Random Drop Congestion Control

Allison Mankin

The MITRE Corporation
7525 Colshire Drive
McLean, VA 22102
mankin@gateway.mitre.org

ABSTRACT

Gateways in very high speed internets will need to have low processing requirements and rapid responses to congestion. This has prompted a study of the performance of the Random Drop algorithm for congestion recovery. It was measured in experiments involving local and long distance traffic using multiple gateways. For the most part, Random Drop did not improve the congestion recovery behavior of the gateways A surprising result was that its performance was worse in a topology with a single gateway bottleneck than in those with multiple bottlenecks. The experiments also showed that local traffic is affected by events at distant gateways.

1. INTRODUCTION

Gateways in very high speed internets will need to have low processing requirements and rapid responses to congestion. This paper reports on a set of experiments on the potential of Random Drop, a statistical congestion control algorithm that is stateless and operates with minimal processing. This algorithm was originally proposed by Jacobson [1]. The performance of Random Drop was measured on a multiple-network testbed over three different gateway topologies. The measurement approach, PDU tracing, used extensive, but efficient, internal tracing of packets for direct examination of the gateway queue dynamics. This approach offered insights into the interaction of a series of gateways dropping packets.

For the most part, Random Drop did not improve the congestion recovery behavior of the gateways. Surprisingly, the performance was worse in a topology with a single gateway bottleneck than in those with multiple bottlenecks. Increasing numbers of congested gateways had measurable effect on throughput in an expected manner, and on local traffic delays in a less expected manner. The experiments suggested that local delays were affected by traffic at distant gateways.

1.1. Stateless Gateways

In the present Internet, gateways between networks maintain no state about the traffic flowing through them. Without reference to their source or destination, IP packets to be forwarded are processed as soon as the processor is available for them, and they are placed on the appropriate outgoing transmission link as soon as it is free. While they wait for the processor or the link, they are held in a queue, typically one that has a size limit and has a simple first-come-first-serve discipline. When the size limit is reached, each new arrival to the queue is discarded. These features reflect the following original Internet goals:

1. The independence of senders from the gateways they use, so that packets from one transport connection in theory can go over different gateway paths, or transparently be moved to a new gateway path if the old one fails.
2. The freedom of the gateways from guaranteeing service, and the expectation that the transport protocols at the sender should recover from losses and highly variable delay.
3. The need for gateway software to be simple, since there are many fewer gateways than hosts, and consequently gateways usually have comparatively large amounts of traffic to process.

This research was sponsored by the Defense Communications Agency under Contract F 19628-89-C-0001.

There has been some re-evaluation of the first two of these goals, as plans are made for the next generation Internet based on very high speed transmission technology [2]. End-point transparent paths and end-point reliability do not fit well with new models of the very high speed internet. However, the third, the need for the fastest possible processing of packets, is even more pressing as the volume of traffic and the number of different traffic flows processed by each gateway increase in high-speed internetworking.

1.2. Random Drop Concepts

The concepts of Random Drop were originally proposed in meetings of the Internet Engineering Task Force [1]. The statistical concept of Random Drop is that a packet randomly selected from all traffic passing through the gateway belongs to a particular connection with a probability matching that connection's proportion of the traffic. The control concept is that dropping random packets from connections sending too much traffic can reduce the total steady state traffic of the gateway.

The appeal of the statistical concept is that the gateway's connections are distinguished without the overhead of keeping track of individual connections. Alternatives require identifying the connection for every packet. A potentially large table of connection state information has to be maintained, since the number of connections whose traffic needs to be tracked grows exponentially with the number of endpoints served simultaneously by the gateway. Random sampling would be an attractive alternative to maintaining this state.

Random Drop samples the population using the gateway as follows: the sample size is some number of arrivals, regardless of how long a time they span. Each round of Random Drop requires one generation of a random number, followed by simple packet counting. The jth packet (j is the random number) is selected from a prespecified number N of arrivals. Once N packets have arrived, a new j is drawn. The result is a uniformly applied $1/N$ chance of being selected. If a connection has p packets in the sample of N, it has an p/N probability of having its packet selected; if it has only one packet in the same sample, it has only a $1/N$ chance of being selected.

