

# A Lightweight, High Performance Communication Protocol for Grid Computing

Phillip M. Dickens  
Department of Computer Science  
University of Maine  
dickens@umcs.maine.edu

## Abstract

This paper describes a lightweight, high-performance communication protocol for the high-bandwidth, high-delay networks typical of computational Grids. One unique feature of this protocol is that it incorporates an extremely accurate classification mechanism that is efficient enough to diagnose the cause of data loss in real time, providing to the controller the opportunity to respond to different causes of data loss in different ways. The simplest adaptive response, and the one discussed in this paper, is to trigger aggressive congestion control measures only when the data loss is diagnosed as network related. However, even this very simple adaptation can have a tremendous impact on performance in a Grid setting where the resources allocated to a long-running, data-intensive application can fluctuate significantly during the course of its execution. In fact, we provide results showing that the utilization of the information provided by the classifier increased performance by over two orders of magnitude depending on the dominant cause of data loss. In this paper, we discuss the Bayesian statistical framework upon which the classifier is based and the classification metrics that make this approach highly successful. We discuss the integration of the classifier into the congestion control structures of an existing high-performance communication protocol, and provide empirical results showing that it correctly diagnosed the cause of data loss in over 98% of the experimental trials.

**Keywords:** Grid computing; High-performance communication protocols; High-performance networking; Bayesian Analysis; Classification Mechanisms.

## 1. Introduction

The ability to efficiently move extreme-scale data sets between nodes on a computational Grid is critical to the success of data-intensive Grid applications. However, even though the underlying networking infrastructure of computational/data Grids can provide very high bandwidth, data-intensive applications are often able to obtain only a small percentage of such bandwidth. One problem is that TCP [12], the transport protocol of choice for most wide-area data transfers, was not designed for and does not perform well in the high-bandwidth, high-delay networks typical of computational Grids [26, 41, 47, 56]. This has led to significant research activity aimed at modifying TCP itself to make it compatible with this new network environment (e.g., Highspeed TCP [35], Fast TCP [49], Scalable TCP [50]), as well as systems that monitor the end-to-end network to diagnose and correct performance problems in TCP networks (e.g., [4, 55]). An alternative strategy has been the development of application-level protocols that can largely circumvent the performance issues of TCP. Some of these approaches are based on UDP (e.g., FOBS [22, 24-26], Tsunami [8], and UDT[40] ), while others spawn multiple TCP streams for a single data flow (e.g., GridFTP [10], PSOCKETS [58]).

UDP-based protocols can be attractive for two reasons: First, some applications require a smooth transfer rate that can be difficult to obtain with TCP. Second, such protocols are well suited to the high-bandwidth, high-delay network environment typical of computational Grids, and are able to obtain a significant percentage of the underlying bandwidth. However, because UDP-based protocols execute at the application level, the protocol developer must provide reliability (i.e., ensure that all

data is successfully received), and must also provide a mechanism to detect and respond fairly to competing traffic flows. Executing at the application level also means UDP protocols can lose data packets for any number of reasons unrelated to network congestion. This can lead to very poor performance if the control mechanisms interpret such loss as congestion in the network and, in response to such perceived network congestion, trigger very aggressive congestion control actions [37, 40].

One of the most challenging aspects of Grid computing, and one that makes the implementation of lightweight communication protocols difficult, is that resource availability can fluctuate significantly during the execution of a long-running application. This includes the CPU cycles allocated to nodes involved in a data transfer, which can be a major contributor to packet loss in UDP-based protocols [39]. This happens when the receiver's socket-buffer becomes full, additional data bound for the receiver arrives at the host, and the receiver is switched out and is thus unavailable to pull packets off of the network. The data receiver could be switched out for any number of reasons including preemption by a higher priority system process, interrupt processing, paging activity, and multi-tasking. To execute successfully in this environment, it is essential that the controller be able to distinguish between such CPU-related data loss and that associated with network contention.

To illustrate the importance of this issue, consider a data transfer with a sending rate of one gigabit per second and a packet size of 1024 bytes. Given these parameters, a packet will arrive at the receiving host on the order of every 7.9 microseconds, which is approximately the amount of time required to perform a context switch on the TeraGrid systems [7] used in this research (as measured by Lmbench [54]). Thus the receiver does not need to be switched-out long before packets begin to get dropped. We have observed, for example, tens to hundreds of packets being dropped when the operating system creates three to four new processes. This same phenomenon has been observed and commented upon within the context of UDT [39], where it was noted that any burst of CPU activity can lead to dropped packets. They further note that such loss can cripple performance because the controller interprets (and responds to) the loss as network contention.

To address these issues, we have been developing an accurate and efficient classification mechanism that can be used by UDP-based protocols to distinguish between data loss caused by contention for network resources from that caused by contention for CPU resources. It is integrated into a high-performance communication protocol for computational Grids, and is efficient enough to execute in real time. The controller invokes the classifier at the end of each transmission window, which returns a probability value associated with each cause of data loss. The controller uses this information to determine whether an aggressive response to observed data loss is required, or whether less aggressive responses can be employed.

The classifier is based on the analysis of what we term *packet-loss signatures* and the application of Bayesian statistics. These signatures show the patterns of packet loss in the current transmission window, and are so named based on our observations that different causes of packet loss result in very different *signatures*. We quantify the differences between such signatures based on their underlying structure, and define two metrics to quantify this structure. One metric is the *complexity* of the packet-loss signatures, which is derived using techniques from symbolic dynamics, and measures the level of randomness or periodicity in the signature. The second metric is based on the distance between successive packet losses. We have chosen these classification metrics because they have strikingly different statistical properties under different causes of packet loss, which is the fundamental determinant of the accuracy of the diagnoses.

In this paper, we discuss the Bayesian statistical framework upon which the classifier is based, and discuss its integration into the congestion control structures of FOBS: a lightweight, UDP-based high-performance communication protocol for Grid computing [22, 24, 25, 59]. We provide

empirical results showing that the classifier correctly diagnosed the causes of data loss in over 98% of the experimental trials, and, in the remaining trials, we show that the problems occurred (largely) at very low loss rates where there was not enough information in the signature to make a diagnosis (in which case an inconclusive rather than incorrect diagnosis was returned). Also, we present empirical results showing that the classifier is efficient enough to execute in real time, and that utilizing the information it provides can increase performance by over two orders of magnitude depending on the cause of loss.

We believe this paper makes three important contributions in the area of high-performance communication for Grid computing. First, it presents a classification mechanism that is extremely accurate and efficient, and generally applicable to UDP-based protocols that use selective acknowledgements for reliability. Second, the communication protocol, augmented with the classification mechanism, can obtain a large percentage of the underlying bandwidth when system conditions permit, can back off in the face of competing network traffic, and can accurately distinguish between these two conditions. Third, the ability to distinguish between the causes of data loss is an important milestone on the path to more adaptive and intelligent communication services for Grid computing. This paper should be of interest to a large portion of the Grid computing community given the critical nature of high-performance communication to the success of data-intensive Grid applications.

The rest of the paper is organized as follows. In Section 2, we discuss the data transfer system that serves as the platform for this research. In Section 3, we describe the development of the Bayesian classifier for the causes of data loss. In Section 4, we present empirical results demonstrating the accuracy of the classification system and the potential improvements that can be achieved by its utilization. In Section 5, we discuss the impact on the performance of the data transfer system itself due to the computational costs of the classifier. In Section 6, we discuss related work, and in Section 7, we discuss current and future research activities. We provide our conclusions in Section 8.

## **2. Data Transfer System**

The platform for this research is FOBS: a lightweight, high-performance, UDP-based data transfer system for computational Grids [22, 24-26]. We began development of the prototype system in 2002, and were primarily interested in obtaining a large percentage of the available bandwidth rather than congestion control (the high-speed, research networks were very lightly loaded at that time). When we began development of the congestion control mechanisms, we noticed large, contiguous groups of packets being dropped even when the network was largely dedicated to our transmissions. We investigated the behavior, and determined that the packet loss was almost always the result of CPU activity. However, such activity still resulted in significant reductions in the sending rate because at that time there was no mechanism to differentiate network contention from CPU contention. This motivated the development of the Bayesian classifier and its integration into FOBS. The design, architecture, and performance of FOBS have been discussed in earlier publications [24-26], and in this paper, we restrict our attention to the congestion control mechanisms and the integration of the classifier into such controls.

Similar to UDT [39], Scalable TCP [50], and HighSpeed TCP [2], FOBS uses packet loss rather than delay as the indicator of resource contention (for both network- and CPU-related contention). UDP-based protocols such as FOBS and UDT must provide their own reliability mechanism, and, in both cases, this is accomplished with a selective acknowledgment (SACK) and retransmission mechanism. Thus each data packet is individually numbered, and the selective acknowledgments show exactly those packets that successfully traversed the end-to-end transmission path and those that did not. These selective acknowledgement packets are what we term packet-loss signatures, and it is the

patterns of packet loss shown in these acknowledgements that are analyzed by the classifier. While the classifier is currently implemented only in FOBS, one area of future research is its integration into the control mechanisms of other UDP-based protocols such as UDT.

FOBS is a lightweight protocol that executes at the application-level and does not require system-level privileges for its installation, execution, or modification. It is a stand-alone application in that it does not require support from a system such as Globus, but could be made available to Globus applications via the XIO interface [11]. A significant advantage of FOBS is that it does not assume (or require) that it has dedicated control of the CPU(s) on which it is executing. Thus it is flexible enough to execute in a Grid environment where the CPU cycles dedicated to a long-running application may vary significantly during its execution.

FOBS is multithreaded to take advantage of nodes with multiple processors or processors with multiple cores. In such cases, the classifier executes as a separate thread that runs concurrently with the ongoing transfer. FOBS can be executed as a window-based protocol where all packets within the current transmission window are put onto the network at a constant sending rate. It can also be used as a rate-based system, where packets are put onto the network at a constant sending rate until a congestion event occurs, at which point a new sending rate is determined based on the information provided by the classifier and the long-term loss rate.

