Understanding Patterns of TCP Connection Usage with Statistical Clustering

Félix Hernández-Campos[1] *Andrew B. Nobel*[2] *F. Donelson Smith*[1] *Kevin Jeffay*[1]

[1]Department of Computer Science
[2]Department of Statistics and Operations Research
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/Research/dirt

Abstract – *We describe a new methodology for understanding how applications use TCP to exchange data. The method is useful for characterizing TCP workloads and synthetic traffic generation. Given a packet header trace, the method automatically constructs a source-level model of the applications using TCP in a network without any a priori knowledge of which applications are actually present in a network. From this source-level model, statistical feature vectors can be defined for each TCP connection in the trace. Hierarchical cluster analysis can then be performed to identify connections that are statistically homogeneous and that are likely exert similar demands on a network. We apply the methods to packet header traces taken from the UNC and Abilene networks and show how classes of similar connections can be automatically detected and modeled.*

1. Introduction

TCP is the standard transport protocol for most Internet applications and services. File-sharing, web browsing, email, instant messaging, and many other applications make use of the reliable, in-order transport service offered by TCP to communicate their data across the Internet.

Our goal is to understand how applications use TCP in the Internet today. There are three primary motivations for studying this question. First, and most pragmatically, to build a synthetic workload generator that can generate realistic application-level inputs for a network simulator or testbed that are provably representative of how a given link, such as a link in the Abilene backbone, is used. Networking research has long relied on simulation as the primary vehicle for demonstrating the effectiveness of proposed algorithms and mechanisms. We are advocates of the simulation philosophy described by Floyd and Paxson [6] of using application-level descriptions of network usage as opposed to packet-level descriptions to populate simulations. We observe that the networking community suffers at present from a dearth of valid, contemporary models of Internet applications. By understanding how applications use TCP in the Internet today, and more importantly, by developing automated methods to characterize how TCP is used, we will enable researchers to construct application-level models that are representative of the workloads found on any network on which they can obtain simple packer header traces.[1]

A second, related motivation is to develop a method that will allow one to measure and characterize, in generic application-level terms, how traffic differs from one network to another. That is, we seek an abstract characterization of how one network is used differently than another network and to frame this characterization in terms of the data objects carried on TCP connections. An additional and more refined motivation is that we seek to discover and characterize different patterns of use within a given application protocol such as HTTP. Given the large variety of uses of the web and the large fraction of network resources consumed by web traffic, rather than simply characterize web traffic as a single entity, we conjecture that it is useful to identify and classify specific uses of the web and develop source-level models of each dominant sub-class of HTTP traffic. That is, we claim that since different uses of HTTP place different demands on a network, rather than construct a source-level model of "HTTP traffic," it is better to develop source-level models of, for example, HTTP single-object file transfer traffic, HTTP interactive traffic, and HTTP streaming media traffic.

This motivates the final goal: to move from studying the impact of specific application-level protocols on the network to studying the impact of common uses of protocols. For example, instances of HTTP, FTP, and many peer-to-peer protocols are essentially used for unidirectional, single-object file transfer. Instead of characterizing these protocols individually, we would like to identify and characterize the "data transfer traffic" that exists in a network independent of the application-level protocol used to carry the traffic. That is, we would like to characterize the use of TCP for "streaming media traffic" across *all* application-level protocols and to do so similarly for "interactive traffic," "file transfer traffic" *etc*. Our thesis is

[1] Because applications are the ultimate sources of all data carried by the network, we use the terms *source-level* and *application-level* interchangeably when referring to models of how TCP is used.

In: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Atlanta, GA, September 2005, pages 35-44.

that the workload placed on a network by a given TCP connection is more a function of the structure (pattern) of how an application uses TCP than it is of the application-level protocol used. Thus we seek to characterize common patterns of TCP usage rather than common protocols.

This approach of characterizing *uses* rather than *protocols* is motivated in part by the fact that both a large and growing percentage of the traffic seen on networks is "unknown" in the sense that the traffic is carried on a connection using an unregistered port number, and that applications misuse the reserved port numbers of other applications to avoid detection or policing.

When constructing source-level traffic models, the only means of identifying applications is to classify connections by port numbers. This is because user privacy concerns dictate that it is inappropriate to record packet data beyond the TCP/IP header without the prior approval of users. For connections that use common reserved ports (*e.g.*, port 80), we can attempt to infer the application-level protocol in use (HTTP) and construct a source-level model of traffic generated by the application [6, 20, 22, 24]. However, this analysis technique does not scale because of the diverse and continually evolving set of applications. A recent measurement study we completed on the UNC campus indicated that while approximately 90% of the bytes on the UNC campus (91% of the packets) were carried by TCP, the four largest TCP-based applications that could be easily identified by "well-known" port numbers (HTTP port 80, FTP-DATA port 20), NNTP port 119, and SMTP port 25), accounted for *less than half* of the total TCP bytes. (Whereas just 5 years ago, HTTP traffic alone accounted for as much as 80% of the bytes on Internet backbones.) The vast majority of the remaining TCP bytes were carried on TCP connections using a very large number (tens of thousands) of unassigned TCP port pairs. Thus close to half of all the traffic observed on our network is unidentified. (Similar data has been reported for Internet2 [4].)

This paper presents an automatic method for understanding how TCP is used by applications. The key to our approach is to study *patterns* of TCP usage within individual connections. The thesis is that (1) fundamental usage patterns exist, (2) one can use knowledge of these patterns to measure and characterize uses of the network by users and applications without explicit knowledge of the application-level protocol(s) in play, and (3) that the identification and characterization process can be performed automatically.

Our approach to discovering and characterizing patterns is based on an abstract representation of a TCP connection that captures the dynamics of both end-user interactions and application-level protocols. The representation, called an *a-b-t* connection vector, models a connection as a series of request/response exchanges separated by inter-exchange think times (a bi-directional ON/OFF model). Network packet traces of TCP/IP headers are "reverse compiled" into a collection of *a-b-t* connection vectors that can serve as inputs to a statistical cluster analysis program that will classify connections into a set of abstract connection types based on a set of feature vectors defined over the space of *a-b-t* connection vectors. The resulting connection types, or traffic classes, correspond to application connections that are generating statistically homogeneous network usage.

