the need for quantitative network comparison, we'll present a methodology which we believe will serve to provide such a comparison. The approach breaks down into two main stages: Derivation of characteristics and weights, which should need to be done once for any given application of the methodology, followed by the actual comparison of network traces, which can be done for any number of trace pairs. We conclude by enumerating the metrics that we can use to evaluate the correctness of this methodology.

4.1 Derivation of Characteristics and Weights

1 Define base cases The first step in the methodology is to collect or build a number of network trace pairs and define the similarity between the each pair of traces. The similarity must be in the range [0..1], and the assignment should be done by a subject matter expert. Each pair will now become a base case.

2 Define first-order characteristics The next step is to define the possible first order characteristics that we can derive from the data sources. Specify which of the characteristics are essential in that they are part of what uniquely identifies a connection; these will be characteristics such as host identifiers or service accessed. We will classify the characteristics as total number characteristics (such as total number of bytes sent out of the target network), discrete characteristics (such as the number of packets sent per host), or continuous characteristics (such as the number of connections established in the last 30 seconds).

[2] `ssh` presents an interesting conundrum to security aware organizations: it's use may be encouraged over protocols such as `telnet` so that adversaries can't see what's being sent over the connection, but it also means that the policy enforcement personnel can't see what the connection is being used for either.

Now run each of our base case test pairs through the following steps:

3 Determine distribution function for each characteristic For each of the characteristics defined above, build its distribution for the first trace:

- The total number characteristics will just be kept as a single value.
- For the characteristics that are labeled as discrete, just keep a table of values.
- For the continuous characteristics, begin with a least-squares quadratic function fit to the data points. The prior work shows that we will need to use more complex representations (such as a fractal distribution, Fourier or Principle Component analysis, wavelet modeling, etc) for some of the characteristics, so use such distributions as the analysis and expertise of a subject matter expert dictates.

4 Build the distributions for the second trace Build the distributions for the same characteristics of the second trace. If two networks differ only by size, this will be reflected in the differences between the total packet and byte counts, so we don't want it to also be reflected in the differences between distributions, such as the distribution of packets to privileged services over the past w seconds. To correct for this, use $\frac{\tau_1}{\tau_2}$ as a multiplier for any distributions (or the individual measurements used to build a distribution) built for the second trace[3]. This will allow us to compare distributions without worrying about one distribution having twice as much traffic as the other distribution. A potential problem with this approach that we will need to remain vigilant of, is that it will also scale traffic that is invariant with load (for instance an automated process that makes a single connection every hour). An alternative approach would be to scale based on the number of active hosts on the target network. Another alternative approach would be to use a size invariant distribution, such as a cumulative distribution function (CDF).

5 Determine similarity function for each characteristic Now we can calculate the similarity between the two distributions for that characteristic:

- For the total counts, use the formula

$$1 - \frac{|x_1 - x_2|}{x_1 + x_2}$$

  as the normalized similarity between the two. This is essentially the Jacquard Coefficient. Alternatively, it is like the percentage similarity (1− the percentage difference) between the measurements, with the variation that we divide by the sum of the values, rather than the mean of the values, as this will naturally constrain our values in the range $[0, 1]$ rather than $[0, 2]$ (hence, "normalized" similarity), as long as all of our measurements are positive numbers (which in the domain of network measurements, they should be).

- For discrete characteristics, use the average of our normalized similarity between the measurements:

$$1 - \frac{\sum_{i=1}^{n} \frac{|x_{(i,1)} - x_{(i,2)}|}{x_{(i,1)} + x_{(i,2)}}}{n}$$

  where $|x_{(i,1)} - x_{(i,2)}|$ is the absolute value of the difference between the i^{th} measurement of the first and second traces.

- For continuous characteristics, the similarity calculation will depend on the function being used to model the data. As long as we're using the quadratic function discussed above, we should be able to sample the curve at given points, and then use the same similarity formula as we do with the discrete characteristics to find the normalized similarity between the two curves. More complex representations will require further analysis to determine the proper similarity method.

Once we've performed the above steps for each of the trace pairs for the base cases, we must consider the usefulness of each characteristic, given its behavior across all the base cases.