## 2.1 Congestion Control

An important design goal of FOBS is that it competes fairly with other network flows, and uses a modified version of the TCP Friendly Rate Control (TFRC) protocol [42] to enforce such fairness. TFRC is equation-based, where the sending rate is computed as a function of the steady-state loss rate. FOBS replaces the TFRC response function with that derived for HighSpeed TCP [35], a more aggressive version of TCP designed for high-performance network environments with very low loss rates. The use of this more aggressive congestion control mechanism is appropriate given that FOBS is designed for the well-provisioned, high-bandwidth, high-delay networks associated with computational Grids, and is not intended for the Internet1 environment. It is thus appropriate in networks within which HighSpeed TCP itself, and Grid FTP [8] with multiple TCP streams are appropriate.

We do not discuss the derivation of the HighSpeed TCP response function here, and direct the interested reader to [35] for a complete analysis. For our purposes, it is sufficient to note that a parameter termed `Low_Window` is defined, which sets the lower bound on the size of the congestion window at which the HighSpeed TCP response function will be used. This more aggressive response function is defined as:

$$(1) \quad w = 0.12/p^{0.835}$$

If the size of the congestion window is lower than `Low_Window`, the standard TCP response function is used<sup>1</sup>:

$$(2) \quad w = 1.2/\sqrt{p}$$

where  $w$  is the size of the next congestion window and  $p$  is the loss rate.

---

<sup>1</sup>As noted by the authors, this equation assumes a loss rate that is less than  $10^{-2}$  where the effects of TCP retransmit timeouts can be largely ignored.

## 2.2. Integration of Classifier into FOBS

While the derivation of the algorithms used by the classification mechanism is somewhat complex, the way in which the information is used by the controller is quite simple. When a congestion event occurs, the classifier is queried to determine the cause of such loss. The classifier then assigns a probability to the event that the data loss was caused by contention for CPU resources (and thus one minus this probability that the cause of loss was network related). If this probability is greater than 95%, then no corrective action is taken. Otherwise, the long-term loss rate is updated appropriately, and Equation (1) or (2) is invoked to determine the new window size.

The classifier is able to properly diagnose the cause of data loss only when there is enough information in a packet-loss signature to do so. We have extended the operating range of the classifier to loss rates as low as 0.00008 while still maintaining a very high level of accuracy. However, below this loss rate, the accuracy of its diagnoses decreases quickly with decreasing loss rates. Also, there are times when the classifier cannot distinguish between the causes of data loss. In both cases, the classifier assigns a probability of 50% to each cause of loss (i.e.,  $P(\text{Network Contention}) =$

$P(\text{CPU Contention}) = 50\%$ ). Because the probability is less than 95% that the cause of data loss is CPU contention, the loss event is treated as network contention.

## 3. Bayesian Classifier

Similar to the approach taken by Liu, *et al.*[53], the classification mechanism is based on the application of Bayesian statistics. This approach is centered on finding a *classification metric* that has different statistical properties under different causes of data loss. We consider two hypotheses: the cause of data loss was network contention ( $H_1$ ), or the cause of loss was CPU contention ( $H_2$ ). Let  $C$  be a random variable representing the value of the metric associated with each loss event, and let  $c_j \in C$  represent a particular observed value of the metric. We wish to evaluate the probability of each hypothesis based on the observed value of the metric. This is termed the *posterior probability* of  $H_i$  given  $c_j$ , and can be computed directly using Bayes' rule:

$$(3) \quad P(H_i | c_j) = \frac{P(c_j | H_i)P(H_i)}{P(c_j | H_1)P(H_1) + P(c_j | H_2)P(H_2)}$$

Here  $P(c_j | H_i)$  is the conditional probability of observing value  $c_j$  assuming hypothesis  $H_i$  is true.

That is, it represents the *likelihood* of observing value  $c_j$  assuming the cause of data loss was network related or assuming the cause of loss was CPU related. The term  $P(H_i)$  is the *prior* probability of hypothesis  $H_i$ , and represents the experimenter's initial estimate of the probability of each cause of data loss occurring in the system. The denominator represents the overall probability of observing value  $c_j$ .

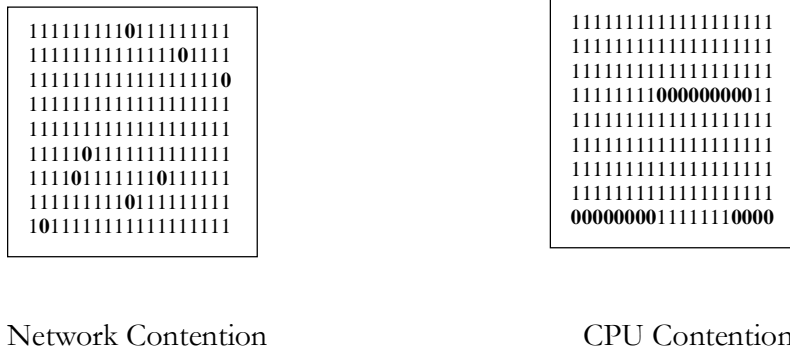
The accuracy of the classifier is maximized when the intersection area of the two conditional probability distributions is minimized. That is, the larger the difference in the statistical properties of the classification metric under the two causes of data loss the better the prediction. Thus a critical task in this research is to develop classification metrics where the conditional probability distribution

for is very different from that of

We have chosen two such classification metrics, both of which are based on packet-loss signatures, and that exhibit strikingly different statistical properties under different causes of data loss. We begin our analysis with a discussion of such signatures.

### 3.1 Packet Loss Signatures

A packet-loss signature is a sequence of 1s and 0s representing those packets that successfully traversed the end-to-end system (1) and those that did not (0). For example, the string ‘1000100’ signifies that packets zero and four were successfully received while the other packets were lost somewhere in the transmission path. When data loss occurs, the resulting signatures will, in all likelihood, be different, even when the fundamental cause of data loss is the same. This is because the number of lost packets can vary significantly (thus affecting the number of 0s in the signature), and the *particular* packets that are lost can also vary significantly (thus affecting the placement of the 0s within the signature). However, the underlying structure of the signatures associated with each cause of data loss remains essentially the same despite such variations. Thus it is possible to talk about “typical” signatures, such as those shown in Figure (1).



**Figure 1.** This figure shows examples of a “typical” packet-loss signature for each cause of data loss. Note that the 0s appear to be randomly placed within the signature associated with network contention and contiguous within the signature associated with CPU contention.

The question then is how to quantify the differences in the underlying structure of packet-loss signatures. We address this issue in the following sections.

#### 3.1.1 Complexity of Packet Loss Signatures

We quantify the differences between packet-loss signatures by applying techniques from *symbolic dynamics*, which have been developed by the nonlinear dynamics community and are highly appropriate for time series of discrete data. In symbolic dynamics [43], the packet-loss signature is viewed as a sequence of symbols drawn from a finite discrete set, which in our case is two symbols: 1 and 0. One diagnostic that quantifies the amount of structure in the sequence is *complexity*. There are numerous ways to quantify complexity, and we have chosen the approach of d’Alessandro and Politi [21], which has been applied with success to quantify the complexity and predictability of time series of hourly precipitation data [34]. The approach of d’Alessandro and Politi is to view the stream of 1s and 0s as a language and focus on subsequences (or *words*) of length  $n$  in the limit of increasing values of  $n$  (i.e., increasing word length). First-order complexity, denoted by  $C^1$ , is a measure of the richness

of the language's vocabulary and represents the asymptotic growth rate of the number of admissible words of fixed length  $\mathbf{n}$  occurring within the string as  $\mathbf{n}$  becomes large. The number of admissible words of length  $\mathbf{n}$ , denoted by  $\mathbf{Na}(\mathbf{n})$ , is simply a count of the number of distinct words of length  $\mathbf{n}$  found in the given sequence. For example, the string **0010100** has  $\mathbf{Na}(\mathbf{1}) = 2$  (0,1),  $\mathbf{Na}(\mathbf{2}) = 3$  (00,01,10), and  $\mathbf{Na}(\mathbf{3}) = 4$  (001, 010, 101, 100). First-order complexity is defined as

$$(4) \quad C^1 = \frac{\lim}{n \rightarrow \infty} [(\text{Log}_2 \mathbf{Na}(n))/2],$$

and characterizes the level of randomness or periodicity in a string of symbols. A string consisting of only one symbol will have one admissible word for each value of  $\mathbf{n}$ , and will thus have a value of  $C^1 = 0$ . A purely random string will, in the limit, have a value of  $C^1 = 1$ . A string that is comprised of a periodic sequence, or one comprising only a few periodic sequences, will tend to have low values of  $C^1$ .

### 3.1.2 Rationale for Complexity Measures

The reason for the differences in the complexity of packet-loss signatures has to do with the different timescales at which loss events occur. In the case of contention for CPU cycles due to multi-tasking, for example, packets are dropped when the data receiver is switched out and its data buffer becomes full, and will continue to drop packets until the receiver regains control of the CPU. The amount of time a receiver is switched out will be on the order of, for example, the time-slice of a higher-priority process for which it has been preempted or the aggregate time-slice of the processes ahead of it in the run queue. Such events are measured in milliseconds, where the packet-arrival rate is on the order of microseconds. Therefore if contention for CPU cycles is the predominant cause of data loss, the packet-loss signatures will consist of strings of 0's (the receiver is switched out), followed by strings of 1's (the receiver is executing and able to pull packets off of the network). Thus the packet-loss signature will be periodic in nature. Different operating systems will have different scheduling policies and time-slices, but the basic structure of the signature will remain periodic.

Data loss caused by network contention, however, operates on a much smaller timescale, that being the precise order in which the bit-streams of competing packets arrive at and are serviced by the hardware. The packet-loss signatures represent those particular packets that successfully competed for network resources and those that did not. This is an inherently random process, a fact that is reflected in the packet-loss signatures. In particular, this loss process generates packet-loss signatures with 0s, which represent those packets that were dropped, scattered randomly throughout the signature.

### 3.1.3 Computation of Complexity Measures

The only computationally expensive factor in Equation (4) is the calculation of the number of admissible words in a signature ( $\mathbf{Na}(\mathbf{n})$ ) as  $\mathbf{n}$  becomes large. The question is how large does the word size ( $\mathbf{n}$ ) need to become in order to provide an accurate picture of the complexity of the packet-loss signature? Our experience has shown that there is very little to gain (in terms of changes in the complexity measures) by increasing the word size past 17, and this is the value used in these experiments. Our approach to computing complexity measures is as follows.