The premise of the cluster analysis work is that while literally tens of thousands of port pairs are in use at any one time, the number of distinct patterns of application behavior that are in use is far smaller, most likely dozens. By representing connections as *a-b-t* vectors we can use statistical cluster analysis to identify connections that are generating "similar" patterns of source-level traffic. This eliminates the need for knowledge of the (sometimes proprietary) application-level protocol to deconstruct a connection and understand its behavior. By being able to identify clusters of statistically homogeneous connections, a researcher or practitioner will be able to characterize uses of their network and thus better understand the fundamental make-up and structure of traffic seen on their networks. For example, instead of seeing 20,000 active connections on seemingly random port pairs they can identify the 5-10 fundamental traffic classes present. Moreover, using knowledge of well-known applications that do use reserved port numbers, these traffic classes can be speculatively labeled (*e.g.*, *ftp*-like bulk transfer, small-object web-like request-response traffic, streaming media class 1 traffic, streaming media class 2 traffic, peer-to-peer class 1 traffic, peer-to-peer class 2 traffic, *etc*). In addition, the characterization of dominant traffic classes will enable one to simulate their networks and to vary the mix of traffic according to actual user-driven usage patterns in a controlled manner.

In the following we present some results from a comprehensive measurement study of TCP connections using *a-b-t* characterizations of TCP connections. We have applied our characterization method to a large collection of packet header traces taken from a variety of Internet locations. We illustrate how features of *a-b-t* traces can be used to identify clusters of similar traffic classes using a set of statistical clustering tools developed originally for the analysis of gene expression arrays.

We claim the *a-b-t* model is a natural step forward: it is simple to describe, interpret, and implement, but flexible enough to accurately capture a wide variety of existing applications *without knowing what those applications are*. More precisely, *a-b-t* models have the important feature that they capture the source-level (*e.g.*, above the socket-layer) behavior of applications and can be constructed

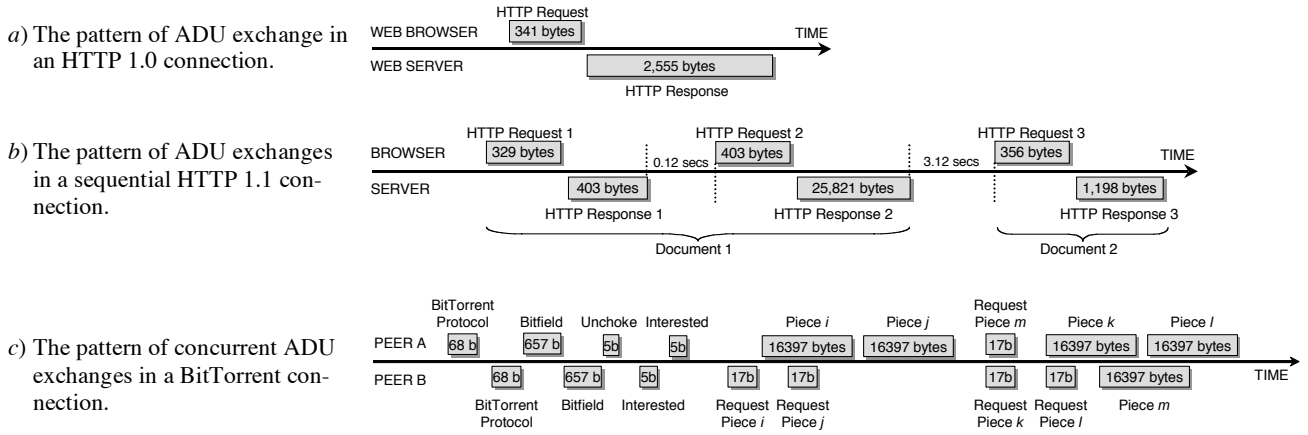

Figure 1: Examples of three patterns of application-level data unit exchanges taken from measurements on the UNC campus.

directly from packet level traces without making any *a priori* assumptions based on port numbers about the type of applications present. The generality of this model makes it possible to compute a set of meaningful statistics (*features*) for connections and to apply statistical clustering to group connections into a small number of *traffic clusters*. These traffic clusters group together connections with similar application-level behavior and usage patterns, and their study helps to understand the most important communication strategies that are used by Internet applications.

2. A Source-Level Characterization of TCP Connections

The foundation of our approach to modeling applications as network-independent entities is the observation that, from the perspective of the network, the vast majority of application-level protocols are based on a few simple patterns of data exchanges within a logical connection between the endpoint processes. Endpoint processes exchange data in units defined by their specific application-level protocol. The sizes of these application-data units (ADUs) depend only on the application protocol and the data objects used in the application and, therefore, are (largely) independent of the sizes of the network-dependent data units employed at the transport level and below. For example, HTTP requests and responses depend on the sizes of headers defined by the HTTP protocol and the sizes of files referenced but not on the sizes of TCP segments used at the transport layer.

The simplest and most common pattern used by TCP applications arises from the client-server model of application structure and consists of a single ADU exchange. For example, given two endpoints, say a web server and browser, we can visualize their behavior over time as shown in Figure 1a. A browser first opens an HTTP/1.0 connection to the server and sends a request for a specific object (*e.g.*, an HTML page or an image). This request is the first ADU in the data exchange. After the server receives the entire request, it replies with the requested object and closes the connection. This object (the response) is the second ADU in this connection.

We model the pattern of ADU exchanges within a TCP connection using a simple notion we call an *a-b-t connection vector*. Each TCP connection is represented as an n-dimensional vector $(c_1, \ldots, c_n)$ where n represents the number of discrete ADU exchanges, called *epochs*, in the connection. An epoch c_i is a triplet of the form $c_i = (a_i, b_i, t_i)$ that describes the sizes of data units exchanged and the duration of any idle time in the connection's i^{th} ADU exchange. An a_i represents the amount of data sent from the initiator of the connection (*e.g.*, a web browser) to the acceptor of the connection (*e.g.*, a web server) during the i^{th} exchange, while a b_i represents the amount of data sent in the reverse direction "in response to" the receipt of a_i. The time parameter t_i is used to model quiet times between data exchanges (epochs), which, if sufficiently long, represent application-level behavior, such as human think times and long application processing delays. Note that we do not record the time interval between the request and its corresponding response as this time depends on network or end-system properties that are not directly related to (or controlled by) the application or the user.