[3]where τ_1 and τ_2 are total packet counts (in and out) for distributions based on packet counts, bytes for distributions built on bytes, etc

6 Find the weights for each characteristic The normalized similarity for each characteristic should perform as noted for all trace pairs in all base cases. This may require that the normalized similarity is multiplied by an adjustment factor. Such an adjustment factor may be non-linear. The determination of this adjustment factor will need to be made by an analyst, taking into consideration the target similarity value for each test case. The normalized similarity will be known as the scaled similarity once multiplied by the adjustment factor.

7 Drop non-differentiating characteristics If we are unable to find an adjustment factor which allows the normalized similarity to reach the target similarity value, for a significant number of base cases, we must determine if that is

1. due to a failure of the distribution algorithm used (for instance if it tells us that two traces are similar when they are not; however, equally likely in other situations),
2. a failure for a characteristic to change between different traces (be non-differentiating), or
3. or a data failure in two traces being more or less similar in some respect than intended.

If a characteristic does not differentiate a significant number of base cases (either because it is invariant, or it is random), then drop it.

8 Verify performance using all characteristics Take the scaled similarity for all the retained characteristics and average (find the arithmetic mean of) them. The average scores should be consistent with the intended scores. If it is not, then steps 5 and 6 (and possibly 2 through 4) will need to be revisited.

Now repeat the above steps for higher order characteristics.

9 Determine second-order characteristics and weights For any of the first order characteristics that were useful, repeat steps 2 through 7 above, treating it as a second order characteristic to each member of the set of essential characteristics (which was specified in the list of characteristics). For example, if the distribution of TCP connection wrong resend rates is useful, then try it as a second order characteristic, such as the distribution of TCP connection wrong resend rates on a per service (dest port), or source IP basis. This will allow us to compare if wrong resend rates on a network are due to a particular service or host. If the second order characteristic proves useful, integrate it with the others as above.

We choose to consider only higher order characteristics of lower order ones which proved useful, and limit them to be higher order to the essential characteristics only as a way of limiting our search space in this combinatorial problem. Granted, there is the possibility that there might be a higher-order characteristic this doesn't find, for instance between the relationship between the number of packets in the past n connections and the distribution of duration times in a given trace; however, if we did that, we would be more likely to find coincidences that have no basis in reality, or obvious relationships that provide no additional insight.

10 Determine higher-order characteristics and weights Continue with this process for higher order characteristics as long as useful characteristics are found. Expanding the above example, if the distribution of TCP connection wrong resend rates on a per service basis is useful, consider the TCP connection wrong resend rate per service per hour after midnight UTC or per destination IP.

4.2 Comparison of IP Network Traces

We are finally ready to use the characteristics and weights derived in the above steps to actually compare two network traces.

1 Build distributions of characteristics Use the function determined in step 3, above, to build the distributions for all characteristics in each trace. Adjust the distributions of the characteristics in the second trace by any multiplier determined in step 4, above, as necessary.

2 Find the scaled similarities for each characteristic Use the similarity function for each characteristic, determined in step 5, above, to find the normalized similarity between each characteristic. Apply the scaling factor determined in step 6, above, to each normalized similarity.

3 Generate overall similarity measure Average (find the arithmetic mean) the scaled similarities to determine the overall similarity measure between the two traces.

4 Generate report The order of the scaled similarities between characteristics, from lowest to highest, determines which were the most significant contributors for the differences between the two traces. These contributors, plus the overall similarity measure, will be compiled into a user-readable report.

4.3 Success metrics

This methodology seems fairly reasonable, however "fairly reasonable" is qualified to the point of not meaning anything; most people would not ride a roller coaster that is labeled as "fairly safe". The reason it seems fairly reasonable is that it is a methodology built from deductive reasoning, based on the assumption that the differences we care about in network traces can be quantified based on a comparison of the traces' characteristics. If this is true, we should be able to test the hypothesis inductively. To do this we have identified six success metrics:

1. A network trace and an anonymized version of the same trace show no difference.
2. The same network at adjacent times (without major sociological difference, such as the start of a workday) shows some minimal difference.
3. The same network at the same time, different weeks, shows some minimal difference.
4. The same network at disparate times (on/off hours) shows more significant difference.
5. Similar networks show some difference at the same time, and different networks (edu/edu vs edu/com) show larger difference.
6. Completely different traces (disjoint networks, disjoint times, disjoint services) show minimal similarity.