The classification mechanism uses a sliding window of length  $\mathbf{n}$  to search for admissible words of length  $\mathbf{n}$  in the signature. If, for example, it is searching for words of length  $\mathbf{n} = 3$ , then the first window would include symbols 0-2, the second window would include symbols 1-3, and so forth. As each word is examined, an attempt is made to insert it into a binary tree whose branches are labeled

as either 1 or 0. Inserting the word into the tree consists of following the path of the symbol string through the branches of the tree until either (1) a branch in the path is not present or (2), the end of the symbol string is reached. If a branch is not present, it is added to the tree and the traversal continues. In such a case, the word has not been previously encountered and the number of admissible words is incremented. If the complete path from the root to the end of the symbol string already exists, then the word has been previously encountered and the count of admissible words is unchanged.

For example, consider the string ‘00100101’ and a word size of  $n = 4$ . The first word to be examined is ‘0010’ (symbols 0-3). Since this word has not been previously encountered, it is inserted into the binary tree (Figure 2-A) and the number of admissible words for  $n=4$  is incremented. The next word to be examined is ‘0100’, which has not been previously seen, and is thus added to the tree causing the number of admissible words to be incremented. The next word is ‘1001’, which is similarly added to the tree. The next word is ‘0010’, which has already occurred in the string, leaving the symbol tree and number of admissible words unchanged.

### 3.1.4. Complexity Measures as a Classification Metric

Having defined complexity measures and shown their derivation, the next step is to establish that such measures behave very differently under different causes of data loss. We did this through extensive experimentation on the TeraGrid [7]: a state-of-the-art computational Grid that connects eleven national research laboratories and universities via high-performance networks. It is the largest computational platform for open scientific research, and provides an excellent platform upon which new approaches to Grid computing can be implemented and evaluated. The TeraGrid facilities used in these experiments were the San Diego Supercomputing Center (SDSC), and the National Center for Supercomputing Applications (NCSA, located at the University of Illinois, Urbana). At NCSA, we used Mercury, an IBM IA-64 Linux cluster that consisted of 887 IBM dual processor 1.3/1.5 Ghz Itanium 2 nodes (we utilized the 1.5 Ghz nodes in these experiments). At SDSC, we used an IBM IA-64 Linux cluster, which consisted of 266 dual processor 1.5 Ghz Itanium 2 nodes. The operating system at both facilities was Linux 2.4.21SMP. Each compute node had a gigabit Ethernet connection to the TeraGrid backbone network.

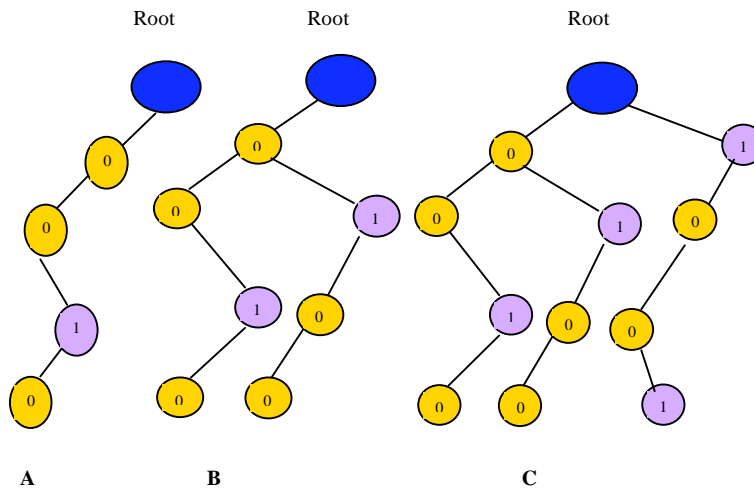
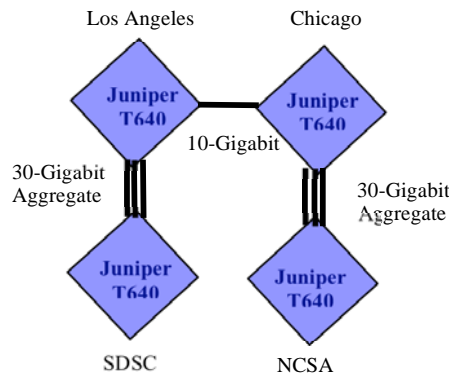


Figure 2. Building a binary tree as the string ‘00100101’ is parsed.

The salient portion of the TeraGrid networking infrastructure at the time of these experiments is shown in Figure (3). To investigate complexity measures resulting from network contention, we performed a cluster-to-cluster data transfer between SDSC and NCSA. The data transfers were initiated at SDSC, and the data was routed through the Juniper T640 in Los Angeles to the 10 gigabit per second backbone network to Chicago. This 10-gigabit connection was the bottleneck link between the two sites. We first launched a “background” communication between 10 nodes on each cluster, with a combined sending rate close to the 10-gigabit maximum. We then initiated a “foreground” communication between a single data sender at SDSC and a single data receiver at NCSA. Because the network was already saturated with the background traffic, this additional data caused contention within the bottleneck link. We computed the complexity of the resulting packet-loss signatures to determine their behavior when the cause of loss was network contention. We controlled the loss rate by varying the send rate of the foreground process, and collected and analyzed approximately 5000 signatures with loss rates in the range of 0 to 5%.

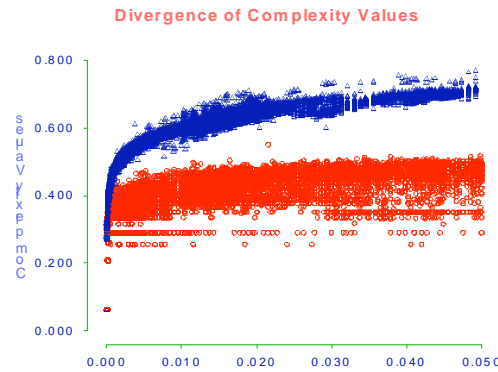


**Figure 3.** This figure shows the TeraGrid backbone network at the time of the experiments. The bottleneck in the transmission path was the 10-gigabit link between Los Angeles and Chicago.

To investigate the behavior of complexity measures associated with CPU contention, we initiated a point-to-point data transfer between a data sender at SDSC and a data receiver at NCSA. We maintained a sending rate of one gigabit per second, and *disabled* the congestion control mechanisms. We developed a simple process generator that spawned a random number of processes (uniformly distributed between 0 and 7) that competed with the data receiver for CPU resources. The generator allowed the additional processes to execute for a random length of time (uniformly distributed between 1 and 7 seconds), and then terminated them. Once the competing processes were terminated, the generator slept for a random length of time (uniformly distributed between 0 and 20 seconds), after which the entire cycle was repeated. We collected and analyzed approximately 5000 packet-loss signatures with loss rates between 0 and 5%.

Figure (4) shows the complexity measures of packet-loss signatures generated for each cause of data loss as a function of the loss rate. As can be seen, there is a very strong relationship between the cause of data loss and the resulting complexity measures. Further, the complexity values resulting from network contention appears to increase exponentially with increasing loss rate, while the measures associated with CPU contention appear to be largely unaffected by the loss rate (except at very low rates). It also shows that there is a very clear separation of complexity measures even at loss

rates as low as approximately 0.0005. These are very encouraging results, and support the claim that the complexity measures of packet-loss signatures have very different statistical properties under different causes of data loss.



**Figure 4.** This figure shows the complexity measure of packet-loss signatures as a function of the cause of data loss and the loss rate. The box highlights the area within which the complexity measures overlap.

However, there are two issues that must be addressed. First, the complexity measures overlap when the loss rate is less than approximately 0.0005, making them unsuitable as a classification metric at this loss rate and below. Second, complexity values are clearly a function of both the cause of data loss and the *loss rate*, where a classification metric must be a function of the cause of data loss only. We addressed each issue as follows.

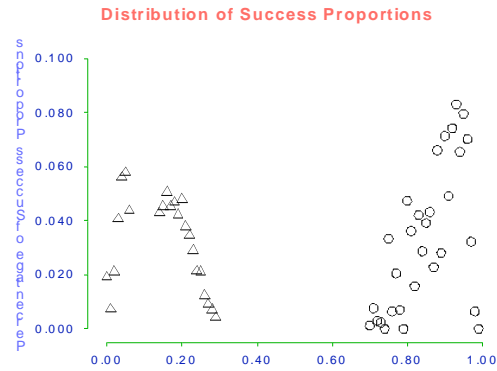
### 3.1.5 Extending the Range of Loss Rates

The reason for the overlap in complexity measures at low loss rates has to do with the size of the words that can be examined in real time without negatively impacting performance. In the experiments reported here, the maximum word size was set to  $n = 17$ , which was sufficient for discerning the basic structure of the signatures (i.e., either random or periodic) when the loss rate was greater than approximately 0.0005. However, as the loss rate decreases, the dropped packets (and the corresponding 0s in the packet-loss signatures) become too far apart to be detected with a word size of 17. While increasing the word size can help, the computational costs of doing so quickly becomes prohibitive, making it necessary to find other metrics to extend the classifier's range of operation.

The second metric that we developed was based on the observation that the basic *structure* of the packet-loss signatures does not change as a function of the loss rate. That is, packet-loss signatures generated by contention for CPU resources are still periodic, and signatures triggered by network contention are still random. Thus it may be possible to discriminate between the causes of data loss, even at very low loss rates, by looking at the proportion of 0s that are contiguous in the signature. In the case of CPU contention, one would expect a large proportion of the 0s to be contiguous, while one would expect a much smaller proportion of 0s to be contiguous in the case of network contention.

To formalize these ideas, we define “success” as two consecutive 0s (packet losses) in a packet-loss signature, and “success proportion” as the percentage of successes in the signature. During the experiments discussed above, the classification system scanned the packet-loss signatures and computed the success proportions for all signatures with a loss rate less than or equal to 0.0005. On the order of 1200 samples fell into this category for each cause of data loss. We wanted to find the

distribution of success proportions resulting from each cause of data loss, and proceeded as follows. We created one hundred data bins, labeled them from 0 to 0.99, and maintained a counter for each bin. Once the success proportion for a given signature was computed, we incremented the counter of the corresponding bin. Thus, for example, if a packet-loss signature had a success proportion of 98.2%, we would assign it to data bin 0.98 and increment the associated counter. Once all of the data were binned, we computed the percentage of success proportions that fell into each bin.