Thus a TCP application can be characterized as a process that generates a number of time-separated epochs where each epoch is characterized by ADU sizes and an inter-epoch time interval. For example, the HTTP/1.1 connection in Figure 1b represents a persistent HTTP connection over which a browser and server engage in a series of sequential request-response exchanges to download a pair of documents. A browser sends three requests of 329, 403, and 365 bytes, respectively, and a server responds to each of them with HTTP responses (including object content) of 403, 25821, and 1198 bytes, respectively. The second request was sent by the browser 120 milliseconds after the last segment of the first response was received

and the third request was sent 3,120 milliseconds after the second response was received. This connection would be represented as the *a-b-t* connection vector

(329, 403, 0.12), (403, 25821, 3.12), (356, 1198, 0).

Abstractly we say that the connection vector consists of three epochs corresponding to the three HTTP request-response exchanges. Note that the *a-b-t* characterization admits the possibility of an application protocol omitting one of the ADUs during an exchange (*e.g.*, epochs of the form (a_i, 0, t_i) or (0, b_i, t_i), are allowed). Single ADU epochs are commonly found in *ftp*-data and streaming media connections.

a-b-t connection vectors are capable of representing the request-response-like patterns of interaction common to a wide variety of application-layer protocols including SMTP and FTP-CONTROL, and most uses of HTTP/1.1, and NNTP. The essential invariant of request-response patterns is that endpoints of a connection do not transmit data concurrently. Connections satisfying this invariant are called *sequential connections*. A second class of connections allows for ADU transmissions by endpoints to overlap in time as shown in Figure 1c. This pattern of concurrent ADU exchange is not commonly found in TCP connections on the Internet today, but can occur in application-level protocols such as HTTP/1.1, NNTP, and BitTorrent. However, while uncommon, it is important to be able to model such concurrent connections as they often carry a significant fraction of the total bytes seen in a network trace (*e.g.*, 15%-35% of the total bytes in traces we have processed).

To represent concurrent ADU exchanges, the actions of each endpoint are considered to occur independently of each other so each endpoint is a separate source generating ADUs that appear as a sequence of epochs following a unidirectional flow pattern with one or more epochs of the form (a_i, 0, t_i) or (0, b_i, t_i).

3.1 Computing *a-b-t* connection vectors from measurement data

Modeling TCP connections as a pattern of ADU transmissions provides a unified view of connections that does not depend on the specific applications driving each TCP connection. The first step in the process is to acquire a trace of TCP/IP headers from a network link of interest. (We use *tcpdump* for this purpose.) The trace is then processed to produce a set of *a-b-t* connection vectors; one vector for each TCP connection in the trace. *a-b-t* vectors can be computed using either unidirectional or bidirectional traces. Here we primarily describe the (more complex) method for processing unidirectional traces.

a-b-t connection vectors for sequential connections can be computed from unidirectional traces. The basic method for determining ADU boundaries and sizes, as well as

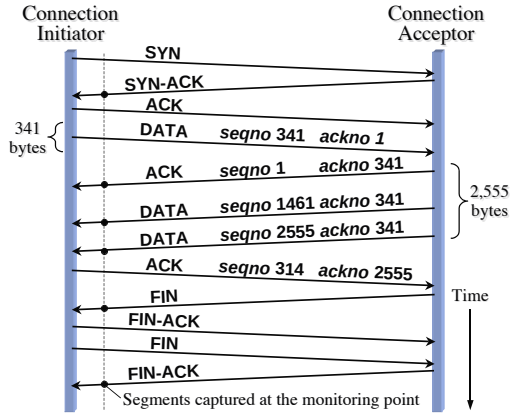

Figure 2: An illustration of the process of computing ADU boundaries and sizes from TCP sequence numbers and acknowledgement numbers in a unidirectional trace.

user think times, is described in detail in [24]. While this method was originally specific to the analysis of HTTP connections, we have generalized it to "reverse compile" arbitrary TCP connections into an abstract source-level model (an *a-b-t* connection vector).

The analysis proceeds by examining sequence numbers and acknowledgement numbers in TCP segments. Changes in sequence numbers are used to compute ADU sizes flowing in the direction traced and changes in ACK values are used to infer ADU sizes flowing in the opposite (not traced) direction. There will be an alternating pattern of advances in the ACK values followed by advances in the data sequence values (or vice versa). This observation is used to construct a rule for inferring the beginning and ending TCP segments of an ADU and the boundary between exchanges. Put another way, an advance in the data sequence numbers marks the end of an ADU flowing in the direction opposite to the traced direction and an advance in the ACK sequence number marks the end of an ADU flowing in the direction of the trace. Of course, other events can mark ADU boundaries as well. These include events such as the receipt of a FIN or RST segment, or the occurrence of idle times greater than a threshold.

Figure 2 shows a TCP time-sequence number diagram for the HTTP/1.0 connection shown in Figure 1a and illustrates the process of computing *a*'s and *b*'s. Timestamps on the *tcpdump* of segments marking the beginning or end of an ADU are used to compute the inter-epoch times and timestamps on the SYN segments are used to compute connection inter-arrival times. The complexity of this analysis (per connection) is $O(sW)$ where s is the number of segments in the connection and W is the receiver's maximum advertised window size. (The window size impacts the *a-b-t* connection vector computation process when reconstructing connections that experience lost,

duplicate, and out-of-order segments. See [24] for details.)