The control concept says that dropping appropriate packets is sufficient for gateway congestion control. When the source detects its packet is dropped, this is used as a signal to carry out an endpoint congestion control algorithm, such as Slow-start in TCP [3]. Our previous measurement study [4] shows how the response to dropped packets by Slow-start cannot clear the gateway under persistent overload. Jacobson proposed Random Drop as a way of sending an earlier slow-down signal for Slow-start TCP, in hopes of improving this behavior, in other words, for congestion avoidance. The randomly selected packets would be dropped when the gateway was starting to experience overload.

Several parameters would be needed to carry out a Random Drop Congestion Avoidance algorithm, principally, the rate of drop (a dynamically determined N), and the function for varying it in time. These must be related to many performance factors in the gateway: whether the gateway is moving toward congestion, and how quickly; the number of connections that send at too great a rate; the time it takes the drop signal to reach senders; and the responsiveness of senders to the drop signal. The premise of Random Drop Congestion Avoidance is that these factors can be combined to compute an N, and possibly a number of random draws to make within the N (not always 1). Then the drop mechanism, operated at the computed rate, would bring about control.

Internet researchers have pointed out problems with Random Drop for congestion avoidance. Random Drop's control concept of dropping packets may be subject to too much "inertia" to work effectively [5]. The computation of the drop rate and the passage of the dropped packet signal to users both entail delay. As stated in Reference 5:

> Even long after the switch enters the dropping region, packets keep coming at the same speed as if there were no dropping, resulting in most connections, including the well-behaved ones, getting packets lost.

When used in advance of the necessity of dropping, the added probability of drop at each gateway may causes excessive loss for connections traversing many gateways.

2. RANDOM DROP CONGESTION RECOVERY

As a result of the negative arguments summarized, we chose to experiment with Random Drop only for congestion recovery, and not for congestion avoidance.

Random Drop Congestion Recovery (RDCR) is an enhancement for packet dropping when the gateway is congested enough for a queue to overflow. Congestion avoidance (CA) algorithms, such as the binary feedback scheme [6], would prevent the occurrence of such overflows most of the time, but the Internet does not currently have any CA scheme. Even if effective CA was in place, however, congestion calling for recovery mechanisms would still occur when there was a sudden increase in the network load or decrease in the available capacity.

In RDCR, instead of dropping the last packet to arrive at the overflowing queue, a randomly chosen packet from the queue is dropped. The mechanism uses uniform random number generation and packet counting as described earlier, but N (the sample size controlling the probability of drop) is fixed at the maximum size of the queue.

RDCR is potentially valuable if arrival when the queue is full appears to be more likely for connections with less traffic than for connections with more. Then, because selecting the random packet identifies a heavier user, the drop for overflow is from a connection that is contributing substantially to the overflow. If there was no correlation between packets arriving at the gateway (in other words, the gateway was like an M/M/1 queue, with

Poisson arrivals and exponential service times), packets' arrival at any queue position would be equally probable for all connections. But there is correlation, such as the "packet trains" [7] and host-pair localizations [8] found in measuring operational internets. TCP arrivals at the gateway are far from independent of each other. The window means past delays and (for Slow-start) past congestion determine the gateway arrival times of current packets.

Simulations of Random Drop Congestion Recovery by other researchers have indicated that RDCR does not control short and long round trip time connections proportionately [5,9]. On the other hand, random selection of packets to receive Source Quench has been shown to improve the fairness of an IP rate reduction algorithm [10]. In particular, it produced nearly perfect throughput fairness among equal senders. The use of random selection with Source Quench is one of the major recommendations made by Reference 10. That the throughput is close to equal for equal connections is visible in the throughput graphs in Reference 5 as well. Our measurements of RDCR confirm and extend these results.