**Figure 5.** This figure shows the distribution of success proportions for each cause of data loss. Note that there is no overlap in the two conditional distributions, an important determinant of the accuracy of a classification metric.

The results are shown in Figure (5), and, as can be seen, the distributions of success proportions are once again strikingly different for different causes of data loss. We note that these results were obtained for loss rates in the range of 0.00008 – 0.0005, because signatures resulting from loss rates below this threshold did not, in general, contain enough information to classify the cause of loss. However, within this range, success proportions performed quite well as a classification metric, and increased significantly our ability to classify the cause of loss.

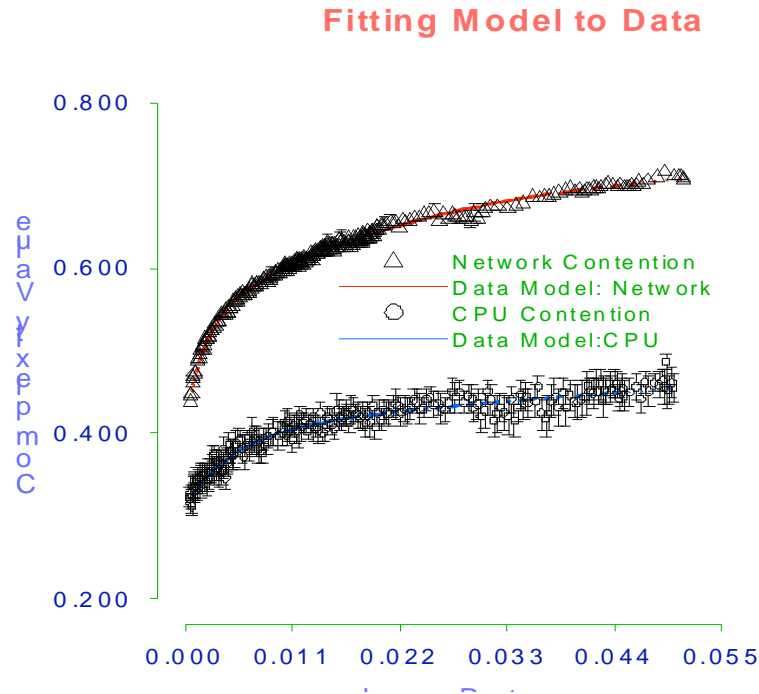
Another important characteristic of this metric is that its behavior is largely independent of the loss rate when the rate falls between 0.00008 and 0.0005. However, when the loss rate exceeds 0.0005, the behavior of success proportions does begin to be impacted. Thus to utilize success proportions as a classification metric outside of this range, we would need to develop models relating success proportions to loss rates. In the following sections, we develop such models for complexity measures and note that the development of such models for success proportions is an area of current research.

### 3.1.6 Mapping between Loss Rates and Complexity Measures

The conditional probability distributions  $P(c_j | H_i), H_i \in (H_1, H_2)$  require a metric that is a function of  $H_1, H_2$  only, while complexity values are affected by both the cause of loss and the loss rate. We address this issue in the following way.

The graphs of the complexity measures appear to be well behaved suggesting that it may be possible to develop simple empirical models that can capture the relationship between loss rates and complexity values. To develop such models, we fed the  $\{loss\ rate, complexity\ value\}$  pairs through a statistical program for fitting models to observed data (Simfit [13]). Given the shape of the curve for network contention, it seemed reasonable to investigate models involving exponentials. We tested many models, and the best fit for the data was obtained using the sum of two exponentials. Similarly,

we tested many models when the loss was caused by contention for CPU resources, and the best fit for the data was obtained using a single exponential. The observed complexity measures and the derived data models are shown in Figure (6). This figure shows the mean value of the observed complexity measures within each data bin, 95% confidence intervals around the means, and the fitted data model. As can be seen, the empirically derived models fit the observed data quite well. The utilization of these models in our analysis is discussed in the following sections.



**Figure 6.** This figure shows the mean complexity measures in each data bin, 95% confidence intervals around the means, and the fitted data models as a function of the cause of loss and the loss rate.

### 3.2 Estimating the Conditional Probability Density Functions

We are now ready to estimate the conditional probability density functions,  $P(c_j | H_i), H_i \in (H_1, H_2)$ , for both classification metrics. Our approach is to assume the fitted data models provide the “theoretical” mean complexity measure at a given loss rate. Our working hypothesis is that the data values are normally distributed around the mean of the empirical data. We go about justifying this working conclusion as follows. The obvious first step is to try and rule out a Normal distribution by checking to see if the distribution is clearly *non*-Normal, and this is clearly not the case.

The next step is to try to rule out a Normal distribution by testing the numbers of data points contained within the various standard deviations away from the mean. This does not force us to rule out a Normal distribution, either. The final step is to perform a hypothesis test using a statistical test for Normality. The one we have chosen is the Shapiro-Wilks test statistic. We applied this test to the data in each data bin, and when outliers were removed from the data, we were, without exception, not required to reject our hypothesis. Based on this, we accept our working hypothesis that this distribution is Normal.

We also have a working hypothesis that the success proportions are normally distributed around the mean of the empirical data. We followed the same approach to justify this working hypothesis (including the Shapiro-

Wilks test statistic), and were not required to reject this hypothesis either. We therefore accept the hypothesis that this distribution is also Normal.

### 3.3 Classifying the Cause of Data Loss

We are now able to compute the posterior probability,  $P(H_i | c_j)$ , using the conditional probabilities (likelihoods),  $P(c_j | H_i)$ , as discussed above. Also, we have a working hypothesis that the complexity measures and success proportions are normally distributed around the empirical means, and we have a good estimate of the standard deviation around the means based on the empirical data. Finally, we take  $P(H_i)$  to be drawn from the discrete uniform distribution (i.e., equally likely).

First, the classification mechanism computes the loss rate for the current transmission window. If the loss rate is greater than or equal to 0.0005, the complexity value of the packet-loss signature is computed. Next, the loss rate is used as a parameter to the empirically derived data models, which returns the complexity value, for each cause of data loss, at that particular loss rate. These values are used as the “theoretical” mean complexity value, and the standard deviation around both means is that derived from the empirical data. We then compute the distance between the observed complexity metric and the “theoretical” mean assuming each of the hypotheses. This distance is then standardized to provide an equivalent distance in the standard normal distribution equal to

$$(5) \quad z = \frac{x_c - \mu_c}{\sigma_c} .$$

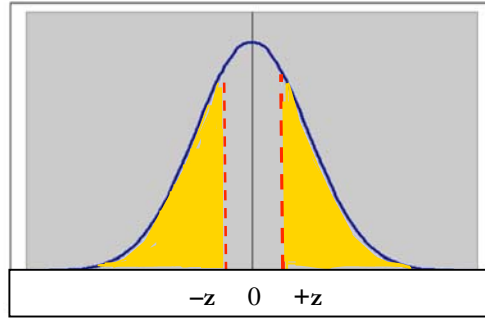
In this equation,  $\sigma_c$  is the empirically derived standard deviation,  $x_c$  is the observed complexity value, and  $\mu_c$  is the “theoretical” mean complexity value for the given loss rate.

Intuitively, the closer the observed value is to the mean of either normal distribution (i.e., the distribution associated with network or CPU contention), the more likely it is that it belongs to that distribution. One way to think about the problem is to consider the distance between the observed value and the mean of each distribution. Consider a given  $z$  value  $z = |z|$  (absolute value of the normalized observation), which is a measure of this distance. Because of the symmetry of the normal distribution, we must consider the distance on both sides of the mean, creating an interval  $\{-z, +z\}$  (this is shown in Figure 7). The area under the standard Normal curve representing this probability is:

$$P(Z < -z) + (1 - P(Z < +z)) = 1 - P(-z < Z < +z).$$

This probability represents the likelihood of the observation assuming each distribution (i.e., the Normal distribution associated with contention for CPU or network resources), and is the final piece of the Bayesian probability model. We can now compute the posterior probabilities of both causes of data loss, and provide this information to the controller. The controller can then use this information to determine the appropriate response to observed data loss.

While this description assumed the loss rate was greater than or equal to 0.0005, the approach is the same when the loss rate is between 0.00008 and 0.0005, except that the means and standard deviations used are those computed for the success proportions. When the loss rate is less than 0.00008, the classifier returns a probability of 50% for each cause of loss (which will be treated as network contention).



**Figure 7.** This figure shows the interval  $\{-z, +z\}$  in a standard normal distribution. The shaded area represents the probability of the observed value assuming a given distribution (i.e., the Normal distribution associated with contention for CPU or network resources).

#### 4. Utilization of the Classifier

The ability to distinguish the cause of data loss significantly increases the intelligence of the communication service and the range of adaptive responses it can employ. The simplest response has to do with congestion control: If the probability is very high that the cause of loss is CPU contention rather than network contention, then it is not necessary (or necessarily advisable) to trigger harsh congestion control measures. While simple, this adaptation can have a significant impact on the performance of large-scale data transfers in Grid systems.

In the experiments that follow, we explore the performance gains that can be obtained when the cause of data loss is in fact contention for CPU resources, and the controller utilizes the results returned by the classifier. In such cases, and assuming the classifier is able to correctly diagnose the cause of loss, performance is improved because the controller does not have to invoke harsh congestion control measures. However, if the controller does not utilize the classifier, or if the classifier returns an incorrect diagnosis, the controller must (incorrectly) assume the loss is network related forcing it to invoke harsh congestion control measures. Thus congestion control is used in these experiments, but is only invoked when the controller does not know the cause of loss (i.e., not using the classifier), or if the contention for CPU resources is incorrectly classified as network contention.

##### 4.1. Experimental Design

We performed several experiments to study the potential performance improvements made possible by utilization of the classifier. In these experiments, we used the same TeraGrid systems and the same approach to generating contention for CPU resources as described in Section 3.1.4. We performed three large sets of point-to-point data transfers between a sender executing on the SDSC cluster, and a receiver executing on the NCSA cluster. For each set of experiments, we conducted one data transfer where the controller utilized the probabilities returned by the classifier and another transfer where these probabilities were discarded. In the latter case, the controller assumed that all data loss was caused by network contention, and invoked the HighSpeed TCP congestion control

algorithms at each congestion event. When the controller did utilize the results from the classifier, it triggered the HighSpeed TCP congestion control algorithms at all congestion events *except* when the classifier returned a probability of at least 95% that the cause of data loss was contention for CPU resources.