As a more complex example, consider again the pattern of ADU exchanges illustrated in Figure 1b. This example, taken from real measurement data, manifested itself in a packet header trace as 29 TCP segments (including SYNs and FINs). Applying the *a-b-t* connection vector computation method to this sequence of packet headers generated the 3-epoch connection vector listed above. Note that while the actual sizes and timing of the TCP segments represented in the original packet header trace are operating system and network-dependent, the analysis has produced a summary representation that models the source-level (browser and server) behaviors.

Application protocols that overlap rather than alternate ADU exchanges between endpoints (the pattern in Figure 1c) are not handled correctly by the above method. To fully detect and process a concurrent TCP connection, a bidirectional packet header trace is required. To detect instances of concurrent ADU exchanges in a TCP connection we look for instances in which both end points have unacknowledged data in flight. Specifically, we look for situations within a connection between end points *A* and *B* in which there exists at least one pair of non-empty TCP segments *p* and *q* such that *p* is sent from *A* to *B*, *q* is sent from *B* to *A*, and the following two inequalities are satisfied: *p.seqno* > *q.ackno* and *q.seqno* > *p.ackno*. If the conversation between *A* and *B* is sequential, then for every pair of segments *p* and *q*, either *p* was sent after *q* reached *A*, in which case *q.seqno* ≤ *p.ackno*, or *q* was sent after *p* reached *B*, in which case *p.seqno* ≤ *q.ackno*. Thus, every non-concurrent connection will be classified as such by our algorithm. Situations in which all the segments in potentially concurrent data exchanges are sent sequentially (purely by chance) are not detected by our algorithm and the connection is treated as sequential.

3. Clustering Communication Patterns

The *a-b-t* model provides a framework for the systematic identification and study of application-level communication patterns in Internet traffic. Internet measurement, modeling, and other research areas can benefit from the analysis and classification of such patterns. For example, the performance of transport protocols depends heavily on the patterns of data exchange within transport connections, hence an understanding of these patterns and their impact is needed for balancing among the tradeoffs that exist in protocol design. For instance, TCP can be tuned to provide better performance for transferring small ADUs at the price of higher instability and less fair allocation of bandwidth. One can analyze the benefits of this tuning by using simulations, however, only those simulations that make use of a broad and representative set of data exchange patterns in their inputs can help to draw general conclusions about the effectiveness of new network mechanisms.

Statistical clustering techniques (*e.g.*, [14, 17, 10]) provide a useful and flexible tool for grouping connections into traffic classes that represent similar communication patterns. Formally, a clustering scheme is a procedure that divides a given set of *d*-dimensional feature vectors $v_1, v_2, \ldots, v_m \in \mathbb{R}^d$ into *k* disjoint groups $S_1, S_2, \ldots, S_k$, known as clusters. The goal of clustering is to find a small number *k* of clusters such that feature vectors within the same clusters (a set of points in a *d*-dimensional space) are close together, while vectors in different clusters are far apart.

In our approach, each *a-b-t* connection vector, *i.e.*, each TCP connection, is first summarized using a vector of statistical features. Each feature captures some relevant characteristic of the connection, such as the number of exchanges, the total number of bytes sent by the initiator, the homogeneity in the sizes of the data units, and so on. Each feature is appropriately normalized so that its values lie between 0 and 1. We then measure the similarity between two *a-b-t* connection vectors by the similarity between their associated feature vectors. We consider two alternative distance measures, the standard Euclidean distance between vectors $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$,

$$d(x,y) = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{1/2}$$

and the Pearson correlation coefficient [10],

$$r_p(x,y) = \frac{S_{xy}}{\sqrt{S_x^2 S_y^2}}$$

where $S_{xy} = \sum_{i=1}^{d} (x_i - \bar{x})(y_i - \bar{y})$, $S_x^2 = \sum_{i=1}^{d} (x_i - \bar{x})^2$, $S_y^2 = \sum_{i=1}^{d} (y_i - \bar{y})^2$, and $\bar{x}$ and $\bar{y}$ are the mean values of *x* and *y* respectively.

Once the distance between each pair of *a-b-t* connection vectors has been defined, these vectors can be grouped using any number of standard clustering algorithms. We have applied a number of different clustering schemes to our data, but have focused on *agglomerative* and *divisive hierarchical* methods [10]. These methods have proven to be effective in other applications such as clustering gene expression arrays, and their graphical representation as trees (dendrograms) provides a useful way of identifying and analyzing groups of related communication patterns.

Figure 3 provides an overview of the basic steps in our methodology. As a first step in clustering source level communication patterns, we extract from each *a-b-t* connection vector a number of numerical features that are designed to capture important aspects of the two-way data transfer described by this vector. Let $v = (c_1, \ldots, c_n)$ be an *a-b-t* connection vector whose j^{th} epoch is given by the triple $c_j = (a_j, b_j, t_j)$, as described above. The most critical

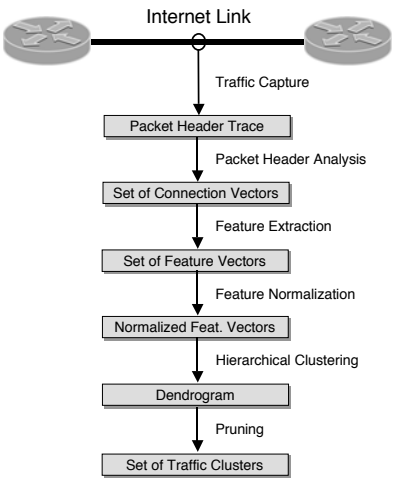

Figure 3: Overview of our approach for clustering patterns of TCP connection usage.

features of v are the number of epochs e, and the total number of bytes sent by each of the connection endpoints, a_{tot} and b_{tot}. Let $A = a_1, \ldots, a_n$ be the collection of a-type ADUs found in a connection. Other useful features include the size of the largest and smallest a-type ADU, a_{max} and a_{min} respectively, the mean a_μ and standard deviation a_σ of A, and the first, second, and third quartiles of A, denoted by a_{1q}, a_{2q}, and a_{3q} respectively. In order to better capture the structure of the sequence A, we measure the total variation a_{vs}, maximum first difference a_{fd}, lag-1 autocorrelation a_ρ, and homogeneity a_h of a-type ADUs for cases where $n \geq 2$ (*i.e.*, for connections with more than 1 epoch). Analogous features can be extracted from the collection $B = \{b_1, \ldots, b_n\}$ of b-type ADUs. Given that inter-epoch times are influenced to a degree by network and end-system properties (as opposed to the a's and b's which depend only on application-level behavior), we restrict our attention to a few time features: t_{max}, t_{2q}, and t_{tot}. These features are defined more formally in Tables 1 and 2.