3. IMPLEMENTATION

RDCR was implemented in the BSD 4.3 UNIX* kernel on MicroVax II** systems. In this implementation, a set of uniformly distributed random numbers between zero and the maximum queue occupancy is precomputed at regular intervals. Whenever a packet must be dropped, the packet in the queue position corresponding to the number is deleted and the arriving packet is enqueued. The chance for the arriving packet to be dropped is retained. A number selection of zero means drop the head packet on the queue. The queue maximum number means drop the currently arriving, not yet enqueued packet.

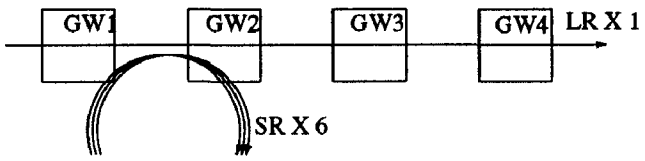
Care was taken to minimize the operations for computing the number of the packet to be dropped, since this low overhead was the major attraction of Random Drop Congestion Recovery. But since the algorithm operates only when packets await link resources, and therefore the processor is not the bottleneck during execution, we did not change the BSD queue data structure to optimize locating the selected packet on the queue. In a production implementation, deleting the selected packet could be made as efficient as the random number generation.

The random number generator had to be devised for use in the kernel. In the kernel, particularly in the networking code, which executes at a processor level blocking other networking events, it is necessary to reduce the operations used for generating each number. The Subtractive Random Number Generator described by References 11 and 12 is low in computational requirements. Each number generation requires only one subtraction and one addition. The standard UNIX linear congruential random number generator needs at least one multiply and one addition. One modulo operation is also needed, to convert the number between 0 and 2^{29} generated by the implemented generator to a number between 0 and the queue maximum packet number, but this is also the case for the UNIX generator.

4. STUDY DESIGN

The experiments with Random Drop Congestion Recovery look at the interaction of congested gateways using the RDCR policy. All traffic through these gateways is Slow-start TCP. The size and other characteristics of the traffic load are such that there is persistent congestion; that is, until most connections have completed their sending, there is no chance for the gateways to clear their queued packets completely. The load selected is enough to maintain this state, but to avoid an infinite queueing overflow. This is done with the following parameters: the maximum burst arrivals at the gateway, when all the Slow-start connections have their windows fully open, is fifty-six packets (seven connections times eight packets); each gateway output queue is configured to a maximum of thirty packets; each connection traverses at least two of the gateways, so that its window of packets is at times spread over two or more queues.

The traffic is divided into two classes, Short Round-Trip, or SR, and Long Round-Trip, or LR. The SR class provides a background of load on the gateways. In each SR group, six nearly equal Slow-start TCP connections run between hosts that are two gateway hops apart. In the LR class, a single Slow-start TCP connection runs across more than two gateway hops. The SR connections and the LR connection are cross-traffic to each other relative to the gateway resources. Figure 1 illustrates the two classes.



SR = SHORT ROUND TRIP TRAFFIC CLASS

LR = LONG ROUND TRIP TRAFFIC CLASS

Figure 1: Traffic Classes Example

For the purposes of the experiment, Source Quench was turned off in the gateways. As a result, all connections waited until retransmission timeout before closing their congestion window following a drop; this probably increased the "inertia." The window size of connections in both classes, SR and LR, had a fixed maximum of four kilobytes (eight maximum segment size packets). With a modification of the destination discard servers, the LR connections could have grown their window beyond this, but the convention of a conservative window in TCP was

* Trademark of AT&T Bell Laboratories.

** Trademark of Digital Equipment Corporation.

followed.

4.1. Testing Environment

The experiments were carried out using the Defense Communications Agency Internet Engineering Net, a testbed recently established by MITRE for the Defense Communications Engineering Center. The gateways were MicroVax II systems linked by RS-232 connections run at 19.2 kilobits per second speeds, and passing data using the Serial Line IP (SLIP) protocol. The hosts were a mixture of MicroVax II's and Sun 3/60's. To minimize the coordination of traffic by host processing, each host was the source or destination of no more than two connections, and each host-pair was unique.