The number of packets dropped per congestion event was a function of operating system behavior and the number of background processes that were spawned to compete with the data receiver. We wanted to evaluate the classifier's potential impact on performance given relatively low loss rates, and manipulated the maximum number of competing processes to achieve long-term loss rates of approximately 0.00035, 0.00045, and 0.00055 for the three transfers. Each transfer lasted approximately four hours and transferred on the order of 1.6 terabytes of data.

#### 4.1.1 Direct Execution Simulation

Our intention was to study the impact of the classification mechanism using the same parameters as those discussed in the RFC for the HighSpeed TCP response function [2]. However, this assumes a 10-gigabit link while the TeraGrid systems we used had only a 1-gigabit link between the compute nodes and the backbone network. To address this problem, we utilized a technique termed *direct-execution* simulation [23, 27-30, 33]. In this approach, part of the system of interest is actually executed to increase the accuracy of the simulation, and part of the system is simulated. Thus the simulation consists of two components: the *physical* components that are executed directly on the host system, and the *virtual* components that are simulated. There are events that occur during the execution of the physical system that are trapped by the simulator, which then determines how the trapped event affects the state of the simulated system and updates its state variables accordingly. The simulator then returns control to the physical system.

Our simulation was quite simple. The physical components of the simulation included an actual point-to-point data transfer between a sending node at SDSC and a receiving node at NCSA. Also, there were physical processes competing with the data receiver at NCSA (these processes were generated using the techniques described in Section 3.1.4). The physical transmission window was 100 MB, and the classifier was invoked at the end of each transmission window (in which data loss occurred) to compute the probability that the cause of loss was contention for CPU resources. The packet-loss signatures analyzed by the classifier were the real signatures generated by the loss process. Also, the long-term loss rate was that associated with the physical data transfer. The simulated part of the system was a *virtual data transfer* with its own virtual transmission window, send rate, and clock. The simulated transfer maintained the state variables necessary to compute the HighSpeed TCP response function, which included the size and send rate of the previous (virtual) transmission window. The loss rate used in the response functions (both HighSpeed and traditional) was that maintained by the physical system.

The interactions between the physical and virtual aspects of the simulation came at the end of each physical transmission window (at which time the simulator was invoked). From the simulation's point of view, this invocation represented the event that it had just successfully completed the transmission of its virtual transmission window. The amount of virtual data transmitted was set to the size of the virtual transmission window, and simulation time was updated by computing the length of time required to complete the transmission window at the current virtual send rate. If there was data loss in the corresponding physical system (i.e., the physical transmission window whose completion invoked the simulation), the classifier was called to compute the probability of each cause of data loss before the simulation was invoked.

In the event of packet loss, the simulation responded as follows:

1. If the simulated system was not utilizing the results returned by the classifier, then either Equation (1), the HighSpeed response function, or Equation (2), the traditional response function, was used to compute the size of the next virtual transmission window and virtual send rate. The choice of Equation (1) or Equation (2) was based on the size of the previous virtual transmission window and the physical long-term loss rate.
2. If the simulated system was utilizing the results of the classifier, then the next response was determined by the probabilities it returned. In particular:
  - a. If the probability was less than 95% that the cause of loss was CPU related, then the simulator treated the loss as if the results of the classifier were not used (i.e., number (1) above).
  - b. If this probability was at least 95%, then the long-term loss rate was updated in a manner consistent with no packet loss, and the size of the next virtual transmission window was computed using one of the HighSpeed TCP equations (based on the previous virtual window size and long-term loss rate).

## 4.2 Experimental Results

Table (1) shows the results of these experiments. The first column is the long-term loss rate, which we kept low to test the effectiveness of the classification system in a low loss rate environment. The second column shows the total number of congestion events, and the number of such events with a loss rate within the classifier's range of operation (0.00008 – 0.0005). As noted above, a congestion event with a loss rate of less than 0.00008 was classified as a network congestion event, which triggered the HighSpeed TCP response functions. The third column shows the probability of correctly diagnosing CPU contention given that the loss rate was within the operating range of the classifier. Column four shows the probability of incorrectly diagnosing the cause of loss as network contention when it was, in fact, CPU contention. The next column shows the probability that the classifier was unable to determine the cause of data loss. In such cases, the classifier returned a probability of 50% that the cause of loss was CPU contention. The next column shows the average throughput when the controller utilized the information returned by the classifier, and the final column shows the average throughput achieved when all packet loss was assumed to be network related.

Long-term loss rate	Total no. of congestion events/no. with LR>0.00008	P (CPU   CPU) when diagnosed (LR>=0.00008)	P (Net   CPU) when diagnosed (LR>=0.00008)	Probability of inconclusive diagnoses (P (CPU)= P (Net)= . 5) when LR>=0.00008	Av. TP with classifier	Av. TP without classifier
0.00035	1110 / 924	98.9%	0.22 %	0.86%	8883 Mb/s	74 Mb/s
0.00045	1429/1425	99.8%	0%	0.2%	9490 Mb/s	35 Mb/s
0.00056	1051/1044	99.3%	0%	0.7%	9397 Mb/s	41.5 Mb/s

**Table 1.** This table highlights the accuracy of the classifier and the potential benefits that can be achieved by its utilization.

As can be seen, the classifier performed extremely well when the loss rate was within its range of operation. There were only two incorrect diagnoses across all of the experiments, and only eighteen inconclusive classifications. These results also show that diagnosing the causes of data loss, rather

than assuming all loss is network congestion, can have a profound impact on performance, even in a very low loss rate environment. In particular, throughput was improved by a factor of 120, 271, and 226 for the three sets of experiments respectively.

While the throughput obtained in the first experiment was quite good (8883 Mb/s), it was less than that obtained in the other sets of experiments. We believe this is because of the (relatively) large number of inconclusive results (8), the large number of out-of-range results (186), and the two incorrect diagnoses of network contention (all of which were treated as network contention). The throughput achieved during the second and third sets of experiments was reasonably close, and we attribute the difference to the larger number of inconclusive diagnoses suffered in the third set (0.7% compared to 0.2%).

Clearly, these results represent the “best-case” scenario because all of the data loss was caused by contention for CPU resources. If the number of congestion events caused by contention for network resources were to increase, the performance benefits obtained from using the classifier would decrease. In the limit, when all data loss is caused by network contention, there would be no improvement in performance obtained from utilization of the classifier.

### 4.3 Classification of Network Contention

In previous sections, we examined the differences in throughput that are possible when the cause of loss is CPU related, the classifier correctly diagnoses the cause of loss, and the controller utilizes the diagnoses returned by the classifier. However, we have not yet examined the ability of the classifier to correctly diagnose network contention, and, for completeness, do so in this section.

Toward this end, we performed several long-running data transfers in which we injected contention into the network. The experimental design was the same as that discussed in Section 3.1.4. In particular, we initiated a cluster-to-cluster background transfer between SDSC and NCSA that consumed most of the 10-gigabit bandwidth, and then introduced contention by executing a point-to-point transfer between the two systems. We were largely able to control the loss rate by varying the rate at which the foreground transfer placed packets on the network. We collected approximately 160,000 classification samples with loss rates in the range of 0.0005 – 0.05 (using complexity as the classification metric), and 2800 samples with loss rates between 0.00008 – 0.0005 (using success proportions as the metric). The results are shown in Table (2).

Cause of data loss	Metric	P(Net Net)	P(Net CPU)	P(CPU CPU)	P(CPU Net)	P(Net) = P(CPU)=50%
Network	Complexity	99.94%	--	--	0.052%	0.001%
Contention	Success P.	98.2%	--	--	0.18%	1.5%

**Table 2.** This table shows the classifier’s ability to correctly diagnose network contention.

These results show that the classifier was also extremely accurate in its diagnosis of network contention. In particular, the probability of returning an incorrect diagnosis was extremely small with both classification metrics, while the probability of returning an inconclusive diagnosis was more significant when success proportions were used as the classification metric. This is because there is less information in packet-loss signatures when the loss rate is within the range of 0.00008 and 0.0005, making it more difficult to conclusively diagnose the cause of loss. However, such misclassifications present no problem to the overall system since they are treated as network contention.

## 5. Overhead of Classification Mechanism

The final question is the cost of executing the classifier in parallel with the data transfer. One way to measure this cost is to perform two data transfers, with identical parameters, except that one is executing the classifier concurrently with the transfer and the other is not. Toward this end, we executed on the order of 500,000 transmission windows in a point-to-point data transfer between SDSC and NCSA. In these experiments, the sender pushed packets onto the network at wire speed (1-gigabit per second), and, to eliminate differences in the sending rates of the two transfers, the congestion control mechanisms were *disabled*.

The round trip time between these facilities was approximately 65 milliseconds, and the transmission window was set to 100 Megabytes. The bottleneck in the transmission path was a 1-gigabit link between the compute nodes and the TeraGrid backbone network. In these experiments, FOBS was able to obtain on the order of 83.5% of the maximum bandwidth without the execution of the classifier and on the order of 81.2% of this bandwidth when the classifier was executing concurrently with the data transfer. There are two factors that are responsible for the relatively small drop-off in performance due to execution of the classifier:

- FOBS is multi-threaded to take advantage of multiple processors per node and/or multiple cores per processor. The nodes on the TeraGrid facilities used were dual-processor, and the basic sending and protocol processing functionality most likely executed on a separate processor from the classifier.
- The computation of complexity measures is highly efficient because it is implemented via simple binary tree operations (insertion, traversal).

## 6. Related Work

This research represents the intersection of two important topics: high-performance communication protocols for Grid computing and classifiers for the causes of data loss. We begin with a discussion of high-performance communication protocols.

### 6.1 Application Level Protocols

One approach to obtaining better utilization of high-bandwidth, high-delay networks has been to develop application level protocols that can circumvent many of the performance problems associated with TCP. In this section, we discuss two representative approaches.

#### 6.1.1 GridFTP

The most widely known communication protocol for Grid computing is GridFTP [10], which enhances and extends the FTP protocol [57] for execution in Grid environments. It provides reliable and secure data movement, partial file transfers, third party transfers, and data and protocol extensibility. It uses TCP as the underlying transport mechanism, and improves performance by allocating multiple TCP streams per processor (parallel transfer) and/or multiple processors per transfer (striped transfer). The current version of GridFTP [19] is implemented within the Globus Toolkit [6] version 4 (GT4), and leverages the Globus components for security and I/O. It is based on the Globus eXtensible Input/Output (XIO) system [11] which provides an interface for modifying and extending the GridFTP server.