If an instance of the proposed methodology is able to pass all six success metrics, we can be confident that the methodology is sound.

5 Conclusion and future work

Hopefully by now, you are convinced of the need for a more holistic form of network comparison than what is currently available. We believe that the presented network comparison methodology should better serve the needs of the network forensics, administration, intelligence, application and protocol development communities, and allow for better artificial traffic generation. We are currently working on an implementation of the proposed methodology, and hope this paper will spurn further research and discussion in this area.

References

Bartoletti, A. and N. A. Tang (2005, 1 April). Characterizing network services through cluster-set variations. Technical Report UCRL-TR-211020, University of California, Lawrence Livermore National Laboratory, Livermore, CA.

Bartoletti, T. (2004, 24–27 May). Visualizations in hostile rapid scan forensics. In *DOE Computer Security Group Training Conf.*, Kansas City, MO. DOE. Available from LLNL Library as UCRL-CONF-204182.

Brugger, S. T. and J. Chow (2007, January). An assessment of the DARPA IDS Evaluation Dataset using Snort. Technical Report CSE-2007-1, University of California, Davis, Department of Computer Science, Davis, CA. `http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-1.pdf`.

Danzig, P. B., S. Jamin, R. Cáceres, D. J. Mitzel, and D. Estrin (1992, March). An empirical workload model for driving wide-area TCP/IP network simulations. *Journal of Internetworking 3*(1), 1–26.

Erramilli, A., R. P. Singh, and P. Pruthi (1994, 6–10 June). Chaotic maps as models of packet traffic. In *Proc. 14th Int. Teletraffic Cong.*, Volume 1, North-Holland, pp. 329–338. Elsevier Science B.V.

Floyd, S. and E. Kohler (2003, January). Internet research needs better models. *Computer Communication Review*.

Floyd, S. and V. Paxson (2001, August). Difficulties in simulating the Internet. *IEEE/ACM Trans. Networking 9*, 392–403.

Lan, K.-C. and J. Heidemann (2002, July). Rapid model parameterization from traffic measurements. *ACM Trans. on Modeling and Computer Simulation 12*(3), 201–229.

Ledesma, S. and D. Liu (2000, 21–25 August). A fast method for generating self-similar network traffic. In *Int. Conf. on Communication Technology (WCC-ICCT) Proc.*, Volume 1, Beijing, pp. 54–61.

Leland, W. E., M. S. Taqqu, W. Willinger, and D. V. Wilson (1994, February). On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Networking 2*, 1–15.

Medina, A., M. Allman, and S. Floyd (2005, April). Measuring the evolution of transport protocols in the internet. *Computer Communication Review*, 37–52.

Mellia, M., A. Carpani, and R. L. Cigno (2002, November). Measuring IP and TCP behavior on edge nodes. In *Proc. IEEE Globecom 2002*, Taipei. IEEE.

Park, K. (1997, 7–10 December). On the effect and control of self-similar network traffic: A simulation perspective. In *Proc. of the 1997 Winter Simulation Conference*, pp. 989–996.

Parno, B. and T. Bartoletti (2004, 9–13 August). Internet ballistics: Retrieving forensic data from network scans. Poster Presented at the 13th USENIX Security Symp., San Diego, CA.

Paxson, V. and S. Floyd (1995, June). Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Networking 3*(3), 226–244.

Racz, P. I., T. Matsuda, and M. Yamamoto (2003, 28–30 August). Contribution of the application, transport and network layers to the self-similarity of Internet traffic. In *Proc. of 2003 IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing*, Volume 2, pp. 760–763. IEEE.

Sikdar, B. and K. S. Vastola (2001, 21–23 March). The effect of TCP on the self-similarity of network traffic. In *Proc. of the 35th Conf. on Information Sciences and Systems*, Baltimore, MD.

Willinger, W., V. Paxson, and M. S. Taqqu (1998). *A Practical Guide to Heavy Tails: Statistical Techniques and Applications*, Chapter Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. Boston: Birkhäuser.

Yook, S.-H., H. Jeong, and A.-L. Barabási (2002, 15 October). Modeling the Internet's large-scale topology. *Proc. of the Nat'l Academy of Sciences 99*(21), 13382–13386.