The RDCR algorithm was tested with three multi-gateway topologies. These had four or six gateways, one LR-class connection traversing all the gateways, and one, two or three SR-class groups of six connections. All three topologies are illustrated in Figure 2. An 'X' symbol marks the links (bottlenecks) at which outbound queue congestion developed during the experiments.

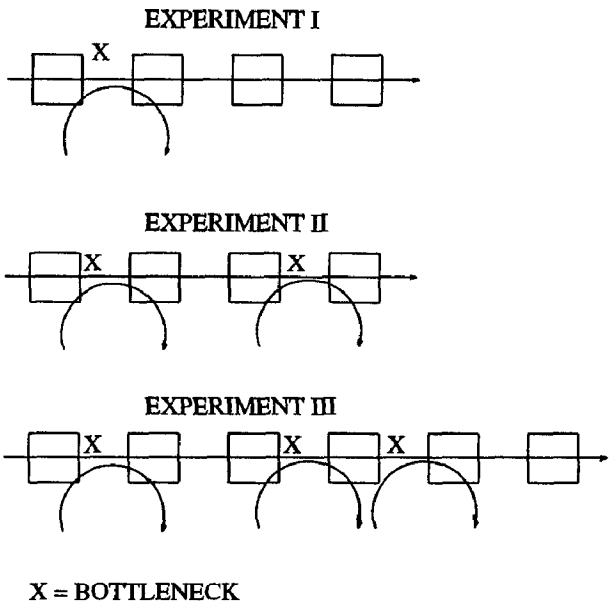


Figure 2: Multiple Gateway Topologies

In the first topology (Single Bottleneck), there was one SR-class group using Gateways 1 and 2. The LR-class connection traversed four gateways. The bottleneck in this configuration was the SLIP link between Gateways 1 and 2.

In the second (Two Bottleneck), two groups of SR-class connections were used, one traversing Gateways 1 and 2, the other Gateways 3 and 4. The LR-class connection again traversed four gateways. The two bottlenecks in this configuration were at the SLIP links between Gateways 1 and 2, and Gateways 3 and 4.

Finally, the Three Bottleneck topology added two more gateways and a third SR group. The LR-class connection traversed six gateways. The third SR group traversed Gateways 4 and 5 (not 5 and 6, since it was desired to make a case less balanced in the load at the bottlenecks than the other two cases). The bottlenecks were in Gateways 1, 3 and 4 in this configuration.

Slow-start TCP connections for both classes were produced using MITRE's *traffgen* tool [4]. In all the experiments, the LR connection was allowed to become established briefly before the SR connections were started. The startup of the SR groups was synchronized, but within each group, connections began with a slight randomness (of three to five seconds). This randomness ensured that all connections in the experiment became established and started sending data within a few seconds of each other. Too closely synchronizing the start causes one or more connections to lag. Each connection transmitted three hundred kilobytes of data from memory to discard servers across the experimental gateways. With these data amounts and execution methods, the execution of each experiment took over six hundred seconds.

4.2. Measurement Methods

The performance of the connections was measured by tapping the LAN and by internal monitoring in the gateway, using our *NETMON/iptrace* tools. Reference 13 fully documents *NETMON* and methods for using it. According to both measurements and estimates, *NETMON* adds 5% or less overhead per packet, largely in copying part of the packet header packet header into the trace buffers [13]. *NETMON* produces timestamped traces of all queue arrivals, departures, and overflows. In addition, when RDCR was on, *NETMON* trace calls recorded which packets arrived in overflow position and which packets were dropped in their stead.