While GridFTP provides excellent support for file transfers across the Grid, it is not the protocol of choice in all circumstances. For example, spawning multiple TCP streams for enhanced performance

effectively circumvents the congestion control mechanisms within TCP. This is because increasing the number of TCP streams increases the probability that, at any given time, there is at least one stream ready to fire. Thus while the individual streams are using TCP's congestion control mechanisms, in the aggregate there is no such control. This effect can be mitigated to some extent by using long virtual round trip times, but doing so requires root privileges [41]. Also, GridFTP requires the machinery of GT4 for its execution [11, 19], and some researchers do not want or need to deal with the complexities of installing, configuring, and using the Globus Toolkit.

Our research is focused on developing a lightweight transfer protocol that is easily integrated with applications and easy to deploy. It is able to obtain a large percentage of the underlying bandwidth when system conditions permit, can recognize and back off in the face of growing network contention, and can distinguish between these two situations via a highly accurate and efficient classification mechanism. The tradeoff is that FOBS does not provide all of the functionality of GridFTP. However, now that the basic congestion control protocol and the classification mechanism have been completed, current work is focused on adding third party transfers, failure recovery, and a simple security model. Also, we are exploring the possibility of writing a XIO driver for FOBS such that it can be utilized by Globus applications.

### **6.1.2 UDP-based Data Transfer (UDT)**

UDT [9, 39] is a lightweight, UDP-based protocol that incorporates a new congestion control mechanism termed DAIMD (additive increase, multiplicative decrease with decreasing increases). This approach has been shown to be efficient and fair to competing flows in a general network environment [40], allowing it to execute in less well-provisioned networks. UDT also serves as the basis for the UDT/CCC library (UDP-based Data Transfer Library with Configurable Congestion Control [38]), which provides support for developing and evaluating alternative congestion control mechanisms.

The developers of UDT and UDP/CCC have discussed at length the problem of CPU-related data loss [37, 40]. They note that because all loss is interpreted as network congestion, the performance of AIMD protocols (such as that employed in UDT) can be significantly degraded even when there is no contention for network resources [37, 40]. They are addressing this issue through rigorous code optimization to reduce CPU utilization, and, in the case of noisy links, by reducing the sending rate only when there is more than one packet lost in a congestion event. The authors also make the observation that there has been very limited exploration of the patterns of packet loss in real networks because most of the work in loss-based congestion control has been performed via simulation or within experimental test beds [58]. We certainly agree with all of these important observations.

Our research has focused extensively on the patterns of packet loss in real networks, and we have developed a sophisticated classifier for the causes of data loss based on the analysis of such patterns and the application of Bayesian statistics. This makes FOBS intelligent enough to respond aggressively only in the case of network contention, which, we believe, is critical in a Grid environment. Also, it appears that the classifier developed in this research could be integrated with UDT in a straightforward manner, and this represents an area of future research.

### **6.1.3 Other Application Level Protocols**

PSockets [58] is similar to GridFTP in that it spawns multiple TCP streams for improved performance. As discussed above, however, this approach circumvents the congestion control mechanisms within TCP. Other UDP-based approaches do not implement any form of congestion control and are thus only suitable for dedicated (or QOS-enabled) networks. Such protocols include

Tsunami [8], RBUDP [44], SABUL, and earlier versions of FOBS [22, 24, 26].

## 6.2 Modifications to TCP

Another approach to overcoming the inefficiencies of TCP is to modify the congestion control mechanisms within TCP itself. One such approach is HighSpeed TCP [2], which was motivated by an analysis showing that maintaining a steady-state throughput of 10 Gigabits per second requires that there be no more than one congestion event every 1 2/3 hours [2]. The authors noted that this is unrealistic in current network environments, and thus leads to very poor network utilization. HighSpeed TCP improves performance by modifying the TCP response function to be more aggressive in connections with large congestion windows. In connections with small congestion windows, it uses the same response function as standard TCP to compete fairly in slower (or more congested) network environments. The authors note that applications employing the TCP Friendly Rate Control protocol (TFRC [42]) can utilize the HighSpeed response function for improved performance in very high-speed network environments, which is the approach taken by FOBS.

Scalable TCP [50] is based on HighSpeed TCP and similarly modifies the TCP response function for increased performance in high-delay, high-bandwidth networks. In particular, it increases the congestion window more aggressively for each acknowledgement received in a round trip time (RTT) without congestion ( $W = W + 0.01$  rather than  $W = W + 1/W$ ), and decreases the congestion window less aggressively upon a congestion event ( $W = W - [0.125 * W]$  rather than  $W = W/2$ ). While Scalable TCP only requires sender-side modifications, such modifications are extensive and include increasing the size of the kernel interface queues, modifying the NIC interrupt handler, and changing the congestion window algorithms [50]. However, the changes to the computation of the congestion window are simple, and could be implemented and evaluated within the context of an equation-based protocol such as FOBS.

FAST TCP [48] uses equation-based congestion control that uses queuing delay, rather than packet loss, as the primary measure of network congestion. The implementation strategy is to explicitly estimate the difference between the current state and the equilibrium state, and to use this measure as the determinate of the degree to which the congestion window should be adjusted. The authors provide theoretical analysis showing that this approach can rapidly drive the system toward equilibrium in a fair and stable manner, while maintaining high network utilization [45]. Simulation and experimentation in a testbed network environment support these theoretical predictions. However, it has been observed that in a real network environment protocols using delay as the congestion indicator have throughput problems in the presence of reverse traffic [40, 46]. Also, it has been observed that there is not necessarily a strong correlation between delay and packet loss [40, 48].

### 6.2.1 Obstacles to the Deployment of Modified Versions of TCP

There are challenges associated with utilizing modified versions of TCP in real high-speed networks and applications. As discussed by the developers of FAST TCP [45], the performance of TCP in such networks behaves very differently from that predicted by theoretical analysis, execution in experimental testbeds, or simulation studies, and it is quite difficult to pinpoint the issues that are causing such differences. The problem is that any number of factors in the host, network, or application, which are completely unrelated to the modified protocol itself, can negatively impact performance. For example, they found logic errors in the implementation of slow start (in the standard Linux implementation) that resulted in zero goodput for 113 seconds [45]. They also found an implementation bug in the way Linux increases the receiver's advertised window leading to poor performance. Similarly, the developers of Scalable TCP had to modify the NIC interrupt handlers

and increase the size of the kernel interface queues to achieve good performance [50]. It has also been noted that modified implementations of TCP may have to interact with legacy versions of the protocol, where the resulting performance is dependent upon the level of compatibility between such versions [40, 45].

Our research into UDP-based protocols is not intended to replace TCP, but rather to offer an alternative approach that is simple, easy to install and use, and which provides excellent performance in high-speed networks. While it may be difficult to separate out the effects of host- and network-related issues in TCP in high-speed networks, it is less difficult to do so in FOBS because much of this information is provided by the classification mechanism. Also, implementing an application level, equation-based congestion control mechanism significantly reduces the number of “moving parts” outside of the protocol itself, which leads to consistent and understandable performance characteristics.

### 6.3 Classifiers for the Causes of Data Loss

The issue of classifying the cause of data loss has received significant attention within the context of TCP for hybrid wired/wireless networks (e.g., [14-17, 36, 51, 53]). The idea is to distinguish between losses caused by network congestion and losses caused by wireless errors, and to trigger TCP’s aggressive congestion control mechanisms only in the case of congestion-induced losses. Similar to our approach, Liu *et al.* [53] use a Bayesian statistical framework to classify the cause of data loss. Their approach is based on *loss pairs* [52], a technique with which network delay around the time of a packet loss can be measured, and Hidden Markov Models. The classification metric is the distribution of network delay (RTT) around the time of a loss event. The idea is that different types of data loss (i.e., congestion/wireless) will result in different delay distributions, and that a Hidden Markov Model can be trained to capture these different distributions as different states within the HMM. Thus each state is characterized by the mean and standard deviation of the associated delay distribution. In [14], Barman and Matta develop a Bayesian classification model that is similar to the one developed by Liu *et al.* They also use the distribution of RTT’s as the classification metric, and Hidden Markov Models to infer the conditional probability distributions of such delay under congestion losses and wireless losses. These related works were able to obtain accurate classifications for either wireless losses or congestion losses, but it was very difficult to obtain accurate classifications for both.

Our research also uses a Bayesian statistical framework, but classifies network- or CPU-related losses rather than congestion or wireless losses. Also, we are focused on the well-provisioned, high-performance (wired) networks typical of computational Grids, while they appear to be operating within the Internet1 environment. Another difference is that our approach uses the patterns of packet loss rather than RTT as the classification metric. Also, based on the results presented in these related works, complexity measures and success proportions appear to be more robust classification metrics than the probability distribution of round trip times. Further, their approach requires that there be exactly one bottleneck link in the network while our approach has no such restrictions. Finally, our classifier is implemented and used within a high-performance communication protocol, while these related works separate the issue of developing the classifier from its actual use.

In [15], Biaz and Vaidya develop a classifier for hybrid TCP networks based on the sending of packets with different discard priorities and biased queue management that “de-randomizes” congestion losses. The idea is that at low loss rates congestion losses appear “random”, and thus it is difficult to distinguish between congestion losses and wireless losses (which also appear to be random). Their solution is to “de-randomize” congestion losses by marking a subset of packets with the label “out”, coupled with a router discipline that drops such packets first in the event of congestion. Their discriminator is based on the probability of losing  $x$  packets marked as “out” in a

random sample of  $r$  packets among a population of  $S$  packets, which follows a hyper-geometric distribution. When a loss event occurs, they look at the number of packets marked as “out”, and determine the probability of observing that number of “out” packets in a random sample. If the loss pattern has a very low probability of occurring, then it is determined that the pattern of packet loss was not drawn from a random sample and is thus classified as congestion related. They define a function that is intended to summarize the pattern of packet loss, and, based on a threshold value of this function, classify the loss as either congestion or wireless.