Another important feature is whether more total bytes are transmitted from connection initiator to connection acceptor or vice versa. We capture the "directionality" of a connection by the ratio a_{tot}/b_{tot} and define a feature *dir* to represent the magnitude of directionality, $dir = \log(a_{tot}/b_{tot})$. To further asses the structure of a- and b-type ADUs, we also compute the lag 0 and 1 cross-correlations between B and A, denoted ρ_1 and ρ_2 respectively. In our preliminary analysis we found that rank correlations exhibited a more diverse and meaningful spectrum of values across different connections. Thus all correlation measurements are based on Spearman's rank correlation coefficient,

$$r_s(x,y) = \left(\sum_{i=1}^{d} \mathsf{rank}_i(x_i)\mathsf{rank}_i(y_i) - u_d\right)v_d^{-1}$$

where $u_d = d(d+1)^2/4$ and $v_d = d(d^2-1)/12$, and rank_i is the rank of the i^{th} feature of vector x among the i^{th} feature of vectors $v_1, \ldots, v_m$. This is the non-parametric equivalent of Pearson's correlation coefficient.

The features we have selected to investigate are simply our first cut on this problem. However, whichever features one ultimately chooses, there are a number of practical issues that need to be addressed before they can profitably be used to cluster connections. The first issue involves numerical scale. While correlations will range between -1 and +1, features such as the number of epochs in a connection (e) and the total number of bytes transmitted in given direction (a_{tot}, b_{tot}), can range anywhere from one to several million. To address this disparity, we first normalize features by taking logarithms of those features that vary over several orders of magnitude. Each feature is then translated and scaled so that, for the vast majority (more than 96%) of measured connections, its value is between 0 and 1. In exceptional cases, *e.g.*, a connection with 10^7 epochs, we allow features greater than 1 or less than 0. Allowing features to take values outside the unit interval avoids the possible compression of their true dynamic range by a small fraction of outliers.

Once normalized, each feature plays a role in determining the Euclidean distance between two feature vectors. One may weight the contributions of different features differently, but we have not done this in our experiments. A second practical issue is that some features (*e.g.*, correlations and total variation) are either not well-defined or not meaningful for *a-b-t* connection vectors with fewer than three epochs. When comparing a connection with ten epochs to one with two epochs, we look only at the Euclidean distance (or correlation) between those features that are defined in both associated vectors, and then normalize by the number of such "active'' features, so that the resulting distance can be compared to distances between longer connections.

We initially tested our approach by clustering training data sets with a small number of connections. Figure 4 shows the result of clustering 20 connections collected at UNC. We analyzed this data set using divisive hierarchical clustering as implemented in R [16], after converting each *a-b-t* connection vector into a feature vector that included all of the statistical features described above. Ten of the connections in the data set carried Telnet traffic (*i.e.*, interactive remote shell), while the other ten carried persistent web traffic (HTTP 1.1). The communication patterns used by these two protocols are quite different, so appropriate clustering should be able to split the data set into two subpopulations. As shown in Figure 4, two distinct clusters, emanating from the root of the dendrogram, are readily apparent. This visualization is a binary tree in which internal nodes represents a split of the set of connections (with a y-axis height that correspond to

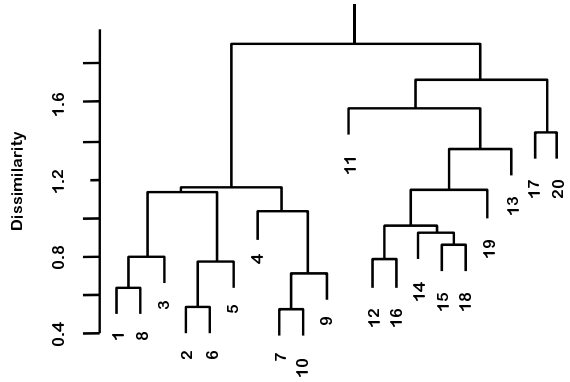

Figure 4: Result of clustering a training set of 20 connections using agglomerative hierarchical clustering. Leaves labeled from 1 to 10 correspond to Telnet connections, while those labeled from 11 to 20 correspond to HTTP connections.

the dissimilarity between its children). Leaves represent individual connections. The first split in the example cleanly separates Telnet connections from web connections.

4. Clustering Examples

4.1 Divisive Hierarchical Clustering Example

In our first example of clustering traffic, we study a packet header trace collected during April 2002 at the main (1 Gbps) network link that connects UNC to the Internet. We first converted this trace into a set of several million *a-b-t* connection vectors, from which we drew a random sample of 5,000 connection vectors with 2 epochs or more. We then computed the feature vectors of the connections in this sample, using the features reported in Table 1. After normalizing the feature vectors, we analyzed them using the diana procedure with Euclidean distance as implemented R [16]. This algorithm is described in [19], and its basic idea is to sequentially split the cluster with the largest diameter by finding its most dissimilar observation. This observation is used as the seed of a new cluster, which will be populated with some number of similar observations from the original cluster.

The results of clustering the set of 5,000 are shown in Figure 5, using a new visualization function that we implemented in the R language. The dendrogram shown is the result of pruning the full dendrogram at depth 4. The plot depicts pruned internal nodes as green triangles with a cluster number, and leaves as red squares with a connection vector number below them. Each internal node is annotated with the number of *a-b-t* connection vectors grouped under its branches. For example, the root of the tree is annotated with 5,000, since all of the connection vectors fall under this internal node. The first triangle on the left, marked as cluster number 1, groups 954 connection vectors.

Table 1: The 26 statistical features used in the divisive hierarchical clustering example shown in Figure 5.