5. RESULTS AND ANALYSIS

PDU traces were collected during the experiments at each gateway using our *NETMON/iptrace* software. Our *tcptrace* [4] tool collected timestamped TCP headers on each Ethernet that had data sources. Since the data sets from each collection point (individual gateway or Ethernet) have independent frames of reference, that is, the tracing timestamps of the individual gateways and origin Ethernets are not synchronized, care was taken to analyze these sets independently.

The measures of performance derived from the data (and discussed in the results) are the probability of drop, the rate of arrival in overflow position, the queueing delay, throughput, and throughput fairness. The gateway measures, drop rate and delay, are derived individually for the LR connection and collectively for the SR groups, though individual SR results could also have been obtained from the data. Only data from the steady-state time periods of each experiment were included in computing the summary statistics. The interval analyzed was 360 seconds.

The direct measurement of gateway queueing delays is a noteworthy feature of this study. The gateway measurements (of the queues, and the times and locations

of congestion recovery actions) verified that the bottlenecks occurred as planned. Data was analyzed for all the output and input queues. Delays at the non-bottleneck queues were minimal, averaging a few hundredths of a second. One noteworthy confirming measurement was that the traffic load of acknowledgements, flowing in the reverse direction to the data, also did not cause significant queueing or delays exceeding a few hundredths of a second. In the experiment results, only measures at the bottlenecks, Gateways 1, 3 and 4, are shown.

Tables 1-5 show the summary statistics for the three experiments in the study, covering probabilities of drop and overflow position, queueing delays, throughput, and throughput fairness.

5.1. Probability of Drop

The probabilities of drop for LR when Random Drop Congestion Recovery is off show that the traffic of the least frequent sender is indeed disproportionately likely to be in the overflow position compared with other senders (Table 1). The fact that LR is the least frequent sender during steady state can be confirmed by looking ahead to the throughputs shown in Table 4. RDCR improves LR's chances in the multibottleneck experiments, but the SR connections still have lower drop rates, even though they occupy more of each queue. In the Single Bottleneck experiment, RDCR worsens LR's probability of drop. The contrast between the single bottleneck and multiple bottleneck results is discussed at the end of this section.

Table 1: Probability of Drop (Percent)

Experiment	GW1	GW3	GW4
I LR On	5.56	0.00	0.00
I LR Off	3.87	0.00	0.00
I SR On	3.72	--	--
I SR Off	3.51	--	--
II LR On	8.33	8.51	0.00
II LR Off	22.73	17.65	0.00
II SR On	3.46	3.49	--
II SR Off	2.97	2.72	--
III LR On	9.52	0.00	5.00
III LR Off	20.00	16.67	0.00
III SR On	2.81	3.08	3.17
III SR Off	2.79	2.34	3.17

5.2. Overflow Position Arrivals

In the experiments with RDCR on, a related measure was the overflow position arrival rate of LR. These rates are shown in Table 2. The rate of LR overflow position arrivals increases with the number of bottlenecks.

Table 2: Overflow Position Arrivals (Percent) for LR in RDCR

	GW1	GW3	GW4
Exp. I	3.17	0.00	0.00
Exp. II	14.58	8.51	0.00
Exp. III	23.81	9.52	5.00

5.3. Queueing Delays

Table 3 presents the measured queueing delays at each of the bottleneck links during the analyzed 360 second interval of steady-state. The delay at a bottleneck is consistently lower for LR with RDCR than without. With RDCR, packets are on the queue for a time before they are dropped, while without it dropped packets spend no time waiting. In averaging the delays, the time on the queue of packets randomly dropped was not included. Even without averaging in their short waiting times, the dropping of these packets is the reason for the delay decrease brought about RDCR, because their presence for their shortened queue occupancy times mean shortened waiting times for the undropped packets behind them.

The delays of the SR connections increase with the number of bottlenecks. This is an odd result, because the SR connections each traverse only one bottleneck.