The approach of Biaz and Vaidya is, in some respects, closely related to the models developed in this research. They are similarly interested in the patterns of packet loss, and whether such patterns arise out of random processes. They also discuss the need for a “fingerprint” (a particular pattern of losses), which could be used to distinguish congestion losses. We agree that the focus on the patterns of packet loss and the level of randomness within such patterns is the correct approach for this problem. However, we believe that the complexity of packet-loss signatures is a superior metric with which to gauge the level of “randomness” in a given pattern of packet loss. Also, our approach does not require router support to classify the causes of data loss.

TCP WestwoodVT [51] classifies the cause of data loss using simple statistics on round trip times as the classification metric. It defines the *expected* transmission rate as the current window size divided by the minimal observed RTT, and the *actual* transmission rate as the current window size divided by the current RTT. It then defines  $\Delta$  as the difference between these two values, and classifies the cause of loss based on  $\Delta$  being above or below simple threshold values. It is difficult to gauge the accuracy of their approach because the authors did not provide such information. However, TCP WestwoodVT is based largely on TCP Vegas [18], which has been shown to perform poorly in its classification of network state ([53], [17], [15]).

Other approaches have also used very simple classification metrics to infer the causes of data loss in TCP networks. Examples of such metrics include one way round trip times (e.g., Spike [20]), the inter-arrival rate of packets and the number of losses detected ([16], [20]), or a combination of all three (ZBS [20]). However, based on the analysis presented in [20], all of these approaches suffer from either very high misclassifications of congestion losses, wireless losses, or both.

## 7. Current and Future Research

We have demonstrated that it is possible to construct a highly accurate classifier for the causes of data loss, and that utilizing the resulting diagnoses can significantly improve the performance of a large-scale data transfer. In this section, we discuss some of the remaining issues that require further research.

### 7.1 Finer Grained Diagnoses

In this paper, we have studied extensively packet-loss signatures associated with pure CPU contention and pure network contention (that was caused by the injection of streaming background data flows). This raises the question of whether other kinds of contention should (or could) be considered, and, if so, whether the classifier should be trained to recognize such contention. For example, we have noticed that network contention caused by bursty traffic flows have somewhat higher complexity measures than those associated with the streaming data flows with which the classifier was trained. Similarly, we have observed that the signatures associated with multiple data streams converging at a router also have somewhat higher complexity measures. We have also observed that packet-loss signatures generated when there is significant CPU contention combined with network contention (what we term *hybrid* signatures), have complexity measures that are higher

than those associated with pure CPU contention but lower than those associated with pure network contention. The first case, where network contention is caused by bursty traffic or multiple data streams, is not problematic because the classifier will return a diagnosis of network contention or will be unable to make a diagnosis (because the observed complexity measures are outside of the range of both causes of data loss). In either case, the loss event will be treated as network contention thus triggering aggressive congestion control measures.

The issue of hybrid signatures, however, could be problematic, particularly when the loss rate is high and a large portion of the total packet loss is due to severe contention at the CPU. The problem stems from the fact that the resulting packet-loss signatures would reflect a small proportion of random behavior and a larger proportion of periodic behavior. Thus even though the resulting complexity measures would be larger than those associated with pure CPU contention, it could be much lower than those associated with pure network contention. This could cause the classifier to return an incorrect diagnosis of CPU contention, which would result in the sender not backing off aggressively enough for the current system conditions.

The question then is whether the classifier should be trained to recognize finer-grained contention (e.g., hybrid packet-loss signatures, signatures from bursty traffic), or whether simple heuristics could be developed to address such issues. The answer, we believe, is dependent upon whether the unrecognized packet-loss signatures could lead to an incorrect diagnosis, and, if not, whether the controller could use the additional diagnostic information to better respond to the current system conditions. For example, if the controller knows that there is hybrid contention, it would seem appropriate to trigger more aggressive congestion control measures than it would in the case of pure network contention. However, it is not (yet) clear that training the classifier to recognize bursty traffic would be of significant benefit, and it seems sufficient to let the classifier return a diagnosis of network contention or no diagnosis at all. These are important issues, and more research is needed to determine those cases where finer-grained diagnoses would be helpful, and to determine how to make the best use of such information once it becomes available.

## 7.2 Network Infrastructure

This study was conducted on the backbone network connecting the research facilities on the TeraGrid, which consists of a small number of high-speed (Juniper T640) routers. Thus one question is whether the approach used here could be expected to be successful on other networking architectures. That is, will contention for network resources, across a wide range of networking architectures, generate packet-loss signatures that are easily distinguishable from those generated by contention for CPU resources? Based on very extensive experimentation, taken over a wide range of CPU architectures [31, 32], we are firmly convinced that contention at the CPU, as observed from the application layer, is periodic in nature. We thus focus our attention on the question of whether signatures generated by network contention on other architectures would likely be random in nature.

This issue is strongly related to the queuing discipline(s) employed at the routers or high-speed switches in the end-to-end transmission path. If there is contention within a router or set of routers employing active queue management with Random Early Detection [5], then it stands to reason that the resulting loss signatures would be random. The most problematic case would be routers using tail drop queue management, which drops all incoming packets once the queue becomes full. Under the right conditions, this could lead to a large number of contiguous packets being dropped, which would result in a periodic signature.

In considering this issue, it is important to note that all of the routers used in this study did, in fact, employ tail drop queue management [60], and, in all of our experimentation to date, we have not

observed a periodic signature when we created contention for network resources. Thus there is at least anecdotal evidence that this is not a serious issue, at least on the high-performance networks for which this protocol was designed. This is supported by similar experiments we have conducted on other facilities connected by high-speed networks (the Illinois Wired/Wireless Infrastructure for Research and Education [3]), where we obtained similar results [32]. However, these are limited results, and we are investigating this issue more fully through empirical and simulation studies. We are also studying the impact on the packet-loss signatures resulting from multiple routers with different queue management policies in the end-to-end transmission path.

### **7.3 Execution at the Application Level**

There were several important considerations that led us to implement the classifier (and FOBS) at the application level. First, it significantly simplified the design of the system and resulted in few “moving parts” to complicate its implementation and evaluation. Second, one of our most important goals was to build a lightweight communication protocol that did not require kernel-level privileges for either its implementation, utilization, or evaluation. Third, the simplicity of the data transfer engine, combined with the fact that complexity measures are obtained as a function of a constant sending rate, means that the values of the variables collected are (largely) unaffected by the protocol itself. Perhaps most importantly, it is the fact that the system dynamics are observed from the application layer that produces packet-loss signatures with strikingly different characteristics depending on the cause of data loss. Thus while it is possible to observe system dynamics from a lower level (e.g., protocol stack at the receiver), changing the perspective from which such dynamics are observed would significantly alter the characteristics of the resulting packet-loss signatures. Having said this however, there may certainly be advantages to observing from a lower level perspective, and we are currently exploring whether the general approach used in this research could be extended to the kernel level.

## **8. Conclusions**

In this paper, we have presented an extremely accurate and efficient classifier of the causes of data loss for UDP-based communication protocols executing in high-performance Grid environments. The classifier is designed to allow UDP-based protocols to distinguish between data loss caused by network contention from that caused by contention for CPU resources. This is an important issue because contention for CPU resources can be a significant cause of packet loss in UDP-based protocols executing in Grid environments. Without such classification, all losses are assumed to be representative of network contention, which can lead to significant performance degradation if aggressive congestion control mechanisms are invoked in response to such perceived congestion. The approach should be applicable to any UDP-based protocol that provides reliability through the use of selective acknowledgments and retransmission.

We discussed the integration of the classification mechanism into the control structures of FOBS, an existing UDP-based protocol, and evaluated its performance and accuracy. We showed that the classifier was able to correctly diagnose contention for CPU resources as the cause of data loss in between 98.9% and 99.8% of the cases, and as contention for network resources in between 98.2% and 99.4% of the cases (when the loss rate was within the classifier’s range of operation). Also, it was shown that the classifier returned an incorrect diagnosis only 0.22% of the time (and thus returned a correct or inconclusive diagnosis in 99.9978% of the cases). These are excellent results, particularly considering that they were achieved over real networks in real time. We believe these results also justify the use of somewhat more complicated classification metrics than others that have been utilized.

This research also demonstrated that utilizing the information provided by the classifier can have a tremendous impact on performance. In particular, our experiments using direct-execution simulations showed that performance was enhanced by up to a factor of 270 when contention for CPU resources was the cause of data loss. Further, it was shown that the classification system was efficient enough to execute in real time, making it a practical tool for high-performance, UDP-based communication protocols. Its practicality is extended by the fact that it executes at the application layer and thus does not require kernel-level privileges for its execution.

It is also important to note that this approach separates the issue of diagnosing the cause of data loss from that of responding appropriately to such loss. In this paper, we used the HighSpeed TCP response function to determine the new sending parameters after a congestion event. However, other equation-based congestion control algorithms (e.g., TFRC) could be used in place of HighSpeed TCP.

One remaining question is the generality of the empirical models that provide the mapping between the loss rate and the “theoretical” mean complexity measure. It is our belief, based on extensive experimentation on different systems and networks, that this basic mapping does not change across architectures (see [32, 33, HPDC05]). That is, the mapping between loss rates and network contention can be modeled quite accurately as the sum of two exponentials, and the mapping between loss rates and complexity measures associated with CPU contention can be modeled as a function of a single exponential. However, it may be the case that the parameters to the empirical models do vary between architectures, making it necessary to “retrain” the classifier for different systems. This is an area of current research.

The ability to classify the temporal dynamics of packet loss behavior (as expressed by the packet-loss signatures) offers two significant advantages. First, such classification allows the control mechanisms to apply corrective actions based on the cause of data loss. For example, the control mechanisms may be able to migrate the data receiver, rather than drastically reducing the sending rate, when the cause of data loss is classified as CPU contention. Second, given that the underlying dynamics have structure, it may be possible to construct simple predictors that allow the data transmitter to shape its behavior in such a way as to increase the probability that a sent packet is successfully received. These are enticing possibilities, and the exploration, evaluation, and integration of these techniques to the problem of large-scale data transfers represents an important area of current research.