Feature	Description
n	Number of epochs
a_{tot}, b_{tot}	Total bytes ($x_{tot} = \sum_{j=1}^{n} x_j$)
a_{max}, b_{max}, t_{max}	Maximum bytes, seconds ($x_{max} = \max\{x_j \in X\}$)
a_{min}, b_{min}	Minimum bytes ($x_{min} = \min\{x_j \in X\}$)
a_{μ}, b_{μ}	Mean bytes
a_{σ}, b_{σ}	Standard deviation
a_{1q}, b_{1q}	First quartile
a_{2q}, b_{2q}	Second quartile
a_{3q}, b_{3q}	Third quartile
a_{vs}, b_{vs}	Total variation ($x_{vs} = \sum_{j=2}^{n} \lvert x_j - x_{j-1} \rvert$)
a_h, b_h	Homogeneity ($(x_{max}+1)/(x_{min}+1)$)
a_{ρ}, b_{ρ}	Lag-1 autocorrelation
$\rho_1(a_{1..n}, b_{1..n})$	Spearman's rank correlation
$\rho_2(b_{1..n-1}, a_{2..n})$	Spearman's rank correlation with lag 1

The dendrogram reveals some useful structure in the set of connection. Connections in cluster 1 mostly correspond to HTTP, HTTPS (encrypted web traffic) and AOL traffic, while those in cluster 3 correspond to mail transfer protocols, such as SMTP and the Post Office Protocol (POP). The composition of clusters 2 and 4 was not so clear. The clustering algorithm accurately separated two clearly different communication patterns. Clusters 5 and 6 include connections in which all the *b*-type ADUs are zero, and whose port numbers did not map to known applications. Finally, cluster 7 grouped together HTTP, HTTPS, Microsoft Directory Service and RTSP connec-

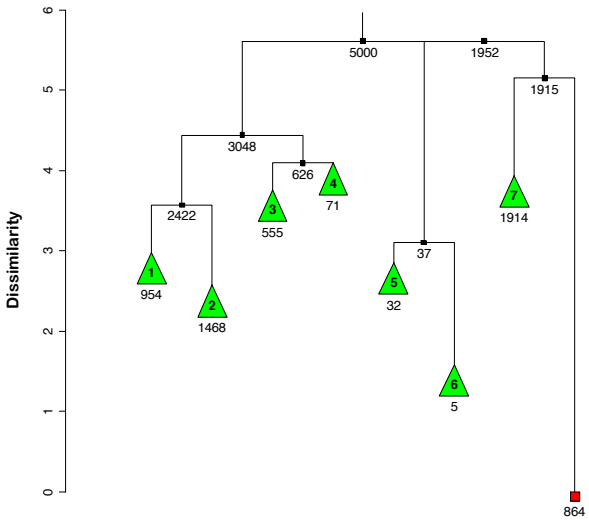

Figure 5: Dendrogram obtained from the divisive hierarchical clustering of data set of 5,000 connections, pruned at depth 4.

tions. The only leaf shown in the dendrogram (connection vector 864) was an FTP-DATA connection with $n = 2$, $a_{tot} = 50K$ and $b_{tot} = 0$.

While the revealed structure is suggestive, it is difficult to explain the observed hierarchy. This motivated the use of a different tool as described next. Furthermore, computation of the full dendrogram was slow; this 5,000-connection example required several hours of processing time. Another difficulty experienced was the $O(n^2)$ memory requirement, present in most statistical clustering algorithms, which comes from the need to compute the distance between each pair of *a-b-t* connection vectors as the first step.

4.2 Agglomerative Hierarchical Clustering

We applied our methodology to the clustering of a sample of connections from the Abilene-I data set [21]. The sample consisted of 717 TCP connections.[2] As before, each connection was first transformed into an *a-b-t* connection vector, and then summarized into a feature vector. Half of these connections were a random sample of port 80 connections, while the other half were a random sample of connections on other ports. The result was a matrix of 717 rows and 14 columns. Table 2 describes the 14 statistical features that were part of each vector.

Feature vectors were clustered using the average-linkage agglomerative method proposed by Sokal and Michener [25], with Pearson correlation coefficient as the similarity measure. For this clustering, we employed the implementation of the algorithm and the visualization tool developed by Eisen *et al.* in the context of gene expression arrays (microarrays) [7]. The result of the clustering is shown in Figure 6. Similar results were obtained using Euclidean distance.

The shared region in the center of Figure 6 is a *heat map* that represents the matrix of feature vectors. Each row in the matrix corresponds to one TCP connection, and each column corresponds to one statistical feature. The fourteen colored cells within a row represent the values of the statistical features of a single connection. Values are displayed using a scale of increasingly lighter shades of blue (in other words, the larger the value, the lighter the color). On the left side of the array, a rotated dendrogram displays the hierarchical clustering of connections. On the right side of the array, seven rectangles (labeled from A to G) are used to highlight seven clusters that exhibit a high degree of internal cohesion (correlation is 0.6 or more) and substantial separation from other clusters (dissimilarity sharply increases when any of these clusters is joined to another cluster).

[2] While this number is relatively small, we believe it is representative of the coarse-grained structure in the data set, and it makes it possible to include graphical output in this paper. We have applied our method to larger sets with up to 25,000 connections.

Table 2: The 14 statistical features used in the agglomerative hierarchical clustering example shown in Figure 6.