Table 3: Queueing Delay/Standard Deviation (Secs)

Experiment	GW1	GW3	GW4
I LR On	6.85/0.929	--	--
I LR Off	7.03/1.204	--	--
I SR On	7.05/0.965	--	--
I SR Off	7.25/1.156	--	--
II LR On	6.59/1.103	7.03/1.479	--
II LR Off	6.86/1.14	7.40/1.059	--
II SR On	6.92/1.206	7.14/1.208	--
II SR Off	7.36/1.118	7.33/1.084	--
III LR On	7.40/0.749	7.13/1.093	8.55/0.777
III LR Off	7.51/0.896	7.78/0.442	9.48/0.407
III SR On	7.29/0.862	7.23/0.888	8.57/1.320
III SR Off	7.60/0.837	7.60/0.911	8.80/1.156

5.4. Throughputs

With respect to LR, throughput in this study offers a dismal view of Slow-start TCP traversing multiple congested gateways. The throughput of LR is slightly improved by RDCR in the multibottleneck experiments, but it is dauntingly low either way.

The maximum throughput for a TCP connection is on the order of:

WINDOW/RTT

Just looking at the gateway queueing delays, the LR connection should have throughputs proportional to those of SR:

I: LR = 1 X SR
II: LR = 2 X SR
III: LR = 3 X SR

A ceiling on the throughput of LR in the multibottleneck experiments would be 1/2-1/3 of each SR connection's. We see from Table 4 that LR's throughputs are considerably lower. This is an effect of packet drops, necessitating retransmissions. With the long round-trip time of LR, the retransmissions are slow to come.

Table 4: Avg Connection Throughput (Bytes/Sec)

Class	Exper. I	Exper. II	Exper. III
LR On	184	75	35
LR Off	229	30	17
SR On	280	288	281
SR Off	258	265	284

5.5. Throughput Fairness

As the simulation studies [5, 9, 10] predict, the measurements from our experiments show a benefit to equal senders from randomized drops at congestion. This effect is consistent whether or not RDCR also benefits the LR connection.

The fairness imposed by RDCR on the SR throughputs is especially clear when quantified by the following fairness metric [14]:

$$F = \left[\frac{(\sum_{i=1}^{n} x_i)^2}{n(\sum_{i=1}^{n} x_i^2)} \right]$$

where x_i = ith connection's throughput

n = number of connections

Table 5 presents the computed throughput fairness of the SR connections.

Table 5: SR Throughput Fairness (Maximum Fairness = 1.0)

	Exper. I	Exper. II	Exper. III
On	0.99	0.98	0.97
Off	0.79	0.79	0.88

5.6. More on the Single Bottleneck Results

The figure of merit for RDCR is the probability of drop. In the simplest test configuration, with a single bottleneck, RDCR *increases* the probability of drop of the long round-trip time connection. We have to ask if, in the single bottleneck case, the number of LR packets in the queue at any given time is comparable to the numbers from the SR connections. The LR throughput is indeed close to that of SR. With the single bottleneck, the overflow position arrival rate of LR is low. Passing through multiple bottlenecks increases how often LR packets arrive to find a full queue, and therefore increases their benefits from RDCR.

6. CONCLUSIONS

Random Drop Congestion Recovery improves the fairness of homogeneous connections that have the same bottleneck, but beyond that, it has limited value. It reduces only somewhat the disproportionate probability that long round-trip time traffic is dropped during congestion recovery, The harm that any drops cause the LR traffic is seen in its low throughputs in the multibottleneck experiments. Moreover, from the results of the Single Bottleneck experiment, RDCR may worsen the performance of long round-trip time traffic.

There are interactions among internet gateways, despite their lack of explicit intercommunication. Multiple bottlenecks exert cumulative influence, and they need not be balanced to do so, as the three bottleneck experiment, with extra high loads at the final bottleneck, shows. As might be expected, local traffic affects the gateway's through traffic performance, but the reverse is also true. Our measurements show how the queueing delays of the local traffic increase when there is a significant downstream bottleneck for the through traffic. This measurable impact of through traffic on local traffic means that connections can be affected by events at gateways distant from them. The role of the through traffic in creating performance interactions across gateways and localities should be further studied. Gateway congestion control and performance management algorithms should be evaluated in topologies that have both local and through traffic. Finally, a single bottleneck configuration, though easiest to measure, should not be used to predict algorithm performance.