## **Acknowledgments**

This material is based in part upon work supported by the National Science Foundation under Grant Number 0702748. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## **References**

- [1]. Advanced Networking for Leading-Edge Research and Development  
<http://abilene.internet2.edu/>
- [2]. HighSpeed TCP for Large Congestion Windows  
[www.ietf.org/rfc/rfc3649.txt](http://www.ietf.org/rfc/rfc3649.txt)
- [3]. I-WIRE: The Illinois Wired and Wireless Infrastructure for Research and Education  
<http://www.iwire.org>
- [4]. Net100: Development of Network Aware Operating Systems  
<http://www.csm.ornl.gov/~dunigan/net100/>

- [5]. RFC 2309. Recommendations on Queue Management and Congestion Avoidance in the Internet. B.Braden et al, April 1998.
- [6]. The Globus Alliance.  
<http://www.globus.org>
- [7]. The Teragrid Project  
<http://www.teragrid.org>
- [8]. Tsunami Home Page (Associated with the Advanced Network Management Lab at the University of Indiana)  
<http://www.indiana.edu/~anml/anmlresearch.html>
- [9]. UDT Software Release  
<http://sourceforge.net/projects/UDT>
- [10]. Allcock, W., Bester, J., Breshahan, J., Chervenak, A., et al., Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In the *Proceedings of the IEEE Mass Storage Conference*, (2001).
- [11]. Allcock, W., Bresnahan, J., Kettimuthu, R. and Link, J., The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid. In the *Proceedings of the Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models*, (April, 2005).
- [12]. Allman, M., Paxson, V. and Stevens, W. TCP Congestion Control  
<http://www.faqs.org/rfcs/rfc2581.html>
- [13]. Bardsley, W.G. SimFit: A Package for Simulation, Curve Fitting, Graph Plotting and Statistical Analysis.  
<http://www.simfit.man.ac.uk>
- [14]. Barman, D. and Matta, I. Model-based Loss Inference by TCP over Heterogeneous Networks. *WiOpt 2004*, Cambridge, UK, 2004.
- [15]. Biaz, S. and Vaidya, N. "De-randomizing" Congestion Losses to Improve TCP Performance over Wired-Wireless Networks. *ACM/IEEE Transactions on Networking*, 13 (3). Pages 596-608. June, 2005.
- [16]. Biaz, S. and Vaidya, N., Discriminating Congestion Losses from Wireless Losses Using Inter-Arrival Times at the Receiver. In the *Proceedings of the IEEE Symposium ASSET '99*, (1999).
- [17]. Biaz, S. and Vaidya, N., Distinguishing Congestion Losses From Wireless Losses: A Negative Result. In the *Proceedings of the 7th International Conference on Computer Communications and Networks*, (Lafayette, LA., October, 1998).
- [18]. Brakmo, L., O'Malley, S. and Peterson, L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *ACM/SIGCOMM Computer Communications Review*, (24). Pages 24-35.
- [19]. Bresnahan, J., Link, M., Khanna, G., Imani, Z., et al., Globus GridFTP: What's New in 2007. In the *Proceedings of the First International Conference on Networks for Grid Applications*, (2007).
- [20]. Cen, S., Cosman, P. and Voelker, G. End-to-end Differentiation of Congestion and Wireless Losses. *IEEE/ACM Transactions on Networking*, 11 (5). 703-717. October, 2003.
- [21]. D'Alessandro, G. and Politi, A. Hierarchical Approach to Complexity with Applications to Dynamical Systems. *Physical Review Letters*, 64 (14). 1609-1612. April 1990.
- [22]. Dickens, P., A High Performance File Transfer Mechanism for Grid Computing. In the *Proceedings of the The 2002 Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, (Las Vegas, Nevada, 2002).
- [23]. Dickens, P., A Workstation-Based Direct Execution Simulator. In the *Proceedings of the The 11th Workshop on Parallel and Distributed Simulation*, (1997).
- [24]. Dickens, P., FOBS: A Lightweight Communication Protocol for Grid Computing. In the *Proceedings of the Europar 2003*, (2003).

- [25]. Dickens, P. and Gropp, B., An Evaluation of Object-Based Data Transfers Across High Performance High Delay Networks. In the *Proceedings of the The 11th Conference on High Performance Distributed Computing*, (Edinburgh, Scotland, 2002).
- [26]. Dickens, P., Gropp, B. and Woodward, P., High Performance Wide Area Data Transfers Over High Performance Networks. In the *Proceedings of the 2002 International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, (2002).
- [27]. Dickens, P., Heidelberger, P. and Nicol, D., A Distributed Memory LAPSE: Parallel Simulation of Message-Passing Programs. In the *Proceedings of the The 8th Workshop on Parallel and Distributed Systems*, (1994), 32-38.
- [28]. Dickens, P., Heidelberger, P. and Nicol, D. Parallel Simulation of Multicomputer Programs. *ICASE Research Quarterly*, 3 (2). June 1994.
- [29]. Dickens, P., Heidelberger, P. and Nicol, D. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems*, 7 (10). 1090-1105. October 1996.
- [30]. Dickens, P., Heidelberger, P. and Nicol, D., Parallelized Network Simulators for Message-Passing Parallel Programs. In the *Proceedings of the MASCOTS '95. The Third International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, (1995), IEEE Computer Society Press.
- [31]. Dickens, P., Larsen, J. and Nicol, D., Diagnostics for Causes of Packet Loss in a High Performance Data Transfer System. In the *Proceedings of the 2004 IPDPS Conference: The 18th International Parallel and Distributed Processing Symposium*, (Santa Fe, New Mexico, 2004).
- [32]. Dickens, P. and Larson, J., Classifiers for the Causes of Data Loss Using Packet-Loss Signatures. In the *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID '04)*, (2004).
- [33]. Dickens, P., Nicol, D., Reynolds, P. and Duva, J. Analysis of Bounded Time Warp and a Comparison With YAWNS. *ACM Transactions on Modeling and Computer Simulation*, 6 (4). 297-320. October 1996.
- [34]. Elsner, J. and Tsonis, A. Complexity and Predictability of Hourly Precipitation. *Journal of the Atmospheric Sciences*, 50 (3). 400-405.
- [35]. Floyd, S. Modifying TCP's Congestion Control for High Speeds  
<http://www.aciri.org/floyd>
- [36]. Fonseca, N. and Crovella, M., Bayesian Packet Loss Detection for TCP. In the *Proceedings of the IEEE Infocom2005*, (2005).
- [37]. Gu, Y. and Grossman, R., Optimizing UDP-based Protocol Implementations. In the *Proceedings of the Third International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet 2005)*, (Lyon, France, 2005).
- [38]. Gu, Y. and Grossman, R., Supporting Configurable Congestion Control in Data Transport Services. In the *Proceedings of the SC05*, (2005).
- [39]. Gu, Y. and Grossman, R. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51 (7). 1777-1799. May, 2007.
- [40]. Gu, Y., Hong, X. and Grossman, R.L., Experiences in Design and Implementation of a High Performance Transport Protocol. In the *Proceedings of the SC 2004*, (Pittsburgh, PA).
- [41]. Hacker, T., Noble, B. and Athey, B., Improving Throughput and Maintaining Fairness using Parallel TCP. In the *Proceedings of the IEEE INFOCOM '04*, (2004).
- [42]. Handley, M., Floyd, S., Padhye, J. and Widmer, J. [RFC 3448] TCP Friendly Rate Control (TFRC): Protocol Specification.  
<http://community.roxen.com/developers/idocs/rfc/rfc3448.html>
- [43]. Hao, B.-L. *Elementary Symbolic Dynamics and Chaos in Dissipative Systems*. World Scientific, 1988.
- [44]. He, E., Leigh, J., Yu, O. and DeFanti, T., Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. In the *Proceedings of the IEEE Cluster Computing*, (Chicago, Illinois, 2002).

- [45]. Hegde, S., Lapsley, D., Wydrowski, B., Lindheim, J., et al., FAST TCP in High Speed Networks: An Experimental Study. In the *Proceedings of the GridNets*, (San Jose, CA, 2004).
- [46]. Herrera-Alonso, S., Rodriguez-Perez, M., Suarez-Gonzalez, A., Fernandez-Veiga, M., et al. Improving TCP Vegas Fairness in Presence of Backward Traffic. *IEEE Communications Letters*, 11 (3). March, 2007.
- [47]. Jacobson, V., Braden, R. and Borman, D. RFC [1323] TCP Extensions for High Performance., 1992.
- [48]. Jin, C., Wei, D., Low, H., Buhrmaster, G., et al. FAST TCP: From Theory to Experiments. *IEEE Network*. January/February 2005.
- [49]. Jin, C., Wei, D., Low, H., Buhrmaster, G., et al. FAST TCP: From Theory to Experiments, submitted to IEEE Communications, 2003.
- [50]. Kelly, T. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *Computer Communication Review*, 32 (2). April, 2003.
- [51]. Koo, J., Mun, S. and Choo, H. TCP WestwoodVT: A Novel Technique for Discriminating the Cause of Packet Loss in Wireless Networks *NETWORKING 2007, LNCS 4479, pages 391-402*, 2007.
- [52]. Liu, J. and Crovella, M., Using Loss Pairs to Discover Network Properties. In the *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001*, (San Francisco, California, 2001).
- [53]. Liu, J., Matta, I. and Crovella, M., End-To-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment. In the *Proceedings of the Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt '03)*, (Sophia-Antipolis, France, 2003).
- [54]. LMBench.  
<http://www.bitmover.com/lmbench/>
- [55]. Mathis, M., Heffner, J. and Reddy, R. Web100: Extended TCP instrumentation for research, education and diagnosis. *ACM Computer Communications Review*, 33 (3). July 2003.
- [56]. Ostermann, S., Allman, M. and Kruse, H., An Application-Level Solution to TCP's Satellite Inefficiencies. In the *Proceedings of the Workshop on Satellite-Based Information Services (WOSBIS)*, (1996).
- [57]. Postel, J. and Reynolds, J. RFC 959- File Transfer Protocol.  
<http://www.w3.org/Protocols/rfc959/>
- [58]. Sivakumar, H., Bailey, S. and Grossman, R., Pockets: The Case for Application-Level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In the *Proceedings of the Super Computing 2000 (SC2000)*, (2000).
- [59]. Vinkat, R., Dickens, P. and Gropp, B., Efficient Communication Across the Internet in Wide-Area MPI. In the *Proceedings of the The 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, (Las Vegas, Nevada, 2001).
- [60]. Wefel, P. Personal Communication, 2008.