Feature	Description
n	Number of epochs
a_{tot}, b_{tot}, t_{tot}	Total bytes, seconds
a_{2q}, b_{2q}, t_{2q}	Second quartile (median)
a_{fd}, b_{fd}	Maximum first difference ($x_{fd} = \max_j \lvert x_j - x_{j-1} \rvert$)
a_h, b_h	Homogeneity ($(x_{max}+1)/(x_{min}+1)$)
dir	Directionality ($\log(a_{tot}/b_{tot})$)
$\rho_1(a_{1..n}, b_{1..n})$	Spearman's rank correlation
$\rho_2(b_{1..n-1}, a_{2..n})$	Spearman's rank correlation with lag 1

The interpretation of the resulting clusters confirms the effectiveness of our approach for grouping connections into homogeneous communication patterns. Note that this interpretation is based on port numbers (that we know are not very accurate), and it is only meant to illustrate the power of the method. Clusters A and B group together connections with small *a*-type ADUs. By looking at the destination port numbers of these connections, we found that most correspond to file sharing applications, mainly Kazaa (port number 1,214), eDonkey (4,662), and Gnutella (6,346). Connections in cluster A show substantially smaller *b*-type ADUs than those in cluster B, and they also exhibit much longer inter-exchange times. We believe that connections in the former cluster mainly correspond to file-sharing sessions in which only searches and no file downloads took place, while file downloads did occur in the connections grouped in the latter cluster. Some number of connections in these two clusters used other destination ports, such as 80, but their intra-connection dynamics did match those of file-sharing applications. These connections provide a good example of port number *hijacking*, a technique frequently employed to overcome firewalls and bandwidth caps.

Cluster C includes connections that have small *a*-type ADUs, and a number of exchanges that is significantly larger than that in the connections contained in clusters A and B. The destination port numbers correspond to a variety of applications, including Gnutella, HTTPS and Telnet.

Connections in cluster D are almost exclusively destined to port 119 (NNTP), and they show a clearly different pattern of data exchanges (large *a*-type ADUs and moderate *b*-type ADUs). Cluster E groups together connections destined to ports 80 (HTTP), 443 (HTTPS) and other ports that are also used for the web traffic, such as 8080 and 8443. Cluster F is mostly composed of SMTP connections (port 25) and some number of POP (110) and Oracle (1521) connections. Finally, cluster G contains FTP-Data connections. Some of these connections used source port 20, but the vast majority used other dynami-

cally negotiated port numbers. We have confirmed that these connections carried FTP-Data traffic by verifying that parallel FTP-Control connections existed.

It is important to remember that even though we have used knowledge of assigned port numbers to describe each of the above clusters, the uses of the clustering approach for workload modeling described in Section 1 do not depend on cluster labeling from port usage or any other means. The fundamental advantage of the *a-b-t* characterization and the clustering of TCP connections based on *a-b-t* features is that we can model and generate realistic Internet workloads without identifying specific applications.

The seven clusters described above can be further explored and decomposed into sub-clusters, an operation naturally supported by the hierarchical structure of the binary tree. For instance, we found other smaller clusters that group together other types of communication dynamics, such as those exhibited by streaming media and FTP-Control connections.

5. Related Work

Measuring and modeling traffic at the source-level has been an active area of research over the last ten years. Two important measurement efforts that focused on application-specific traffic models, but which preceded the growth of the web, were conducted by Danzig *et al.* [6, 2] and by Paxson [22]. Web traffic has been studied in numerous papers (*e.g.*, [20, 5, 15]), and file-sharing applications are the focus of much current work (*e.g.*, [13, 23]).

Traffic classification is known to be a difficult problem that has not received much attention in the past. There are a number of papers (*e.g.*, [18, 9]) that study how to identify groups of traffic that are "remarkable," *e.g.*, consume a large fraction of the traffic, but their focus is not on understanding the source-level structure of traffic. Other relevant papers evaluate existing monitoring techniques and propose more powerful alternatives (*e.g.*, [8]).

A compelling case for identifying traffic generation as one of the key challenges in Internet modeling and simulation is made by Floyd and Paxson in [11]. Prominent examples of research in traffic generation are Danzing *et al. tcplib* [6], the work by Barford and Crovella on web workload generation [1], and the SAMAN project by Lan and Heidemann [3].

6. Conclusions

We presented an abstract model of Internet communication and developed a methodology for clustering TCP connections into a small set of groups based on their patterns of network usage. Recently developed visualization techniques make it possible to easily interpret clustering results. Our results demonstrate that an important short-

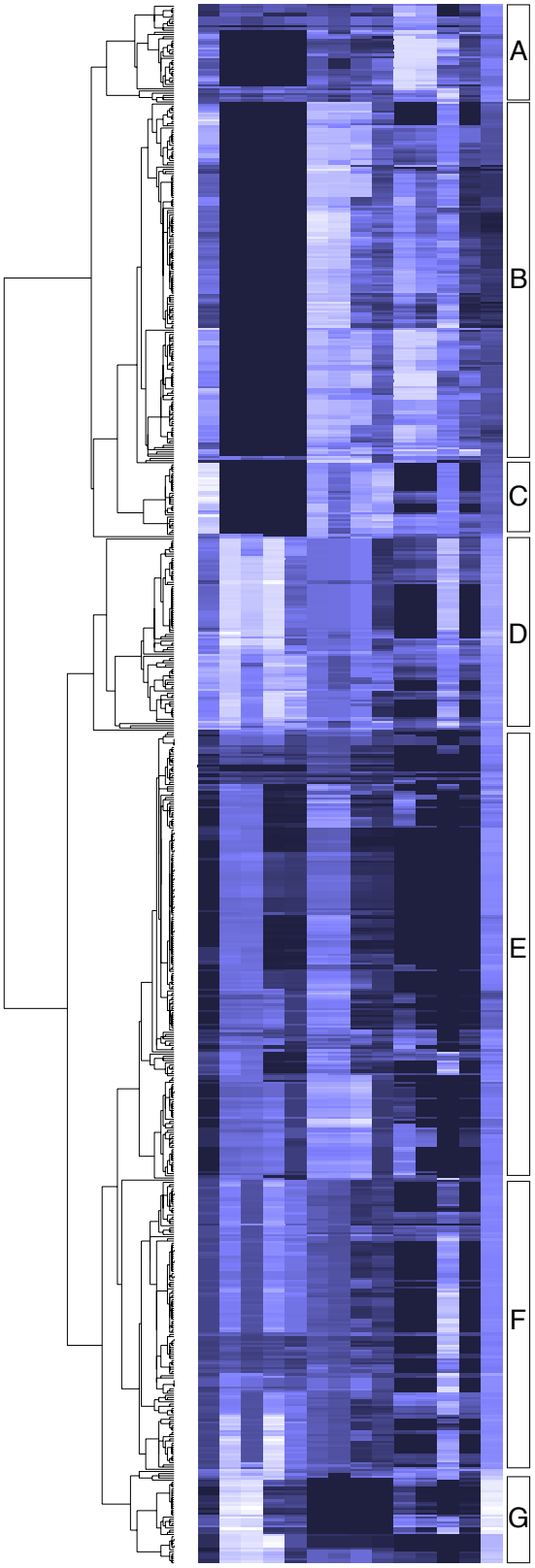

Figure 6: Result of clustering a sample of connections from the Abilene-I data set. From left to right, the columns of the heat map correspond to features: n, a_{tot}, a_{2q}, a_{fd}, a_h, b_{tot}, b_{2q}, b_{fd}, b_h, t_{tot}, t_{2q}, *dir*, ρ_1, and ρ_2.