Based on the throughput fairness results, RDCR appears to be an ideal supplement to congestion avoidance schemes that group traffic together more or less by round trip time, such as the binary selective feedback scheme [6]

and Fair Queueing [15]. These are not stateless so the attractions of the statelessness of Random Drop are no longer primary. Rather, the primary benefit is that Random Drop provides a completely uniform environment.

Researchers are just beginning to consider whether randomizing algorithms beyond RDCR, for instance, Stochastic Fairness Queueing [16], will pay off in Internet gateway performance. Whatever these algorithms, measurements will be needed and some of the revealed effects of the algorithms will be unexpected.

REFERENCES

[1] Jacobson, V. (April, 1989), reported in "Minutes of the Performance Working Group", *Proceedings of the Cocoa Beach Internet Engineering Task Force*, Reston, VA: Corporation for National Research Initiatives.

[2] Parulkar, G. (January, 1990), "The Next Generation of Internetworking", *ACM Computer Communications Review*, Vol. 20, Number 1.

[3] Jacobson, V. (1988), "Congestion Control and Avoidance", *SIGCOMM '88, ACM Computer Communications Review*, Vol. 18, Number 4.

[4] Mankin, A. and K. Thompson (February, 1989), "Limiting Factors in the Performance of Slow-start TCP", *Conference Proceedings of the Winter 1989 USENIX in San Diego.*

[5] Zhang, L. (1989), *A New Architecture for Packet Switching Network Protocols*, Ph.D Thesis, Massachusetts Institute of Technology, Department of Computer Science.

[6] Ramakrishnan, K.K., D.M. Chiu and R. Jain (1987), *A Selective Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer*, DEC TR-510, Littleton, MA: Digital Equipment Corporation.

[7] Jain, R. and S. Routhier (September, 1986), "Packet Trains—Measurements and a New Model for Computer Network Traffic", *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 6.

[8] Feldmeier, D. (1988), *Estimated Performance of a Gateway Routing-Table Cache*, MIT Lab. for Comp. Sci. Technical Memorandum MIT/LCS/TM-352.

[9] Hashem, E. (1990), *Random Drop Congestion Control*, M.S. Thesis, Massachusetts Institute of Technology, Department of Computer Science.

[10] Finn, G. (October, 1989), "A Connectionless Congestion Control Algorithm", *ACM Computer Communications Review*, Vol. 19, Number 5.

[11] Knuth, D. (1983), *The Art of Computer Programming Vol. 2—Seminumerical Algorithms*, Reading, MA: Addison-Wesley, pp. 171-172.

[12] Abrahams, P. (August, 1989), "Technical Correspondence—Random Number Generators and the Minimal Standard", *Communications of the ACM*, Vol. 32, No. 8, pp. 1021-1022.

[13] Mankin, A. (1989), *Design and Implementation of An Instrumented Gateway.* MTR-88W00238, McLean, VA: The MITRE Corporation.

[14] Jain, R., D.M. Chiu and W. Hawe (1984), *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Systems.* DEC TR-301, Littleton, MA: Digital Equipment Corporation.

[15] Demers, A., S. Keshav and S. Shenker (1989), "Simulation and Analysis of a Fair Queueing Algorithm", *SIGCOMM '89, ACM Computer Communications Review*, Vol. 19, Number 4.

[16] McKenney, P. (1990), "Stochastic Fairness Queueing", To appear in *Proceedings of INFOCOM '90.*

ACKNOWLEDGEMENTS

The author thanks the Internet Engineering Task Force Performance and Congestion Control Working Group, and especially Scott Shenker, for discussions of Random Drop. K.K. Ramakrishnan, Lixia Zhang, and the anonymous reviewers all raised helpful points about the analysis of the measurement data.