coming of traditional clustering approaches is the difficulty of interpreting the results in a way that is meaningful for a network researcher. We overcame this problem by adapting to traffic analysis visualization techniques developed for the study of genetic data (in particular, gene expression arrays, or microarrays). The key contribution was the combination of traditional hierarchical visualization (known as a *dendrogram*) and a heat map of connection features, making it possible to easily interpret the clustering result in terms of application types.

We believe this provides a good starting point for understanding the types of application behaviors, and study how they change over time and across different vantage points. We are currently working on refining our measurement techniques and systematically examining the clustering structure of the application mixes at a large number of sites. We are also developing a new traffic generation tool that makes use of this structure to enable flexible traffic generation that is more representative of the wide variety of applications found on the Internet today.

7. Acknowledgements

We would like to thank the NLANR Measurement and Network Analysis Group for making their data and tools publicly available. We are also indebted to Michael Eisen, for making his clustering and visualization tools freely available for academic researchers, and to the developers of the R language.

This work was supported in parts by the National Science Foundation (grants ITR-0082870, CCR-0208924, EIA-0303590, ANI-0323648, and DMS-9971964), Cisco Systems Inc., the IBM Corporation, and a doctoral fellowship from the Computer Measurement Group.

7. References

[1] P. Barford and M. Crovella, *Generating representative web workloads for network and server performance evaluation*. Proc. of ACM SIGMETRICS, pages 151-160, 1998.

[2] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel, *Characteristics of wide-area TCP/IP conversations*. Proc. of the conference on Communications architecture and protocols, pages 101-112. ACM Press, 1991.

[3] K. chan Lan and J. Heidemann, *Rapid model parameterization from traffic measurement*. Technical Report 561, USC/Information Sciences Institute, August 2002.

[4] Internet 2 Consortium, *Internet2 Netflow Weekly Repor*t, http://netflow.internet2.edu/weekly/20040621, June 2004.

[5] M. Crovella and A. Bestavros, *Self-similarity in World Wide Web traffic: evidence and possible causes*. IEEE/ACM Trans. on Networking, 5(6):835-846, 1997.

[6] P. B. Danzig and S. Jamin, *tcp-lib: A library of TCP/IP traffic characteristics*. USC Networking and Distributed Systems Laboratory TR CS-SYS-91-01, October 1991.

[7] M. Eisen, P. Spellman, P. Brown, and D. Botstein, *Cluster analysis and display of genome-wide expression patterns*. Proc. Natl. Acad. Sci., 95:14863-14868, December 1998.

[8] C. Estan, K. Keys, D. Moore, and G. Varghese, *Building a better netflow*. Proc. of the ACM SIGCOMM, 2004.

[9] C. Estan, S. Savage, and George Varghese, *Automatically inferring patterns of resource consumption in network traffic*. Proc. of the ACM SIGCOMM, 2003.

[10] B. S. Everitt, S. Landau, and M. Leese, *Cluster Analysis*. Arnold, 4th edition, 2001.

[11] S. Floyd and V. Paxson, *Difficulties in simulating the Internet*. IEEE/ACM Transactions on Networking, 9(4):392-403, August 2001.

[12] Sprint Advanced Technology Laboratory. *IP monitoring project: Data management system*, 2004. *http://ipmon.sprintlabs.com.*

[13] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, *Measurement, modeling, and analysis of a peer-to-peer file-sharing workload*. Proc., 19th ACM Symposium on Operating Systems Principles, 2003.

[14] J. A. Hartigan, *Clustering Algorithms*. Wiley, 1975.

[15] F. Hernández-Campos, F. D. Smith, K. Jeffay, *Tracking the evolution of web traffic: 1995-2003*. 11th IEEE/ACM Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, October 2003.

[16] R. Ihaka and R. Gentleman, *R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics*, 5(3):299-314, 1996.

[17] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[18] B. Krishnamurthy and J. Wang, *Traffic classification for application specific peering*. Proc. of the ACM Internet Measurement Workshop, 2002.

[19] P. Macnaughton-Smith, W. T. Williams, M. B. Dale, and L. G. Mockett, *Dissimilarity analysis: A new technique of hierarchical sub-division*. Nature, 202:1034-1035, 1965.

[20] B. A. Mah, *An empirical model of HTTP network traffic*. Proc. of IEEE INFOCOM, pages 592-600, 1997.

[21] NLANR Measurement and Network Analysis Group. Trace IPLS-CLEV-20020814-090000-0 (Abilene-I data set). *http://pma.nlanr.net/Traces/long/ipls1.html.*

[22] V. Paxson, *Empirically derived analytic models of wide-area TCP connections*. IEEE/ACM Transactions on Networking, 2(4):316-336, 1994.

[23] D. Qiu and R. Srikant, *Modeling and performance analysis of bittorrent-like peer-to-peer networks*. Proc. of the ACM SIGCOMM, 2004.

[24] F. D. Smith, F. Hernández-Campos, and K. Jeffay, *What TCP/IP Protocol Headers Can Tell Us About the Web*, Proc. ACM SIGMETRICS '01, June 2001.

[25] R. R. Sokal and C. D. Michener, *A statistical method for evaluating systematic relationships*. Univ. Kansas Science Bulletin 38:1409-1438, 1958.